**Universidade Federal de Santa Catarina**
**Departamento de Informática e Estatística**

Quenio Cesar Machado dos Santos

# Reusando Modelos Conceituais: Linguagem e Compilador Extensível

Florianópolis

2017

Quenio Cesar Machado dos Santos

# Reusando Modelos Conceituais: Linguagem e Compilador Extensível

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação
Orientador: Prof. Dr. Raul Sidnei Wazlawick

Florianópolis

2017

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

dos Santos, Quenio Cesar Machado
Reusando Modelos Conceituais : Linguagem e Compilador Extensível / Quenio Cesar Machado dos Santos ; orientador, Raul Sidnei Wazlawick, Dr., 2017.
445 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2017.

Inclui referências.

1. Ciências da Computação. 2. Modelagem conceitual. 3. Geração de código. 4. Desenvolvimento de software dirigido por modelos. 5. Metaprogramação. I. Wazlawick, Dr., Raul Sidnei. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

Quenio Cesar Machado dos Santos

**Reusando Modelos Conceituais: Linguagem e Compilador Extensível**

Este Trabalho de Conclusão de Curso foi apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Florianópolis, 10 de julho de 2017.

---

Prof. Dr. Renato Cislaghi
Coordenador

**Banca Examinadora:**

---

Prof. Dr. Raul Sidnei Wazlawick
Orientador

Prof. Dr. Jerusa Marchi

Prof. Dr. Ricardo Azambuja Silveira

Este trabalho é dedicado a minha querida vó, a Dona Alice; mulher humilde, mas com grande visão. Com muito esforço e trabalho, foi lavando "roupa pra fora" que conseguiu formar quatro filhos na Faculdade e, por consequência, o fez com tantos netos. Apesar da sua origem sem oportunidades e de todas as adversidades da sua vida, foi capaz de melhorar a vida de todas as gerações futuras de nossa família. Obrigado, vó Lice! Sempre te guardaremos em nossos corações...

## Agradecimentos

Agradeço ao meu orientador, Professor Raul Sidnei Wazlawick pelas inumeráveis revisões, sugestões e contribuições que foram indispensáveis na realização deste trabalho e que o enriqueceram enormente, permitindo até sua publicação.

Agradeço ao Professor Ricardo Azambuja Silveira que, desde as primeiras aulas de Construção de Compiladores, incentivou-me a realizar este trabalho e ajudou-me nas fundações necessárias para a sua implementação.

Agradeço à Professora Jerusa Marchi pela motivação, pelo incentivo, pelas correções e pelo valioso feedback, os quais resultaram numa melhora signiticativa deste trabalho.

Agradeço ao Professor Rosvelter Coelho da Costa, que abriu meus olhos sobre Model-Driven Development e assim me colocou no caminho certo quanto à pesquisa dos trabalhos correlatos.

Agradeço aos meus pais por terem me incentivado desde os meus 12 anos de idade a seguir esta carreira na área da Computação, que tanto me realiza.

Agradeço à minha meiga e adorável esposa, e aos meus amados filhos, pelo apoio e pela compreensão nas minhas ausências durante os últimos quatro anos em que estive tão ocupado com este Curso.

Agradeço ao meu gerente, e à empresa em que trabalho, pela flexibilidade e pelo suporte, sem os quais teria sido impossível realizar este sonho de me formar.

Enfim, agradeço à minha família e aos meus amigos que sempre torceram por mim e estiveram ao meu lado.

"O movimento na direção de abordagens dirigidas por modelo é na verdade o mesmo fenômeno que já vem ocorrendo há décadas – o movimento para níveis de abstrações cada vez mais altos."

(Stephen W. Liddle, Model-Driven Software Development, 2011)

**Resumo**

Este relatório apresenta uma linguagem textual para modelagem conceitual (baseada em classes/associações da UML e em restrições da OCL) e um compilador que pode gerar código em qualquer linguagem ou tecnologia através de *templates* de texto extensíveis. A linguagem e o compilador permitem a especificação da informação gerenciada por sistemas de software cada vez mais distribuídos e em constante mudança. A partir de uma única fonte, a geração de código automática mantém as implementações consistentes com sua especificação através das diferentes plataformas e tecnologias. Além disso, na medida em que o horizonte tecnológico se expande, os *templates* textuais podem ser modificados para adotar novas tecnologias. Diferentemente de outras abordagens, tais como MDA e MPS, espera-se que o suporte ferramental acompanhando esta linguagem, juntamente com sua natureza textual, facilite a integração do desenvolvimento de software dirigido por modelos no fluxo de trabalho dos desenvolvedores de software.

**Palavras-chave:** Modelagem conceitual. Geração de código. Desenvolvimento de software dirigido por modelos. Engenharia dirigida por modelos. Metaprogramação. Programação generativa.

**Abstract**

This report presents a textual language for conceptual modeling (based on UML classes/associations and OCL constraints) and a compiler that can generate code in any target language or technology via extensible textual templates. The language and compiler allow the specification of information managed by ever-changing, increasingly distributed software systems. From a single source, automated code generation keeps implementations consistent with the specification across the different platforms and technologies. Furthermore, as the technology landscape evolves, the target templates may be extended to embrace new technologies. Unlike other approaches, such as MDA and MPS, the built-in tooling support, along with the textual nature of this modeling language and its extensible templates, is expected to facilitate the integration of model-driven software development into the workflow of software developers.

**Keywords:** Conceptual modeling. Code generation. Model-driven software development. Model-driven engineering. Metaprogramming. Generative programming.

## Lista de Figuras

## Lista de Tabelas

**Sumário**

# 1 INTRODUÇÃO

O desenvolvimento de um sistema de software se dá em parte pela codificação do seu domínio. Como ilustrado de forma simplificada na figura 1, esta codificação é proveniente de uma conceitualização do domínio que se encontra na mente do desenvolvedor do sistema.

Na realidade atual, os sistemas de software são bem mais complexos do que ilustrado na figura 1. O desenvolvimento de um sistema típico pode envolver dezenas ou centenas de desenvolvedores, que codificam um número cada vez maior de componentes distribuídos pela infra-estrutura tecnológica das organizações e da Internet.

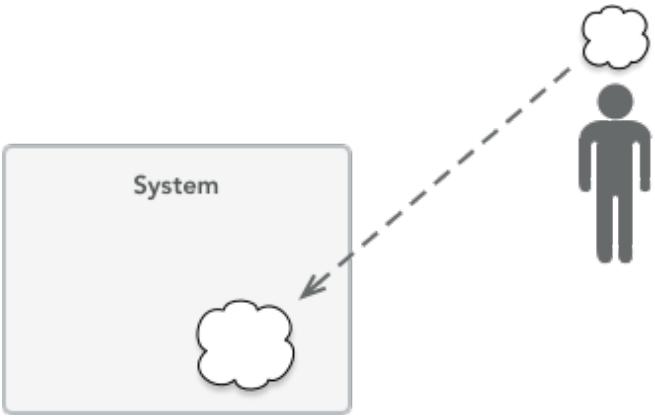


Figura 1 – Sistema Simples

Como ilustrado na figura 2, não somente o próprio sistema é distribuído, mas também a sua conceitualização é distribuída pelas mentes de um grande número de desenvolvedores e por todos os componentes do sistema. Isto gera um enorme custo de desenvolvimento e manutenção, a fim de manter estas conceitualizações em sincronia e atualizadas.

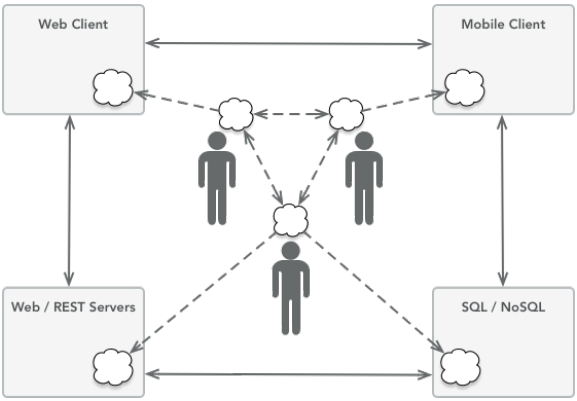


Figura 2 – Sistemas Atuais

Para resolver este desafio imposto por sistemas de software cada vez mais distribuídos e em constante mudança, *Model-Driven Architecture*, ou MDA (OMG, 2014a), é uma iniciativa que tem sido promovida pelo *Object Management Group* (OMG). Como ilustrado na figura 3, MDA tem guiado o uso de modelos de alto nível – criados com padrões do próprio OMG, como *Unified Modeling Language* (UML) (OMG, 2015a), *Object Constraint Language* (OCL) (OMG, 2014b) e *Meta Object Facility* (MOF) (OMG, 2016) – para produzir artefatos de software e implementações através de tranformações automatizadas.

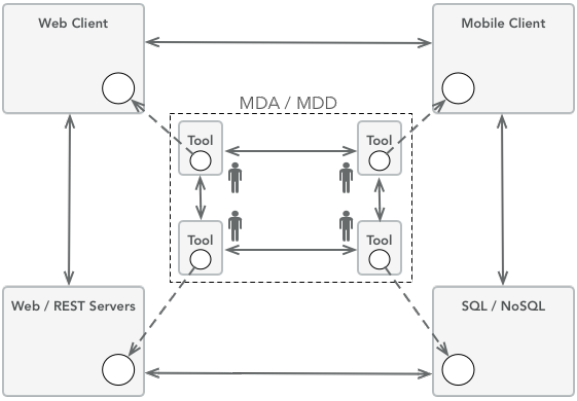


Figura 3 – Model-Driven Architecture / Development - MDA / MDD

Como uma de suas propostas de valor, o guia do MDA (OMG, 2014a) advoga que:

> "Automação reduz o tempo e o custo de se efetuar um design, reduz o tempo e o custo de mudanças e da manutenção e produz resultados que garantem a consistência em todos os artefatos derivados. Por exemplo, manualmente produzindo todos os artefatos [...] necessários para implementar um conjunto de processos e serviços para uma organização é difícil e propenso a erros. Produzir artefatos através de um modelo é mais confiável e mais rápido. "

MDA fornece orientações e padrões para se colocar em prática esta visão, mas deixa para os fornecedores da área de desenvolvimento de software a tarefa de fornecer as ferramentas (ilustradas na figura 3) que devem automatizar o processo de geração de implementações a partir de modelos. Na prática, apesar dos padrões, os desenvolvedores acabaram dependentes dos ambientes de desenvolvimento e das ferramentas de um determinado fornecedor. *XML Metadata Interchange* (XMI) (OMG, 2015b) é o padrão da OMG que permite interoperabilidade entre ferramentas, mas, sendo baseado em *Extensible Markup Language* (XML), este não é adequado para edição de modelos em editores de texto, independemente das ferramentas. O mesmo problema afeta Metaprogramming System (MPS), apresentado por Voelter (VOELTER, 2014), que requer editores projecionais para a edição de seus modelos.

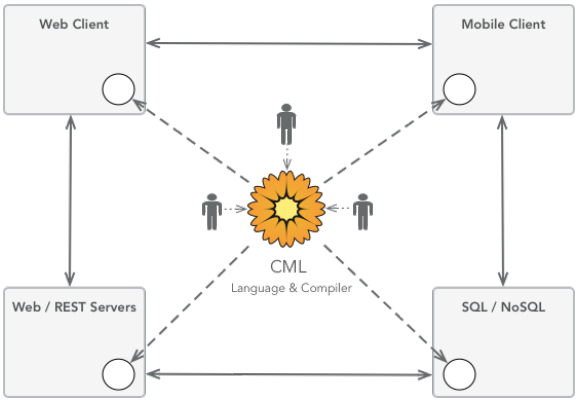


Figura 4 – CML-Driven Development

A linguagem de modelagem conceitual (CML) e o seu compilador extensível (apresentados neste relatório e ilustrados na figura 4) são conjuntamente uma alternativa a MDA e MPS. Enquanto MPS é um ambiente integrado de desenvolvimento baseado em linguagens de domínio específico (DSLs), a proposta aqui apresentada (CML) provê um compilador que tem:

- como *entrada*, arquivos-fonte escritos na sua própria linguagem, que provê uma sintaxe abstrata similar a (mas menos abrangente que) uma combinação de UML (OMG, 2015a) e OCL (OMG, 2014b).

- como *saída*, arquivos em qualquer outra linguagem, através de *templates* de texto extensíveis que podem ser fornecidos por uma biblioteca básica do próprio compilador, por bibliotecas de terceiros, ou ainda pelos próprios desenvolvedores de software.

Os arquivos de entrada representam a modelagem conceitual de um determinado sistema e os arquivos de saída são os artefatos que podem ser utilizados na implementação do sistema.

## 1.1 OBJETIVOS

De acordo com Thalheim (THALHEIM, 2011), a escolha de uma linguagem de modelagem conceitual tem a ver com seu próposito. O autor sugere que a linguagem é simplesmente um *atravessador* mapeando algumas propriedades da *origem* (o espaço do problema) que podem ter utilidade aos seus usuários para se alcançar o *destino* – ou seja, encontrar uma solução.

Neste contexto, o propósito de CML – a linguagem e compilador propostos neste trabalho – é permitir que desenvolvedores de software possam transformar modelos conceituais (representados em forma de texto) em código executável numa gama extensível de tecnologias. Para atingir este propósito, esta linguagem é concebida com os objetivos específicos apresentados nas próximas subseções.

### 1.1.1 Experiência do Desenvolvedor

CML segue o princípio estabelecido pelo manifesto de *Conceptual-Model Programming* (CMP) (EMBLEY; LIDDLE; PASTOR, 2011):

> "Modelagem conceitual é programação. A instância do modelo conceitual é código, ou seja, ao invés de 'o código é o modelo', a frase se torna 'o modelo é o código'. Um compilador de modelos conceituais assegura que a execução do programa corresponde à especificação conceitual, desta forma fazendo a instância do modelo conceitual diretamente executável."

Além disso, CML também tem a intenção de permitir desenvolvedores de software a fazer *modelagem conceitual* no mesmo fluxo de trabalho que eles estão acostumados a fazer *programação*. CML se esforça a ser *não apenas* o código (como advogado por CMP), mas também *parecer-se* com código (sintaticamente falando), procurando compatibilidade com a mentalidade, as ferramentas e o fluxo de trabalho dos desenvolvedores de software.

### 1.1.2 Evolução da Linguagem

Espera-se que CML evolua juntamente com seu compilador e as ferramentas relacionadas. Diferentemente do poder de expressidade visto em UML (OMG, 2015a) e OWL 2 (W3C, 2012), com sua grande quantidade de opções para modelagem, esta primeira versão da linguagem de CML e seu compilador intencionalmente suportam um número limitado de cenários.

As opções de modelagem disponíveis nesta primeira versão de CML suportam generalização/especialização, associações unidirecionais e bidirecionais (com cardinalidade zero-ou-um e zero-ou-mais) e a habilidade de definir atributos e associações derivados através de expressões inspiradas em OCL (OMG, 2014b).

Estas construções da linguagem CML já foram utilizadas na especificação e implementação do próprio metamodelo do compilador CML.

### 1.1.3 Geração de Código Extensível

Algumas das declarações suportadas pela linguagem CML possibilitam a geração de código numa gama ampla de linguagens e tecnologias. Entre as funcionalidades que precisam ser suportadas por esta linguagem, encontram-se a habilidade de:

- dividir modelos em vários módulos que podem ser compartilhados

como bibliotecas;

- especificar como deve proceder a geração de código para as platformas-alvo;

- e anotar os elementos do modelo para fornecer informação adicional sobre estes durante a geração de código.

Para que seja possível então dar o suporte necessário à geração de código, estas funcionalidades devem estar disponíveis numa única linguagem e devem ser compatíveis entre si e com *backend* do compilador.

### 1.1.4 Independência de Fornecedores

Como colocado na seção introdutória, MDA (OMG, 2014a), através de seus padrões, incentivou desenvolvedores de ferramentas de desenvolvimento de software a prover soluções para o desenvolvimento dirigido por modelos. Porém, os desenvolvedores acabaram dependentes dos ambientes de desenvolvimento e das ferramentas de um determinado fornecedor.

CML, sendo uma linguagem textual, cuja única dependência é em seu compilador *open-source*, pretende seguir a tradição da linguagem C, que permitiu a evolução de ferramentas e programas tanto na comunidade de desenvolvedores *open-source* – nos sistemas Unix/Linux – como no âmbito dos fornecedores comerciais – como Microsoft, IBM e tantos outros. O desenvolvimento de CML, portanto, deve garantir que seu código-fonte possa ser editado, tanto por editores de texto convencionais, quanto por ferramentas fornecidas comercialmente. Da mesma forma, o compilador CML, sendo *open-source*, serve como uma implementação de referência; e a linguagem CML, através de sua especificação (em anexo A), permite a implementação de outros compiladores e ferramentas.

Espera-se assim que os modelos conceituais criados com CML serão independentes não apenas das tecnologias e linguagens para as quais se gera código, mas também independentes das ferramentas que se utiliza para criar os modelos conceituais.

## 1.2 LIMITAÇÕES

Em termos de modelagem, esta primeira versão da linguagem CML permite modelar somente os aspectos estruturais de um sistema de informação, como caracterizado por Wazlawick no capítulo 6 do seu livro (WAZLAWICK, 2014). De forma sucinta, em CML, modela-se *conceitos*, seus *atributos* e *associações*, permitindo a definição de *derivações* através de *expressões*. Ainda nos aspectos estruturais, existem algumas limitações em CML, tais como um limitado número de cardinalidades (somente *um*, *zero-ou-um* e *zero-ou-mais*).

Os aspectos funcionais de um sistema de informação, tais como os *contratos* de componentes do sistema (capítulo 8 do livro de Wazlawick), ainda não podem ser modelados em CML. Também não é possível implementar em CML as *operações* descritas pelos *contratos*.

Apesar destas limitações de modelagem conceitual, o compilador extensível de CML permite a criação de *templates* textuais que podem gerar código implementando *operações* a partir dos modelos conceituais, tais como operações CRUD (Create-Read-Update-Delete) e também consultas a partir de *expressões* em CML. De fato, o módulo básico fornecido com o compilador possui *templates* que implementam modelos orientados a objetos com operações de consulta e modificação nas linguagens Java e Python.

## 1.3 ORGANIZAÇÃO

O capítulo 2 apresenta os trabalhos relacionados que influenciaram a concepção e o desenvolvimento de CML. São linguagens, ferramentas e *frameworks* que foram projetadas para modelagem conceitual ou para geração de código a partir de modelos conceituais.

O capítulo 3 apresenta a linguagem de modelagem conceitual proposta neste trabalho. Já o capítulo 4 discorre sobre o compilador e o ferramental de suporte a este. No capítulo 5, descreve-se o processo de desenvolvimento da linguagem CML e de seu compilador. Na sequência, o capítulo 6 faz uma análise de custo-benefício do uso de CML no desenvolvimento de software. Por último, o capítulo 7 faz as considerações finais deste trabalho.

Salienta-se aqui também a importância dos documentos anexados a este relatório. O anexo A é uma cópia integral da especificação da linguagem. Logo após, o anexo B contém o artigo (resumo deste relatório) que foi submetido a *International Conference on Concep-*

*tual Modeling 2017*. O último anexo (C) contém a listagem de todo código-fonte que implementa o compilador CML, assim como exemplos de código gerado por CML.

# 2 TRABALHOS CORRELATOS

As próximas seções apresentam outras linguagens, ferramentas e *frameworks* que se comparam de alguma forma com CML – a solução proposta neste trabalho. Assim como CML, estas soluções permitem a geração de código a partir de modelos conceituais. Cada seção apresenta uma categoria diferente, enumerando soluções específicas e caracterizando-as de acordo com suas diferenças e sua relevância a CML.

## 2.1 LINGUAGENS TEXTUAIS

Esta seção apresenta as linguagens baseadas em texto projetadas para geração código a partir de modelos conceituais. Quando comparadas a CML, as linguages seguintes são as mais relevantes por causa da sua natureza textual. Apesar da sua relevância, é importante salientar que a maior parte das soluções apresentadas nesta seção promovem a modelagem através de linguagens de domínio específico (DSLs), enquanto CML é uma linguagem genérica para modelagem em qualquer domínio.

### 2.1.1 MPS

Meta Programming System (MPS) é um ambiente de desenvolvimento desenvolvido por JetBrains que, de acordo com Voelter (VOELTER, 2014), possui as seguintes características:

- desenvolvedores podem definir linguagens – usualmente, linguagens de domínio específico (DSLs), mas não limitado a estas – que são totalmente integradas umas com as outras;
- o "código-fonte" é uma árvore da sintaxe abstrata (ou ainda uma instância do metamodelo da linguagem), persistido em XML, que nunca é diretamente editado pelo desenvolvedor;
- cada linguagem é definida em três partes: o metamodelo (a definição da linguagem), os editores projecionais (de natureza textual ou gráfica), e os geradores de código;
- desenvolvedores manipulam o "código-fonte" (na verdade, os mo-

delos conceituais na forma de uma árvore sintática abstrata - AST) através dos editores projecionais.

- o ambiente de desenvolvimento é capaz de persistir modelos incompletos ou inválidos (ou seja, não seguindo as regras da linguagem) para posterior resolução de erros, assim como é possível salvar programas com erros de sintaxe ou semântica em qualquer editor de texto.

Uma característica fundamental de MPS é a dependência no ambiente de desenvolvimento – e de seus editores projecionais – para a edição dos modelos conceituais. Também é característico a necessidade de se definir linguagens (ou de se reutilizar linguagens pré-definidas por terceiros) antes de se iniciar o processo de modelagem.

### 2.1.2 Linguagem de Modelagem "M"

Apesar do seu nome, a linguagem de modelagem "M" desenvolvida pela Microsoft (MICROSOFT, 2009), como parte do projeto Oslo (que foi descontinuado posteriormente), é essencialmente uma linguagem que permite especificar outras linguagens textuais, tais como linguagens de domínio específico (DSLs). Estas linguagens, consequentemente, permitem criar modelos (em forma de texto) usando conceitos de um determinado domínio. Outra característica importante da linguagem "M" é que, ao se definir a sintaxe de uma DSL, também é possível se definir transformações do modelo textual em estruturas de dados que representam a sintaxe abstrata do modelo (AST). Esta representação em AST, por sua vez, pode ser processada posteriormente para se gerar código em outras tecnologias.

No exemplo da figura 5, uma DSL é definida para se especificar modelos de qualquer tipo de loja. Cada sentença que começa com **syntax** representa uma regra de produção da gramática. A regra *Main* é a primeira regra a ser avaliada pelo compilador. Nesta linguagem de exemplo, chamada de *StoreLanguage*, é possível definir um modelo textual de uma loja que contém determinados tipos de produtos, cada um com uma lista de atributos específicos, como mostra a figura 6.

Uma vez definido um modelo, como o da figura 6, pode-se processá-lo pelo compilador "M" a fim de gerar uma representação, como ilustrado na figura 7. A estrutura desta representação foi definida pelas projeções especificadas após o símbolo de derivação (`=>`) de cada regra da figura 5.

```
language StoreLanguage
{
    syntax Main =
        "store" name:Name "{" list:Product* "}"
        => id(name)[valuesof(list)];

    syntax Product =
        "product" name:Name "{" list:Attribute* "}"
        => id(name)[valuesof(list)];

    syntax Attribute =
        "attribute" name:Name ":" type:Type ';'
        => id(name)[type];

    syntax Type = ("String" | "Decimal");

    token Name =
        ("A".."Z" | "a".."z" | "_")
        ("A".."Z" | "a".."z" | "0".."9" | "_")*;

    interleave Whitespace = (" " | "\n")+;
}
```

Figura 5 – Especificação de DSL para lojas na linguagem “M”.

```
store BookStore
{
    product Book
    {
        attribute title: String;
        attribute price: Decimal;
    }

    product Maganize
    {
        attribute name: String;
        attribute issue: String;
        attribute price: Decimal;
    }
}
```

Figura 6 – Modelo criado com a DSL *StoreLanguage* definida na figura 5.

```
BookStore[
    Book[
        title[String],
        price[Decimal]
    ],
    Magazine[
        title[String],
        issue[String],
        price[Decimal]
    ]
]
```

Figura 7 – Representação do modelo definido na figura 6.

Possuindo a estrutura da figura 7, é possível então gerar código em várias tecnologias diferentes, tais como:

- Um esquema em SQL para criação de um banco de dados para a loja modelada.
- Um modelo orientado a objetos em C#, que permite consultar produtos da loja armazenados no banco de dados.
- Uma API REST para accessar os produtos da loja através da Internet.

Para que isto seja possível, entretando, é necessário que ferramentas adicionais sejam fornecidas para processar a representação da figura 7. Antes de ser descontinuado, o projeto Oslo da Microsoft pretendia fornecer tais ferramentas de geração de código no Visual Studio e no SQL Server.

### 2.1.3 Xtext

Xtext, como descrito por Bettini (BETTINI, 2016), é uma linguagem textual disponível no ambiente de desenvolvimento Eclipse para a implementação de linguagens de programação e linguagens de domínio específico (DSLs). A partir da especificação da gramática de uma linguagem em Xtext, é possível gerar automaticamente o *lexer*, o *parser*, a sintaxe abstrata (o metamodelo), o gerador de código Java e até mesmo editores no Eclipse. Se for necessário, também é possível adaptar o metamodelo e a geração de código para resolver questões específicas de cada domínio ou aplicação. Recentemente, Xtext também

está disponível no ambiente de desenvolvimento da JetBrains: IntelliJ IDEA.

Similarmente ao exemplo usado para demonstrar a linguagem "M" na seção 2.1.2, o exemplo da figura 8 apresenta uma DSL criada em Xtext que permite especificar modelos de qualquer tipo de loja. Observe que a especificação da linguagem é basicamente uma lista de regras, cada uma definindo uma produção gramatical e também a sintaxe abstrata, com suas classes (*Store*, *Product*, *Attribute*), associações (*products*, *attributes*) e propriedades (*name*, *type*), a ser construída a partir dos elementos de texto.

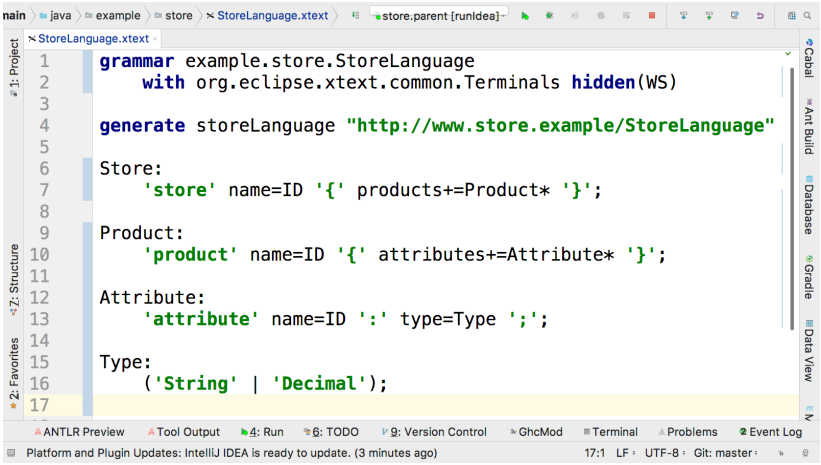


Figura 8 – DSL para lojas especificada em Xtext no IntellIJ IDEA.

A regra *Store* é a primeira regra a ser avaliada pelo compilador. Esta regra (similarmente à regra *Main* no exemplo da linguagem "M" na seção 2.1.2), juntamente com as demais, permite definir um modelo textual de uma loja que contém determinados tipos de produtos, cada um com uma lista de atributos específicos, como mostra a figura 9. Observe na figura que este é o mesmo modelo criado com a linguagem "M" na figura 6. Também observe na figura 9 que Xtext gerou um *plugin* para IntelliJ IDEA, permitindo a edição do modelo com *highlight syntax* e com erros sublinhados em vermelho no editor.

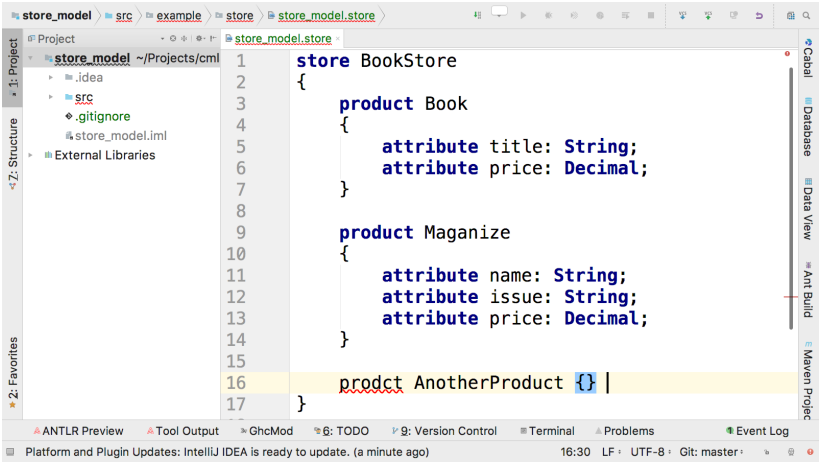


Figura 9 – Modelo criado no IntelliJ IDEA com a DSL *StoreLanguage* definida na figura 8 em Xtext.

A partir de um modelo, como ilustrado na figura 9, é possível então criar geradores de código através da linguagem Xtend (seção 2.4). Também é possível manipular os modelos diretamente em EMF, apresentado na seção 2.3.

### 2.1.4 MM-DSL

MM-DSL (VISIC; KARAGIANNIS, 2014) permite a definição de metamodelos (sintaxe abstrata) que servem de entrada para a geração de ferramentas de domínio específico (DSTs).

### 2.1.5 ThingML

ThingML (HARRAND et al., 2016) inclui uma linguagem de modelagem de sistemas embutidos e uma ferramenta extensível para geração de código em várias plataformas. Entre as plataformas suportadas estão: Arduino, Linux, Java, Android, e JavaScript.

O conceito principal em ThingML é chamado de "thing", ou *coisa* em Português. Uma *coisa* em ThingML define um componente que pode enviar e receber *mensagens* de forma assíncrona. As *mensa-*

*gens* são enviadas e recebidas através de *portas*. O comportamento de uma *coisa* é definido através de uma máquina de estado.

```
thing HelloWorldClient {
    message hello();
    message msg(m : String);

    required port hello {
        sends hello
        receives msg
    }

    statechart behavior init Init {
        state Init {
           on entry do hello!hello() end

           internal event m : hello?msg
           guard m.m == "world"
           action do print(m.m) end
        }
    }
}

thing HelloWorldServer {
    message hello();
    message msg(m : String);

    provided port hello {
        receives hello
        sends msg
    }

    statechart behavior init Init {
        state Init {
            transition t -> B event hello?hello
        }

        state B {
            transition t -> Init action hello!msg("world")
        }
    }
}

configuration helloWorldConf {
    instance client : HelloWorldClient
    instance server : HelloWorldServer
    connector client.hello => server.hello
}
```

Figura 10 – Exemplo de modelo conceitual na linguagem ThingML.

No exemplo da figura 10, definiu-se duas *coisas* (um cliente chamado **HelloWorldClient** e um servidor chamado **HelloWorldServer**) que trocam duas *mensagens*. Através da porta *hello*, *HelloWorldClient* envia uma mensagem **hello** e *HelloWorldServer* responde com uma mensagem **msg(m : String)**. Um *statechart* (ou máquina de estado) define o comportamento do servidor, que deve enviar a mensagem **hello!msg("world")** logo após receber **hello**. No bloco *configuration*, são definidas uma instância do cliente e outra do servidor, e a porta *hello* do cliente é conetada à porta *hello* do servidor.

Uma vez definido um modelo conceitual em ThingML, é possível gerar código para diferentes plataformas. Por exemplo, na figura 11, um modelo em ThingML (no editor que se encontra na parte de baixo) gerou código em C (no editor de cima) que está rodando num Arduino.

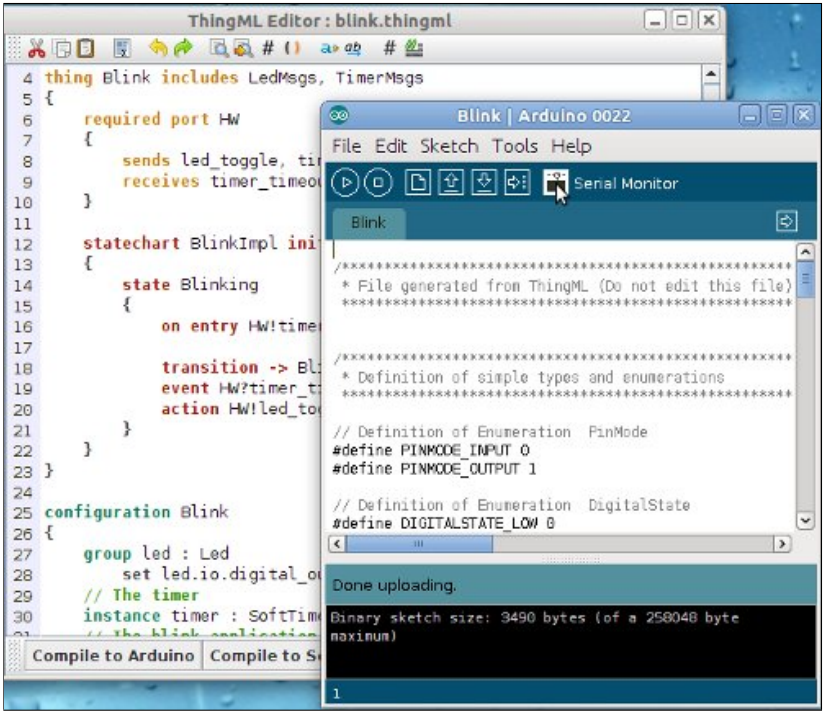


Figura 11 – Modelo de ThingML gerando código C que roda no Arduino. Origem: ThingML web site (FLEUREY; MORIN, 2017).

### 2.1.6 XML

Como demonstrado por Gheraibia e Bourouis (GHERAIBIA; BOUROUIS, 2012), é possível utilizar XML para se definir um modelo conceital e posteriormente definir templates em XSLT para a geração de código. Porém, antes de se definir modelos em XML, é preciso definir um metamodelo em *XML Schema*. Esta seção demonstra esta abordagem através do mesmo exemplo usado nas seções anteriores para se modelar lojas.

Neste caso, como ilustrado na figura 13, ao invés de se definir uma DSL para lojas, define-se um *XML Schema*. Um modelo de lojas, como definido por este *schema*, inicia-se com um elemento *store*, que contém uma lista de elementos *product*, que por sua vez contém elementos *attribute*.

```
<?xml version="1.0" encoding="UTF-8" ?>

<store name="BookStore"
       xmlns="http://www.example.com/store">
       <product name="Book">
           <attribute name="title" type="String"/>
           <attribute name="price" type="Decimal"/>
       </product>
        <product name="Magazine">
            <attribute name="name" type="String"/>
            <attribute name="issue" type="String"/>
            <attribute name="price" type="Decimal"/>
        </product>
</store>
```

Figura 12 – Um modelo em XML de uma loja de livros.

A figura 12 mostra um modelo de uma loja de livros usando o *XML Schema* da figura 13. Observe como o conteúdo é similar ao do modelo apresentado na figura 6 (na linguagem "M") e ao conteúdo do modelo apresentado na figura 9 (em Xtext). Também como naqueles casos, é possível gerar código para o modelo da figura 12, mas neste caso usando *Extensible Stylesheet Language Transformation* (XSLT).

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:st="http://www.example.com/store"
    targetNamespace="http://www.example.com/store"
    elementFormDefault="qualified">

    <xs:element name="store">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="product"
                            type="st:Product"
                            maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="name"
                          type="xs:string"/>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="Product">
        <xs:sequence>
            <xs:element name="attribute"
                        type="st:Attribute"
                        maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="name"
                      type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="Attribute">
        <xs:attribute name="name"
                      type="xs:string"/>
        <xs:attribute name="type"
                      type="st:AttributeType"/>
    </xs:complexType>

    <xs:simpleType name="AttributeType">
        <xs:restriction base="xs:string">
            <xs:pattern value="String|Decimal"/>
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

Figura 13 – *XML Schema* para modelos de lojas.

## 2.2 LINGUAGENS GRÁFICAS

Esta seção enumera as linguagens gráficas projetadas para geração de código a partir de modelos conceituais. Apesar de CML – a solução proposta neste trabalho – ser uma linguagem textual, as seguintes linguagens têm uma certa relevância, pois também têm como próposito a geração de código em linguagens de programação:

- FCML (KARAGIANNIS et al., 2016) incorpora and expande linguagens de modelagem (tais como ER, UML, and BPMN) através da ferramenta OMNILab a fim de gerar código.
- MetaEdit+ (TOLVANEN, 2004) é um ambiente de desenvolvimento que permite a criação de ferramentas de modelagem e geradores de código para linguagens de domínio específico (DSLs) visuais.
- MDA (OMG, 2014a) é a iniciativa da OMG para difundir o uso de UML (OMG, 2015a) em desenvolvimento baseado em modelos.
- IFML (BRAMBILLA; MAURI; UMUHOZA, 2014) é uma linguagem de alto nível da OMG que pode ser usada para gerar interfaces de usuário em diferentes plataformas, tais como na Web e em dispositivos móveis.
- MPS (VOELTER, 2014), como mostrado na seção 2.1, além de modelos textuais, também permite a criação de modelos gráficos.

O maior obstáculo das linguagens gráficas, como abordado na seção 1.1.1, é a dificuldade de integrá-las no fluxo de trabalho, nas ferramentas e na mentalidade de grande parte dos desenvolvedores de software, que estão habituados com as linguagens de programação textuais, seus compiladores e com editores de texto.

A seleção da ferramenta gráfica também envolve tempo e custo (licenças, treinamento, atualizações), estabelecendo uma depêndencia num determinado fornecedor ou num ambiente de desenvolvimento com um certo grau de risco.

## 2.3 FRAMEWORKS

Alguns *frameworks* permitem a geração de código a partir de modelos conceituais, mas sem uma linguagem de modelagem – gráfica ou textual. *Eclipse Modeling Framework*, ou EMF (STEINBERG et al.,

2008), é o exemplo clássico de *framework* que permite modelagem sem uma linguagem para tal. A modelagem é feita através de editores no ambiente de desenvolvimento Eclipse, ou via uma interface de programação, e os modelos são armazenados no formato ECORE/XML.

*Frameworks* podem ser usadas também como a infra-estrutura de linguagens de modelagem. EMF, por exemplo, é a framework que supporta Xtext (BETTINI, 2016), apresentado na subseção 2.1.3. É possível imaginar que outras linguagens de modelagem possam usar EMF. De fato, o compilador extensível de CML permite a implementação de *templates* que tenham EMF como alvo.

## 2.4 LINGUAGENS DE TEMPLATE

Como apresentado na seção 4.3, o compilador CML usa a linguagem StringTemplate (PARR, 2004) na geração de código. Existem outras linguagens projetadas para geração de código a partir de *templates*, enumeradas a seguir:

- EGL (ROSE et al., 2008) é uma linguagem de *templates* que possibilita a geração de código a partir de modelos.
- MOFScript (OLDEVIK et al., 2005) permite a geração de código a partir de modelos definidos em qualquer tipo de metamodelo.
- Xpand (GREIFENBERG et al., 2016) permite a definição de *templates* com múltiplas regiões de variabilidade.
- Xtext/Xtend (BETTINI, 2016) permite a definição de DSLs textuais para se gerar código de modelos conceituais editados em Eclipse.
- JET (BOAS, 2004) permite a geração de código a partir de modelos em EMF (STEINBERG et al., 2008).
- XSLT (GHERAIBIA; BOUROUIS, 2012) é a linguagem de *templates* normalmente utilizada para geração de código a partir de modelos XML.

Um dos pontos fortes de StringTemplate, a linguagem de *templates* adotada por CML, é seu mecanismo de extensibilidade. É possível definir um conjunto padrão de *templates* e depois extendê-los com as partes específicas de cada linguagem-alvo ou tecnologia. Também é possível compartilhar *templates* em bibliotecas. Este mesmo nível de

extensibilidade também está disponível em Xpand (GREIFENBERG et al., 2016).

## 2.5 LINGUAGENS SUPORTANDO ASSOCIAÇÕES

Tal como CML, existe um número de linguagens de programação que permitem a declaração de associações bidirecionais:

- DSM (BALZER; GROSS; EUGSTER, 2007) é uma linguagem de programação orientada-a-objetos com suporte a associações.
- Fibonacci (ALBANO et al., 1993) é uma linguagem de programação para bancos de dados orientados a objetos que permite a modelagem de papéis de associações.
- ASSOCIATION# (CARDOSO, 2011) é uma extensão da linguagem C# que permite a modelagem de associações.
- RelJ (BIERMAN; WREN, 2005) é uma extensão da linguagem Java que suporta associações.

Um desvantagem chave da maior parte das linguagens acima é o fato de seus modelos conceituais não poderem ser reutilizados para gerar código em outras linguagens ou tecnologias; estas linguagens são, para todos os efeitos, a linguagem-alvo.

## 2.6 OUTRAS LINGUAGENS

Existem outras linguagens conceituais cujo foco original não foi a geração de código, mas sim servir de artefatos de modelagem.

Linguagens tais como OWL (W3C, 2012) and Telos (MYLOPOULOS et al., 1990) foram projetadas como metamodelos de ontologias para suportar a representação e o armazenamento de conhecimento, e também para automatizar raciocínio a partir de bases de conhecimento. OWL sendo a *lingua franca* da Web semântica, enquanto Telos foi criado para armazenar ontologias em bancos de dados orientados a objetos.

Outras linguagens, como UML (OMG, 2015a) and ER (CHEN, 2011), foram originalmente imaginadas como ferramentas para dar suporte à análise e ao projeto de sistemas de software; somente depois de algum tempo estas foram reorientadas para o desenvolvimento de software baseado em modelos (MDSD).

A relevância destas linguagens para CML vem da força de expressividade na modelagem conceitual provida por seus metamodelos. CML deve continuar expandindo suas capacidades de modelagem inspirado nas construções disponíveis nestas linguagens.

UML e ER, juntamente com OCL (OMG, 2014b), serão usadas no capítulo 3 como bases de comparação do metamodelo da linguagem CML.

# 3 CML: A LINGUAGEM

Este capítulo apresenta CML, a linguagem de modelagem conceitual proposta neste trabalho. A seção 3.1 apresenta CML através de um exemplo. Uma versão simplificada do metamodelo de CML (da estrutura da sintaxe abstrata), que é suficiente para representar o exemplo da seção 3.1, é apresentada na seção 3.2. As seções seguintes apresentam em mais detalhe cada um dos elementos do metamodelo de CML. (O anexo A, contendo a especificação da linguagem, provê uma descrição formal da sintaxe concreta, do metamodelo e das invariantes do metamodelo da CML.)

## 3.1 EXEMPLO

Esta seção apresenta através de um exemplo a sintaxe concreta de CML. No exemplo da figura 14, alguns *conceitos*, tais como *Book* and *Customer*, são modelados em CML. A sintaxe declarando cada *conceito*, baseada em blocos, lembra a sintaxe da linguagem C (ISO, 2011).

Cada *conceito* declara uma lista de *propriedades*. A declaração de uma *propriedade* é inspirada na forma como a linguagem Pascal (JENSEN; WIRTH, 1974) declara variáveis, onde o nome é seguido por dois-pontos (":") e depois pela declaração do tipo.

A sintaxe de *expressões* em CML, tal como a expressão usada para definir *orderedBooks* no conceito *BookStore*, é parcialmente inspirada em expressões de OCL (OMG, 2014b). A sintaxe da expressão em *goldCustomers* é original, mas sua semântica também é baseada em expressões de consulta em OCL.

Como visto no bloco *CustomerOrder*, CML também permite a declaração de *associações bidirecionais* através do pareamento de propriedades de diferentes conceitos. As *associações unidirecionais* são representadas apenas por uma única propriedade partindo da origem da associação, como é o caso da propriedade *books* em *BookStore*.

Na subseção 3.1.1, o modelo do exemplo da figura 14 é ilustrado como uma árvore sintática abstrata, que se assemelha ao modelo de representação interna usado pelo compilador. O mapeamento do exemplo para UML (OMG, 2015a) e OCL (OMG, 2014b) é descrito na subseção 3.1.2.

```
@concept BookStore
{
    books: Book*;
    customers: Customer*;
    orders: Order*;

    /goldCustomers = customers | select: totalSales > 1000;
    /orderedBooks = orders.items.book;
}

@concept Book
{
    title: String;
    price: Decimal;
    quantity: Integer = 0;
}

@concept Customer
{
    orders: Order*;
    /totalSales = sum(orders.total);
}

@concept Order
{
    customer: Customer;
    items: Item*;
    total: Decimal = sum(items.amount);
}

@concept Item
{
  book: Book;
  quantity: Integer;
  amount: Decimal;
}

@association CustomerOrder
{
    Order.customer: Customer;
    Customer.orders: Order*;
}
```

Figura 14 – Modelo em CML adaptado da loja de livros fictícia Livir; um estudo de caso no livro de Wazlawick (WAZLAWICK, 2014).

### 3.1.1 Árvore de Sintaxe Abstrata

O conceito *BookStore* especificado no modelo da figura 14, quando lido pelo compilador CML, gera uma árvore de sintaxe abstrata[1] (ou simplesmente AST) semelhante à ilustrada na figura 15.

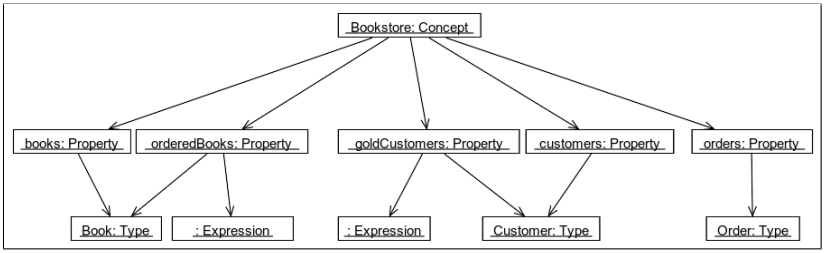


Figura 15 – AST gerada pelo compilador CML após a leitura da especificação do conceito *BookStore* apresentado na figura 14.

Para o compilador CML, esta AST resultante corresponde à sua representação interna do conceito *BookStore*, assim como especificado pelo código em CML da figura 14. Usando a terminologia para metamodelos de EMOF (OMG, 2016), cada nodo na figura 15 representa uma instância de uma classe do metamodelo da linguagem CML. O nó-raíz é uma instância da classe *Concept* e seus subnós são instâncias da classe *Property*. Esta, por sua vez, tem como subnós instâncias das classes *Type* e *Expression*.

Uma vez que a AST da figura 15 é gerada e validada internamente pelo compilador CML, esta serve como a entrada dos *templates extensíveis* (apresentados na seção 4.3) durante a fase de geração de código.

Para que o compilador CML possa gerar a AST, é necessário definir o metamodelo da linguagem CML, que é formalizado pela especificação da linguagem no anexo A, e ilustrado na seção 3.2. Antes desta, a subseção 3.1.2 faz um mapeamento de modelos conceituais criados em CML (como o da figura 14) para modelos correspondentes em UML (OMG, 2015a) e OCL (OMG, 2014b).

[1] Na figura 15, mostra-se apenas parte da árvore do modelo CML definido na figura 14; somente a parte que se refere ao conceito *BookStore*. Uma árvore similar seria gerada pelo compilador para todos os conceitos do modelo.

### 3.1.2 Mapeamento para UML/OCL

Parte do metamodelo de CML, como apresentado na seção 3.2, pode ser considerado um pequeno subconjunto do metamodelo de UML (OMG, 2015a). Portanto, os elementos estruturais (ou estáticos) de modelos escritos em CML podem ser transformados em diagramas de classes em UML. O exemplo do modelo em CML mostrado na figura 14 é mapeado para o modelo em UML da figura 16.

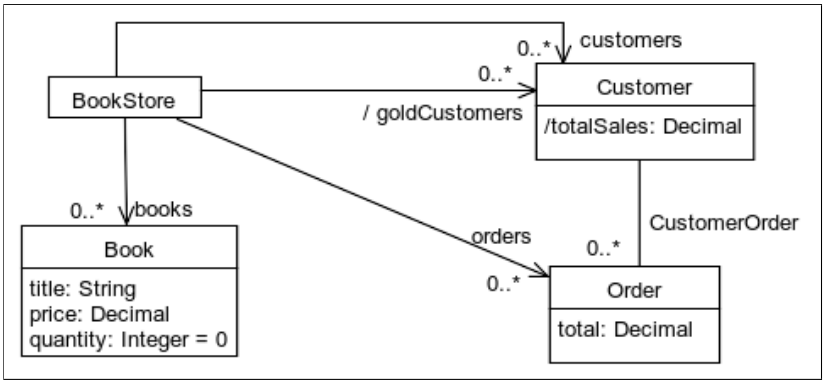


Figura 16 – O diagrama de classe representando em UML (OMG, 2015a) o mesmo modelo codificado em CML pela figura 14.

No diagrama de classe da figura 16, os *conceitos* em CML (*BookStore*, *Book*, *Customer* e *Order*) são mapeados para *classes* em UML. As *propriedades* em CML que representam atributos, tais como *title*, *quantity* e *price* do conceito *Book*, são mapeadas para atributos em UML dentro de cada *classe*. As *propriedades* em CML que representam o destino de associações unidirecionais (*books*, *customers* e *orders* de *BookStore*) são mapeadas para *associações* em UML com os respectivos *papéis* (mostrando a direção de navegação e a cardinalidade junto ao nome de cada papel). A associação bidirecional *CustomerOrder* (formada pelas propriedades: *Customer.orders* e *Order.customer*) é mapeada para uma associação em UML com navegabilidade bidirectional, ou seja, sem setas na linha de associação.

Como demonstrado por este exemplo, CML permite criar modelos em texto com o mesmo nível conceitual que UML. Ao mesmo tempo, quando comparado com o metamodelo de UML, o metamodelo

de CML disponibiliza apenas um núcleo dos elementos fornecidos por UML, como é apresentado na seção 3.2.

Além dos elementos estruturais de um modelo conceitual, CML também permite o uso de *expressões* para definir o valor inicial de *atributos* e para definir *propriedades derivadas*; tanto as que representam *atributos derivados*, quanto aquelas que representam *associações derivadas*. Estas *expressões* são apenas parcialmente baseadas na sintaxe de OCL (OMG, 2014b), mas seguem fortemente as semânticas de OCL, como formalmente definido pelo anexo A na especificação da linguagem CML.

Por exemplo, a expressão em CML mostrada a seguir (extraída da figura 14) é uma expressão de navegação tal qual como em OCL:

```
/orderedBooks = orders.items.book;
```

Usando *propriedades* que representam o destino de *associações*, a expressão acima "navega" (uma força de expressão) a partir de uma instância de *BookStore*, passando por todas as instâncias de *orders* ligadas a instância de *BookStore*, e então por todas as instâncias de *items* de todas as *orders*, a fim de resultar em todos os livros (*books*) que já foram pedidos por clientes.

Como outro exemplo, a seguinte expressão em CML (também extraída da figura 14) já não segue a sintaxe de OCL:

```
/goldCustomers = customers | select: totalSales > 1000;
```

Entretanto, a expressão acima é semanticamente idêntica à expressão seguinte em OCL:

```
derive: customers->select(totalSales > 1000)
```

Ambas as expressões acima (uma, em CML; outra, em OCL) resultam na mesma coleção de instâncias de *Customer* que compraram mais de 1000 em dinheiro na loja (*BookStore*). A seção 3.6 explica as diferenças sintáticas e as equivalências semânticas de expressões entre CML e OCL.

## 3.2 METAMODELO

Representado como um diagrama de classe EMOF[2] (OMG, 2016), o metamodelo ilustrado na figura 17 é uma versão simplificada do me-

[2]EMOF é basicamente um subconjunto bem definido da UML que permite especificar qualquer tipo de metamodelo, tais como a própria UML, o metamodelo ER e, como mostrado aqui, o metamodelo da linguagem CML.

tamodelo de CML. Como mostrado neste diagrama, uma instância de *Concept* é composta de zero-ou-mais instâncias de *Property*. Cada instância de *Property* pode estar ligada a uma instância de *Type* e, opcionalmente, a uma instância de *Expression*. Se duas ou mais instâncias de *Property* representam uma associação bidirecional, deve haver uma instância de *Association* que as liga. Associações unidirecionais são representadas somente por uma instância de *Property* (na verdade, representando o papel-destino da associação), permitindo a navegação de uma instância-origem (o *conceito* contento a *propriedade*) para uma instância-destino (determinada pela instância *Type* de uma *propriedade*).

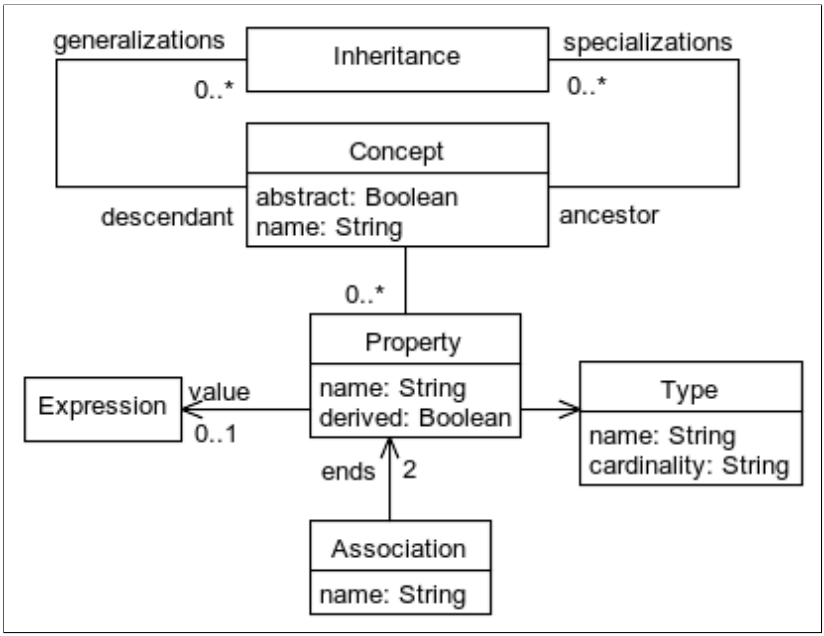


Figura 17 – This class diagram renders the EMOF (OMG, 2016) model defining the CML metamodel.

No artigo *UML and OCL in Conceptual Modeling*, Gogolla (GOGOLLA; THALHEIM, 2011) apresenta, através do mapeamento do metamodelo de UML (OMG, 2015a) para o metamodelo de ER (CHEN, 2011), como modelos UML, somado às expressões definidas em OCL (OMG, 2014b), podem ser usados na especificação de modelos concei-

tuais. Também, em seu livro, Wazlawick (WAZLAWICK, 2014) sistematicamente preescreveu um método de modelagem conceitual usando UML and OCL.

Já que um dos objetivos de CML é permitir a especificação de modelos conceituais, tais como aqueles especificados em ER e UML/OCL, para apresentar os elementos principais do metamodelo da linguagem CML, uma abordagem semelhante àquela de Gogolla é usada nas próximas seções para mostrar a correspondência de cada um dos elementos apresentados no metamodelo de CML aos metamodelos ER e UML/OCL. O anexo A contém uma especificação formalizada dos elementos da linguagem CML.

## 3.3 CONCEITOS

Em CML, um *conceito* pode representar qualquer coisa que tem um significado coerente, coeso e relevante no domínio do sistema sendo modelado. Por exemplo, como visto no modelo escrito em CML da figura 14, conceitos como *Book* e *Customer* têm signifcado e relavência no domínio de uma loja de livros.

A tabela 1 resume o mapeamento de um *conceito* entre o metamodelo de CML e os metamodelos de UML (OMG, 2015a) e ER (CHEN, 2011).

| Metamodelo | Termo | Observações |
|---|---|---|
| CML | Concept | Somente possuem atributos e associações. |
| UML | Class | Possuem atributos, associações e operações (comportamento). |
| ER | Entity Type | Possuem atributos e papéis de relacionamentos. Cada instância é chamada de entidade. |

Tabela 1 – Mapeamento de *Conceito* para UML (OMG, 2015a) e ER (CHEN, 2011).

Em UML, um *conceito* corresponde a uma *classe*, que descreve um conjunto de objetos contendo a mesma estrutura e comportamento. CML, porém, não modela o comportamento; apenas os *atributos* e as *associações* relacionadas a um *conceito*. Já, em ER, cada *conceito* corresponderia a conjunto de *entidades* possuindo os mesmos *atributos*,

enquanto cada *entidade* individualmente corresponderia a uma única instância de um *conceito*.

Um *conceito* pode possuir uma lista de *propriedades* que representam *atributos* ou *associações*. *Propriedades* são apresentadas na subseção 3.3.1. Também é possível especializar um *conceito* a partir de outro *conceito* mais genérico. Generalização e especialização são apresentadas na subseção 3.3.2. Por fim, a subseção 3.3.3 aborda *conceitos abstratos*, ou simplesmente *abstrações*.

### 3.3.1 Propriedades

Um *conceito* possui uma lista de *propriedades* que representam *atributos* ou *destinos de associações unidirecionais*. Por exemplo, no modelo da figura 14 escrito em CML, o conceito *Book* possui as propriedades *title*, *price* e *quantity*, todas declaradas com *tipos primitivos* (descritos na subseção 3.4.1). Enquanto o conceito *BookStore* possui as propriedades *books* e *orders*, que já representam o *destino de associações* (descritos na subseção 3.5.1).

A tabela 2 faz o mapeamento de *propriedades* no metamodelo de CML para os metamodelos de UML (OMG, 2015a) e ER (CHEN, 2011).

| Metamodelo | Termo | Observações |
|---|---|---|
| CML | Property | Modela um *atributo* ou o papel-destino de uma *associação*. |
| UML | Attribute / Association End | UML possui classes específicas para *atributos* e *associações*. |
| ER | Attribute / Role | ER também modela *attributos* como parte de uma *entidade* e *papéis* como parte de *relacionamento*. |

Tabela 2 – Mapeamento de *Property* para UML (OMG, 2015a) e ER (CHEN, 2011).

Uma *propriedade* em CML, portanto, pode conter um valor de um *tipo primitivo*, correspondendo neste caso a *atributos* nos metamodelos UML (OMG, 2015a) e ER (CHEN, 2011); ou pode conter uma referência (ou até uma sequência de referências) ligando-se a uma instância de outro *conceito*, correspondendo então à *papéis de relacionamentos*

em ER e ao *destino de associações* em UML.

### 3.3.2 Generalização/Especialização

Em CML, um *conceito* pode ser generalizado por outro *conceito*. Em outras palavras, um *conceito* pode ser considerado uma especialização de outro *conceito*. O primeiro pode ser chamado de *especialização*; o segundo, de *generalização*.

Por exemplo, no modelo CML da figura 18, o conceito **Rectangle** é uma *especialização* do conceito **Shape**, que por consequência é a *generalização* de **Rectangle**. Uma *generalização*, como **Shape**, possui *propriedades*, como **color** e **area**, que se aplicam a um conjunto maior de instâncias, enquanto uma *especialização*, como **Rectangle**, possui *propriedades* que se aplicam a apenas um subconjunto daquelas instâncias, como é o caso de *width* e *height*.

A tabela 3 mostra o mapeamento de *generalizações* entre o metamodelo de CML e os metamodelos de UML (OMG, 2015a) e ER (CHEN, 2011).

Em CML, de acordo com a figura 17, existe uma metaclasse chamada *Inheritance* que se associa de um lado com a *generalização* e de outro com a *especialização*; cada *conceito* possui uma associação com *Inheritance* chamada *generalizations*, que mantém uma *sequência* de *conceitos* que são suas generalizações, e outra associação *specializations*, que permite acesso às especializações de um conceito. No metamodelo da UML, o relacionamento de *generalização/especialização* ocorre entre *classes* e é representado pela metaclasse *Generalization*. Na versão original do metamodelo ER, o relacionamento de *generalização/especialização* não estava disponível.

Explorando mais o exemplo da figura 18, observa-se que *propriedades*, tais como o atributo **area** de **Shape**, podem ser redefinidas pelas *especializações*, como é feito por **Rectangle**, **Rhombus** e **Square**. Algumas *especializações* podem também definir novas *propriedades*, como **p** e **q**, que são características somente das instâncias de **Rhombus**.

A figura 18 mostra ainda que um *conceito* pode ser a *especialização* de dois ou mais *conceitos*, como é o caso de **Square**, que especializa **Rectangle** e **Rhombus**, e portanto pode redefinir as *propriedades* de ambas as generalizações.

Se uma *propriedade* for definida por mais de uma *generalização*, como acontece com a propriedade **area** em **Square**, então é preciso redefinir esta *propriedade* na *especialização* a fim de evitar conflito na

```
-- Generalização de Rectangle e Rhombus:
@concept Shape
{
    -- As especializações abaixo compartilham "color":
    color: String;

    -- As especializações abaixo redefinem "area":
    area: Double;
}

-- Especialização de Shape:
@concept Rectangle: Shape
{
    -- Novos atributos que caracterizam um retângulo:
    width: Double; height: Double;

    -- Redefinição do atributo "area":
    /area = width * height;
}

-- Losângo é outra especialização de Shape:
@concept Rhombus: Shape
{
    -- Atributos com o comprimento das diagonais
    -- caracterizam um losângo:
    p: Double; q: Double;

    -- Outra redefinição do atributo "area":
    /area = (p * q) / 2;
}

-- Especialização de ambos Rectangle e Rhombus:
@concept Square: Rectangle, Rhombus
{
    -- Único atributo que caracteriza um quadrado:
    side_length: Double;

    -- Redefinindo os atributos de Rectangle:
    /width = side_length;
    /height = side_length;

    -- Redefinindo os atributos de Rhombus:
    /p = side_length * 1.41421356237d; // raiz quadrada de 2
    /q = p;

    -- É obrigatória a redefinição do atributo "area"
    -- para resolver o conflito de definição
    -- entre Rectangle e Rhombus:
    /area = side_length ^ 2;
}
```

Figura 18 – Modelo em CML demonstrando *especialização/generalização* entre *conceitos*. Adaptado do exemplo de Lee (LEE, 2004).

| Metamodelo | Termo | Observações |
|---|---|---|
| CML | Inheritance | Em CML, a metaclasse *Inheritance* mantém as associações entre *generalizações* e *especializações*. |
| UML | Generalization | O relacionamento de *generalization/specialization* em UML ocorre entre *classes*. *Generalização* é o nome da metaclasse no metamodelo da UML. |
| ER | — | A versão original de ER não possui o relacionamento de especialização/generalização. |

Tabela 3 – Mapeamento de *Generalização* para UML (OMG, 2015a) e ER (CHEN, 2011).

definição. Se uma redefinição apropriada para ambas as generalizações não for possível, isto pode ser uma indicação que a *especialização* ou as *generalizações* são incoerentes do ponto de vista do domínio sendo modelado.

### 3.3.3 Abstrações

Uma *abstração* em CML é um conceito que não representa instâncias específicas, mas serve somente como uma *generalização* para outros *conceitos* mais específicos. Portanto, todas as instâncias de uma *abstração* são primeiro instâncias de suas *especializações*.

Por exemplo, no modelo CML da figura 19, o conceito **Shape** é marcado como **@abstraction**. Como tal, nenhuma instância de **Shape** é diretamente instanciada. Além disso, **Shape** pode definir *propriedades abstratas*, tais como **area**, que é simplesmente uma *propriedade derivada* sem uma *expressão* definida. Uma *abstração* também pode definir *propriedades concretas*, como é o caso de **color** em **Shape**. O conceito **Circle** já é uma *especialização* de **Shape** que obrigatoriamente redefine a propriedade **area** (provendo uma *expressão*), e portanto é considerado um *conceito concreto*. Como tal, **Circle** pode então ter suas próprias instâncias, que são indiretamente instâncias de **Shape**. **Circle** também pode redefinir *propriedades concretas*, tais como **color**, mas a redefinição não é um requisito neste caso. Finalmente, em **UnitCircle**, pode-se observar que a redefinição de uma *propriedade*

*abstrata*, como **area**, pode ser feita de forma *concreta*; ou seja, esta não precisa ser necessariamente uma *propriedade derivada*. A situação oposta também é permitida em CML, onde uma *propriedade concreta* é redefinida como uma *propriedade derivada*, como ilustrado pela propriedade **radius** em **UnitCircle**.

A tabela 4 mostra o mapeamento de *conceitos abstratos* entre o metamodelo de CML e o metamodelo de UML (OMG, 2015a). Enquanto no metamodelo de CML a metaclasse *Concept* possui um atributo chamado *abstract* do tipo *Boolean* para marcar um *conceito* como abstrato, o metamodelo da UML tem um atributo chamado *isAbstract* (também Boolean) na metaclasse *Class*. A versão original do metamodelo ER (CHEN, 2011), como uma consequência da falta de relacionamentos do tipo *generalização/especialização*, não considera a noção de *entidades abstratas*.

| Metamodelo | Termo | Observações |
|---|---|---|
| CML | Abstract Concept | Em CML, a metaclasse *Concept* possui um atributo chamado *abstract* do tipo *Boolean* que marca um *conceito* como *abstrato*. |
| UML | Abstract Concept | Em UML, a metaclasse *Class* tem um atributo chamado *isAbstract* para o mesmo próposito. |
| ER | — | A versão original de ER não possui *entidades abstratas*. |

Tabela 4 – Mapeamento de *Conceitos Abstratos* para UML (OMG, 2015a) e ER (CHEN, 2011).

Em CML, além de *conceitos abstratos*, é possível definir *propriedades abstratas*, como mostra a tabela 5. Em UML, *atributos* são sempre concretos; somente *operações* podem ser abstratas. No caso de uma *operação*, UML possui um meta-atributo chamado *isAbstract* do tipo *Boolean*. Em CML, porém, existe um meta-atributo derivado chamado *abstract* na metaclasse *Property*. Este tem o valor **true** quando uma *propriedade* é derivada mas não é definida por uma *expressão*.

Vale também salientar que, em CML, somente *abstrações* podem definir *propriedades abstratas*. Por consequência, as especializações de *abstrações* precisam necessariamente redefinir as *propriedades abstratas*, ou então devem ser declaradas elas mesmas como *abstrações*.

```
-- Como um conceito abstrato,
-- instâncias nunca são diretamente criadas
--  a partir de Shape.
@abstraction Shape
{
    -- Uma propriedade derivada sem uma expressão
    -- é considerada uma propriedade abstrata.
    -- Somente conceitos abstratos podem ter propriedades
    -- abstratas.
    /area: Double;

    -- Conceitos abstratos podem ter propriedades concretas:
    color: String;
}

-- Todas as instâncias de Circle são também instâncias
-- de Shape.
@concept Circle: Shape
{
    radius: Double;

    -- Para ser considerado um conceito concreto,
    -- Circle tem que redefinir as propriedades
    -- abstratas herdadas de Shape.
    /area = 3.14159d * radius ^ 2.0d;

    -- Circle pode também definir propriedades de Shape.
    -- Porém, a redefinição não é obrigatória neste caso.
    color = "Blue";
}

@concept UnitCircle: Circle
{
    -- Observe que a redefinição de uma propriedade
    -- abstrata pode ser concreta;
    -- ou seja, esta não precisa ser mais uma
    -- propriedade derivada, como ocorria em Circle.
    area = 3.14159d;

    -- No caso acima, porém, seria melhor redefinir
    -- "area" ainda como uma propriedade derivada,
    -- para se evitar que o valor de "area" seja
    -- modificado após a instanciação de UnitCircle.
    -- Isto foi feito na redefinição de "radius" abaixo.
    -- Observe que, em Circle, "radius" era concreta,
    -- mas sua redefinição abaixo a fez derivada.
    -- Isto também é permitido em CML,
    -- da mesma forma que a situação oposta,
    -- como foi feita acima com "area".
    /radius = 1.0d;
}
```

Figura 19 – Modelo em CML demonstrando *conceitos abstratos* e *propriedades abstratas*.

| Metamodelo | Termo | Observações |
|---|---|---|
| CML | Abstract Property | Em CML, *propriedades* podem ser abstratas. A metaclasse *Property* possui um meta-atributo derivado chamado *abstract* do tipo *Boolean* que retorna *true* quando uma *propriedade* é *derivada* sem definir uma *expressão*. |
| UML | Abstract Attribute / Operation | Em UML, *propriedades* são sempre concretas, somente *operações* podem ser abstratas. A metaclasse *Operation* tem um meta-atributo chamado *isAbstract*. |
| ER | — | A versão original de ER não possui *atributos abstratos*. |

Tabela 5 – Mapeamento de *Propriedades Abstratas* para UML (OMG, 2015a) e ER (CHEN, 2011).

## 3.4 ATRIBUTOS

Um *atributo* em CML é uma *propriedade* de *tipos primitivos* (apresentados na subseção 3.4.1). No modelo da figura 14 em CML, as propriedades **title**, **price** e **quantity** do conceito **Book** são *atributos*, pois seus *tipos* são *primitivos*, respectivamente: **String**, **Decimal** e **Integer**.

A tabela 6 apresenta o mapeamento de um *atributo* entre o metamodelo de CML e os metamodelos de UML (OMG, 2015a) e ER (CHEN, 2011).

| Metamodelo | Termo | Observações |
|---|---|---|
| CML | Attribute | São *propriedades* de tipos *primitivos*. |
| UML | Attribute | São representados pela associação *attribute* entre a metaclasse *Classe* e a metaclasse *Property*. |
| ER | Attribute | Formalmente definido como uma função de um *Entity Set* para um *Value Set*. |

Tabela 6 – Mapeamento de *Attributo* em CML para UML (OMG, 2015a) e ER (CHEN, 2011).

Em UML, *atributos* são representados como uma meta-associação chamada *attribute* entre a metaclasse *Class* e a metaclasse *Property*. Em ER, um *atributo* é definido formalmente como uma função que tem como domínio um *Entity Set* (ou *Entity Type*) e como contra-domínio um *Value Set* (ou seja, um *Primitive Type*).

Em CML, *atributos não-derivados* servem como uma variável (ou um *slot*), que faz parte da instância de um *conceito*, e que mantém um valor de um determinado *tipo primitivo*. Um valor inicial pode ser especificado como uma *expressão*.

A subseção 3.4.1 enumera os *tipos primitivos* suportados por CML. A subseção 3.4.2 trata de *atributos derivados*. A seção 3.6 cobre em detalhes os tipos de *expressões* que podem ser usadas em CML para inicializar *atributos* e para definir *atributos derivados*.

### 3.4.1 Tipos Primitivos

Um *tipo primitivo* em CML é um dos *tipos de dados* pré-definidos pela linguagem, de acordo com as tabelas 7 e 8. Um *tipo primitivo* define o *tipo de dado* que pode ser armazenado (guardado) pelo *atributo* de uma *instância*.

No metamodelo ER, um *tipo de dado* é formalmente definido como um *conjunto* de valores que servem como o contra-domínio de uma função *atributo*. De acordo com o artigo original de ER (CHEN, 2011), para cada *conjunto de valores* (ou seja, *tipo de dado*), existe um *predicado* que pode ser usado para verificar se um determinado *valor* pertence ao *conjunto*, ou seja, é de um determinado *tipo*. Em CML, *expressões literais* são sintaticamente definidas para cada *tipo primitivo*, permitindo assim que o *tipo* seja inferido através da *expressão literal*. Pode-se dizer assim que a forma sintática de uma *expressão literal* em CML (subseção 3.6.1) serve como o *predicado* do metamodelo ER.

Ainda no artigo original de ER, é dito que *valores* num determinado *conjunto* podem ser equivalentes a *valores* em outro *conjunto*. Em CML, também, *expressões literais* do tipo *Integer*, por exemplo, são equivalentes a *expressões literais* do tipo *Decimal*, e da mesma forma com outros *tipos numéricos*. Isto permite que *expressões* de um *tipo primitivo* possam ser promovidas em *expressões* de outro *tipo*, a fim de permitir a inferência de *tipo* em *expressões* compostas, tais como em *expressões com operadores* (subseção 3.6.3).

O metamodelo UML (OMG, 2015a) contém uma metaclasse específica para representar *tipos primitivos*, chamada de *PrimitiveType*.

Assim como em CML, UML também define notação sintática de *expressões literais* para cada *tipo primitivo*. CML, entretanto, não tem correspondência direta aos *tipos primitivos* e *expressões literais* da UML. Ao invés disto, CML procura ter uma correspondência mais direta com linguagens de programação, visto que *atributos* e *expressões* em CML são traduzidos para várias linguagens-alvo, como enumerado pelas tabelas 7 e 8.

| CML | Java | C# | C++ | Python | TypeScript (JS) |
|---|---|---|---|---|---|
| String | String | string | wstring | str | string |
| Sequência de caracteres Unicode de 16 bits. | | | | | |
| Boolean | boolean | bool | bool | bool | boolean |
| Únicos valores são as expressões literais: **true**, **false**. | | | | | |
| Integer | int | int | int32_t | int | number |
| Inteiro positivos e negativos de 32 bits em complemento de 2. | | | | | |
| Decimal* | BigDecimal | decimal | decimal128 | Decimal | number |
| Precisão arbitrária, ponto-fixo, ou ponto-flutuante no sistema decimal, dependendo da linguagem-alvo. | | | | | |

*A especificação do tipo *Decimal* varia de acordo com a linguagem-alvo. Comparado aos tipos ponto-flutuante binários, o tipo Decimal é mais adequado aos cálculos monetários, porém com um custo de *performance*.

Tabela 7 – Tipos Primitivos Essenciais de CML.

### 3.4.2 Atributos Derivados

Em CML, quando o *valor* de um *atributo* precisa ser determinado pela avaliação de uma *expressão*, este é chamado de *atributo derivado*. Seu *valor* não pode ser alterado ou definido de nenhuma outra forma.

No exemplo da figura 18, o atributo **area** de **Rectangle** é um exemplo de *atributo derivado*. O valor de **area** é sempre calculado a

| CML | Java | C# | C++ | Python | TypeScript (JS) |
|---|---|---|---|---|---|
| Byte | byte | byte | int8_t | int | number |
| Inteiros positivos e negativos de 8 bits em complemento de 2. | | | | | |
| Short | short | short | int16_t | int | number |
| Inteiros positivos e negativos de 16 bits em complemento de 2. | | | | | |
| Long | long | long | int64_t | long | number |
| Inteiros positivos e negativos de 64 bits em complemento de 2. | | | | | |
| Float | float | float | float* | float | number |
| Ponto-flutuante de 32 bits - IEEE 754. | | | | | |
| Double | double | double | double* | float | number |
| Ponto-flutuante de 64 bits - IEEE 754. | | | | | |

*A especificação dos tipos ponto-flutuante em C++ podem variar, dependendo do hardware e compilador.

Tabela 8 – Tipos Primitivos Adicionais em CML.

partir dos valores de **width** e **height**, que não são *atributos derivados* e ao invés armazenam seus *valores*. Já no conceito **Square**, **width** e **height** são *atributos derivados* de **site_length**.

Em outro caso, agora na figura 19, observa-se que o atributo **area** de **UnitCircle** não é um *atributo derivado*, pois não tem o prefixo "/" antes do seu nome, apesar de ter uma *expressão* na sua definição. Neste caso, a *expressão* serve apenas para definir um valor inicial ao *atributo* na instanciação do conceito **UnitCircle**.

A tabela 9 apresenta o mapeamento de um *atributo derivado* do metamodelo de CML para os metamodelos de UML (OMG, 2015a) e ER (CHEN, 2011).

No metamodelo de UML, a metaclasse *Property* tem um meta-atributo chamado *isDerived*, que determina se um *atributo* é derivado ou não. Um *atributo derivado* em UML pode ser definido por uma expressão em OCL (OMG, 2014b); enquanto CML tem *expressões* como parte da linguagem, como é visto na seção 3.6.

| Metamodelo | Termo | Observações |
|---|---|---|
| CML | Derived Attribute | Em CML, a metaclasse *Property* possui um meta-atributo *derived*, e permite a definição usando *expressões* da própria linguagem. |
| UML | Derived Attribute | Em UML, a metaclasse *Property* possui um meta-atributo *isDerived*. A definição pode ser feita com *expressões* em OCL. |
| ER | — | *Atributos derivados* não são diferenciados de *atributos* de memória, ou são definidos como *operações de consulta*. |

Tabela 9 – Mapeamento de *atributos derivados* em CML para UML (OMG, 2015a) e ER (CHEN, 2011).

O metamodelo ER, na sua forma original, não permite a qualificação de *atributos derivados* como parte de um *Entity Type*, mas é possível definir *operações de consulta* nas quais o resultado pode ser equivalente a *valores* de *propriedades derivadas* em CML. Entretanto, pode-se dizer que ER, ao definir um *atributo* como uma função de um *conjunto de entidades* para um *conjunto de valores*, não prescreve que todos *atributos* devem ser baseados em memória, nem impede que a definição de uma função *atributo* seja feita por uma *expressão*. No me-

tamodelo de CML e em sua sintaxe, por outro lado, é necessário definir que um *atributo*, ou é baseado em memória (um *atributo não-derivado, não-abstrato*, ou seja um *slot*), ou então é derivado de uma *expressão*.

## 3.5 ASSOCIAÇÕES

Em CML, uma *associação* representa uma relação entre dois *conceitos*. Cada tupla desta relação representa uma ligação entre instâncias destes *conceitos*. Quando dois conceitos $A$ e $B$ possuem uma *associação* entre si, suas instâncias são ligadas de tal forma que é possível acessar uma instância do conceito $A$ a partir de uma instância do conceito $B$, ou vice-versa.

A tabela 10 apresenta o mapeamento de uma *associação* entre o metamodelo de CML e os metamodelos de UML (OMG, 2015a) e ER (CHEN, 2011).

| Metamodelo | Termo | Observações |
|---|---|---|
| CML | Association | Uma associação pode ser representada apenas por uma *Propriedade* no *Conceito* de origem, se for unidirecional e não haver restrição de cardinalidade na origem; ou pela metaclasse *Associação*, se for bidirecional. |
| UML | Association | Sempre representada pela metaclasse *Associação*. |
| ER | Relationship | Formalmente definido como uma tupla contendo referências às *entidades* que fazem parte do relacionamento. |

Tabela 10 – Mapeamento de *Associação* em CML para UML (OMG, 2015a) e ER (CHEN, 2011).

O metamodelo UML (OMG, 2015a) tem uma metaclasse chamada *Associação* que possui instâncias de *Propriedade* para as quais os tipos são instâncias de *Classe* participantes da *associação*.

No metamodelo CML, por outro lado, a metaclasse *Associação* é necessária somente quando se define *associações bidirecionais* com restrição de cardinalidade para os dois *papéis* da *associação*, como descrito na subseção 3.5.2. Enquanto *associações unidirecionais* (que não possuem restrição de cardinalidade na *origem* da *associação*) são re-

presentadas por instâncias da metaclasse *Propriedade*, como descrito na subseção 3.5.1.

No metamodelo ER (CHEN, 2011), cada *associação* é conhecida como um *conjunto de relacionamento*, sendo cada tupla neste conjunto chamada de *relacionamento*. Diferentemente de CML e UML, num modelo ER, as tuplas de um *conjunto de relacionamento* podem ser consultadas diretamente, não havendo a noção de *propriedade* em cada *Tipo de Entidade*, que permitiria acesso aos *relacionamentos*.

Assim como UML, CML também permite a definição de *associações derivadas* através de *expressões*. Este tipo de *associação* será descrito na subção 3.5.3.

### 3.5.1 Associações Unidirecionais

*Associações unidirecionais* em CML são representadas por instâncias da metaclasse *Propriedade*. O *conceito* que dá origem à associação possui a *propriedade* cujo o *tipo* é o nome do outro *conceito* que faz parte da *associação*. Somente instâncias do *conceito-origem* podem acessar instâncias do *conceito-destino* através de sua *propriedade*, que serve como o *papel* do *conceito-destino* se usarmos a terminologia da UML.

No exemplo da figura 14, as *propriedades* em CML que representam o destino de associações unidirecionais (*books*, *customers* e *orders* de *BookStore*) seriam equivalentes a *associações* em UML com os respectivos *papéis* (mostrando a direção de navegação e a cardinalidade junto ao nome de cada papel). Porém, em CML, é necessário apenas definir as *propriedades*.

No metamodelo ER, as tuplas de um *conjunto de relacionamento* (equivalente a uma *associação* em CML) podem ser consultadas diretamente, portanto não há necessidade de se expressar a direcionalidade da *associação*.

É importante esclarecer que, se existir a necessidade de restringir a cardinalidade na *origem* de uma *associação unidirectional*, esta não pode ser representada em CML pela metaclasse *Propriedade*. É necessário utilizar-se da metaclasse *Associação* e, portanto, transformá-la numa *associação bidirecional*.

### 3.5.2 Associações Bidirecionais

*Associações bidirecionais* possuem ligações que podem ser acessadas a partir de cada *instância* participante da *associação*. O acesso se dá através de uma *propriedade* (*papel*) que faz referência a uma ou mais *instâncias* do outro *conceito* participando da *associação*. O metamodelo de CML representa uma *associação bidirecional* com a metaclasse *Associação*.

No exemplo da figura 20, **Employment** é uma instância da metaclasse *Associação*. **Employment** define uma associação bidirecional entre **Employee** e **Organization** através da propriedade **employer** de **Employee** e da propriedade **employees** de **Organization**.

```
@concept Employee
{
    name: String;
    employer: Organization;
}

@concept Organization
{
    name: String;
    employees: Employee*;
}

@association Employment
{
    Employee.employer;
    Organization.employees;
}
```

Figura 20 – Exemplo de *associação bidirecional*.

Devido à bidirecionalidade da associação **Employment**, se a propriedade **employer** de uma instância de **Employee** faz referência a uma instância de **Organization**, torna-se possível acessar aquela instância de **Employee** a partir da propriedade **employees** da instância de **Organization**. Isto se dá porque toda instância de **Employee** que atribuir à sua propriedade **employer** uma referência a uma instância de **Organization** terá sua referência automaticamente incluída na propriedade **employees** daquela instância de **Organization**.

Uma restrição importante em CML é não ser possível definir uma *associação bidirecional* entre dois *conceitos* onde o *tipo* de am-

bas as *propriedades* participantes da *associação* tenha cardinalidade maior que zero. Esta restrição é necessária porque pelo menos uma das *instâncias* participantes precisa ser criada antes que uma *ligação* seja estabelecida.

### 3.5.3 Associações Derivadas

*Associações derivadas* são representadas por *propriedades derivadas*, que associam um *conceito-origem* a um *conceito-destino* através de *expressões*. Para se obter referência a um conceito-destino a partir de um conceito-origem, é preciso avaliar a *expressão* da *propriedade*. Entre as sub-expressões que formam a expressão da associação derivada, existem *caminhos* para outras *associações*, que servem como base para a *associação derivada*.

No exemplo da figura 14, a expressão de **orderedBooks** em **BookStore** é um exemplo de expressão usando um *caminho* (seção 3.6.2) para definir uma associação derivada. Compreensões (seção 3.6.7) também podem ser usadas para definir *associações derivadas*.

Enquanto *associações derivadas* em CML são necessariamente *unidirecionais*, por serem definidas por *expressões*, UML não impõe tal restrição, permitindo modelar *associações bidirecionais derivadas* em seu modelo. Este caso pode ser modelado em CML através de duas *associações derivadas*, uma definida na *origem* e outra no *destino*, desde que ambas sejam derivadas a partir de *expressões* baseadas na mesma *associação bidirecional*.

## 3.6 EXPRESSÕES

Em CML, *expressões* são usadas para atribuir valores iniciais a *atributos* ou para definir *propriedades derivadas*. Como visto nos exemplos das seções anteriores, existem vários tipos de *expressões* em CML, permitindo a definição de valores simples de *tipos primitivos* até *associações derivadas* mais complexas. Diferentemente de UML, que utiliza OCL como uma linguagem auxiliar para definir *expressões*, CML incorpora *expressões* em sua própria sintaxe, permitindo assim que modelos conceituais sejam completamente definidos pela linguagem.

As subseções seguintes vão apresentar cada um dos tipos de expressões suportadas por CML.

### 3.6.1 Expressões Literais

Para cada um dos *tipos primitivos* suportados por CML, existem *expressões literais* que permitem representar seus valores.

A tabela 11 apresenta exemplos e a sintaxe das *expressões literals* para os *tipos primitivos* essenciais, os quais são suficientes para modelar um grande conjunto de domínios.

| Tipo | Sintaxe | Exemplo |
|---|---|---|
| String | `'"' ('\\'[btnr"\\] \| . )*? '"'` | `"Hello!\n"` |
| Boolean | `'true' \| 'false'` | `true` |
| Integer | `('0'..'9')+` | `123456` |
| Decimal | `('0'..'9')* '.' ('0'..'9')+` | `1234.567` |

Tabela 11 – Expressões Literais dos Tipos Primitivos Essenciais

A tabela 12 apresenta as *expressões literais* dos *tipos primitivos* adicionais. Este são necessários quando se precisa ter acesso aos tipos correspondentes das linguagens de programação convencionais, tais como inteiros curtos/longos e ponto-flutuantes.

| Tipo | Sintaxe | Exemplo |
|---|---|---|
| Byte | `('0'..'9')+ 'b'` | `127b` |
| Short | `('0'..'9')+ 's'` | `1234s` |
| Long | `('0'..'9')+ 'l'` | `12345671` |
| Float | `('0'..'9')* '.' ('0'..'9')+ 'f'` | `1234.567f` |
| Double | `('0'..'9')* '.' ('0'..'9')+ 'd'` | `1234.567d` |

Tabela 12 – Expressões Literais dos Tipos Primitivos Adicionais

É importante observar que, como cada *tipo primitivo* tem seus próprios valores literais, é possível determinar o tipo de cada *expressão literal* em CML. Este recurso é necessário porque CML faz inferência automática do tipo de *propriedades* definidas por uma *expressão*, e também porque permite evitar a combinação de expressões *numéricas* e de *ponto-flutuante*, que poderiam gerar perda de precisão. Em contraste, OCL possui uma única sintaxe para todos os tipos númericos, e por isto OCL não pode inferir os *tipos primitivos* a partir de suas *expressões literais*.

### 3.6.2 Expressões de Caminho

*Expressões de caminho* permitem acessar os valores de *propriedades* de um *conceito*, sejam *atributos* ou *destinos de associações*, e também valores de *parâmetros* em *expressões lambda*.

No exemplo da figura 21, no conceito **Rectangle**, o atributo **area** é definido pela multiplicação dos valores dos atributos **width** e **height**, que são referenciados por duas *expressões de caminho*. O resultado destas expressões serão os respectivos valores das *atributos* correspondentes.

Também na figura 21, o conceito **BookStore** contém a associação derivada **ordered_books**, que é definida por uma *expressão de caminho*. Esta percorre os destinos de associação **orders** de **BookStore**, **items** de **Order** e **book** de **Item**. O resultado é uma sequência contendo referências a todas as instâncias de livro que já foram pedidos na *BookStore*.

```
@concept BookStore
{
    orders: Order*;

    /ordered_books = orders.items.book;
}

@concept Order
{
    items: Item*;
}

@concept Item
{
    book: Book;
    qty: integer;
    /description = book.title;
    /amount = qty * price;
}

@concept Book
{
    title: string;
    price: decimal;
}
```

Figura 21 – Exemplos de Expressões de Caminho.

Outra forma de interpretar o caminho de expressão *orders.items.book* é transformá-lo na seguinte expressão equivalente em OCL:

```
orders->collect(items)->flatten()->collect(book)
```

Na expressão OCL acima, enumera-se a coleção de **orders** para coletar as coleções de **items** de cada instância de **Order** em **orders**. Como o resultado é uma coleção de outras coleções, usa-se **flatten()** para transformá-la numa única coleção de instâncias de **Item**. Finalmente, enumera-se esta última coleção a fim de gerar uma coleção com as instâncias de **Book** que são associadas às instâncias de **Item**.

Pela explicação acima, é possível perceber, que apesar de sucintas, as *expressões de caminho* envolvem uma série de operações, sendo assim uma forma poderosa de acessar instâncias e valores de *propriedades*.

### 3.6.3 Expressões com Operadores

A linguagem CML permite o uso de operadores aritméticos, relacionais, lógicos e referenciais. A maior parte dos operadores são infixados, com apenas três prefixados. O uso de parênteses é permitido a fim de estabelecer precedência.

A tabela 13 mostra os operadores aritméticos em ordem de precedência. Estes operadores só aceitam como operandos as expressões de tipos primitivos númericos e de ponto-flutuante. Os operadores de adição e subtração também podem ser prefixados.

| Operadores | Operações | Associatividade | Exemplo |
|---|---|---|---|
| `^` | Exponenciação | Direita | `3 ^ 2` |
| `*`, `/`, `%` | Mult., Div., Mod. | Esquerda | `6 * 2 / 3 % 4` |
| `+`, `-`* | Adição, Subtração | Esquerda | `6 + 2 - 1` |

*Os operadores de adição e subtração também podem ser prefixados.

Tabela 13 – Operadores Aritméticos em Ordem de Precedência

A tabela 14 mostra os operadores relacionais. Estes operadores só aceitam como operandos as expressões de tipos relacionais; o que inclui **String**, os tipos númericos e os de ponto-flutuante; são excluídos o tipo **Boolean** e os tipos referenciais.

A tabela 15 mostra os operadores referenciais. Estes operadores só aceitam como operandos as referências a instâncias de conceitos; o

| Operadores | Operações | Exemplo |
|---|---|---|
| `<`, `<=`, `>`, `>=` | Ordenação | `3 < 2` |
| `==`, `!=` | Igualdade, Desigualdade | `6 != 3` |

Tabela 14 – Operadores Relacionais

que exclue todos os tipos primitivos. O operador de conversão incondicional de tipo (`as!`) pode causar uma exceção em tempo de execução se o tipo real da instância não for compatível com o tipo esperado. O operador de conversão condicional (`as?`) seleciona automaticamente somente as instâncias compatíveis e nunca causa uma exceção.

| Oper. | Operação | Tipo Resultante | Exemplo |
|---|---|---|---|
| `===` | Igualdade Referencial | Boolean | `c1 === c2` |
| `!==` | Desigualdade Referencial | Boolean | `c1 !== c2` |
| `is`, `isnt` | Verificação de Tipo | Boolean | `member isnt Task` |
| `as!` | Conversão Incondicional de Tipo | Tipo e cardinalidade do operando 2 | `members as! Task*` |
| `as?` | Conversão Condicional de Tipo | Tipo do operando 2, somente cardinal. opcional (?) ou sequência (*) | `member as? Task?` |

Tabela 15 – Operadores Referenciais

A tabela 16 mostra os operadores lógicos em ordem de precedência. Estes operadores só aceitam como operandos as expressões do tipo Boolean. Com exceção do operador de negação (`not`), que é prefixado, todos os outros operadores lógicos são infixados. A associatividade dos operadores infixados é sempre da esquerda para a direita.

Finalmente, a tabela 17 mostra a ordem de precedência entre as diferentes classes de operadores.

| Operadores | Operações | Exemplo |
|---|---|---|
| `not` | Negação | `not p` |
| `and` | Conjunção | `p and q` |
| `or` | Disjunção | `p or q` |
| `xor` | Disjunção Exclusiva | `p xor q` |
| `implies` | Implicação | `p implies q` |

Tabela 16 – Operadores Lógicos em Ordem de Precedência

| Operadores | Classe |
|---|---|
| `+`, `-`, `not` | Prefixados |
| `+`, `-`, `*`, `/`, ... | Aritméticos |
| `<`, `>`, ... | Relacionais |
| `===`, `!==`, ... | Referenciais |
| `and`, `or`, ... | Lógicos |

Tabela 17 – Classes de Operadores em Ordem de Precedência

### 3.6.4 Expressões Condicionais

As expressões condicionais permitem alternar a avaliação de uma ou mais expressões. O operando da condição que determina a alternância é sempre uma expressão do tipo Boolean. Os demais operandos podem ser de qualquer tipo, incluindo os tipos primitivos.

Estas expressões se dividem em três categorias: unária – somente avalia a única expressão se a condição for verdadeira; binária – resulta na primeira expressão se a condição for verdadeira e na segunda expressão se a condição for falsa; opcional – resulta na expressão que tem um valor presente.

A tabela 18 mostra as expressões condicionais.

| Forma | Semântica | Exemplo |
|---|---|---|
| `<expr> given <cond>` | Se <cond> é verdadeira, retorne <expr>, senão resultado é vazio. | `10 if i > 2` |
| `<expr> unless <cond>` | Se <cond> é falsa, retorne <expr>, senão resultado é vazio. | `10 unless i <= 2` |
| `if <cond> then <expr1> else <expr2>` | Se <cond> é verdadeira, retorne <expr1>, senão retorne <expr2>. | `if i > 2 then 10 else 5` |
| `<expr1> or? <expr2>` | Retorne <expr1> se resultar num valor, senão retorne <expr2> se resultar num valor, senão o resultado é vazio. | `item.book or? item.other` |

Tabela 18 – Expressões Condicionais

Para as expressões condicionais unárias (`if` e `unless`), o tipo resultante sempre é o tipo do operando `expr`. Se a cardinalidade do tipo do operando for sequência (`*`), a cardinalidade do resultado também será sequência. Senão, a cardinalidade do resultado é opcional (`?`).

No caso da expressão condicional binária (`if-then-else`), o tipo resultante é o mais próximo super-tipo entre `expr1` e `expr2`. A cardinalidade do resultado será aquela que abrange a cardinalidade das duas expressões. Se não foi possível encontrar um super-tipo, a expressão é considerada inválida.

Já na expressão condicional opcional, se `present(<expr1>)` é verdadeiro, retorna `<expr1>`. Se `present(<expr1>)` é falso e `present(<expr2>)` é verdadeiro, retorna `<expr2>`. Senão, o resultado é vazio. Assim como nas expressões binárias, tipo resultante é o mais próximo super-tipo entre o primeiro e o segundo operando, e a cardinalidade do resultado será aquela que abrange a cardinalidade das duas expressões.

### 3.6.5 Invocações

*Invocações* permitem aplicar uma *função* a um número de argumentos que correspondem aos seus parâmetros. Em CML, existem algumas *funções* pré-definidas pelo módulo **cml_base**, mas novas funções podem ser definidas através de **@function**.

No exemplo da figura 22, o conceito **Order** define um atributo derivado **total_items** através da *invocação* da função **count()**, que é definida em **cml_base**. Como único argumento, a função **count()** recebe uma sequência de valores provenientes do caminho **items** e, quando avaliada, retorna o número de elementos nesta sequência.

```
@concept Order
{
  items: Item*;
  total_items = count(items);
}

@concept Item;
```

Figura 22 – Exemplos de Invocações.

Diferentemente de CML, OCL não permite a *invocação* de *funções*. As *invocações* em OCL são de *operações* ou *métodos* sobre uma instância de uma *classe* (usando o operador ".") ou de uma *coleção* (usando o operador "->").

### 3.6.6 Lambdas

CML permite o uso de *expressões lambda* como um argumento na *invocação* de uma *função*. Existem várias *funções* pré-definidas pelo módulo **cml_base** que se utilizam de *expressões lambda*.

No exemplo da figura 23, o conceito **BookStore** define uma associação derivada **premium_orders**, que é definida pela invocação da função **select**. O primeiro argumento da invocação define a associação **orders** como a origem da derivação. O segundo argumento define uma expressão lambda que seleciona elementos da propriedade **orders**. A invoção de **select** retorna, neste caso, somente as instâncias em **orders** cuja a avaliação da expressão $\boldsymbol{total > 1000.00}$ seja verdadeira.

Observe que *expressões lambda* são colocadas entre colchetes, como no exemplo de **premium_orders**. Os colchetes delimitam sin-

```
@concept BookStore
{
    orders: Order*;

    /premium_orders = select(orders, { total > 1000.00 });
}

@concept Order
{
    total: Decimal;
}
```

Figura 23 – Exemplo do Uso de *Expressões Lambda*

taticamente o escopo em que será executada a expressão. Em ***total* > 1000.00**, o **total** será avaliado dentro do escopo da *expressão lambda*, sempre que esta for invocada pela função **select**. Neste caso, **total** refere-se ao atributo de uma instância de **Order** encontrada em **orders**.

Em OCL, também é permitido o uso de *lambda* (chamado de *closure* em OCL) nas *operações* sobre *coleções*.

### 3.6.7 Compreensões

CML permite um tipo de expressão chamada de *compreensão*, que é uma forma flexível e expressiva de construir novas sequências, ou de calcular resultados, a partir de sequências existentes. As *compreensões* se assemelham à notação de construção de conjuntos em Matemática, ou às *compreensões de lista* na linguagem Python, ou ainda às *compreensões* ***for*** na linguagem Scala.

No exemplo da figura 24, o conceito **BookStore** define uma associação derivada **premium_orders**, que é definida por uma *compreensão*. A sequência de entrada definida pela propriedade **orders**, que serve como a associação original, é seguida por uma barra vertical. Ao lado direito da barra, encontra-se uma expressão com a palavra-chave *select* seguida de dois-pontos e da expressão ***total* > 1000.00**.

Da mesma forma como correu com a invocação do exemplo da figura 23, a *compreensão* da figura 24 retorna somente as instâncias em **orders** cuja a avaliação da expressão ***total* > 1000.00** seja verdadeira. De fato, o compilador converte a *compreensão* da figura 24 na invocação da figura 23.

```
@concept BookStore
{
    orders: Order*;

    /premium_orders = orders | select: total > 1000.00;
}

@concept Order
{
    total: Decimal;
}
```

Figura 24 – Exemplos de uma expressão de *compreensão*.

As *compreensões* em CML são encadeadas usando barras verticais, enquando *operações sobre coleções* em OCL usam "->", como ilustrado na figura 25.

```
// Em OCL:

top_orders->select(items->exists(qty > 10))
          ->select(total > 100)

// Em CML:

top_orders | select: (items | exists: qty > 10)
           | select: total > 100;
```

Figura 25 – Comparação sintática de uma *compreensão* em CML com seu equivalente em OCL.

## 3.7 COMPARAÇÃO DE CML COM UML / OCL E ER

A tabela 19 resume a comparação de CML com os metamodelos de UML / OCL e ER.

| CML | UML / OCL | ER |
|---|---|---|
| Conceito | Classe | Tipo de Entidade |
| Propriedades | Atributos / Association Ends | Atributos / Papéis |
| Associações (incl. unidirecionais) | Associações | Tipo de Relacionamento |
| Atributos / Associações Derivados | Idem | Operações de Consulta |
| Generalização / Especialização / Herança Múltipla | Idem, incluindo operações | Não Disponível |
| Abstrações | Classes Abstratas | Não Disponível |
| Atributos / Associações Abstratas | Não Disponível | Não Disponível |
| Consultas Extensíveis | Parcialmente através de closures em OCL | Não Disponível |

Tabela 19 – Comparação de CML com UML / OCL e ER

A seguir, algumas das diferenças são enumeradas:

- CML define *conceitos* através de *atributos* e *associações*, a fim de manter-se agnóstico de paradigmas de programação, enquanto UML também permite a definição de *operações* em *classes*.
- CML define tanto *atributos* quanto *associações* através de *propriedades*, enquanto UML requer a definição explícita de *associações*, sejam elas *unidirecionais* ou *bidirecionais*.
- ER não permite a definição explícita de *atributos* e *associações derivados*, tal como em CML e OCL.

- Generalização / especialização são suportados tanto por CML quando UML, mas não na versão original de ER. UML também permite a herança de *operações*.
- *Abstrações*, ou *classes abstratas*, não estão disponível em ER.
- *Atributos* e *associações abstratas* são possíveis em CML, mas não em UML e ER.
- CML permite maior extensibilidade nas consultas usadas na definição de atributos e associações derivadas através de *funcões*, enquanto OCL já é mais restrito, não permitindo a definição de *funções* nas linguagens-alvo. Já, ER, baseado no modelo relacional, só permite as *operações* pré-definidas deste modelo.

Pode-se então verificar que CML possui um conjunto básico de construções em seu metamodelo que permitem a modelagem conceitual de forma textual. CML tem um poder de exprevisidade similar a UML (embora limitado, comparado a este) e mais rico que o metamodelo de ER.

# 4 CML: O COMPILADOR

Este capítulo apresenta o compilador CML, que permite gerar código em qualquer linguagem de programação. Para tanto, este precisa de *templates* escritos especificamente para gerar código numa determinada linguagem. Um conjunto padrão de *templates* são fornecidos com a biblioteca base do compilador CML. Bibliotecas de terceiros podem fornecer seus próprios *templates* a fim de gerar código para tecnologias ou plataformas específicas. Desenvolvedores também podem estender *templates* existentes para adaptar as implementações às características específicas de seus projetos.

A seção 4.1 apresenta a arquitetura do compilador CML, enquanto as próximas seções apresentam seus componentes: a seção 4.2 introduz as noções de *construtores* e *tarefas*; a seção 4.3 apresenta em mais detalhes os *templates* que suportam a geração de código; e, por fim, a seção 4.4 aborda *módulos* e *bibliotecas*.

## 4.1 ARQUITETURA DO COMPILADOR

A arquitetura do compilador CML é apresentada na figura 26.

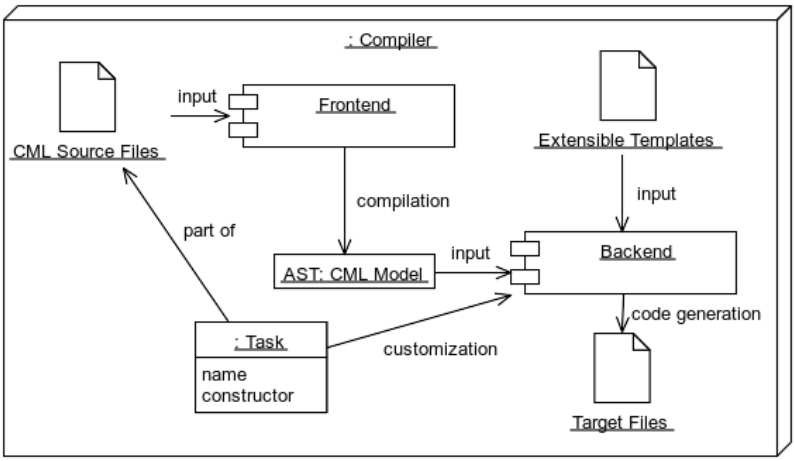


Figura 26 – Uma visão geral da arquitetura do compilador CML.

Os dois componentes principais do compilador CML, juntamente com os tipos de artefatos que eles manipulam, são:

- *Frontend:* processa os *arquivos-fonte CML*, gerando uma representação interna de um *modelo CML*. Durante o processamento, validações sintáticas e semânticas são executadas, e erros são apresentados ao desenvolvedor. Se os arquivos-fonte são processados e validados de forma bem sucedida, a representação interna (AST) do modelo CML serve de entrada para o *Backend*.
- *Backend:* recebe a AST do modelo CML como entrada. Baseado na especificação-alvo provida juntamente com o modelo CML, o Backend seleciona que *templates* usar na geração de código. Assim, os *arquivos-alvo* são gerados e ficam disponíveis para serem consumidos por outras ferramentas.

Observa-se na figura 26 a importância dos *templates* na geração de código. Estes servem de entrada, juntamente com a AST, para o *Backend*. De certa forma, é possível considerá-los uma espécie de *código-fonte*, assim como o código de entrada em CML, pois não seria possível executar a geração de código sem os *templates* e estes também podem ser fornecidos pelos próprios desenvolvedores.

## 4.2 CONSTRUTORES E TAREFAS

Uma vez capturada a essência de um modelo conceitual usando a linguagem CML (como apresentado no capítulo 3), é possível então gerar código para diversas finalidades a partir deste modelo. Diferentemente de compiladores tradicionais, o processo de geração de código do compilador CML pode ser modificado ou complementado a fim de se gerar código apropriado para diferentes tipos de aplicações, tecnologias e linguagens de programação.

Foi necessário então adicionar ao metamodelo da linguagem CML dois termos – *construtores* e *tarefas* – que organizam a estrutura de templates de geração de código, permitindo sua extensão e execução. Os dois termos, e a teoria por trás deles, foram definidos por David Deutsch no artigo Teoria de Construtores (DEUTSCH, 2013). Esta teoria foi concebida como um arquetipo para expressar teorias científicas; porém, devido à sua aplicabilidade, CML usa esta teoria para organizar o processo de geração de código.

Um *construtor* é algo que pode causar a transformação de um determinado substrato em outro. Em CML, um construtor representa

um grupo de templates de geração de código; o substrado de entrada é a representação interna (AST) do modelo definido na linguagem CML; e o substrato de saída são os arquivos-alvo gerados pelos templates. O processo de geração de código é chamado de *construção*.

Para que ocorra uma *construção* a partir de um *modelo de entrada*, é preciso definir o *alvo* desejado como saída e o *construtor* que executará a transformação. Em CML, estes elementos são definidos por uma *tarefa*. Um módulo CML pode conter uma ou mais tarefas que definem, além de seu construtor, alguns atributos que orientam o construtor a gerar arquivos-alvo de acordo com certos parâmetros.

A figura 27 mostra dois exemplos de tarefas definidas em CML: a tarefa **livir_poj** usa o construtor **poj** para gerar código em Java; e a tarefa **livir_pop** usa o construtor **pop** para gerar código em Python. Cada uma das tarefas especifica alguns atributos que irão definir o nome de módulos, diretórios e arquivos gerados.

```
@task livir_poj: poj
{
    groupId = "livir";
    artifactId = "livir-books";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "livir.books";
    packagePath = "livir/books";
}

@task livir_pop: pop
{
    moduleName = "livir_books";
    moduleVersion = "1.0";
}
```

Figura 27 – Exemplos de *tarefas* definidas em CML.

## 4.3 TEMPLATES DE GERAÇÃO DE CÓDIGO

Os templates extensíveis usados pelo compilador CML são implementados em StringTemplate. Parr desenvolveu a linguagem StringTemplate para geração de código (PARR, 2004). Esta linguagem foi adotada para uso no compilador CML por garantir a separação entre os modelos definidos em CML e os templates utilizados na geração de código.

Os templates são definidos através de expressões sem efeito colateral, ou seja, as expressões permitem percorrer os modelos sem causar a alteração do estado do compilador. As expressões também são independentes de ordem de execução, ou seja, um template pode ser invocado a partir de qualquer ponto de outro template e sempre gerará o mesmo resultado se os mesmos parâmetros forem utilizados. Outra vantagem de StringTemplate é o fato de reforçar um estilo mais declarativo/funcional (ao invés de imperativo), impedindo assim a inclusão de código mais complexo nos próprios templates, o que melhora a manutenibilidade dos mesmos.

Se considerarmos uma gramática de linguagem como uma função que gera modelos a partir de texto (função esta especificada de forma declarativa através de produções), pode-se dizer que um template escrito em StringTemplate é basicamente a função inversa de uma gramática (também especificada de forma declarativa), gerando texto a partir de um modelo.

O compilador CML usa StringTemplate para dois propósitos:

- *Definição de nomes de arquivos e estrutura de diretórios:* cada construtor (definido na seção 4.2) gera sua própria estrutura de diretórios através de um arquivo de template chamado "files.stg". Este template pode usar os atributos definidos por uma tarefa (também definida na seção 4.2) para derivar o nome dos arquivos-alvo e o nome de seus diretórios. Um exemplo é mostrado abaixo:

```
model_files(task, model) ::= <<
pom_file|pom.xml
>>

concept_files(task, concept) ::= <<
concept_file|<task.packagePath>/<concept.name; format="pascal-case">.java
>>
```

- *Geração do conteúdo dos arquivos-alvo:* Cada arquivo-alvo listado no template "files.stg" precisa fazer referência a um outro template que gera o contéudo do arquivo-alvo. O *template de conteúdo* recebe como entrada um elemento-raiz do modelo CML, que provê toda a informação necessária para gerar o seu conteúdo. O tipo de elemento do modelo recebido como entrada pelo template de conteúdo depende de qual função do template "files.stg" listou o arquivo-alvo. Por exemplo, se o arquivo-alvo foi listado por **concept_files**, então o elemento-raiz passado para o template de conteúdo será um *conceito* do modelo CML. Um arquivo de conteúdo típico é mostrado abaixo:

```
import "/design/poj.stg"

concept_file(task, concept) ::= <<
package <task.packageName>;

import java.util.*;

public <class(concept)>
>>
```

No exemplo que mostra um template "files.stg", dois tipos de arquivo-alvo são criados pelo construtor. Um arquivo para o módulo "pom.xml" (baseado no template "pom_file"); e um arquivo com a extensão ".java" para cada conceito do modelo CML (baseado no template "concept_file"). No exemplo do template de conteúdo, "concept_file" gera uma classe DTO (Data Transfer Object) em Java. O template que define uma DTO é importado de "/design/poj.stg" e é aplicado pela invocação **class(concept)**.

## 4.4 MÓDULOS E BIBLIOTECAS

Ao desenvolver um único componente, é suficiente manter um único diretório com todos os arquivos-fonte de um modelo CML. Porém, quando mais de um componente está sendo desenvolvido como parte de um sistema maior, é necessário separar o código-fonte em CML compartilhado entre os vários componentes. Alguns sistemas também lidam com vários domínios, o que torna benéfico a separação do código-fonte em vários modelos.

Para permitir isto, CML introduz a noção de *módulos*. Agrupando um conjunto de elementos de um modelo CML, um módulo em CML é conceitualmente similar aos pacotes em UML (OMG, 2015a). Fisicamente, cada módulo é um diretório no sistema de arquivos contendo os seguintes sub-diretórios:

- *source*: onde fica o código-fonte escrito em CML.
- *templates*: um diretório opcional contendo arquivos de templates.
- *tests*: outro diretório opcional contendo testes que verificam o código gerado.
- *targets*: criado pelo compilador CML; contém os arquivos-alvo gerados por cada tarefa executada pelo módulo.

A figura 28 é um exemplo de uma estrutura de diretório típica de um módulo CML.

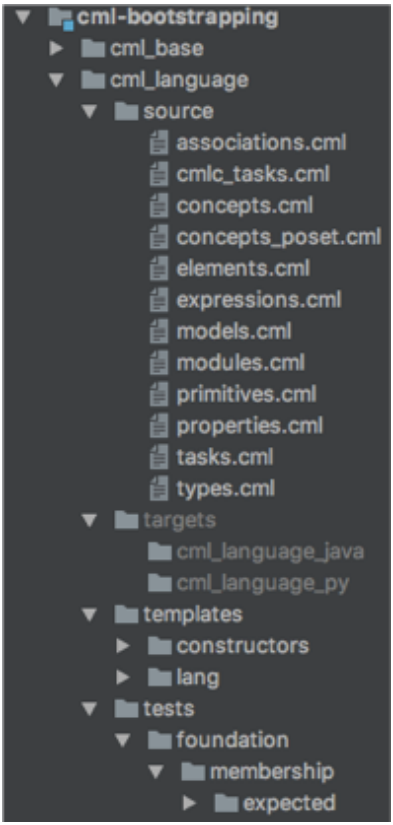


Figura 28 – Estrutura de Diretório de Módulo CML

Um módulo é definido em CML pela declaração **@module**, como ilustrado na figura 29. Se um módulo precisa fazer referência a elementos de outro módulo, a declaração **@module** pode incluir uma declaração **@import** que define o nome do módulo contendo os elementos desejados. O compilador irá então compilar o módulo importado antes do módulo corrente.

```
@module livir_books
{
    @import livir_products;
}
```

Figura 29 – Exemplo de declaração de *módulo* em CML.

Um módulo CML não tem especificação de versão, sendo mantido dentro do mesmo repositório de código que contém os módulos que importa. Entretanto, é planejado para uma versão futura de CML o empacotamento de um módulo em uma biblioteca, a qual teria uma versão e o mesmo nome do módulo. A biblioteca então poderia ser publicada num repositório público (ou de uma organização) para que seja compartilhada com outros desenvolvedores e projetos. Uma biblioteca CML seria desta forma o empacomento de um módulo para distribuição com uma versão específica.

# 5 O PROCESSO DE DESENVOLVIMENTO DE CML

A figura 30 ilustra o processo de desenvolvimento da linguagem CML e do seu compilador. A seção 5.1 descreve as atividades envolvidas na implementação do compilador, enquanto a seção 5.2 aborda o processo de *bootstrapping* do metamodelo do compilador.

O processo iniciou-se com o desenvolvimento de um protótipo para validar a viabilidade das tecnologias usadas no projeto, tais como ANTLR (PARR, 2007) e StringTemplate (PARR, 2004); e também para definir a arquitetura do compilador, como apresentada no capítulo 4. A seguir, pesquisou-se as soluções existentes (relatadas no capítulo 2), que serviram de inspiração e de bases de comparação durante o desenvolvimento.

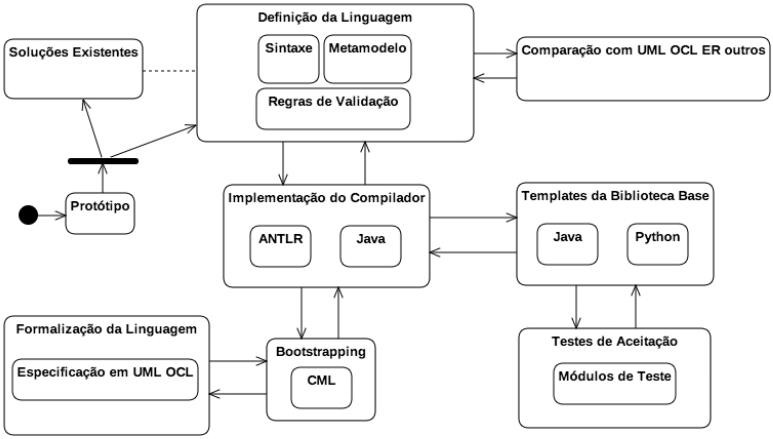


Figura 30 – Diagrama do Processo de Desenvolvimento de CML

Como pode-se verificar no diagrama, o processo de desenvolvimento de CML não tem estado final, ou seja, o processo é iterativo e contínuo; cada uma das atividades se repete continuamente, aprimorando a linguagem e o compilador. O resultado apresentado neste trabalho é um *snapshot* inicial do que ainda se pretende realizar em trabalhos futuros, como enumerados na seção 7.1.

## 5.1 IMPLEMENTAÇÃO E TESTE DO COMPILADOR

Como mostra a figura 30, a definição da linguagem se deu em paralelo com sua implementação. Na medida em que a sintaxe era definida em ANTLR, o metamodelo correspondente da linguagem era definido em Java, e os templates da biblioteca básica eram definidos em StringTemplate. Módulos de testes eram então definidos (usando a própria infra-estrutura de teste do compilador) para se verificar as novas construções acrescentadas na linguagem. Erros na execução dos testes exigiam modificações tanto na sintaxe, quanto no metamodelo e nos templates. Este processo iterativo se repetia para cada nova construção da linguagem.

Outro aspecto importante durante a definição da linguagem foi a comparação com o metamodelo e as construções existentes nas linguagens UML e OCL, a fim de guarantir que a linguagem CML possua um número essencial de construções de modelagem, e também para que CML seja bem fundamentada semanticamente. O resultado destas comparações foi relatado no capítulo 3.

A infra-estrutura de teste permitiu a definição de módulos cuja a entrada é código de teste em CML. Ao executar um módulo de teste, o próprio compilador CML verifica automaticamente se a saída corresponde ao código desejado na linguagem-alvo, reportando as diferenças encontradas entre o código gerado e o código esparado. A verificação de código gerado em Java e Python permitiu validar a flexibilidade do metamodelo da linguagem CML e dos templates da biblioteca básica do compilador CML; visto que Java é uma linguagem estática, fortemente tipada, enquanto Python é uma linguagem dinâmica com sintaxe muito característica, bem diferente da família de linguagens que se basearam na sintaxe do C, como é o caso do Java.

A infra-estrutura de teste do compilador também permite executar o código gerado em Python e em Java, a fim de verificar seu comportamento em tempo de execução. A figura 31 mostra a execução de um módulo de teste pelo compilador CML.

```
Testing associations/bidirectional with task vehicles_cmlc_py:
- Verifying the compiler's output ...OK
- Verifying missing target files ...OK
- Verifying target file: setup.py ...OK
- Verifying target file: cml_associations_bidirectional_vehicles_cmlc/__init__.py ...OK
- Checking types of Python module: cml_associations_bidirectional_vehicles_cmlc ...OK
- Installing Python package: vehicles_cmlc_py ...OK
- Running Python client: clients/vehicles_cmlc_py/client.py ...OK
- SUCCESS

Testing associations/bidirectional with task vehicles_pop:
- Verifying the compiler's output ...OK
- Verifying missing target files ...OK
- Verifying target file: setup.py ...OK
- Verifying target file: cml_associations_bidirectional_vehicles_pop/__init__.py ...OK
- Checking types of Python module: cml_associations_bidirectional_vehicles_pop ...OK
- Installing Python package: vehicles_pop ...OK
- Running Python client: clients/vehicles_pop/client.py ...OK
- SUCCESS

Testing specializations/redefinitions with task shapes_cmlc_java:
- Verifying the compiler's output ...OK
- Verifying missing target files ...OK
- Verifying target file: pom.xml ...OK
- Verifying target file: src/main/java/shapes/cmlc/Rectangle.java ...OK
- Verifying target file: src/main/java/shapes/cmlc/Circle.java ...OK
- Verifying target file: src/main/java/shapes/cmlc/UnitCircle.java ...OK
- Verifying target file: src/main/java/shapes/cmlc/Square.java ...OK
- Verifying target file: src/main/java/shapes/cmlc/RedUnitCircle.java ...OK
- Verifying target file: src/main/java/shapes/cmlc/Shape.java ...OK
- Verifying target file: src/main/java/shapes/cmlc/Rhombus.java ...OK
- Building Maven module: shapes_cmlc_java .....................................................OK
- Running Java client: clients/shapes_cmlc_java/pom.xml ........................OK
- SUCCESS

Testing specializations/redefinitions with task shapes_cmlc_py:
- Verifying the compiler's output ...OK
- Verifying missing target files ...OK
- Verifying target file: setup.py ...OK
- Verifying target file: cml_specializations_redefinitions_shapes_cmlc/__init__.py ...OK
- Checking types of Python module: cml_specializations_redefinitions_shapes_cmlc ...OK
- Installing Python package: shapes_cmlc_py ...OK
- Running Python client: clients/shapes_cmlc_py/client.py ...OK
- SUCCESS

Testing expressions/types with task poj:
- Verifying the compiler's output ...OK
- Verifying missing target files ...OK
- Verifying target file: src/main/java/types/poj/Descendant.java ...OK
- Verifying target file: src/main/java/types/poj/Ancestor.java ...OK
- Verifying target file: src/main/java/types/poj/Types.java ...OK
- Building Maven module: poj ..................................................OK
- SUCCESS

Testing expressions/types with task pop:
- Verifying the compiler's output ...OK
- Verifying missing target files ...OK
- Verifying target file: cml_expressions_types_pop/__init__.py ...OK
- Checking types of Python module: cml_expressions_types_pop ...OK
- Installing Python package: pop ...OK
- SUCCESS

Testing specializations/compatible_generalizations with task compatible_generalizations:
- Verifying the expected errors from compiler ...OK
- SUCCESS

Testing specializations/abstract_property_redefinition with task ...
- Verifying the expected errors from compiler ...OK
- SUCCESS

Testing specializations/generalization_compatible_redefinition with task ...
- Verifying the expected errors from compiler ...OK
- SUCCESS

Testing specializations/conflict_redefinition with task conflict_redefinition:
- Verifying the expected errors from compiler ...OK
- SUCCESS

Testing specializations/not_own_generalization with task not_own_generalization:
- Verifying the expected errors from compiler ...OK
- SUCCESS
```

Figura 31 – Execução de Módulos de Teste

## 5.2 BOOTSTRAPPING DO METAMODELO

Assim que uma versão relativamente completa do compilador estava disponível, iniciou-se o processo de *bootstrapping*, onde o metamodelo da linguagem CML começou a ser convertido da linguagem Java para a própria linguagem CML, como ilustrado na figura 32.

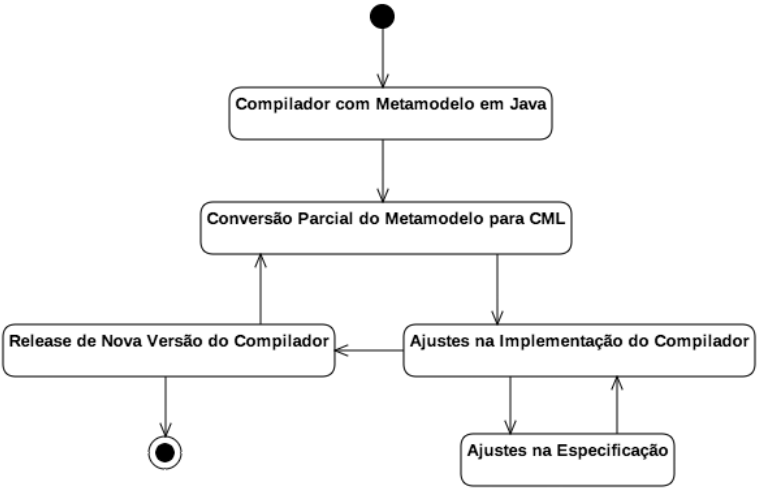


Figura 32 – Diagrama do Processo de Bootstrapping do Metamodelo de CML

Para que o processo de *bootstrapping* fosse auto-sustentável, apenas parte do metamodelo foi selecionada para conversão em cada iteração. Uma vez validada a conversão de uma parte do metamodelo, uma nova versão do compilador era gerada e usada para se converter a próxima parte. Assim foi possível guarantir que o compilador continuasse funcionando e seu metamodelo fosse gradativamente convertido para CML.

Durante o processo de *bootstrapping* foram detectadas uma série de deficiências tanto no metamodelo, como no código gerado, que não haviam sido verificadas originalmente com os módulos de teste. Assim, alterou-se a linguagem e os testes para se corrigir as deficiências, melhorando tanto a linguagem CML, quanto a qualidade do código

gerado.

Vale salientar também que a modelagem do metamodelo da linguagem CML na própria linguagem auxiliou na sua formalização através de sua especificação em UML/OCL (anexo A), visto que modelagens em CML podem ser facilmente convertidas para UML/OCL, como demonstrado nas comparações do capítulo 3.

# 6 ANÁLISE DE CUSTO-BENEFÍCIO

Esta análise foi inspirada pelo trabalho *A General Economics Model of Software Reuse* (GAFFNEY JR.; CRUICKSHANK, 1992) e procura estabelecer algumas equações que modelam o custo do desenvolvimento de software com o auxílio de modelagem conceitual. Diferentemente do trabalho mencionado, esta análise estabelece apenas uma tese baseada no modelo matemático. Esta análise, entretanto, abre a porta para que trabalhos futuros utilizem este modelo na verificação desta tese através de experimentos que avaliem o desempenho de CML em projetos concretos.

Dado um sistema distribuído composto de um ou mais componentes ($N$) que precisam implementar o mesmo modelo conceitual, e desconsiderando as bibliotecas reutilizadas pelos componentes, tem-se o custo total de desenvolvimento ($T_o$):

$$T_o = \sum_{i=1}^{N} (A_i + M_i), \tag{6.1}$$

onde, para cada componente do sistema, $A_i$ é o custo de desenvolvimento do código de aplicação e $M_i$ é o custo de desenvolvimento do modelo conceitual, includindo o código de suporte para persistência e transporte de dados.

Supondo que já existam construtores disponíveis em módulos CML para cada um dos tipos de componentes do sistema, e que seja necessário apenas definir um modelo conceitual para tal sistema, pode-se redefinir o custo de desenvolvimento total sem o custo de projeto, implementação e teste do modelo conceitual ($T_m$):

$$T_m = M + \sum_{i=1}^{N} A_i, \tag{6.2}$$

onde $M$ é o custo de análise e definição do modelo conceitual em CML, que é incorrido uma única vez, ou seja, não há custo de desenvolvimento do modelo para cada componente do sistema, pois os construtores se encarregação de gerar o código do modelo.

Supõe-se então que:

$$M \ll M_o = \sum_{i=1}^{N} M_i, \tag{6.3}$$

ou seja, que o custo de definição do modelo conceitual em CML é provavelmente bem menor que o custo de desenvolvimento do modelo individualmente para cada componente.

Assim, pode-se concluir que:

$$M \ll M_o \rightarrow T_m \ll T_o, \tag{6.4}$$

ou seja, que CML pode reduzir consideravelmente o custo de desenvolvimento se o custo de análise e definição do modelo conceitual em CML é bem menor que o custo de projeto, implementação e teste de um modelo individual desenvolvido para cada componente.

É possível aceitar a conclusão na expressão 6.4 tendo-se como premissas: que os construtores existentes fazem a "parte pesada" da implementação; que não é necessário testar o código gerado pelos construtores; e que a linguagem CML é mais sucinta e expressiva na definição de modelos do que a linguagem de programação usada em cada componente. Na prática, porém, é preciso fazer mais algumas considerações:

- Mesmo que CML traga consigo construtores que sirvam uma série de sistemas, é bem provável que alguns sistemas precisem, ou complementar os construtores existentes, ou implementar novos construtores para uma determinada situação. Isto vai gerar custo adicional de desenvolvimento. Vale salientar, porém, que este custo pode ser amortizado se mais de um componente do sistema usar o novo construtor.

- Além disso, mesmo que os construtores existentes sejam bem testados e de boa qualidade, provavelmente haverá a necessidade de testes de integração entre o código de aplicação e o código gerado que implementa o modelo; o que pode incorrer em custo adicional.

- Ainda, mesmo que a linguagem CML seja mais adequada para a modelagem conceitual do que uma linguagem de programação convencional, o modelo em CML precisará levar em consideração os cenários de todos os componentes do sistema. Portanto, este custo ($M$) será provavelmente maior que o custo individual de

desenvolvimento do modelo de cada componente ($M_i$) – mesmo que $M \ll M_o$, provavelmente $M > M_i$.

Levando em consideração as questões acima, revisa-se a equação 6.2:

$$T_f = C + \varepsilon \frac{M_o}{N} + \sum_{i=1}^{N} (A_i + I_i), \tag{6.5}$$

onde $C$ é o custo de implementação de um ou mais novos construtores (ou o custo de extensão de construtores existentes); $\varepsilon$ representa a taxa de eficiência de modelagem em CML (se $\varepsilon > 1$, incorre-se em custo adicional para modelar em CML, acima da média do custo de modelagem individual de cada componente: $\frac{M_o}{N}$); e $I_i$ representa o custo de integração do modelo CML em cada componente.

Para que a modelagem conceitual em CML traga prejuízo ao desenvolvimento, seria preciso que uma ou mais das seguintes conjecturas fossem verdade:

- o custo de implementação de um ou mais construtores seja bem maior que o custo de modelagem e implementação individual de todos os componentes ($C >= M_o$);
- a eficiência de modelagem em CML seja muito baixa ($\varepsilon >= N$);
- ou o custo de integração do código gerado por CML seja maior que o custo de modelagem e implementação individual de cada componente ($I_i >= M_i$).

Para evitar os cenários acima, o objetivo então é fazer com que o custo real ($T_f$) seja o mais próximo possível do custo ideal ($T_m$). Para alcançá-lo ($T_f \approx T_m$):

- É necessário a disponibilidade de um grande número de construtores de alta qualidade, suportando várias tecnologias e linguagens. Espera-se que isto seja possível na medida em que CML atinja uma número crítico de usuários e que vários módulos de construtores estejam disponíveis publicamente ou dentro de organizações. Desta forma, o custo dos construtores ($C$) deve ser amortizado ao longo do tempo, assim como o custo de integração ($I_i$) deve ser reduzido.
- Com o uso contínuo de CML, espera-se que o processo de modelagem se torne mais eficiente. Espera-se que os desenvolvedores

habituem-se a desenvolver modelos conceituais, que acostumem-se a englobar no modelo os cenários de vários componentes do sistema, e que o compilador CML seja integrado na ferramentália disponível aos desenvolvedores, tais como editores de texto, IDEs, *build systems*, etc. Inicialmente, espera-se $\varepsilon \gg 1$; porém, difundindo-se o uso de CML, espera-se uma gradual melhora na eficiência até que se atinja $\varepsilon < 1$.

- Mesmo nos primeiros projetos que adotarem CML (quando as condições esperadas pelos dois itens acima não sejam ainda uma realidade), é improvável que: o desenvolvimento de construtores seja tão custoso ($C >= M_o$); a modelagem em CML seja tão ineficiente ($\varepsilon >= N$); e o custo de integração seja tão alto ($I_i >= M_i$); quanto descrito anteriormente pelos cenários de pior caso. Espera-se assim que os pequenos ganhos iniciais deverão incentivar os *early adopters* que eventualmente criarão as condições esperadas descritas nos dois itens anteriores.

Sem considerar outras questões que não foram abordadas aqui, esta análise parece então trazer a tese de que, quanto maior o uso de CML (ou qualquer compilador/linguagem que suporte de forma equivalente a modelagem conceitual com geração de código), maior o potencial desta linguagem causar um impacto positivo na diminuição do custo de desenvolvimento.

# 7 CONSIDERAÇÕES FINAIS

Neste trabalho apresentou-se uma nova linguagem textual para modelagem conceitual – CML. Seu compilador permite a geração de código em várias linguagens-alvo e tecnologias através de *templates* extensíveis e modulares.

Apesar do seu foco em modelagem, a sintaxe da linguagem CML é semelhante a de linguagens de programação, o que a torna familiar a um grande grupo de desenvolvedores de software e permite o uso de editores de texto para modelagem conceitual. Seu compilador também permite a integração da modelagem conceitual dentro do fluxo de trabalho dos desenvolvedores e nas ferramentas que estes utilizam. CML propicia, desta forma, a aplicação do princípio de que *o modelo é o código*, divulgado pelo manifesto *Conceptual Model Programming* (EMBLEY; LIDDLE; PASTOR, 2011).

Embora não possua a expressidade da linguagem UML (OMG, 2015a), as opções de modelagem disponíveis nesta primeira versão de CML, tais como generalização/especialização, associações unidirecionais/bidirecionais e atributos/associações derivados, se mostraram suficientes para a modelagem dos aspectos estruturais de alguns sistemas de software. A especificação e implementação do próprio metamodelo do compilador CML, usando versões sucessivas do mesmo compilador (*bootstrapping*), mostrou a aplicabilidade da linguagem e do compilador CML.

Esta versão inicial do compilador CML fornece, como parte da sua biblioteca base, quatro *construtores* para geração de código nas linguagens Java e Python. Dois construtores básicos permitem a geração de *Data Transfer Objects* (DTOs), um para Java e outro para Python. Outros dois construtores geram código em Java e Python que sintetiza herança múltipla nestas linguagens. (A versão Java destes dois últimos foi usada na própria implementação do compilador CML. Além disso, todos os construtores acima implementam associações bidirecionais.) Através destes construtores, o compilador CML demonstrou sua flexibilidade na geração de código em diferentes linguagems-alvo e em diferentes paradigmas de projeto e implementação. Sendo assim, é possível extrapolar seu uso para gerar código em outras linguagens, em outros tipos de *design* e em outras tecnologias.

Enquanto outras soluções, como a linguagem M (MICROSOFT, 2009) e Xtext (BETTINI, 2016), requerem a criação de uma DSL, CML tem uma linguagem básica para modelagem. As outras soluções de-

pendentem dos IDEs para criação e edição dos modelos baseados em DSLs; CML só depende do compilador; a linguagem CML pode ser usada em qualquer editor de texto. Além disso, CML sabe construir seus próprios artefatos, que podem posteriormente ser consumidos por outros sistemas de *build*, e permite ainda a combinação de *tarefas* para gerar artefatos mais complexos.

Com a liberação do compilador CML como um projeto *open-source*, mesmo ainda com suas limitações atuais, CML dá o primeiro passo para eventualmente se tornar uma alternativa a MPS (VOELTER, 2014) e a MDA (OMG, 2014a) na geração de código a partir de modelos conceituais. De fato, no capítulo 6 foi demonstrado que, dado a disponibilidade do compilador CML e o surgimento de um número maior de *construtores*, é possível vislumbrar o impacto positivo que CML pode trazer na redução do custo de desenvolvimento de software.

## 7.1 TRABALHOS FUTUROS

Abaixo, sugerem-se alguns trabalhos que poderiam dar continuidade ao projeto CML:

- Seria muito útil para a validação e o aprimoramento de CML, assim como para diminuir o custo de desenvolvimento de software em projetos que eventualmente adotassem CML, a implementação de construtores para vários tipos de tecnologias, tais como:
  - servidores e clientes REST;
  - a camada de acesso e o esquema de banco de dados;
  - sistemas do tipo MapReduce e *serverless*;
  - camada de acesso remoto e de banco de dados local de aplicativos móveis;
  - além de outras tecnologias atuais.
- A modelagem e a construção de alguns tipos diferentes de sistemas também possibilitaria a validação e o aprimoramento de CML, como por exemplo:
  - a modelagem de sistemas de informação, como a loja de livros fictícia Livir (WAZLAWICK, 2014).
  - sistemas de computação gráfica, modelando os elementos gráficos em CML e criando construtores que geram código em C/C++ (Cairo), Java e JavaScript.

  - sistemas embutidos, tal qual é feito com ThingML (HARRAND et al., 2016).

- Usar construtores para portar o compilador CML para outras linguagens (Python, JavaScript, C#, C/C++, etc.) seria uma forma de validar a capacidade de CML suportar várias tecnologias e linguagens.

- Permitir a especificação de DSLs através de uma extensão de CML – uma linguagem de especificação de linguagens – que geraria ASTs usando o metamodelo da própria CML e usaria o compilador da CML para gerar código.

- Expandir a linguagem CML para permitir a especificação de componentes e seus contratos, como já é feito com OCL (WAZLAWICK, 2014), mas com a habilidade de geração de código.

- Prover mais mecanismos de modelagem conceitual na linguagem CML, como suportar outras cardinalidades, conjuntos e outros tipos de coleções, invariantes, associações qualificadas, além de outros formalismos provenientes de linguagens como UML e OWL.

- Permitir a definição de outros tipos de dados, além dos tipos primitivos, que seriam úteis em domínios específicos, tais como números complexos na área científica ou moedas na área financeira.

- Pesquisa de campo: adotar CML em determinados projetos, acompanhar seu uso e coletar dados quantitativos e qualitativos do seu desempenho, usando o modelo de custo-benefício apresentado no capítulo 6.

Qualquer uma das sugestões de trabalho acima seria uma contribuição significativa no amadurecimento de CML e na redução do custo de desenvolvimento de software com a prática de modelagem conceitual.

# REFERÊNCIAS

ALBANO, A. et al. An object data model with roles. In: **VLDB**. [S.l.: s.n.], 1993. v. 93, p. 39–51.

BALZER, S.; GROSS, T. R.; EUGSTER, P. A Relational Model of Object Collaborations and Its Use in Reasoning About Relationships. In: ERNST, E. (Ed.). **ECOOP 2007 – Object-Oriented Programming: 21st European Conference, Berlin, Germany, July 30 - August 3, 2007. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 323–346. ISBN 978-3-540-73589-2. Disponível em: <http://dx.doi.org/10.1007/978-3-540-73589-2_16>.

BETTINI, L. **Implementing domain-specific languages with Xtext and Xtend**. [S.l.]: Packt Publishing Ltd, 2016. ISBN 978-1786464965.

BIERMAN, G.; WREN, A. First-Class Relationships in an Object-Oriented Language. In: BLACK, A. P. (Ed.). **ECOOP 2005 - Object-Oriented Programming: 19th European Conference, Glasgow, UK, July 25-29, 2005. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 262–286. ISBN 978-3-540-31725-8. Disponível em: <http://dx.doi.org/10.1007/11531142_12>.

BOAS, G. van E. Template programming for model-driven code generation. In: **19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Vancouver, CA**. [S.l.: s.n.], 2004.

BRAMBILLA, M.; MAURI, A.; UMUHOZA, E. Extending the Interaction Flow Modeling Language (IFML) for Model Driven Development of Mobile Applications Front End. In: AWAN, I. et al. (Ed.). **Mobile Web Information Systems: 11th International Conference, MobiWIS 2014, Barcelona, Spain, August 27-29, 2014. Proceedings**. Cham: Springer International Publishing, 2014. p. 176–191. ISBN 978-3-319-10359-4. Disponível em: <http://dx.doi.org/10.1007/978-3-319-10359-4_15>.

CARDOSO, I. S. **Inserindo suporte a declaração de associações da UML 2 em uma linguagem de programação orientada a objetos**. Dissertação (Mestrado) — Universidade Federal de Santa

Catarina, 2011. Disponível em:
<http://repositorio.ufsc.br/xmlui/handle/123456789/96031>.

CHEN, P. P.-S. The Entity-Relationship Model (Reprinted Historic Data). In: EMBLEY, D. W.; THALHEIM, B. (Ed.). **Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 57–84. ISBN 978-3-642-15865-0. Disponível em:
<http://dx.doi.org/10.1007/978-3-642-15865-0_3>.

DEUTSCH, D. Constructor theory. **Synthese**, v. 190, n. 18, p. 4331–4359, Dec 2013. ISSN 1573-0964. Disponível em:
<https://doi.org/10.1007/s11229-013-0279-z>.

EMBLEY, D. W.; LIDDLE, S. W.; PASTOR, O. Conceptual-Model Programming: A Manifesto. In: EMBLEY, D. W.; THALHEIM, B. (Ed.). **Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 3–16. ISBN 978-3-642-15865-0. Disponível em:
<http://dx.doi.org/10.1007/978-3-642-15865-0_1>.

FLEUREY, F.; MORIN, B. **ThingML Web Site**. 2017. Disponível em: <http://thingml.org>.

GAFFNEY JR., J. E.; CRUICKSHANK, R. D. A General Economics Model of Software Reuse. In: **Proceedings of the 14th International Conference on Software Engineering**. New York, NY, USA: ACM, 1992, (ICSE '92). p. 327–337. ISBN 0-89791-504-6. Disponível em: <http://doi.acm.org/10.1145/143062.143150>.

GHERAIBIA, Y.; BOUROUIS, A. Ontology and automatic code generation on modeling and simulation. In: **2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)**. [s.n.], 2012. p. 69–73. Disponível em:
<http://doi.org/10.1109/SETIT.2012.6481892>.

GOGOLLA, M.; THALHEIM, B. UML and OCL in Conceptual Modeling. In: EMBLEY, D. W.; THALHEIM, B. (Ed.). **Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 85–122. ISBN 978-3-642-15865-0. Disponível em:
<http://dx.doi.org/10.1007/978-3-642-15865-0_4>.

GREIFENBERG, T. et al. Modeling Variability in Template-based Code Generators for Product Line Engineering. **CoRR**, abs/1606.02903, 2016. Disponível em: <http://arxiv.org/abs/1606.02903>.

HARRAND, N. et al. Thingml: A language and code generation framework for heterogeneous targets. In: **Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems**. New York, NY, USA: ACM, 2016. (MODELS '16), p. 125–135. ISBN 978-1-4503-4321-3. Disponível em: <http://doi.acm.org/10.1145/2976767.2976812>.

ISO. **ISO/IEC 9899:2011 Information technology — Programming languages — C**. International Organization for Standardization, 2011. Disponível em: <https://www.iso.org/standard/57853.html>.

JENSEN, K.; WIRTH, N. **PASCAL User Manual and Report**. [S.l.]: Springer-Verlag New York, Inc., 1974. ISBN 3-540-06950-X.

KARAGIANNIS, D. et al. Fundamental Conceptual Modeling Languages in OMiLAB. In: KARAGIANNIS, D.; MAYR, H. C.; MYLOPOULOS, J. (Ed.). **Domain-Specific Conceptual Modeling: Concepts, Methods and Tools**. Cham: Springer International Publishing, 2016. p. 3–30. ISBN 978-3-319-39417-6. Disponível em: <http://dx.doi.org/10.1007/978-3-319-39417-6_1>.

LEE, K. **CSE341 Lecture Notes 21**. 2004. Disponível em: <http://courses.cs.washington.edu/courses/cse341/04wi/lectures/21-oop-multi.html>.

MICROSOFT. **The "M"Modeling Language Specification**. 2009. Disponível em: <http://msdn.microsoft.com/en-us/library/dd285282.aspx>.

MYLOPOULOS, J. et al. Telos: Representing Knowledge About Information Systems. **ACM Trans. Inf. Syst.**, ACM, New York, NY, USA, v. 8, n. 4, p. 325–362, out. 1990. ISSN 1046-8188. Disponível em: <http://doi.acm.org/10.1145/102675.102676>.

OLDEVIK, J. et al. Toward Standardised Model to Text Transformations. In: HARTMAN, A.; KREISCHE, D. (Ed.). **Model Driven Architecture – Foundations and Applications: First European Conference, ECMDA-FA 2005, Nuremberg,**

**Germany, November 7-10, 2005. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 239–253. ISBN 978-3-540-32093-7. Disponível em: <http://dx.doi.org/10.1007/11581741_18>.

OMG. **Model Driven Architecture (MDA) MDA Guide rev. 2.0**. 2014. Disponível em: <http://www.omg.org/cgi-bin/doc?ormsc/14-06-01>.

OMG. **Object Constraint Language (OCL), Version 2.4**. 2014. Disponível em: <http://www.omg.org/spec/OCL/2.4>.

OMG. **Unified Modeling Language (UML), Superstructure, Version 2.5**. 2015. Disponível em: <http://www.omg.org/spec/UML/2.5>.

OMG. **XML Metadata Interchange (XMI), Version 2.5.1**. [S.l.], 2015. Disponível em: <http://www.omg.org/spec/XMI/2.5.1/>.

OMG. **Meta Object Facility (MOF) Core Specification, Version 2.5.1**. 2016. Disponível em: <http://www.omg.org/spec/MOF/2.5.1>.

PARR, T. **The Definitive ANTLR Reference: Building Domain-specific Languages**. [S.l.]: Pragmatic Bookshelf, 2007. (Pragmatic Bookshelf Series). ISBN 9780978739256.

PARR, T. J. Enforcing Strict Model-view Separation in Template Engines. In: **Proceedings of the 13th International Conference on World Wide Web**. New York, NY, USA: ACM, 2004, (WWW '04). p. 224–233. ISBN 1-58113-844-X. Disponível em: <http://doi.acm.org/10.1145/988672.988703>.

ROSE, L. M. et al. The Epsilon Generation Language. In: SCHIEFERDECKER, I.; HARTMAN, A. (Ed.). **Model Driven Architecture – Foundations and Applications: 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 1–16. ISBN 978-3-540-69100-6. Disponível em: <http://dx.doi.org/10.1007/978-3-540-69100-6_1>.

STEINBERG, D. et al. **EMF: Eclipse Modeling Framework**. [S.l.]: Pearson Education, 2008.

THALHEIM, B. The Theory of Conceptual Models, the Theory of Conceptual Modelling and Foundations of Conceptual Modelling. In: EMBLEY, D. W.; THALHEIM, B. (Ed.). **Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 543–577. ISBN 978-3-642-15865-0. Disponível em: <http://dx.doi.org/10.1007/978-3-642-15865-0_17>.

TOLVANEN, J.-P. MetaEdit+: Domain-specific Modeling for Full Code Generation Demonstrated [GPCE]. In: **Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications**. New York, NY, USA: ACM, 2004. (OOPSLA '04), p. 39–40. ISBN 1-58113-833-4. Disponível em: <http://doi.acm.org/10.1145/1028664.1028686>.

VISIC, N.; KARAGIANNIS, D. Developing Conceptual Modeling Tools Using a DSL. In: BUCHMANN, R.; KIFOR, C. V.; YU, J. (Ed.). **Knowledge Science, Engineering and Management: 7th International Conference, KSEM 2014, Sibiu, Romania, October 16-18, 2014. Proceedings**. Cham: Springer International Publishing, 2014. p. 162–173. ISBN 978-3-319-12096-6. Disponível em: <http://dx.doi.org/10.1007/978-3-319-12096-6_15>.

VOELTER, M. **Generic Tools, Specific Languages**. Tese (Doutorado) — Delft University of Technology, June 2014. Disponível em: <http://repository.tudelft.nl/view/ir/uuid%3A53c8e1e0-7a4c-43ed-9426-934c0a5a6522>.

W3C. **OWL 2 Web Ontology Language. Structural Specification and Functional-Style Syntax (Second Edition)**. 2012. Disponível em: <http://www.w3.org/TR/owl2-syntax>.

WAZLAWICK, R. S. **Object-Oriented Analysis and Design for Information Systems: Modeling with UML, OCL, and IFML**. Morgan Kaufmann, 2014. ISBN 978-0-12-418673-6. Disponível em: <http://www.sciencedirect.com/science/book/9780124186736>.

# ANEXO A – Especificação da Linguagem

# *Conceptual Modeling Language* Specification

Version 1.0 (Draft)

Quenio Cesar Machado dos Santos

Universidade Federal de Santa Catarina*

November 2017

* Initially developed as part of the author's Bachelor Technical Report in Computer Sciences

# Contents

# Listings

# Figures

# Tables

# Part I

# Language and Compiler

# One

# The Language

This document specifies the *Conceptual Modeling Language*, or CML for short. CML enables the modeling of the information of software systems. It focuses on modeling the structural aspects of such systems, having less emphasis on the behavioral aspects. Using CML, it is possible to represent the information as understood by the system users, while disregarding its physical organization as implemented by target languages or technologies.

In this first part of the CML specification, the first chapter will provide an overview of the CML compiler's architecture, and the second chapter describes the organization and notation used in the remainder of this document. The second part describes that structural constructs of the language that enable conceptual modeling. The third part covers values and expressions. The fourth part focuses on the semantics of type checking. The fifth part describes code generation. The last part will cover organization and sharing of conceptual models.

# Two

# The Compiler

The CML compiler has as *input*, source files defined using its own conceptual language (as specified in this document), which provides an abstract syntax similar to (but less comprehensive than) a combination of UML [4] and OCL [3]; and, as *output*, any target languages based on extensible templates, which may be provided by the compiler's base libraries, by third-party libraries, or even by developers.

The CML compiler's overall architecture follows the standard compiler design literature [2]. An overview diagram of the architecture is shown in figure 2.1. The two main components of the compiler, and the

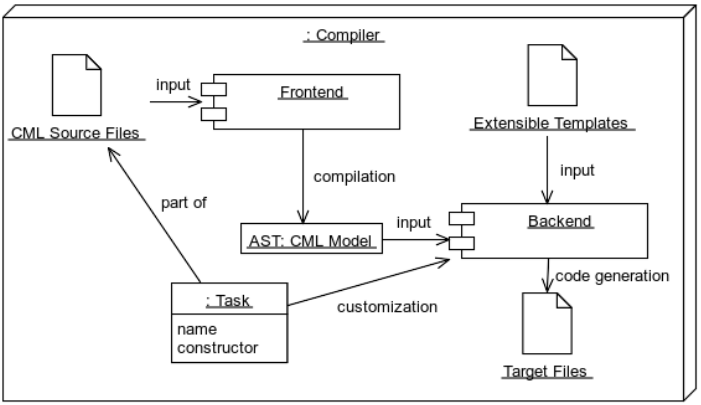


Figure 2.1: An architectural overview of the CML compiler.

artifacts they work with, are presented in the next sections.

## 2.1 The Compiler Frontend

The frontend receives as input the *CML source files*. It will parse the files and generate an internal representation of the *CML model*.

Syntactical and semantic validations will be performed at this point. Any syntax and constraint errors are presented to the developer, interrupting the progress to the next phase. If the *source files* are parsed and validated successfully, then the internal representation (the AST) of the *CML model* is provided as the input to the *backend* component.

## 2.2 The Compiler Backend

The backend receives the *CML model AST* as input. Based on the *target specification* provided by the AST, chooses which *extensible templates* to use for code generation. The *target files* are then generated, and become available to be consumed by other tools. The *task declaration* plays the key role of determining the kind of *target* to be generated.

CML extensible templates are implemented in StringTemplate [7]. The CML compiler uses StringTemplate for two purposes:

- *File names and directory structure:* each type of target generated by the CML compiler requires a different directory structure. The CML compiler expects each target type to define a template file named "files.stg" (also known as *files template*), which will contain the path of all files to be generated. The *files template* may use information provided by the *task declaration* in order to determine the file/directory names.
- *File content generation:* each file listed under the *files template* will have a corresponding *content template* that specifies how the file's content must be generated. The *content template* will receive as input one root-level element of the CML model, which will provide information to generate the file's content. Each type of top-level model element should have a corresponding *content template*. Templates are described in *Code Generation* part of this specification.

# Three

# Specification and Notations

The first two parts of this specification specify all elements of CML metamodel needed for conceptual modeling. Every chapter in these two parts starts with a definition, which is followed by: an example; the specification of the concrete syntax; the abstract syntax; and then how to transform the concrete syntax into the abstract one.

The first paragrah of each chapter has an informal description of a CML metamodel element. If a correspondence exists to an element of the Entity-Relationship (ER) [1] metamodel, or to an element of the Unified Modeling Language (UML) [4] metamodel, it is provided with some comments of their differences.

Examples are provided in a separate section for each declaration of a metamodel element. These sections refer to a `verbatim` figure containing the examples, and describes them as needed. The examples are provided for illustrative purposes only, and they are *not* intended to be normative. They may be excerpts of larger CML source files, and thus may not be successfully compiled on their own.

The concrete syntax of each CML metamodel element is described on its own section. This type of section refers to a `verbatim` figure, which contains the actual ANTLR [6] grammar specifying the syntax for the CML metamodel element in question, and it must be considered normative. The appendix § A presents all the grammar rules in a single listing.

The abstract syntax of each CML metamodel element is described on the next section. This type of section refers to two types of figure: the first figure presents a class diagram with the EMOF [5]-based metamodel of the element being described; the second figure specifies the transformation from the concrete syntax into instances of the metamodel classes, which are the nodes of the abstract syntax tree (the intermediate representation described in the *Compiler Overview*).

The constraints of each CML metamodel element are described on its own section. These sections refer to a `verbatim` figure, which contains the OCL [3] invariants (and its definitions) of the CML metamodel element in question, and it must be considered normative. Each invariant has a name in the format `inv_name` so that it can be referred by the compiler's error messages and users. Derived properties may also be defined before the constraints in order to simplify the constraint expressions.

All metamodel elements referred by one of the descriptions defined above (definitions, examples, etc.) are emphasized in *italic*. If the descriptions of a CML metamodel element refer to another metamodel element, the corresponding chapter or section defining the other element is provided in parenthesis, like so (§ 3).

Some sections may not follow the structure defined above. These normally provide additional semantic information in plain English, which cannot be described using the notations presented above.

# Part II

# Conceptual Modeling

# Four

# Concepts

A *concept* in CML represents anything that has a coherent, cohesive and relevant meaning in a domain. In the ER [1] metamodel, it corresponds to an *entity set* (or an *entity type*); in UML [4], to a *class*. The CML *concept* differs, however, from the UML *class*, because it has only *properties* (§ 5), while the UML *class* may also have *operations*.

## 4.1 Example

Listing 4.1 presents some examples of *concepts* declared in CML. As shown, a *concept* may have zero or more *properties* (§ 5), and a *property* may optionally declare a *type* (§ 26). Also, as shown in the concept **EBook** of the example, a *concept* may specialize (§ 8) another *concept*.

## 4.2 Syntax

Listing 4.2 specifies the syntax used to declare a *concept*. The **concept** keyword is followed by a NAME. Optionally, a list of other NAMEs may be enumerated, referring to other *concepts* that are generalizations (§ 8) of the declared *concept*. A list of *properties* (§ 5) may be declared under the **concept** block. And the **abstract** keyword may precede the **concept** keyword, making a *concept* abstract (§ 9).

Figure 4.1 presents the *Concept* metaclass in an EMOF [5] class diagram, and listing 4.3 specifies the *concept* transformation from its concrete syntax to its abstract syntax. For each *concept* parsed by the compiler, an instance of the *Concept* class will be created, and its properties will be assigned according to parsed information:

```
// Empty concept:
@concept Book;

// Property without a type:
@concept TitledBook
{
    title;
}

// Property with the String type:
@concept StringTitledBook
{
    title: String;
}

// Specializing another concept:
@concept Ebook: Book;
```

Listing 4.1: Concept Examples

```
conceptDeclaration returns [TempConcept concept]:
  (ABSTRACTION | CONCEPT) NAME
  (':' generalizations)?
  (';' | propertyList);

generalizations:
  NAME (',' NAME)*;
```

Listing 4.2: Concept Concrete Syntax

- *name*: assigned with the value of the terminal node NAME.
- *abstract*: set to *true* if the **abstract** keyword is found before the **concept** keyword; otherwise, set to *false*.
- *elements*: an *ordered set* referencing all *properties* parsed in the **concept** block.
- *generalizations*: an *ordered set* referencing all *concepts* whose NAMEs were enumerated in the *GeneralizationList*.

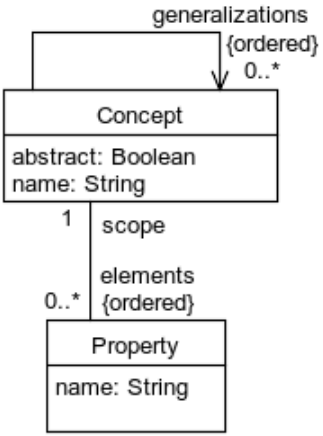


Figure 4.1: Concept Abstract Syntax

```
node Concept:
    'abstract'?
    'concept' NAME
    (':' GeneralizationList)?
    (';' | PropertyList)
{
    name = NAME;
    abstract = 'abstract'?;
    elements = PropertyList.Property*;
    generalizations = for name in GeneralizationList.NAME*
        | yield Model.concept[name];
}

node GeneralizationList: NAME (',' NAME)*;
```

Listing 4.3: Concept AST Instantiation

```
context Concept

inv unique_concept_name:
    parent.concepts
        ->select(c| c != self and c.name = self.name)
        ->isEmpty()
```

Listing 4.4: Concept Constraints

## 4.3 Model Validation

Listing 4.4 presents the invariants of the *Concept* metaclass:

- *unique_concept_name*: Each *concept* must have a unique NAME within its *module*.

# Five

# Properties

A *property* in CML may hold *values* of *primitive types* (§ 26), in which case they correspond to *attributes* (§ 6) on the ER [1] and UML [4] metamodels. Or they may hold references, or sequences of references, linking to instances of other *concepts* (§ 4), in which case they correspond to a *relationship* on the ER metamodel, or to *associations* (§ 7) on the UML metamodel.

## 5.1 Example

Listing 5.1 shows some *properties* declared in CML. As shown, a *property* may be an *attribute* (§ 6) of a *primitive type* (§ 26), or represent the *role* (or *end*) of an *association* (§ 7).

```
— Attributes of primitive types:
@concept Book
{
    title: String;
    quantity: Integer;
}

— Role in unidirectional association:
@concept Order
{
    customer: Customer;
}
```

Listing 5.1: Property Examples

```
propertyList:
  '{' (propertyDeclaration ';')* '}';

propertyDeclaration returns [Property property]:
 DERIVED? NAME (':' typeDeclaration)? ('=' expression)?;

DERIVED: '/';
```

Listing 5.2: Property Concrete Syntax

```
node PropertyList: '{' (Property ';')* '}';

node Property: '/'? NAME (':' Type)? ('=' STRING)?
{
    name = NAME;
    derived = '/'?;
    value = unwrap(STRING?);
    type = Type?;
}
```

Listing 5.3: Property AST Instantiation

## 5.2 Syntax

Listing 5.2 specifies the syntax used to declare a *property*. The NAME is followed by a *typeDeclaration* (§ 26). Optionally, an *expression* (§ 11) may be specified in order to set the initial value.

Figure 4.1 presents the *Property* metaclass in an EMOF [5] class diagram of the CML metamodel, and listing 5.3 specifies the transformation from the *property* concrete syntax to its abstract syntax. For each *property* parsed by the compiler, an instance of the *Property* class will be created, and its properties will be assigned according to parsed information:

- *name*: assigned with the value of the terminal node NAME.
- *type*: if *typeDeclaration* is provided, *type* is set with the instance of the *Type* class matching the *typeDeclaration*.
- *expression*: if provided, it contains the instance of the *Expression* class matching the parsed *expression*.

```
context Property
inv unique_property_name:
    self.scope.properties
        ->select(p| p != self and p.name = self.name)
        ->isEmpty()

context Property
inv property_type_specified_or_inferred:
    type->notEmpty() or expression->notEmpty()

context Property
inv property_type_assignable_from_expression_type:
    type->notEmpty() and expression->notEmpty() implies
        type.isAssignableFrom(expression.type)
```

Listing 5.4: Property Constraints

## 5.3 Model Validation

Listing 5.4 presents the invariants of the *Property* metaclass:

- *unique_property_name*: Each *property* must have a unique NAME within its *concept* (§ 4).
- *property_type_specified_or_inferred*: Either the *property* explicitly defines a *type* or it defines an *expression*, from which the type is inferred. That is required for both regular, slot-based *properties* (which may provide an *initialization expression*) and *derived properties* (which may have an *expression* defining the derivation).
- *property_type_assignable_from_expression_type*: When both a *type* and *expression* are defined for a *property*, the *type* inferred from the *expression* should be assignable to the declared *type*. That is required for both regular, slot-based *properties* (which may provide an *initialization expression*) and *derived properties* (which may have an *expression* defining the derivation).

# Six

# Attributes

In CML, *attributes* are *properties* (§ 5) of *primitive types* (§ 26). They correspond to the *Attribute* metaclass in the ER [1] metamodel; in the UML [4] metamodel, to the association *attribute* between the metaclass *Class* and the metaclass *Property*.

*Attributes* serve as a *slot* that holds a value of the specified *primitive type*. An initial value may be specified as an *expression* (§ 11). Some *attributes*, however, may be continuosly derive their *value* from an *expression* (not only initially), in which case they are called *derived attributes* (§ 6.3).

While initial values are only set when a *concept* (§ 4) is instantiated, the value of *derived attributes* is always evaluated from the given *expression*, and they cannot be set any other way.

## 6.1 Example

Listing 6.1 presents some examples of *attributes* declared in CML. As shown, the attribute **a** is a regular attribute definition that specifies the *primitive type* (§ 26) of the values that can be held by the *attribute*'s slot. The attribute **b** is an example showing how an *attribute* can be defined with an initial value. As shown by the attribute **c**, an attribute may be derived from an *expression* that refers to other *attributes*. In order to differentiate *attributes* with initial values from *derived attributes*, a forward slash ("/") prefixes the name of the latter. Attributes **d** and **e** are examples where the type of the attribute, instead of being specified, is inferred from the given *expression*. Type inference is possible for both regular, slot-based *attributes* and *derived attributes* that provide an *expression*.

```
@concept Attributes
{
    -- Attribute with a slot for values of a primitive type:
    a: Integer;

    -- An attribute with an initial value:
    b: String = "initial value";

    -- A derived attribute:
    /c: Decimal = 2.0 * a;

    -- An attribute with type inferred from its initial value:
    d = 3; -- Inferred as Integer based on constant "3"

    -- A derived attribute with type inferred from expression:
    /e = 2.0d * a; -- Inferred as Double based on "2.0d * a"
}
```

Listing 6.1: Attribute Example

## 6.2 Syntax

Listing 5.2 specifies the syntax used to declare any kind of *property* (§5), including *attributes*. The NAME of an *attribute* is followed by a *typeDeclaration* of a *primitive type* (§26). Optionally, an *expression* (§11) may be specified in order to set the initial value. A *derived attribute* must be prefixed with the forward-slash character, as specified by DERIVED, in which case the given *expression* defines the value of the *attribute* at all times.

Since an *attribute* in CML is just a *property* (§5) with *primitive types* (§26), the *property* metaclass in the CML metamodel is used to represent *attributes*. Figure 4.1 presents the *property* metaclass in an EMOF [5] class diagram, and listing 5.3 specifies the *property* transformation from its concrete syntax to its abstract syntax.

## 6.3 Derived Attributes

A *concept* in CML may have *attributes* (§6) that do not hold specific *values*, but instead provide a *value* derived from an *expression* (§11). These are called *derived attributes*. Unlike an *expression* used to initialize a *non-derived attribute*, the expression of a derived attribute is evaluated every time the attribute is queried.

In the UML [4] metamodel, the *Property* metaclass has a meta-attribute named *isDerived*, which determines whether an *attribute* is derived or not. A *derived attribute* in UML may be defined using a OCL [3] constraint; while CML has *expressions* as part of the language.

The ER [1] metamodel, in its original form, does not allow for the differentiation of *derived attributes* as part of an *entity set*, but it is possible to define *retrieval operations* whose results would equal to *values* of *derived properties* in CML. It can be said, however, that ER, by defining an *attribute* as a function from the *entity set* to the *value set*, does not prescribe that all *attributes* are memory-based, nor does it prevent the definition of an *attribute* function as an *expression*.

The CML metamodel and its syntax, on the other hand, define whether an *attribute* is memory-based (a *non-derived attribute*) or it is derived from an *expression* (a *derived attribute*).

# Seven

# Associations

In CML, an *association* represents a relation between two *concepts* (§ 4), where a reference to an *instance* of each *concept* is found in every tuple that is part of the relation. When *concepts* have an *association* between themselves, its *instances* are linked in such way that it is possible to access an *instance* of one *concept* from an *instance* of the other *concept*.

The UML [4] metamodel has a metaclass named *Association* that has *Property* instances, whose *types* are the *Class* instances that are part of the *association*. In UML, the name of each *Property* instance in the *Association* metaclass is known as the *role* of the corresponding *Class* in the *association*.

On the CML metamodel, on other hand, the *Association* metaclass is only needed when it is necessary to define *bidirectional associations*, whose *links* are accessible from either *association end*. For *unidirectional associations*, where only one *association end* is accessible, only a *property* is defined in the source *concept*, making its *type* the target *concept*.

On the ER [1] metamodel, each *association* is known as a *relationship set*, and each tuple in this set is called a *relationship*. Unlike CML and UML, the tuples in a *relationship set* of an ER model can be queried directly, and no notion of *property* is required as part of the *entity type* in order to access those *relationships*.

As it is case for *attributes* (§ 6), *associations* in CML can also be derived from other *associations* (just as well as in UML); they are called *derived associations* (§ 7.4).

## 7.1 Example

Listing 7.1 presents some examples of *associations* declared in CML. The concept **Vehicle** contains the property **driver**, which may optionally refer

to an instance of **Employee**, meaning that a **driver** may or may not be assigned to a single **Vehicle**. The concept **Vehicle** also has the property **owner**, which always refers to an instance of **Organization**, meaning that an **owner** must always be assigned to each instance of **Vehicle**. Similarly, the concept **Employee** has the property **employer**, which must always be assigned to an instance of **Organization**.

Just below the declaration of **Organization**, we observe an association named **Employment**, which enumerates two *properties*: the first is **employer** from the concept **Employee**; the second is **employees** from the concept **Organization**. What this *association* implies is a correspondence between these two properties. Every time a reference to an instance of **Organization** is assigned to the slot **employer** of an instance of **Employee**, a reference to this same instance of **Employee** must be assigned to the slot **employees** of the **Organization** instance. However, since the *type* of **employees** in the concept **Organization** is a sequence of **Employee** instances, the reference to the instance of **Employee** will actually be appended to the sequence being held by the slot **employees** of the concept **Organization**, and maintained along with the other **Employee** instances already found in the sequence. Thus, the association **Employment** actually characterizes a *bidirectional association*.

The association **VehicleOwnership** is another example of a *bidirectional association*; in this case, between **Vehicle**'s **owner** property and **Organization**'s **fleet** property. It can be noticed, though, in this second *bidirectional association*, that the *types* of the *properties* are declared along with their names; such a *type* declaration, in the *association* declaration, is optional in CML, but must match the original *property* declaration under the *concept* declaration, if present.

The **driver** property in the concept **Vehicle** is a different case, since this *property* does not participate in any *association* declaration on listing 7.1. That's because there is no corresponding *property* in the concept **Employee** representing the other end of the *association*. As such, the property **driver** is representing the source end of a *unidirectional association*.

The property **drivers** in the concept **Organization** is *derived association* (§7.4).

## 7.2 Syntax

The concrete syntax used to declare an *association* in CML is specified by listing 7.2. First, the **association** keyword is followed by a NAME. Then, a list of *association ends* are declared under the **association** block. For each

```
@concept Vehicle
{
    plate: String;
    driver: Employee?;
    owner: Organization;
}

@concept Employee
{
    name: String;
    employer: Organization;
}

@concept Organization
{
    name: String;
    employees: Employee*;
    fleet: Vehicle*;
    /drivers = fleet.driver;
}

@association Employment
{
    Employee.employer;
    Organization.employees;
}

@association VehicleOwnership
{
    Vehicle.owner: Organization;
    Organization.fleet: Vehicle*;
}
```

Listing 7.1: Association Example

```
associationDeclaration
  returns [Association association]:
  ASSOCIATION NAME
  '{' (associationEndDeclaration ';')* '}';

associationEndDeclaration
  returns [Association association]:
  conceptName=NAME '.' propertyName=NAME
  (':' typeDeclaration)?;
```

Listing 7.2: Association Concrete Syntax

declaration of an *association end*, The **conceptName** and **propertyName** are optionally followed by a **typeDeclaration**.

The *Association* metaclass is presented in the EMOF [5] class diagram of figure 7.1, and its instantiation from the concrete syntax is specified by listing 7.3. For each parsed *association*, an instance of the *Association* metaclass will be created, and its meta-properties will be assigned according to parsed information:

- *name*: assigned with the value of the token NAME.
- *members*: an *ordered set* referencing all *associationEnd* instances parsed in the **association** block.

## 7.3 Model Validation

The invariants of the metaclasses *Association* and *AssociationEnd* are specified by listing 7.4:

- *association_end_property_found_in_model*: Each *association end* enumerated under an *association* must correspond to a *property* of the same name under the specified *concept*.
- *association_end_type_matches_property_type*: If a *type* is specified for a given *association end*, its *name* and *cardinality* must match the *type* of the corresponding *property*.
- *association_must_have_two_association_ends*: An *association* must have exaclty two *association ends*, since only *binary associations* are supported in CML.

```
node Association:
    'association' NAME '{' (AssociationEnd ';')* '}'
{
    name = NAME;
    members = AssociationEnd*;
}

node AssociationEnd:
    conceptName=NAME '.' propertyName=NAME (':' type=Type)?
{
    concept = Model.concepts
                ->select(concept.name = conceptName)
                ->first()

    property = concept.allProperties
                    ->select(property.name = propertyName)
                    ->first()
}
```

Listing 7.3: Association AST Instantiation

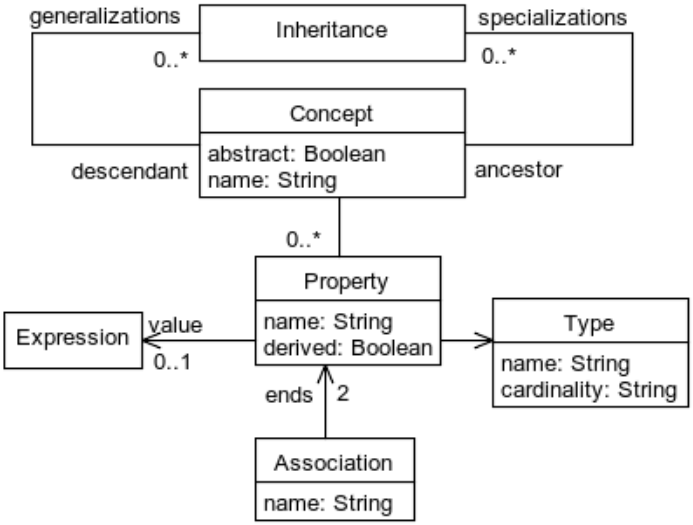


Figure 7.1: Association Abstract Syntax

```
context AssociationEnd
inv association_end_property_found_in_model:
    concept->notEmpty() and property->notEmpty()

context AssociationEnd
inv association_end_type_matches_property_type:
    self.propertyType->notEmpty() and
    self.property->notEmpty() implies
        self.propertyType.name = property.type.name and
        self.propertyType.cardinality = property.type.cardinality

context Association
inv association_must_have_two_association_ends:
    associationEnds->count() = 2

context Association
def: first = associationEnds->first()
def: last = associationEnds->last()
inv association_end_types_must_match:
    associationEnds->count() = 2 and
    first->notEmpty() and last->notEmpty() and
    first.concept->notEmpty() and
    first.property->notEmpty() and
    last.concept->notEmpty() and
    last.property->notEmpty() implies
        first.concept.name = last.property.type.name and
        last.concept.name = first.property.type.name

inv property_must_be_part_of_single_association:

inv no_associations_of_primitive_types:
```

Listing 7.4: Association Constraints

- *association_end_types_must_match*: The *concept* of one *association end* must correspond to the *type* of the *property* of the other *association end*, and vice-versa.

## 7.4 Derived Associations

*Derived associations* are declared by *derived properties* (§ 5), which associate a *source-concept* to a *target-concept* via an *expression* (§ 11). In order to obtain a reference to a *target-concept* (§ 4), it is necessary to evaluate the *expression* of the *derived property*. Among the possible *sub-expressions* that

are part of the *expression* of the *derived association*, there are *path expressions* (§ 21) of other *associations*, which server as the base for the *derived association*.

# Eight

# Generalization / Specialization

A *concept* (§ 4) in CML may be generalized by another *concept*. In other words, a *concept* may be considered a specialization of another *concept*. Generalized *concepts* have *properties* (§ 5) that apply to a larger set of instances, while specialized *concepts* have *properties* that only apply to a subset of those instances.

In the UML [4] metamodel, such generalization/specialization relationship between *classes* is known as *generalization*, which is the name of the metaclass in the UML metamodel. The original version of the ER [1] metamodel lacked this kind of relationship between *entity types*.

## 8.1 Example

Figure 8.1 presents some examples of generalization/specialization relationships declared in CML. As shown, a *concept* (§ 4) may specialize zero or more other *concepts*. The latter are called the generalizations, while the former is called the specialization. A generalization, such as **Shape**, may define *attributes* (§ 6), such as **color** and **area**, or also the *roles* in *unidirecional associations* (§ 7). Both *attributes* and *roles* are *properties* (§ 5) shared among all its specializations. Some of these *properties* may be redefined by the some of the specializations, as it is the case with the *area* property, which is redefined by **Rectangle**, **Rhombus** and **Square**. Some specializations may also define new *properties*, such as **width** and **height** in **Rectangle**, which characterize only instances of this specialization. A *concept* may be a specialization of two or more other *concepts*, as seen with **Square**, which specializes both **Rectangle** and **Rhombus**, and thus can redefine *properties* of both generalizations. If a *property* has been defined by more than one generalization, then it must be redefined by the

specialization in order to resolve the definition conflict, which is the case with **area** in **Square**. If a redefinition suitable for both generalizations is unattainable, it may be an indication that either the specialization or the generalizations are unsound from the domain's prospective.

## 8.2 Syntax

Figure 4.2 specifies the syntax used to declare a *concept* (§ 4), and in turn its generalizations. A list of NAMEs may be enumerated after the declared *concept*'s NAME, referring to other *concepts* that this concept is a specialization of.

Figure 4.1 presents the *Concept* metaclass in an EMOF [5] class diagram of the CML metamodel, and figure 4.3 specifies the *concept* transformation from its concrete syntax to its abstract syntax. There is a unidirecional association in the *Concept* class that keeps track of the generalization/specialization relationships, which is named *generalizations*. It is an *ordered set* referencing all *concepts* whose NAMEs were enumerated in the *GeneralizationList* of the declared *concept*.

## 8.3 Model Validation

Figure 8.2 presents the invariants of the *Concept* and *Property* classes related to *generalizations*:

- *not_own_generalization*: A *concept* (§ 4) may not be listed on its own *GeneralizationList*, nor on the *GeneralizationList* of its direct or indirect generalizations.
- *compatible_generalizations*: The *generalizations* of a *concept* must all be compatible between themselves, that is, no two *generalizations* may have a *property* with the same name but a different type.
- *generalization_compatible_redefinition*: A *property* may only be redefined with the same type defined in the *generalizations*.
- *conflict_redefinition*: A *concept* is required to redefine a *property* that has been defined by two or more of its *generalizations* in order to resolve the definition conflict. That is required only if the *property* has been initialized or derived in at least one of the *generalizations*. Otherwise, the redefinition is not required.

```
-- Generalization of Circle and Rectangle:
@concept Shape
{
    -- Specializations below share the color attribute as-is:
    color: String;

    -- Specializations below redefine the area attribute:
    area: Double;
}

-- Specialization of Shape:
@concept Rectangle: Shape
{
    -- New attributes that characterize a rectangle:
    width: Double;
    height: Double;

    -- Redefinition of the area attribute:
    /area = width * height;
}

-- Another specialization of Shape:
@concept Rhombus: Shape
{
    -- Diagonal attributes that characterize a rhombus:
    p: Double;
    q: Double;

    -- Another redefinition of the area attribute:
    /area = (p * q) / 2.0d;
}

-- Specialization of both Rectangle and Rhombus:
@concept Square: Rectangle, Rhombus
{
    -- Only attribute needed to characterize a square:
    side_length: Double;

    -- Redefinitions of Rectangle's attributes:
    /width = side_length;
    /height = side_length;

    -- Redefinitions of Rhombus' attributes:
    /p = side_length * 1.41421356237d; -- square root of 2
    /q = p;

    -- Required to redefine area in order to resolve conflict
    -- between Rectangle's area and Rhombus' area:
    /area = side_length ^ 2.0d;
}
```

Figure 8.1: Generalization Examples

```
context Concept::all_generalizations: Set(Concept)
derive:
    generalizations->closure(generalizations)

context Concept::all_properties: Set(Property)
pre:
    all_generalizations->excludes(self)
derive:
    elements->union(
        generalizations.all_properties->select(p1|
            not elements->exists(p2| p1.name == p2.name)
        )
    )

context Concept::generalization_pairs
                : Set(Tuple(left: Concept, right: Concept))
derive:
    generalizations->collect(g1|
        generalizations
            ->select(g2| g1 != g2)
            ->collect(g2| Tuple { left: g1, right: g2 })
    )->flatten()

context Concept::generalization_property_pairs
                : Set(Tuple(left: Property, right: Property))
derive:
    generalization_pairs->collect(pair|
        pair.left.all_properties->collect(p1|
            pair.right.all_properties
                ->select(p2| p1 != p2 and p1.name = p2.name)
                ->collect(p2| Tuple { left: p1, right: p2 })
        )->flatten()
    )->flatten()

context Concept
inv not_own_generalization:
    all_generalizations->excludes(self)

context Concept
inv compatible_generalizations:
    generalization_property_pairs
        ->forAll(
            left.type.name = right.type.name and
            left.type.cardinality = right.type.cardinality
        )

context Concept
inv conflict_redefinition:
    generalization_property_pairs
        ->select(left.type = right.type)
        ->select(left.derived or left.expression->notEmpty() or
                 right.derived or right.expression->notEmpty())
        ->forAll(self.elements->exists(name = left.name))

context Property
inv generalization_compatible_redefinition:
    self.scope.generalizations.all_properties
        ->select(property| self.name = property.name)
        ->forAll(property|
            self.type.name = property.type.name and
            self.type.cardinality = property.type.cardinality
        )
```

Figure 8.2: Generalization Constraints

# Nine

# Abstractions

An *abstraction* in CML is a *concept* (§ 4) that cannot create instances on its own, but instead serves as a *generalization* (§ 8) for other *concepts*, which in turn can create their own instances. Thus, all instances of an *abstraction* are first instances of its *specializations*.

In CML, an *abstraction* may also define a *derived property* (§ 5) without providing an *expression* (§ 11) in its definition; such *properties* are called *abstract properties*.

CML's support for *abstractions* matches UML's [4], which allows the declaration of *abstract classes* by setting the *isAbstract* attribute of a *Class* instance to *true*. UML also allows the declaration of *abstract attributes* and *abstract operations*.

The original version of the ER [1] metamodel, however, as a consequence of lacking the *generalization/specialization* relationship, has not considered the notion of *abstract entities*.

## 9.1 Example

Listing 9.1 presents an example of an *abstract concept* declared in CML. As shown, the concept **Shape** is tagged as *abstract*, and as such no direct instances of *Shape* are ever instantiated. As an *abstract concept*, **Shape** can define *abstract properties*, like **area**, which is just a *derived property* (§ 5) without an *expression* (§ 11). An *abstract concept* may also define concrete *properties*, such as **color** in **Shape**. The concept **Circle** is a *especialization* of **Shape** that must redefine the property **area** (and provide an *expression*) if it is to be considered a *concrete concept*. As a *concrete concept*, **Circle** may have direct instances, which are in turn instances of *Shape* as well. **Circle** may also redefine *concrete properties* of **Shape**, like **color**, but the redefinition is not a requirement in this case. In **UnitCircle**, we can observe that

the redefinition of an *abstract property*, such as **area**, may be made *concrete*; meaning it does not need to be redefined as a *derived property*. The converse situation is also allowed in CML, where a *concrete property* is redefined by as a *derived property*, as illustrated with the property **radius** in **UnitCircle**.

## 9.2 Syntax

Listing 4.2 specifies the syntax used to declare a *concept* (§ 4) in CML. It shows that a *concept* may be tagged with the **abstract** keyword in order to convey it as an *abstract concept*. Listing 5.2 specifies the syntax used to declare a *property* (§ 5) in CML. It shows that a *property* may be prefixed with a forward slash ("/") in order to mark it as a *derived property*. If the optional **expression** is not provided, the property is then considered an *abstract property*.

Figure 4.1 presents the *concept* metamodel in an EMOF [5] class diagram, and listing 4.3 specifies the *concept* transformation from its concrete syntax to its abstract syntax. There is a **Boolean** attribute named **abstract** in the *Concept* class that determines whether a *concept* is *abstract* or not.

## 9.3 Model Validation

Listing 9.2 presents the invariants of the *Concept* and *Property* classes in CML's EMOF [5] metamodel related to *abstract concepts*:

- *abstract_property_redefinition*: A *concrete concept* must redefine concretely all *abstract properties* of its *generalizations*.
- *abstract_property_in_abstract_concept*: Only *abstract concepts* may have *abstract properties*.

```
-- As an abstract concept,
-- no direct instances of Shape are ever created.
@abstraction Shape
{
    -- A derived property without an expression
    -- is considered abstract.
    -- Only abstract concepts may have abstract properties.
    /area: Double;

    -- Abstract concepts may also have concrete properties:
    color: String;
}

-- All instances of Circle are in turn instances of Shape.
@concept Circle: Shape
{
    radius: Double;

    -- In order to be considered a concrete concept,
    -- Circle must redefine the abstract properties
    -- inherited from Shape.
    /area = 3.14159d * radius ^ 2;

    -- Circle may also redefine concrete properties of Shape.
    -- However, the redefinition is not required in this case.
    color = "Blue";
}

@concept UnitCircle: Circle
{
    -- Observe below that the redefinition of
    -- an abstract property may be concrete;
    -- that is, it does not have to be derived
    -- as it was done in Circle.
    area = 3.14159d;

    -- In the case above, however,
    -- it is desirable to redefine "area" as a derived property,
    -- in order to guarantee area's value cannot be modified
    -- after the instantiation of UnitCircle.
    -- This is done with the redefinition of "radius" below.
    -- Notice that, in Circle, radius was concrete,
    -- but its redefinition below makes it derived.
    -- That's allowed in CML just as the other way around,
    -- as it was done with "area" above.
    /radius = 1.0d;
}
```

Listing 9.1: Abstract Concept Example

```
context Property::abstract: Boolean
derive:
    self.derived and self.expression->isEmpty()

context Property::concrete: Boolean
derive:
    not self.abstract

context Concept
inv abstract_property_redefinition:
    self.concrete implies
        self.generalizations.all_properties
            ->select(abstract)
            ->forAll(p1|
                self.properties
                    ->select(p2| p1.name = p2.name)
                    ->reject(abstract)
                    ->notEmpty()
            )

context Property
inv abstract_property_in_abstract_concept:
    self.abstract implies self.scope.abstract
```

Listing 9.2: Abstract Concept Constraints

# Ten

# Functions

## 10.1 Built-In Functions

CML provides *built-in functions* in the `cml_base` module. Most *built-in functions* are defined via *template functions* (§ 10.2) for each target programming language or technology. The *templates* that implement the *built-in functions* must ensure they are pure functions (just like *declared functions* § 10.3) so that their *invocations* from *declared functions* (§ 10.3) or *lambdas* (§ 23) do not invalidate their *purity*.

## 10.2 Template Functions

In CML, *template functions* allow the definition of a *function* in the target programming language via StringTemplate [7]. The declaration of a *template function* only defines the *function signature*, while the *function expression* is defined in StringTemplate for each target programming language.

## 10.3 Declared Functions

In CML, *functions* may be defined by a *signature* and a corresponding *expression* (§ 11). The *function signature* declares the *parameter* and the resulting *type* (§ 25). Such a *function* is translated to a target programming language much like a *derived attribute* (§ 6.3), except that it does not belong to any particular *concept* (§ 4) and it may be invoked by any *expression*, in any scope.

*Functions* declared in CML are pure functions, which means they have the following characteristics:

- they always evaluate to the same result given the same arguments;
- their evaluation does not cause any side-effects or state mutation.

If a *declared function* in its definition invokes other *functions*, then those *functions* (being them other *declared functions* or *template functions* § 10.2) must provide the same guarantees listed above in order for the *declared function* to be considered pure.

## 10.4 Comprehension Functions

Some *functions*, being *declared* (§ 10.3) or just *templates* (§ 10.2), may be used in *comprehension expressions* (§ 10.4) if they have a specific *signature*. They may have one or more *parameters* as long as the first *parameter* is named `seq` and it is of a *sequence type*.

# Part III

# Expressions

# Eleven

# Expressions

An *expression* in CML is used to compute *values* (§ 26) or *references* (§ 27.1) to *concept* instances (§ 4). They are used to initialize or derive *properties* (§ 5). On the UML [4] metamodel, it corresponds to the *Expression* metaclass; in OCL [3], to *OclExpressionCS*.

CML *expressions* are designed to provide the same level of expressivity provided by OCL *expressions*, but the CML syntax varies from OCL's; sintactically, they differ especially on OCL's *collection operations*, which correspond to *comprehensions* (§ 24) in CML.

CML allows the use of *operators* in *expressions*. The categories of *operators* in CML are: *arithmetic operators* (§ 13), *relational operators* (§ 15), *logical operators* (§ 14), *referential operators* (§ 16), *type-checking operators* (§ 19) and *type-casting operators* (§ 20). Most of the *operators* in CML are infixed, with just three of them (`not`, `+` and `-`) being used as a prefix. The use of parenthesis is allowed to estabilish precedence.

Besides the *operators* listed above, CML also offers the following *expressions*: *paths* (§ 21), *invocations* (§ 22), *lambdas* (§ 23) and *comprehensions* (§ 24). They are all presented in the following chapters.

## 11.1 Examples

Listing 11.1 has some examples of *expressions* in CML. As shown, there are different types of expressions: *literal values* (§ 12), *prefixed/infixed expressions* (§ 13, § 14, § 15, § 16, § 20, § 20), *conditional expressions* (§ 18), *path expressions* (§ 21) and *comprehension expressions* (§ 24), among others.

```
@concept Expressions
{
    -- Literal Values:
    c: String = "SomeString";
    d: Integer = 123;

    -- Prefix Expression:
    prefixed_subtraction = -2;

    -- Infix Expressions:
    arithmetic = 1 + 2;
    relational = 3 == 3;
    logical = q and p;

    -- Conditional Expression:
    if_then_else = if a > 0 then a else b;

    -- Path Expression:
    path = somePath.bar;

    -- Comprehension Expression:
    selection = items | select: name == "this";
}
```

Listing 11.1: Expression Example

## 11.2 Syntax

Listing 11.2 specifies the syntax of all kinds of *expressions* in CML. It also lists them in their order of precedence.

Observe that the grammar in listing 11.2 is ambiguous. ANTLR [6] resolves the ambiguity based on the order in which the *expression* alternatives are listed. Thus, the order in listing 11.2 defines the precedence order among all *operators* in CML.

Also, according to ANTLR, and as required by CML, all *expressions* in the grammar are left-to-right associative, except for the *exponentiation expression*, which is right-to-left associative, as defined by the **<assoc=right>** clause.

```
expression returns [Expression expr]
  : literalExpression
  | pathExpression
  | lambdaExpression
  | invocationExpression
  | comprehensionExpression

  // Grouping
  | '(' inner=expression ')'

  // Arithmetic Expressions:
  | operator=('+' | '-') expression
  | <assoc=right> expression operator='^' expression
  | expression operator=('*' | '/' | '%') expression
  | expression operator=('+' | '-') expression

  // String Concatenation:
  | expression operator='&' expression

  // Relational Expressions:
  | expression operator=('<' | '<=' | '>' | '>=') expression
  | expression operator=('==' | '!=') expression

  // Referential Expressions:
  | expression operator=('===' | '!==') expression

  // Type-Checking:
  | expression operator=(IS | ISNT) type=typeDeclaration

  // Type-Casting:
  | expression operator=(AS | ASB | ASQ) type=typeDeclaration

  // Logical Expressions:
  | operator=NOT expression
  | expression operator=AND expression
  | expression operator=OR expression
  | expression operator=XOR expression
  | expression operator=IMPLIES expression

  // Conditional Expressions:
  | IF cond=expression
    THEN then=expression ELSE else_=expression
  | then=expression conj=(GIVEN | UNLESS) cond=expression
  | then=expression (ORQ | XORQ) else_=expression
```

Listing 11.2: Expression Syntax

## 11.3 Metamodel

Figure 11.1 displays the metamodel of *expressions* in CML.

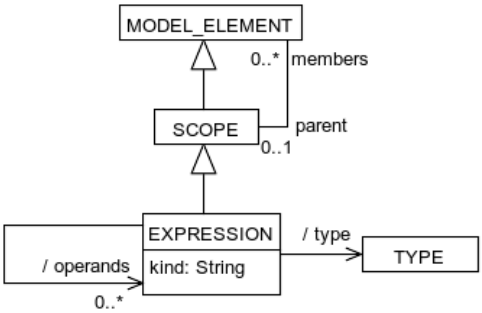


Figure 11.1: Metamodel of Expressions

## 11.4 Model Synthesis

For each kind of *expression* parsed by the compiler, an instance of the *Expression* metaclass is created, and its properties are assigned according to parsed information:

- *kind*: a *String* value matching the *Expression* subclass; for example, for the *Literal* subclass, **kind = "literal"**.
- *type*: a derived attribute that computes the *Type* of the *expression*; each *Expression* subclass will do its own *Type* computation by providing its own definition for this derived attribute.

# Twelve

# Literal Expressions

For each one of the primitive types (§ 26), there are corresponding *literal expressions* that can represent their values. In CML, as it is the case with UML [4], OCL [3] and ER [1], each *primitive type* may considered a mathematical *set*. A *literal value* allowed by a *type* (§ 25) corresponds to an *element* belonging to this *set*.

CML is able to infer the *primitive type* of a *property* (§ 5) based on the syntax of a *literal expression*. Each *primitive type* in CML has a unique syntax. In contrast, OCL has the same *literal* syntax for all *numeric types*, and thus cannot infer a *primitive type* from a *literal expression*.

## 12.1 Examples

Table 12.1 displays examples of *literal expressions* in CML.

| Type | Example |
|---|---|
| String | `"Hello!\n"` |
| Boolean | `true` |
| Integer | `123456` |
| Decimal | `1234.567` |
| Byte | `127b` |
| Short | `1234s` |
| Long | `12345671` |
| Float | `1234.567f` |
| Double | `1234.567d` |

Table 12.1: Literal Expressions for Primitive Types

## 12.2 Syntax

Table 12.2 displays the syntax of *literal expressions* in CML.

| Type | Syntax |
|---|---|
| String | `'"' ('\\'[btnr"\\] \| . )*? '"'` |
| Boolean | `'true' \| 'false'` |
| Integer | `('0'..'9')+` |
| Decimal | `('0'..'9')* '.' ('0'..'9')+` |
| Byte | `('0'..'9')+ 'b'` |
| Short | `('0'..'9')+ 's'` |
| Long | `('0'..'9')+ 'l'` |
| Float | `('0'..'9')* '.' ('0'..'9')+ 'f'` |
| Double | `('0'..'9')* '.' ('0'..'9')+ 'd'` |

Table 12.2: Literal Expressions for Primitive Types

## 12.3 Metamodel

Figure 12.1 displays the metamodel of *literal expressions* in CML.

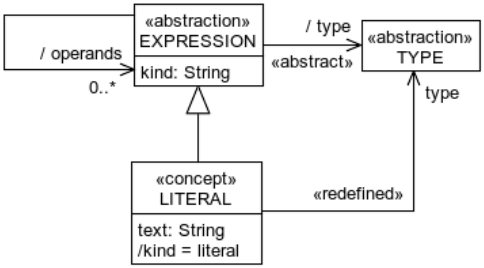


Figure 12.1: Metamodel of Literal Expressions

# Thirteen

# Arithmetic Expressions

*Arithmetic expressions* are composed by the *arithmetic operators*, which only accept as operands the *expressions* of *numeric* (§ 26.4) and *floating-point* (§ 26.5) types.

## 13.1 Examples

Table 13.1 displays examples of *arithmetic expressions* in CML.

| Operator | Operation | Example |
|---|---|---|
| `^` | Exponentiation | `3 ^ 2` |
| `*`, `/`, `%` | Multiplication, Division, Modulo | `6 * 2 / 3 % 4` |
| `+`, `-` | Addition, Subtraction | `6 + 2 - 1` |

Table 13.1: Examples using the Arithmetic Operators

## 13.2 Syntax

Listing 11.2 specifies the syntax of the *arithmetic expressions* in CML. It also lists them in their order of precedence in relation to other kinds of *expressions*.

## 13.3 Metamodel

Figure 13.1 displays the metamodel of *arithmetic expressions* in CML.

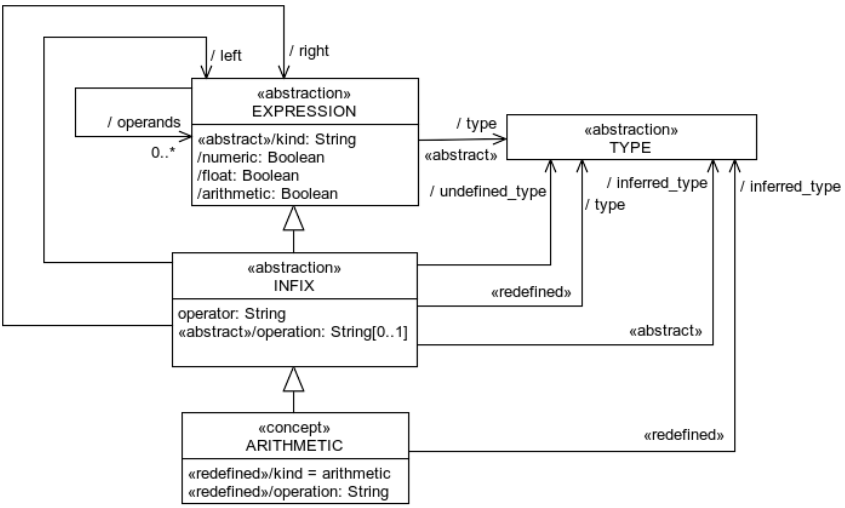


Figure 13.1: Metamodel of Arithmetic Expressions

## 13.4 Model Validation

Table 13.2 shows the precende order and associativity of *arithmetic operators* in CML.

| Operator | Operation | Associativity |
|---|---|---|
| ^ | Exponentiation | Right |
| *, /, % | Multiplication, Division, Modulo | Left |
| +, −* | Addition, Subtraction | Left |

*The addition/subtraction operators may also be prefixed in the unary form.

Table 13.2: Arithmetic Operators in Precedence Order

All *arithmetic operators* are infixed, but the addition (+) and subtraction (−) operators may also be prefixed when used in the unary form. The associativity of the arithmetic operators is from left to right, except for the exponentiation operator (^), where it is from right to left.

A validation error should reported by the compiler if any of the *operands* of an *arithmetic expression* is inferred to be of a *type* other than *numeric* (§ 26.4) or *floating-point* (§ 26.5).

# Fourteen

# Logical Expressions

*Logical expressions* are composed by the *logical operators*, which only accept as operands the *expressions* (§ 11) of the *Boolean* type (§ 26.3).

## 14.1 Examples

Table 14.1 displays examples of *logical expressions* in CML.

| Operator | Operation | Example |
|---|---|---|
| `not` | Negation | `not p` |
| `and` | Conjunction | `p and q` |
| `or` | Disjunction | `p or q` |
| `xor` | Exclusive Disjunction | `p xor q` |
| `implies` | Implication | `p implies q` |

Table 14.1: Logical Operators in Precedence Order

## 14.2 Syntax

Listing 11.2 specifies the syntax of the *logical expressions* in CML. It also lists them in their order of precedence in relation to other kinds of *expressions*.

## 14.3 Metamodel

Figure 14.1 displays the metamodel of *logical expressions* in CML.

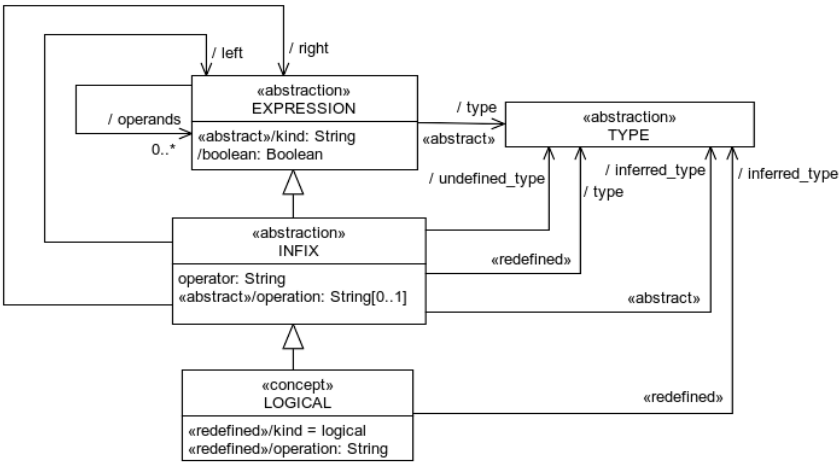


Figure 14.1: Metamodel of Logical Expressions

## 14.4 Model Validation

Table 14.2 shows the precende order of the *logical operators* in CML.

| Operator | Operation |
|---|---|
| `not` | Negation |
| `and` | Conjunction |
| `or` | Disjunction |
| `xor` | Exclusive Disjunction |
| `implies` | Implication |

Table 14.2: Logical Operators in Precedence Order

All logical operators are infixed, except for the negation operator (`not`), which is prefixed and unary. The associativity of the binary operators is always from left to right.

A validation error should reported by the compiler if any of the *operands* of a *logical expression* is inferred to be of a *type* other than *Boolean*.

# Fifteen

# Relational Expressions

*Relational expressions* are composed by the *relational operators*, which only accept as operands the *expressions* (§ 11) of a *relational type*, which include *String* (§ 26.6), the *numeric* (§ 26.4) and *floating-point* (§ 26.5) types, but exclude *Boolean* (§ 26.3) and *reference types* (§ 27.1).

## 15.1 Examples

Table 15.1 shows examples of the *relational expressions* in CML.

| Operators | Operation | Example |
|---|---|---|
| <, <=, >, >= | Ordering | `3 < 2` |
| ==, != | Equality, Unequality | `6 != 3` |

Table 15.1: Relational Operators

## 15.2 Syntax

Listing 11.2 specifies the syntax of the *relational expressions* in CML. It also lists them in their order of precedence in relation to other kinds of *expressions*.

## 15.3 Metamodel

Figure 15.1 displays the metamodel of *relational expressions* in CML.

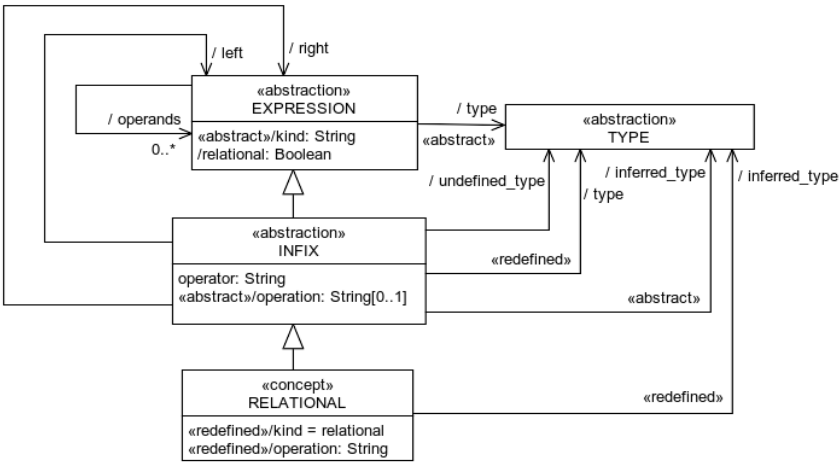


Figure 15.1: Metamodel of Relational Expressions

## 15.4 Model Validation

Table 15.2 shows the precende order of the *relational operators* in CML.

| Operators | Operation |
|---|---|
| <, <=, >, >= | Ordering |
| ==, != | Equality, Unequality |

Table 15.2: Relational Operators

All *relational operators* are infixed. There is no associativity between them.

A validation error should reported by the compiler if any of the *operands* of a *relational expression* is inferred to be of a *type* other than *String* (§ 26.6), *numeric* (§ 26.4) or *floating-point* (§ 26.5).

# Sixteen

# Referential Expressions

*Referential expressions* are composed by the *referential operators*, which only accept as operands the *expressions* (§ 11) resulting in *references* (§ 27.1). That excludes all *primitive types* (§ 26).

The *referential equality operator* (`is`) results in the `true` value if both operands reference the same instance. The *referential inequality operator* (`is`) results in `true` if the *refecences* do not point to the same instance.

## 16.1 Examples

Table 16.1 shows examples of *referential expressions* in CML.

| Operator | Operation | Example |
|---|---|---|
| `===` | Referential Equality | `product === book` |
| `!==` | Referencial Inequality | `product !== book` |

Table 16.1: Referential Operators

## 16.2 Syntax

Listing 11.2 specifies the syntax of the *referential expressions* in CML. It also lists them in their order of precedence in relation to other kinds of *expressions*.

## 16.3 Metamodel

Figure 16.1 displays the metamodel of *referential expressions* in CML.

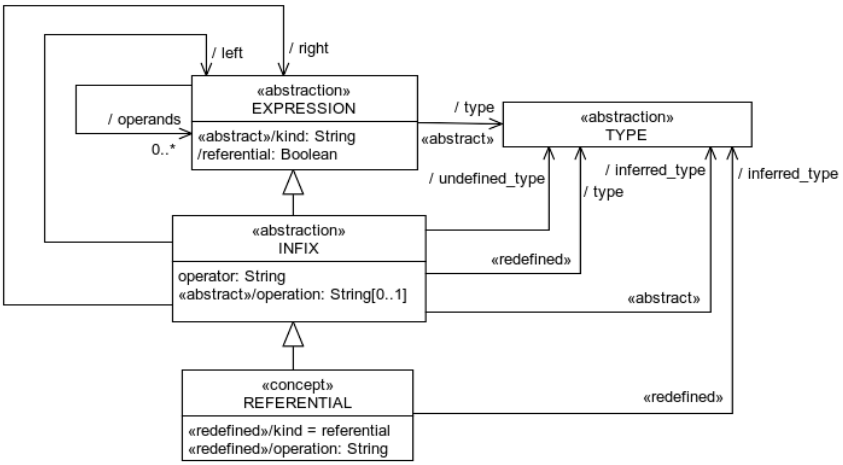


Figure 16.1: Metamodel of Referential Expressions

## 16.4 Model Validation

The *referential operators* are infixed. There is no associativity between them.

A validation error should reported by the compiler if any of the *operands* of a *referential expression* is inferred to be of a *type* other than a *reference type* (§ 27.1).

# Seventeen

# String Concatenation

*Expressions* (§ 11) of the *primitive types* (§ 26), such as the *String* type (§ 26.6), may be combined with the ampersand (*&*) operator for string concatenation.

## 17.1 Examples

Listing 17.1 presents an example of a *string concatenation expression* in CML.

```
@concept Address
{
  number: integer;
  street: string;
  city: string;
  state: string;
  zip: string;

  /display = number & "␣" & street & "\n" &
             city & ",␣" & state & "␣" & zip;
}
```

Listing 17.1: Example of String Concatenation

## 17.2 Syntax

Listing 11.2 specifies the syntax of the *string concatenation expressions* in CML. It also lists them in their order of precedence in relation to other kinds of *expressions*.

## 17.3 Metamodel

Figure 17.1 displays the metamodel of *string concatenations* in CML.

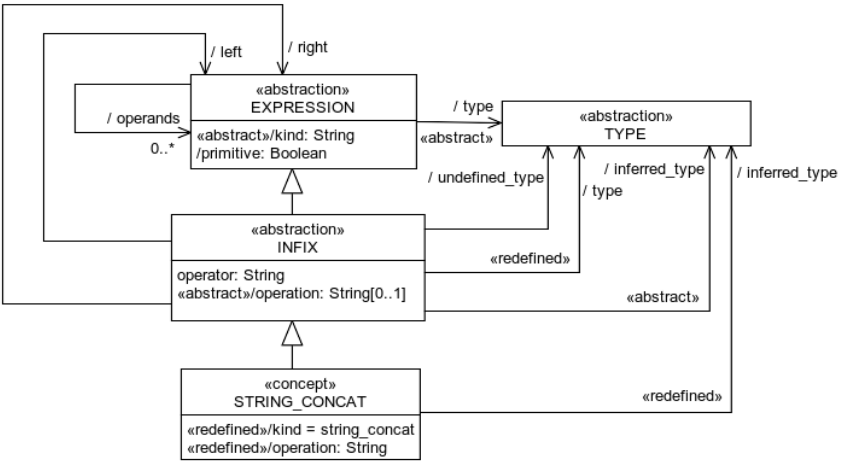


Figure 17.1: Metamodel of String Concatenations

## 17.4 Model Validation

Only an *expression* inferred to be of a *primitive type* (§ 26) may be used as an operand of a *string concatenation* expression. It is allowed that none of the operands are of *String* type. All operands are converted to *String* before concatenation.

The associativity of the concatenation operator is from left to right, just like the *arithmetic operators* (§ 13).

## 17.5 Type Inference / Conformance

The resulting *expression* is always of the *String* type, regardless of the operand types.

## 17.6 Model Translation

The *operands* are concatenated using the string concatenation operation provided by the target language. If the *operand type* is not *String*, then it is evaluated and the resulting value is converted to a *String* value, which is then concatenated with the other *String* values. The conversion of the *operands* is dependent on the target programming language, but it is normally done using the standard methods, such as *Objects.toString()* in Java, or *str()* in Python.

# Eighteen

# Conditional Expressions

In CML, *conditional expressions* allow alternating between one or more *expressions* (§ 11) based on some *condition*, which is an *expression* of *Boolean* type (§ 26.3). The remaining operands of a *conditional expression* – the alternating *expressions* – may be of any type (§ 25), including the *primitive types* (§ 26) and *references* (§ 27.1).

The *conditional expressions* are divided in three categories:

- unary: only evaluates a single *expression* if the *condition* evaluates to `true`.
- binary: results in the evaluation of the first *expression* if the *condition* evaluates to `true`; otherwise, it results in the evaluation of the second *expression*.
- optional: the *condition* is implicit; it results in the first *expression* if it provides a value; else, it results in the second *expression* if it provides a value; otherwise, it results in *none*.

The resulting type of a *conditional expression* is based on the type of its *operands*. The cardinality is also based on the *operands*.

## 18.1 Examples

Table 18.1 displays examples of *conditional expressions* in CML.

| Expression | Example |
|---|---|
| unary given | `item.book given item.qty > 2` |
| unary unless | `item.book unless item.qty <= 2` |
| binary if-then-else | `if item.qty > 10 then 0.9 else 1.0` |
| conditional or | `item.book or? item.accessory` |
| conditional xor | `item.book xor? item.accessory` |

Table 18.1: Conditional Expressions

## 18.2 Syntax

Listing 11.2 specifies the syntax of the *conditional expressions* in CML. The table 18.2 is another representation of the syntax.

| Expression | Example |
|---|---|
| unary given | `<expr> given <cond>` |
| unary unless | `<expr> unless <cond>` |
| binary if-then-else | `if <cond> then <expr1> else <expr2>` |
| conditional or | `<expr1> or? <expr2>` |
| conditional xor | `<expr1> xor? <expr2>` |

Table 18.2: Syntax of Conditional Expressions

## 18.3 Model Validation

In *conditional expressions*, the *condition* must be of *Boolean* type, and the *alternatives* must be of compatible types.

# Nineteen

# Type Checking

The *type-checking operator* (`is`) results in the `true` value if the instance referenced by the first operand is of the *type* (§ 25) specified by the second operand, or it is an *specialization* of such type. The *nagative type-checking operator* (`is`) results in `true` if the *type* does not match.

## 19.1 Examples

The table 19.1 shows examples of the *type-checking expressions*.

| Operator | Operation | Example |
|---|---|---|
| `is` | Type-Checking | `product is Book` |
| `isnt` | Negative Type-Checking | `product isnt Book` |

Table 19.1: Examples of Type-Checking Expressions

## 19.2 Syntax

Listing 11.2 specifies the syntax of the *type-checking expressions* in CML. It also lists them in their order of precedence in relation to other kinds of *expressions*. The table 19.2 is another representation of the syntax.

| Operator | Operation | Syntax |
|---|---|---|
| `is` | Type-Checking | `<expr> is <typeDeclaration>` |
| `isnt` | Negative Type-Checking | `<expr> isnt <typeDeclaration>` |

Table 19.2: Syntax of Type-Checking Expressions

# Twenty

# Type Casting

There are three *type-casting operators* in CML. The *safe type-casting operator* (`as`) may only be applied when the casting is guaranteed to work and the resulting expression has the same cardinality as the original one. The *forced type-casting operator* (`as!`) may raise an exception at runtime if the actual type of the instance is not compatible with the expected type. The *conditional type-casting operator* (`as?`) automatically selects the compatible instances, never raising an expception at runtime.

## 20.1 Examples

The table 20.1 presents the *type-casting expressions*.

| Oper. | Operation | Example |
|---|---|---|
| `as` | Safe Type-Casting | `products as Book*` |
| `as!` | Forced Type-Casting | `products as! Book` |
| `as?` | Condicional Type-Casting | `products as? Book?` |

Table 20.1: Examples of Type-Casting Expressions

## 20.2 Syntax

Listing 11.2 specifies the syntax of the *type-casting expressions* in CML. It also lists them in their order of precedence in relation to other kinds of *expressions*. The table 20.2 is another representation of the syntax.

| Operator | Operation | Syntax |
|---|---|---|
| `as` | Safe Type-Casting | `<expr> as <typeDeclaration>` |
| `as!` | Forced Type-Casting | `<expr> as! <typeDeclaration>` |
| `as?` | Condicional Type-Casting | `<expr> as? <typeDeclaration>` |

Table 20.2: Syntax of Type-Casting Expressions

# Twenty-one

# Path Expressions

*Path expressions* allow accessing the *values* (§ 26) and *references* (§ 27.1) of *properties* (§ 5) in instances of a *concept* (§ 4). Be them *attributes* (§ 6) or *association roles* (§ 7), *path expressions* will traverse through each *property* in the path in order to find the intended *values* or *references*. They can also be applied to *lambda parameters* (§ 23).

## 21.1 Examples

Listing 21.1 presents some examples of *path expressions* in CML.

```
@concept BookStore
{
    orders: Order*;

    /ordered_books = orders.items.book;
}

@concept Order
{
    items: Item*;
}

@concept Item
{
    book: Book;
    qty: integer;
    /description = book.title;
    /amount = qty * price;
}

@concept Book
{
    title: string;
    price: decimal;
}
```

Listing 21.1: Examples of Path Expressions

## 21.2 Syntax

Listing 21.2 specifies the syntax of *path expressions* in CML.

```
  // Let Expressions:
  | LET var=NAME '=' varExpr=expression IN resultExpr=expression;
```

Listing 21.2: Syntax of Path Expressions

# Twenty-two

# Invocation Expressions

*Invocations* apply a *function* (§ 10.1) to some *expressions* (§ 11) – the arguments – that correspond to the *function parameters*. CML provides some *built-in functions* in the base module. New functions may be defined via *template functions* (§ 10.2) or via *declared functions* (§ 10.3).

## 22.1 Examples

Listing 22.1 presents an example of an *invocation expression* in CML.

```
@concept Order
{
  items: Item*;
  total = sum(items.amount);
}

@concept Item
{
  amount: decimal;
}
```

Listing 22.1: Invocation Example

## 22.2 Syntax

Listing 22.2 specifies the syntax of *invocation expressions* in CML.

```
lambdaParameterList returns[Seq<String> params]:
  NAME (',' NAME)* '->';
```

Listing 22.2: Syntax of Invocation Expressions

# Twenty-three

# Lambda Expressions

A *lambda expression* may be used as an argument of the *invocation* (§ 22) of a *function* (§ 10.1). CML provides some *built-in functions* in the base module that accept a *lambda expression* as an argument.

*Lambda expressions* may be seen as an inline function definition. Just like *functions*, *lambda expressions* may have a number of *parameters* that are used in its inner expression.

Just like OCL ([3]), CML allows a parameterless *lambda expression* to have an undeclared, implicit parameter, which serves as its scope. This implicit parameter in the *lambda expression* is inferred from the *function* declaring the *lambda* as one of its parameters. *Functions* may declare a *lambda parameter* with a *function type declaration* (§ 27.3).

Additionally, a *lambda expression* in CML is considered a pure function (just like *declared functions* § 10.3) given that the *invocations* performed by its inner expression are also pure *functions* or *lambdas*.

## 23.1 Examples

Listing 23.1 presents an example of a *lambda expression* in CML.

```
@concept BookStore
{
    orders: Order*;

    /premium_orders = select(orders, { total > 1000.00 });
}

@concept Order
{
    total: decimal;
}
```

Listing 23.1: Lambda Example

## 23.2 Syntax

Listing 23.2 specifies the syntax of *lambda expressions* in CML.

```
pathExpression returns [Path path]:
  NAME ('.' NAME)*;

lambdaExpression returns[Lambda lambda]:
   '{' lambdaParameterList? expression '}';
```

Listing 23.2: Syntax of Lambda Expressions

# Twenty-four

# Comprehension Expressions

CML supports a special kind of *expression* (§ 11) called *comprehension*. It is a flexible and expressive way to calculate *values* (§ 26) or to query *references* (§ 27.1). It is thus a more powerful construct than *path expressions* (§ 21), but it can also combined with the latter.

The *comprehension expressions* have a similar purpose to the *set builder notation* in Mathematics, and also to *list comprehensions* in Python, or *for comprehensions* in Scala. Unlike those languages, in CML, *comprehensions* can be combined using the pipe (|) operator and *functions* (§ 10.4), making them more extensible.

## 24.1 Examples

Listing 24.1 presents an example of a *comprehension expression* in CML.

```
@concept BookStore
{
    orders: Order*;

    /premium_orders = orders | select: total > 1000.00;
}

@concept Order
{
    total: decimal;
}
```

Listing 24.1: Example of Comprehension Expression

## 24.2 Syntax

Listing 24.2 specifies the syntax of *comprehension expressions* in CML.

```
invocationExpression returns [Invocation invocation]:
  NAME '(' expression (',' expression)* ')' lambdaExpression?;

comprehensionExpression returns [Comprehension comprehension]:
  (pathExpression |
   FOR enumeratorDeclaration (',' enumeratorDeclaration)*)
  queryStatement+;

enumeratorDeclaration returns [Enumerator enumerator]:
  var=NAME IN pathExpression;

queryStatement returns [Query query]:
  '|' (NAME | keywordExpression+);
```

Listing 24.2: Syntax of Comprehension Expressions

# Part IV

# Type System

# Twenty-five

# Types

CML is a statically-typed language. All its *properties* (§ 5) and *expressions* (§ 11) must have a *type* declared or inferred at compilation time. Additionally:

- the type of a *property* must be compatible with the type of its *expression*;
- the type of the *operands* in an *expression* must be compatible with its *operator*;
- the type of a *property redefinition* must be compatible with the original *property definition* in the *generalization*;
- and the type of the arguments in a *invocation* must be compatible with the type of the declared parameters of the corresponding *function*.

In order to allow the compiler to verify the types during the model validation, the model elements that must have an associated type are all specializations of the *TypedElement* metaclass, which has the abstract property *type*. Each *TypedElement* specialization must redefine the *type* property in order to be able to infer its type.

All types are specializations of the metaclass *Type*, which represents the *type declaration* or *type inference* of a *TypedElement*.

Among other properties, *Type* defines the *min_cardinality* and the *max_cardinality* allowed on a *TypedElement*. CML only allows specific cardinality options:

- A *type* with the min/max cardinality equal to one is called "required". It is declared as required when no suffix is provided in a *type declaration*.
- A *type* with cardinality zero-or-one is called "optional". It is declared as optional by the question-mark (?) suffix in a *type declaration*.
- A *type* with cardinality zero-or-more is called "sequential". It is declared as sequential by the asterisk-mark (*) suffix in a *type declaration*.

The direct specializations of *Type* are the following metaclasses:

- *ValueType*: for *types* that contain *values*, such as the *primitive types* (§ 26).
- *ReferenceType*: for *types* that contain *references* to the actual *instances*, such as the *concept types* (§ 27.1).
- *TupleType*: for types that declare *tuples* used as an *argument* or a *result* of a *function* (§ 10.1).
- *FunctionType*: used in *functions* to declare that an *argument* accepts a *lambda expression* (§ 23).

The instances of the *Type* specializations listed above are presented in the following chapters.

# Twenty-six

# Primitive Types

A *primitive type* in CML is one of the pre-defined *value types* supported by the language, as shown in tables 26.1 and 26.2.

In the ER [1] metamodel, a *data type* is formally defined as a *set* of *values* that can be held by an *attribute* (§ 6). The original ER paper [1] states that, for each *value set* (i.e. *data type*), there is a *predicate* that can be used to test whether a *value* belongs to the *set*. In CML, instead, *literal expressions* are syntactically defined for each *primitive type*, so that the *type* can be inferred from the *literal expression*.

On the original ER paper, it is also said that *values* in a *value set* may be equivalent to *values* in another *value set*. In CML, also, *literal expressions* of the *Integer* type may be equivalent to *literal expressions* of the *Decimal* type, and so with other *numeric types*. This allows *expressions* (§ 11) of a *primitive type* to be promoted to *expressions* of another *primitive type* in order to allow *type inference* of composite *expressions*.

In the UML [4] metamodel, there is a specific metaclass named *PrimitiveType*, which matches to the same notion in CML.

## 26.1 Example

Figure 26.1 presents examples of *atributes* declared with *primitive types* in CML. Each example corresponds to one of the *primitive types* supported by the language, as shown in tables 26.1 and 26.2. The *target constructors* of CML's base module will translate the primitive types to Java, C#, C/C++, Python, and TypeScript (JavaScript), according to the mapping shown in the tables.

```
@concept PrimitiveTypes
{
    -- Core Primitive Types:

    -- Only values are the literal expressions: true, false
    a: Boolean;

    -- 32-bit signed two's complement integer
    c: Integer;

    -- Arbitrary precision arithmetic.
    -- BigDecimal in Java; decimal in C#; decimal128 in C++.
    d: Decimal;

    -- 16-bit Unicode character sequences
    -- as in Java, C#, C++ (std::wstring), and JavaScript.
    b: String;

    -- Additional Primitive Types:

    -- 8-bit signed two's complement integer
    e: Byte;

    -- 16-bit signed two's complement integer
    f: Short;

    -- 64-bit signed two's complement integer
    g: Long;

    -- 32-bit IEEE 754 floating point
    h: Float;

    -- 64-bit IEEE 754 floating point
    i: Double;
}
```

Figure 26.1: Example of *Primitive Types*

| CML | Java | C# | C++ | Python | TypeScript (JavaScript) |
|---|---|---|---|---|---|
| String | String | string | std::wstring | str | string |
| 16-bit Unicode character sequences. | | | | | |
| Boolean | boolean | bool | bool | bool | boolean |
| Only values are the literal expressions: **true**, **false**. | | | | | |
| Integer | int | int | int32_t | int | number |
| 32-bit signed two's complement integer. | | | | | |
| Decimal* | BigDecimal | decimal | decimal128 | Decimal | number |
| Arbitrary precision, fixed-point, or decimal floating-point, depending on the target language. | | | | | |

*The specification of Decimal type varies by target programming language. Compared to the binary floating-point types (Float and Double), the Decimal type is better suited for monetary calculations at a performance cost.

Table 26.1: Core Primitive Types in CML.

## 26.2 Syntax

Figure 26.2 specifies the syntax used to declare any kind of *type*, including *primitive types*. The NAME of the *type* may be any of the *primitive types* defined in the column named *CML* of the tables 26.1 and 26.2. Optionally, cardinality may also be specified for a *primitive type*. The '*' cardinality suffix allows zero or more values to be stored in a property as a sequence type. The '?' cardinality suffix allows a single value to be stored, or none. If no cardinality is specified, a value must be assigned to the *attribute* when its *concept* is instantiated.

Figure 7.1 presents the *Type* metaclass in an EMOF [5] class diagram of the CML metamodel, and figure 26.3 specifies the transformation from the *type* concrete syntax to its abstract syntax.

| CML | Java | C# | C++ | Python | TypeScript (JavaScript) | Specification |
|---|---|---|---|---|---|---|
| Byte | byte | byte | int8_t | int | number | 8-bit signed two's complement integer |
| Short | short | short | int16_t | int | number | 16-bit signed two's complement integer |
| Long | long | long | int64_t | long | number | 64-bit signed two's complement integer |
| Float | float | float | float* | float | number | 32-bit IEEE 754 binary floating point |
| Double | double | double | double* | float | number | 64-bit IEEE 754 binary floating point |

*C++ floating point types may vary by hardware and compiler

Table 26.2: Additional Primitive Types in CML.

## 26.3 Boolean Type

The *Boolean* type accepts only two literal expressions (§ 12): `true` or `false`. When either of these two literals is found in *expressions* (§ 11), its *type* is inferred to be *Boolean*.

Any *expression* of the *Boolean* type may be used as an operand of *logical expressions* (§ 14).

## 26.4 Numeric Types

The *numeric types* are the following in CML (from the smallest set to the largest one): *Byte*, *Short*, *Integer*, *Long* and *Decimal*. They belong to the group of *primitive types* (§ 26), which are shown in tables 26.1 and 26.2.

There is a specific *literal expression syntax* (§ 12) for each *numeric type*, which allows the type to be inferred uniquely. *Literal expressions* of a narrower-range type may be equivalent to literal expressions of the wider-range one. This allows type inference of *arithmetic expressions* (§ 13) and *relational expressions* (§ 15).

```
typeDeclaration returns [Type type]
  : name=NAME cardinality? // named type
  | tuple=tupleTypeDeclaration // tuple type
  | params=tupleTypeDeclaration '->' result=typeDeclaration // function type
  | '(' inner=typeDeclaration ')';

cardinality:
  ('?' | '*');

tupleTypeDeclaration returns [TupleType type]:
  '(' ( tupleTypeElementDeclaration (',' tupleTypeElementDeclaration)* )? ')'
  cardinality?;

tupleTypeElementDeclaration returns [TupleTypeElement element]:
  (name=NAME ':')? type=typeDeclaration;

typeParameterList returns [Seq<TypeParameter> params]:
  '<' typeParameter (',' typeParameter)* '>';

typeParameter returns [TypeParameter param]:
  name=NAME;
```

Figure 26.2: Type Declaration Syntax

```
node Type: NAME CARDINALITY?
{
    name = NAME;
    cardinality = CARDINALITY?;
}
```

Figure 26.3: Type AST Instantiation

## 26.5 Floating-Point Types

The *floating-point types* in CML are *Float* and *Double* for single-precision (32-bit) and double-precision (64-bit), respectively. They belong to the group of *primitive types* (§ 26), which are shown in tables 26.1 and 26.2.

Each *floating-point type* has a specific *literal expression syntax* (§ 12), which allows them to be inferred uniquely. *Literal expressions* of the *Float* type may be equivalent to *literal expressions* within the same range of the *Double* type. This allows type inference of *arithmetic expressions* (§ 13) and *relational expressions* (§ 15).

In order to prevent data/precision loss, it is disallowed the coercion of a value of a *numeric type* (§ 26.4) to a value of a *floating-point type*, and vice-versa. That means it is only possible to combine *expressions* (§ 11) of *floating-point* and *numeric* types with explicit conversion.

## 26.6 String Type

The *String* type in CML represents a 16-bit Unicode character sequence, matching the same type in other programming languages, as described in table 26.1.

There is a specific *literal expression syntax* (§ 12) for *String* values, which allows them to be inferred uniquely. *Expressions* (§ 11) of the *String* type may be the operands of *relational expressions* (§ 15). They may also be combined with the ampersand (*&*) operator for concatenation (§ 17), in which case the resulting *expression* is of *String* type.

# Twenty-seven

# Other Types

## 27.1 Reference Types

In CML, all *type declarations* referring to the name of a *concept* (§ 4) are instances of the *ReferenceType* metaclass (§ 25); in short, the *type declaration* declares a *reference type*. A *property* (§ 5) of a concept A, whose type is declared or inferred to be of a concept B, holds a reference to an instance of concept B; not the actual instance. This allows the *properties* of a concept C to also reference the same instance of B.

Models in CML do not to keep track of the memory used to store the actual instances. CML expects the target programming language or technology to support some kind of reference management, such as a garbage collector in Java or automatic reference counting in Swift, or still a database. CML does not require any particular implementation.

The *path expressions* (§ 21) whose result is of a *reference type* may be used in *referential expressions* (§ 16), *type-checking expressions* (§ 19), *type-casting expressions* (§ 20) and in *invocations* (§ 22).

## 27.2 Tuple Types

*Tuples* may be declared as the *type* of an *argument* or the *result* of a *function* (§ 10.1). A *type declaration* that declares a *tuple* is an instance of the *TupleType* metaclass.

## 27.3 Function Types

A *function type declaration* may used in *functions* (§ 10.1) to declare that an *argument* accepts a *lambda expression* (§ 23).

# Part V

# Appendices

# A

# CML Grammar

```
// Compilation Units:

compilationUnit:
  declarations*;

declarations
  : moduleDeclaration
  | conceptDeclaration
  | associationDeclaration
  | taskDeclaration
  | templateDeclaration
  | functionDeclaration;


// Module Declarations:

moduleDeclaration returns [TempModule module]:
  MODULE NAME '{' importDeclaration* '}';

importDeclaration returns [Import _import]:
  IMPORT NAME ';';

// Concept Declarations:

conceptDeclaration returns [TempConcept concept]:
  (ABSTRACTION | CONCEPT) NAME
  (':' generalizations)?
  (';' | propertyList);

generalizations:
  NAME (',' NAME)*;
```

```
// Property Declarations:

propertyList:
  '{' (propertyDeclaration ';')* '}';

propertyDeclaration returns [Property property]:
 DERIVED? NAME (':' typeDeclaration)? ('=' expression)?;

DERIVED: '/';

// Association Declarations:

associationDeclaration
  returns [Association association]:
  ASSOCIATION NAME
  '{' (associationEndDeclaration ';')* '}';

associationEndDeclaration
  returns [Association association]:
  conceptName=NAME '.' propertyName=NAME
  (':' typeDeclaration)?;


// Function Declarations:

functionDeclaration returns [Function function]:
  annotationList?
  FUNCTION name=NAME
  typeParams=typeParameterList?
  params=functionParameterList
  ('->' resultType=typeDeclaration)? ('=' expression)? ';';

functionParameterList returns [Seq<FunctionParameter> params]:
  '(' functionParameterDeclaration? (',' functionParameterDeclaration)* ')';

functionParameterDeclaration returns [FunctionParameter param]:
 name=NAME ':' type=typeDeclaration;

// Type Declarations:
```

```
typeDeclaration returns [Type type]
  : name=NAME cardinality? // named type
  | tuple=tupleTypeDeclaration // tuple type
  | params=tupleTypeDeclaration '->' result=typeDeclaration // function type
  | '(' inner=typeDeclaration ')';

cardinality:
  ('?' | '*');

tupleTypeDeclaration returns [TupleType type]:
  '(' ( tupleTypeElementDeclaration (',' tupleTypeElementDeclaration)* )? ')'
  cardinality?;

tupleTypeElementDeclaration returns [TupleTypeElement element]:
  (name=NAME ':')? type=typeDeclaration;

typeParameterList returns [Seq<TypeParameter> params]:
  '<' typeParameter (',' typeParameter)* '>';

typeParameter returns [TypeParameter param]:
  name=NAME;

// Task Declarations:

taskDeclaration returns [Task task]:
  TASK NAME
  constructorDeclaration?
  (';' | propertyList);

constructorDeclaration: ':' NAME;


// Expression Declarations:

expression returns [Expression expr]
  : literalExpression
  | pathExpression
  | lambdaExpression
  | invocationExpression
  | comprehensionExpression

  // Grouping
  | '(' inner=expression ')'
```

```
  // Arithmetic Expressions:
  | operator=('+' | '-') expression
  | <assoc=right> expression operator='^' expression
  | expression operator=('*' | '/' | '%') expression
  | expression operator=('+' | '-') expression

  // String Concatenation:
  | expression operator='&' expression

  // Relational Expressions:
  | expression operator=('<' | '<=' | '>' | '>=') expression
  | expression operator=('==' | '!=') expression

  // Referential Expressions:
  | expression operator=('===' | '!==') expression

  // Type-Checking:
  | expression operator=(IS | ISNT) type=typeDeclaration

  // Type-Casting:
  | expression operator=(AS | ASB | ASQ) type=typeDeclaration

  // Logical Expressions:
  | operator=NOT expression
  | expression operator=AND expression
  | expression operator=OR expression
  | expression operator=XOR expression
  | expression operator=IMPLIES expression

  // Conditional Expressions:
  | IF cond=expression
    THEN then=expression ELSE else_=expression
  | then=expression conj=(GIVEN | UNLESS) cond=expression
  | then=expression (ORQ | XORQ) else_=expression

  // Let Expressions:
  | LET var=NAME '=' varExpr=expression IN resultExpr=expression;

pathExpression returns [Path path]:
  NAME ('.' NAME)*;

lambdaExpression returns[Lambda lambda]:
  '{' lambdaParameterList? expression '}';
```

```
lambdaParameterList returns[Seq<String> params]:
  NAME (',' NAME)* '->';

invocationExpression returns [Invocation invocation]:
  NAME '(' expression (',' expression)* ')' lambdaExpression?;

comprehensionExpression returns [Comprehension comprehension]:
  (pathExpression |
   FOR enumeratorDeclaration (',' enumeratorDeclaration)*)
  queryStatement+;

enumeratorDeclaration returns [Enumerator enumerator]:
  var=NAME IN pathExpression;

queryStatement returns [Query query]:
  '|' (NAME | keywordExpression+);

keywordExpression returns [Keyword keyword]:
  NAME ':' lambdaParameterList? expression;



// Names:

// All reserved words must be declared before NAME.
// Otherwise, they are recognized as a NAME instead.

FOR: 'for';
IN: 'in';
AS: 'as';
ASB: 'as!';
ASQ: 'as?';
IS: 'is';
ISNT: 'isnt';
IF: 'if';
THEN: 'then';
ELSE: 'else';
GIVEN: 'given';
UNLESS: 'unless';
LET: 'let';
ORQ: 'or?';
XORQ: 'xor?';
BOOLEAN: 'true' | 'false';
```

```
AND: 'and';
OR: 'or';
XOR: 'xor';
IMPLIES: 'implies';
NOT: 'not';
TEMPLATE: '@template';
FUNCTION: '@function';
ABSTRACTION: '@abstraction';
CONCEPT: '@concept';
TASK: '@task';
ASSOCIATION: '@association';
MODULE: '@module';
IMPORT: '@import';

NAME:
  ('A'..'Z' | 'a'..'z')
  ( 'A'..'Z' | 'a'..'z' | '0'..'9' | '_' )*;


// Literals:

literalExpression returns [Literal literal]:
  BOOLEAN | STRING |
  INTEGER | LONG | SHORT | BYTE | DECIMAL |
  FLOAT | DOUBLE;

STRING:
  '"' (ESC | . )*? '"';

fragment ESC: '\\'[btnr"\\];

INTEGER:
  ('0'..'9')+;

LONG:
  ('0'..'9')+ 'l';

SHORT:
  ('0'..'9')+ 's';

BYTE:
  ('0'..'9')+ 'b';

DECIMAL:
```

```
  ('0'..'9')* '.' ('0'..'9')+;

FLOAT:
  ('0'..'9')* '.' ('0'..'9')+ 'f';

DOUBLE:
  ('0'..'9')* '.' ('0'..'9')+ 'd';

// Ignoring Whitespace:

WS:
  ( ' ' | '\t' | '\f' | '\n' | '\r' )+ -> skip;


// Ignoring Comments:

COMMENT:
  (('//' | '-') .*? '\n' | '/*' .*? '*/' ) -> skip;
```

# Bibliography


[1] Peter Pin-Shan Chen. The Entity-Relationship Model (Reprinted Historic Data). In David W. Embley and Bernhard Thalheim, editors, *Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges*, pages 57–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[2] Torben ï£¡gidius Mogensen. *Introduction to Compiler Design.* Undergraduate Topics in Computer Science. Springer, 2011.

[3] OMG. Object Constraint Language (OCL), Version 2.4, 2014.

[4] OMG. Unified Modeling Language (UML), Superstructure, Version 2.5, 2015.

[5] OMG. Meta Object Facility (MOF) Core Specification, Version 2.5.1, 2016.

[6] Terence Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition, 2013.

[7] Terence John Parr. Enforcing Strict Model-view Separation in Template Engines. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 224–233. ACM, New York, NY, USA, 2004.

# ANEXO B – Artigo

A seguir, o artigo referente a este trabalho que foi aceito pela Conferência Internacional de Modelagem Conceitual – ER 2017 e apresentado nesta conferência no dia 8 de Novembro de 2017 em Valência, Espanha. (URL de proceedings: `http://ceur-ws.org/Vol-1979`; URL do artigo: `http://ceur-ws.org/Vol-1979/paper-20.pdf`.)

# Reusing Conceptual Models: Language and Extensible Compiler

**Quenio Cesar Machado dos Santos**[1] **, Raul Sidnei Wazlawick**[2]

[1]Computer Sciences,
UFSC - Universidade Federal de Santa Catarina, Brazil,

[2]Associate Professor of Computer Sciences Department,
UFSC - Universidade Federal de Santa Catarina, Brazil

queniodossantos@gmail.com, raul@inf.ufsc.br

***Abstract.*** *This paper presents a textual programming language for conceptual modeling (based on UML classes/associations and OCL constraints) and a compiler that can generate code in any target language or technology via extensible textual templates. The language and compiler allow the specification of information managed by ever-changing, increasingly distributed software systems. From a single source, automated code generation keeps implementations consistent with the specification across the different platforms and technologies. Furthermore, as the technology landscape evolves, the target templates may be extended to embrace new technologies. Unlike other approaches, such as MDA and MPS, the built-in tooling support, along with the textual nature of this modeling language and its extensible templates, is expected to facilitate the integration of model-driven software development into the workflow of software developers.*

## 1. Introduction

In order to address the challenges of the ever-changing, increasingly distributed technologies used on software systems, Model-Driven Architecture (MDA [OMG 2014a]) from Object Management Group (OMG) has been promoting model-driven software development. In particular, MDA has guided the use of high-level models (created with OMG standards, such as UML [OMG 2015], OCL [OMG 2014b] and MOF [OMG 2016]) to derive software artifacts and implementations via automated transformations. As one of its value propositions, the MDA guide [OMG 2014a] advocates:

> "Automation reduces the time and cost of realizing a design, reduces the time and cost for changes and maintenance and produces results that ensure consistency across all of the derived artifacts."

MDA provides guidance and standards in order to realize this vision, but it leaves to software vendors the task of providing the tools that automate the process of generating the implementations from the models. The key role played by tools has been demonstrated by Voelter [Voelter 2014] in his *Generic Tools, Specific Languages* approach for model-driven software development. Voelter [Voelter 2014] has used domain-specific languages (DSLs) with the Metaprogramming System (MPS) in order to generate software artifacts. Unlike MDA, which is based on UML/MOF models, MPS allows the specification of models using domain-specific editors.

The conceptual modeling language and extensible compiler presented here are an alternative approach to MPS. While the latter is a fully integrated development environment based on domain-specific languages and their projectional editors[1], the former (hereby called CML) is a compiler. As *input*, CML has source files defined using its own conceptual language, which provides an abstract syntax similar to (but less comprehensive than) a combination of UML [OMG 2015] and OCL [OMG 2014b]; and, as *output*, any target languages, based on extensible templates that may be provided by the compiler's base libraries, by third-party libraries, or even by developers. Both the CML language and compiler are in its initial stage of development, as part of the author's Computer Sciences Bachelor Technical Report, and available as an open source project online [dos Santos 2017].

Section 2 explains the motivation for creating yet another language for conceptual modeling. The next two sections present the language (section 3) and the compiler with its extensible templates (section 4). Section 5 compares CML to other languages, tools and frameworks that can also generate code from conceptual models. We conclude in section 6, reiterating the objectives being pursued by CML and exploring options to validate the use of the CML compiler.

## 2. Why A New Language?

Thalheim [Thalheim 2011] has observed that the choice of a conceptual modeling language has to do with its purpose. He suggests that a language is just a *carrier* mapping some properties of the *origin* (the problem space) that can provide utility to its users.

In this context, the purpose of the CML language is being a tool that allows software developers to transform text-based conceptual models into executable code of an extensible range of technologies. In order to achieve this purpose, a new language is designed with the following goals (among others):

- *Developer Experience*: CML follows the principle "the model is the code" as laid out on the *Conceptual-Model Programming* (CMP) manifesto [Embley et al. 2011]. Furthermore, CML is also intended to enable software developers to do *conceptual modeling* on the same workflow they are used to doing *programming*; that is, using text editors and a compiler. CML strives to *not only be* the code (as advocated by CMP), but also *look like* code (syntactically speaking), pursuing compatibility with developers' mindset, toolset and workflow. By providing its own syntax based on existing programming languages, CML then promotes the *modeling-as-programming* approach. The UML [OMG 2015] notation, on the other hand, being graphical, is not suited for mainstream, textual programming. However, the *Human Usable Textual Notation* (HUTN) [Rose et al. 2008a] is a textual syntax for MOF-based [OMG 2016] metamodels, and as such, it can also be used for UML models. The syntax of the structural (static) elements of CML models is based on HUTN.
- *Language Evolution*: This initial version is being designed for the validation of the model-driven development approach offered by CML. Unlike the expressive power seen on UML [OMG 2015] and OWL 2 [W3C 2012] with their breadth of

[1]Projectional editors in MPS do not rely on parsers. Instead, the abstract syntax tree (AST) is modified directly. MPS renders the visual representation of the AST based on the DSL editor definition.

features, the CML language initially supports generalization/specialization, bidirectional associations (with zero-or-one and zero-or-many cardinality) and the ability to define derived attributes and associations with OCL-like expressions. These features have already allowed the specification of CML compiler's own metamodel in CML itself. The CML compiler is thus the first system used to validate CML's aplicability, and will continue to do so as the language evolves.

- *Extensible Target Generation*: Some of the language features should enable the generation of code into a wide range of target languages and technologies. Among the features that must be provided by the CML language, it is the ability to break models into modules (already available); the ability to share modules as libraries (planned); the ability to specify different code generation targets (already available); and the ability to annotate model elements in order to provide more information for specific targets during code generation (also planned). In order to effectively support code generation, these language features must be available in a single language, so that they can be compatible with each other and with the compiler backend.

Section 5 provides further motivation for developing CML, comparing it to related work.

## 3. The Language

This section presents an overview of the conceptual modeling language. The *concrete syntax* is presented using an example in subsection 3.1. The mapping of the CML example to UML [OMG 2015] and OCL [OMG 2014b] is illustrated in subsection 3.2. The CML metamodel (the *abstract syntax*'s structure) is presented in subsection 3.3. (The *CML Specification* [dos Santos 2017], which is under development, should eventually provide a formal description of the *concrete syntax*, along with its mapping to the *abstract syntax*.)

### 3.1. An Example

On the example of figure 1, some concepts, such as *Book* and *Customer*, are declared in CML. The block-based syntax declaring each concept resembles the C [ISO 2011] language's syntax. Each concept declares a list of properties. The property declarations are based on the Pascal [Jensen and Wirth 1974] style for variable declarations, where the name is followed by a colon (":") and then the type declaration. Part of the CML syntax for expressions, such as the expression in *BookStore*'s *orderedBooks*, is based on OCL [OMG 2014b] expressions. While the syntax of the expression in *goldCustomers* is new, its semantics also match OCL [OMG 2014b] query expressions.

The key language features: *Book* and *Customer* are concepts; *title* and *price* under the *Book* concept are attributes; *totalSales* under the *Customer* concept is a derived attribute; the properties *books* and *customers* declared under the *BookStore* concept represent unidirectional associations (in UML [OMG 2015], they would correspond to the association roles); *CustomerOrder* binds two unidirectional associations (represented by the *orders* property under the *Customer* concept and by the *customer* property under the *Order* concept) into a single bidirectional association; the properties *goldCustomers* and *orderedBooks* under the *BookStore* concept are examples of derived associations.

These language features are defined in the subsection 3.3.

```
concept BookStore {
  books: Book*; customers: Customer*; orders: Order*;
  /goldCustomers = customers | select: totalSales > 1000;
  /orderedBooks = orders.items.book;
}

concept Book {
  title: String; price: Decimal; quantity: Integer = 0;
}

concept Customer {
  orders: Order*; /totalSales = sum(orders.total);
}

concept Order {
  customer: Customer; total: Decimal;
}

association CustomerOrder {
  Order.customer; Customer.orders;
}
```

**Figure 1. Adapted from the fictional Livir bookstore; a case study by Wazlawick [Wazlawick 2014].**

### 3.2. Mapping CML Source to UML and OCL

Part of the CML metamodel (presented in section 3.3) may be considered a small subset of the UML [OMG 2015] metamodel. Thus, the structural (static) elements of CML models can be transformed into UML class diagrams. The example CML model in the listing of figure 1 is mapped to the UML model in figure 2.

In figure 2, the CML concepts (*BookStore*, *Book*, *Customer* and *Order*) are mapped to corresponding UML classes. The CML properties that represent attributes (such as *title*, *quantity* and *price* of *Book*) are mapped to UML attributes under each class. The CML properties that represent unidirectional associations (*books*, *customers*, and *goldCustomers* of *BookStore*) are mapped to UML associations with corresponding roles (showing the navigability direction, and matching the property names and cardinality.) The CML bidirectional association *CustomerOrder* (comprised by two CML properties: *Customer.orders* and *Order.customer*) is mapped to a UML association with bidirectional navigability (that is, no direction arrows.) As demonstrated by this example, CML strives to enable modeling at the same conceptual level as allowed by UML. That being said, when compared to the UML metamodel, the CML metamodel supports only a core set of its elements, as shown in subsection 3.3.

Besides the structural elements of a conceptual model (as seen above), CML also has expressions that can set initial values to attributes, and define derived properties for both attributes and associations. CML expressions are partially based on the OCL [OMG 2014b] syntax, but they follow closely the OCL semantics. For example, the following CML expression (extracted from figure 1) is a path-based navigation expression

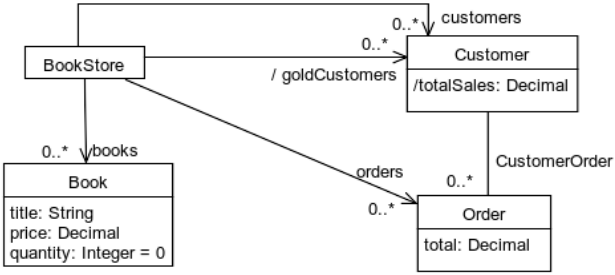


**Figure 2. The UML class diagram [OMG 2015] for the CML model listed in figure 1.**

borrowed from OCL:

```
/orderedBooks = orders.items.book;
```

Using association properties, the expression above navigates from one instance of *BookStore*, passing through all linked *orders*, and then through all *items* of all *orders*, in order to return all books that have been ordered. As another example, the following CML expression (also extracted from figure 1) does not follow the OCL syntax:

```
/goldCustomers = customers | select: totalSales > 1000;
```

However, the expression above closely matches the semantics of the following OCL expression:

```
derive: customers->select(totalSales > 1000)
```

Both the CML expression and the OCL excerpt above evaluate to a set of *Customer* instances that have bought more than 1000 in the *BookStore*.

The OCL syntax for expressions that process collections of instances has the following general form:

```
collection->method_name(predicate or function)
```

The expression above is based on method invocations (an influence from UML's object-oriented paradigm), and thus it has an imperative style. CML, on the other hand, intends to be agnostic towards programming paradigms. By using extensible comprehensions [Trinder 1992] to define derived attributes and associations, CML's syntax is more declarative, similar to SQL [ISO 2016] or C#'s LINQ [Torgersen 2007]. In CML, smaller expressions can also be combined into larger ones. For example:

```
/goldOrders = for order in bookStore.orders,
                  goldCustomer in bookStore.goldCustomers
                  | select: order.customer == goldCustomer | yield: order
```

Above, all *orders* from *goldCustomers* are returned. The sub-expressions are evaluated sequentially: the *for* expression provides a cross join of all (*order*, *goldCustomer*) pairs; the *select* expression selects only the pairs that have matching customers; Finally, the *yield* expression maps selected pairs into a sequence of *orders*. Sub-expressions like *for*, *select* and *yield* can be combined in different configurations in order to derive any required attributes and associations.

### 3.3. The CML Metamodel (Abstract Syntax)

In the article *UML and OCL in Conceptual Modeling*, Gogolla [Gogolla and Thalheim 2011] shows, by mapping the UML [OMG 2015] metamodel to the ER [Chen 2011] metamodel, how UML models (augmented by OCL [OMG 2014b] constraints) can be used to specify conceptual models. Also, Wazlawick [Wazlawick 2014] systematically prescribes a method for conceptual modeling using UML and OCL. Since one key CML goal is enabling the specification of conceptual models (such as those specified by ER models and UML/OCL models), in order to present the key elements of the CML metamodel, a similar approach to Gogolla's is used to map the CML metamodel to the ER metamodel, and to the UML/OCL metamodel.

The EMOF [OMG 2016] model presented by figure 3 is a simplified version of the CML metamodel. As shown, a *Concept* is composed of zero-or-more *Property* instances. Each *Property* must have a *Type* and an optional *Expression*. If two *Property* instances represent both ends of the same bidirectional association, there must be an *Association* instance that binds them. Unidirectional associations are only represented by a single *Property* instance (actually representing the association role) that enables the navigation from the source *Concept* instance to the target one, which is represented by the property's *Type*.

Next, there is a description for the key metamodel elements:

- *Concept*: According to Wazlawick [Wazlawick 2014], a concept represents complex information that has a coherent meaning in the domain. They aggregate attributes and cannot be described as primitive values. They may also be associated with other concepts. On the ER metamodel, it is known as *Entity Type*; on the UML metamodel, as *Class*. CML's *Concept* differs, however, from the UML *Class*, because it has only *Property* instances, while the UML *Class* may also have *Operation* instances.
- *Property*: May hold values of primitive types, in which case they represent an attribute on the *ER* and *UML* metamodels; or may hold references (or collections of references) linking to instances of other concepts. On the ER metamodel, a set of all reference pairs linking one *Entity Type* to another is known as a *Relationship*; on the UML metamodel, it is known as a unidirectional *Association*.
- *Association*: Unlike the ER and UML metamodels, in the CML metamodel, only a bidirectional *Association* is represented with the *Association* class. Using UML terminology, they bind the reference (non-primitive) properties, so that the *Association* links are accessible from each participating association end. It directly represents in the CML metamodel what normally requires additional implementation in programming languages. It is inspired[2] on the work of Cardoso [Cardoso 2011],

[2]The syntax used in CML resembles the syntax of a *struct* in C [ISO 2011], while Cardoso

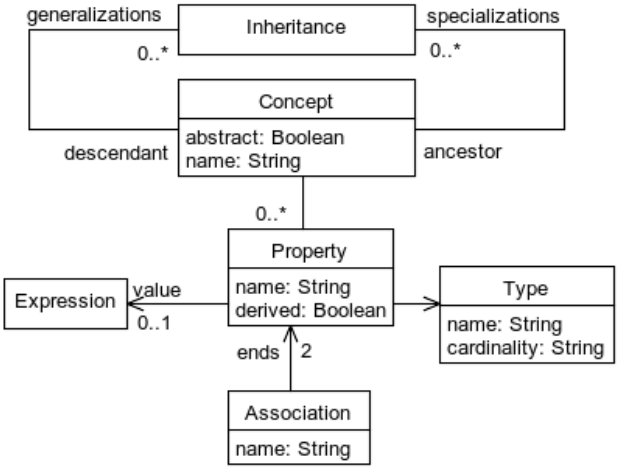


**Figure 3. Simplified EMOF [OMG 2016] model defining the CML metamodel.**

which extends the C# language to represent bidirectional associations; it is also inspired on the work of Balzer et. al. [Balzer et al. 2007], which uses *member interposition* to model relationships.

- *Type*: They may be of: a primitive type (such as Boolean, String, and Decimal); a reference to a *Concept* instance (*cardinality* equal to one); a sequence of references (*cardinality* equal to zero-or-many); or optional, meaning their value may or may not have been set (also defined by the *cardinality* property). CML also supports the tuple and lambda types, which are used in expressions.
- *Inheritance*: Following the the UML [OMG 2015] metamodel, CML provides the generalization/specialization relationship. In CML, a *Concept* may be a specialization of two or more other *Concept* instances. If a *Property* has been defined by more than one generalization, CML requires it to be redefined by the specialization in order to resolve the definition conflict.

## 4. The Extensible Compiler

In order to realize the CMP [Embley et al. 2011] manifesto's vision, the CML compiler can generate code in any target language if the corresponding templates are provided. A set of core templates is provided by CML compiler's base module, which is currently supporting Java and Python. In order to target specific technologies or platforms, third-party modules can also provide their own templates, along with their conceptual models. Developers can also extend existing templates in order to adapt the implementation to characteristics specific to their projects.

[Cardoso 2011] uses a verbose syntax. Also, unlike CML, Cardoso does *not* bind properties that represent each association end; instead, associations – unidirectional or bidirectional – are declared independently of class properties.

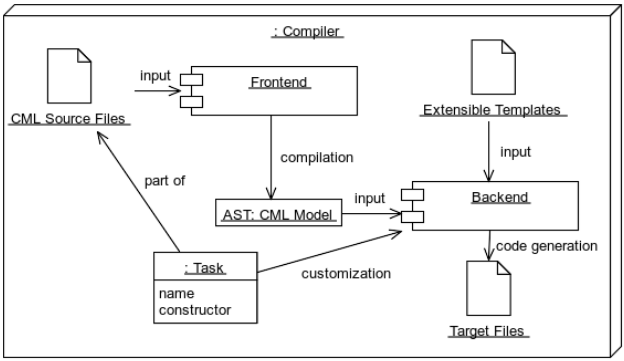


**Figure 4. An architectural overview of the CML compiler.**

Subsection 4.1 provides an overview of the CML compiler's architecture. Next, subsection 4.2 introduces the CML compiler's extensible templates. Finally, subsection 4.3 lays out the CML compiler's mechanism for organizing and sharing conceptual models and extensible templates.

### 4.1. Compiler Overview

An overview diagram of the architecture is shown in figure 4. The two main components of the compiler, and the artifacts they work with, are presented below:

- *Frontend:* receives as input the *CML source files*. It parses the files into an internal representation of the *CML model*. Syntactical and semantic validations are then executed. Any errors are presented to the developer, interrupting the progress to the next phase. If the *source files* are parsed and validated successfully, the *CML model* serves then as the input to the *backend* component.
- *Backend:* receives the *CML model* as input. Based on the *constructor* defined by a *task*, the *backend* chooses which *extensible templates* to use for code generation. The *target files* are then generated to be consumed by other tools. The *task* and associated *constructor* play a key role in determining the kind of *target* to be generated.

### 4.2. Extensible Templates

Parr has formalized and developed the StringTemplate [Parr 2004] language for code generation. CML's extensible templates are implemented in StringTemplate. The CML compiler uses StringTemplate for two purposes:

- *File names and directory structure:* each type of target generated by the CML compiler requires a different directory structure. The CML compiler expects each *constructor* to define a template file named "files.stg" (also known as *files template*), which will contain the path of all files to be generated. The *files template* may use information provided by the *task* (introduced in subsection 4.1) in order to determine the file/directory names. An example of a *files template* is shown below:

```
model_files(task, model) ::= <<
pom_file|pom.xml
>>

concept_files(task, concept) ::= <<
concept_file|src/main/java/<task.packagePath>/<concept.name; format="pascal-case">.java
>>
```

- *File content generation:* each file listed under the *files template* must have a corresponding *content template* that specifies how the file's content must be generated. The *content template* will receive as input one root-level element of the CML model, which will provide information to generate the file's content. The type of model element received as input by the *content template* depends on which function of the *files template* has defined the file to be generated. A typical *content template* is shown below:

```
import "/design/poj.stg"

concept_file(task, concept) ::= <<
package <task.packageName>;

import java.util.*;

public <class(concept)>
>>
```

On the *files template* example, two types of files are created by this *constructor*: one file for the CML module (named "pom.xml", and based on the "pom_file" template); and one for each concept found in the CML model (with the file extension ".java", and based on the "concept_file" template.)

On the *content template* example, the "concept_file" content template is displayed, which can generate a *Data Type Object* (DTO) class in Java. The actual template that knows how to generate the class is imported from "/design/poj.stg".

### 4.3. Modules and Libraries

When developing a single application, having a single directory to maintain all the CML source code is sufficient. But, once more than one application is developed as part of a larger project, and CML model elements are shared among them, it is necessary to separate the common source code. Also, some applications cover different domains, and it may be beneficial to separate the source code into different CML models.

In order to allow that, CML supports *modules*. Grouping a set of CML model elements, a module in CML is conceptually similar to a UML [OMG 2015] package. Physically, each module is a directory containing the following sub-directories:

- *source*: where the CML source files reside.
- *templates*: optional directory containing templates for code generation.
- *tests*: optional directory containing tests that verify the generated code.
- *targets*: created by the CML compiler to contain each target sub-directory, which in turn contains the target files generated for a given target.

Under the *source* directory, the module is defined by a *module specification*. If a module needs to reference CML model elements in other modules, then import statements define the name of the other modules. The CML compiler should then compile the imported modules before compiling the current module.

A CML module has no version as it is maintained in the same code repository with the other modules it imports. However, it is planned that a future version of CML will allow packaging a module as a library, which will have a version and the same name as the module. Such a library will in turn be published into a public (or company-wide) *library site* in order to be shared with other developers. A CML library is expected to become a packaged, read-only, versioned module.

## 5. Related Work

This section compares CML to other languages, tools and frameworks that can also generate code from conceptual models. Each paragraph covers a different category, enumerating specific solutions and characterizing their relevance to CML, and also their differences.

When compared to CML, the text-based languages are the most relevant. MPS [Voelter 2014] is a development environment for DSLs. Strictly speaking, its DSLs are not textual, since their AST is directly edited on projectional editors. However, the editors allow textual representations. Unlike MPS, the DSLs created with the M language [Brunelière et al. 2010] are truly textual. It was part of the discontinued Oslo project from Microsoft, which incorporated into Visual Studio similar capabilities to what is available on MPS. Xtext/Xtend [Bettini 2016] allows the definition of textual DSLs to generate code from conceptual models edited on Eclipse. It is similar to the Oslo project from Microsoft, and based on EMF [Steinberg et al. 2008]. MM-DSL [Visic and Karagiannis 2014], on the other hand, allows the definition of metamodels (abstract syntax; not the actual DSLs), which serve as input to generate domain-specific modeling tools. ThingML [Harrand et al. 2016] is also a language and code generation framework for the development of software in embedded devices. XML may also be used for conceptual modeling, and XSLT then used to create the templates for code generation, as shown by Gheraibia et all [Gheraibia and Bourouis 2012]. Observe that most of the solutions previously mentioned enable modeling via DSLs, while CML is a generic language for modeling in any domain.

Graphical languages also have some relevance to CML, despite the latter being a textual language, because the former have also been used to generate code in other target languages. MPS [Voelter 2014], besides the textual models, also allows the creation of graphical models. FCML [Karagiannis et al. 2016], on other hand, incorporates and extends conceptual modeling languages (ER, UML, and BPMN) via the OMNILab tool in order to generate code. MetaEdit+ [Tolvanen 2004] is another development environment that allows the creation of modeling tools and code generators for visual DSLs. As mentioned previously in this article, MDA [OMG 2014a] is the initiative from OMG to use UML [OMG 2015] for model-driven development. IFML [Brambilla et al. 2014] is an example from OMG of a high-level language that can be used to generate user interfaces on different platforms, such as the Web, or on mobile devices. The major drawback of graphical languages, as covered in section 2, is their difficulty to integrate seamlessly with the text-based, compiler-based workflow of software developers.

Frameworks also allow code generation from conceptual models, but lack a modeling language – graphical or textual. EMF [Steinberg et al. 2008] is a classical example, where modeling is done via editors on Eclipse or via a programming interface, and the

models are stored in the ECORE/XML format. Frameworks may also be used as the infrastructure of modeling languages. EMF, for example, is the framework supporting Xtext [Bettini 2016]. Conceivably, other modeling languages may also target EMF. In fact, CML's extensible compiler allows the implementation of templates that target EMF.

As seen in previous sections, the CML compiler uses StringTemplate [Parr 2004] as the language for its code generation templates. There are other template languages designed for code generation from conceptual models. Xpand [Greifenberg et al. 2016] allows the definition of templates with multiple variability regions. EGL [Rose et al. 2008b] is another language that allows code generation from models. MOFScript [Oldevik et al. 2005] allows code generation from models defined by any type of metamodel. JET [van Emde Boas, Ghica 2004] allows code generation from EMF [Steinberg et al. 2008] models. One strength of StringTemplate is its extensibility mechanisms. It is possible to define a core set of templates that define patterns, and then extend them with the specifics of each target language or technology. It is also possible to share templates as libraries, which can be further extended for specific purposes by third-parties. Xpand also allows this level of extensibility.

Just like CML, there are programming languages that provide the ability to declare bidirectional associations. DSM [Balzer et al. 2007] is an object-oriented programming language with support for associations. Fibonacci [Albano et al. 1993] is programming language for object-oriented databases that allows the modeling of association roles. ASSOCIATION# [Cardoso 2011], on the other hand, is an extension to C# that allows the modeling of associations. Likewise, RelJ [Bierman and Wren 2005] is a Java extension with support for associations. One key drawback of these languages is the fact that their conceptual models cannot be reused to generate code in any other language or technology; they are, for all intents and purposes, the target language.

There are also other conceptual languages whose original focus has not been to support code generation or implementation, but to serve solely as modeling artifacts. Languages, such as OWL [W3C 2012] and Telos [Mylopoulos et al. 1990], have been designed as ontology metamodels to support the representation and storage of knowledge, and to allow automated reasoning from knowledgebases; OWL being the *lingua franca* of the semantic web, while Telos has been created to store ontologies in a object-oriented database. Other languages, like UML [OMG 2015] and ER [Chen 2011], have been originally intended as tools to support the analysis and design of software systems, and only later have been repurposed for model-driven software development. The relevance of these languages to CML comes from the expressivity power their metamodels provide for conceptual modeling. For that reason, CML should continue to expand its capabilities by borrowing features from these languages.

## 6. Conclusion

The CML language and compiler make it possible to specify, in a single high-level language, the concepts of ever-changing, increasingly distributed software systems. As opposed to modeling concepts, their properties and associations in each target language, from a single CML model, the CML extensible templates generate code that keeps the implementations (across the different platforms and technologies) consistent with the specification. Also, as the technology landscape evolves, these textual CML models can be

reused to generate code in new target languages and technologies.

The initial version of CML has been designed to validate this textual, model-driven approach of software development. The use of the CML compiler to model and implement CML's own metamodel has shown to reduce the amount of manually written code, and made the metamodel more readable, maintainable and reusable. Other applications of CML are needed in order to provide qualitative evidence that CML can indeed be used as a single source to implement multiple targets. Quantitative cost-benefit analysis (based on the implementation effort of hand-written vs generated lines-of-code, perhaps using a method adapted from the work of Gaffney et al [Gaffney and Cruickshank 1992]) may also provide data that shows whether the investment – made on the development of CML models – pays off. The data collected, together with the feedback provided by software developers, should then inform the iterative evolution of CML features.

## References

Albano, A., Bergamini, R., Ghelli, G., and Orsini, R. (1993). An object data model with roles. In *VLDB*, volume 93, pages 39–51.

Balzer, S., Gross, T. R., and Eugster, P. (2007). A Relational Model of Object Collaborations and Its Use in Reasoning About Relationships. In Ernst, E., editor, *21st ECOOP. Proceedings*, pages 323–346. Springer.

Bettini, L. (2016). *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd.

Bierman, G. and Wren, A. (2005). First-Class Relationships in an Object-Oriented Language. In Black, A. P., editor, *19th ECOOP. Proceedings*, pages 262–286. Springer.

Brambilla, M., Mauri, A., and Umuhoza, E. (2014). Extending the Interaction Flow Modeling Language (IFML) for Model Driven Development of Mobile Applications Front End. In Awan, I., editor, *MobiWIS 11th. Proceedings*, pages 176–191. Springer.

Brunelière, H., Cabot, J., Clasen, C., Jouault, F., and Bézivin, J. (2010). Towards Model Driven Tool Interoperability. In Kühne, T., editor, *ECMFA 6th. Proceedings*, pages 32–47. Springer.

Cardoso, I. S. (2011). Inserindo suporte a declaração de associações da UML 2 em uma linguagem de programação orientada a objetos. Master's thesis, Universidade Federal de Santa Catarina.

Chen, P. P.-S. (2011). The Entity-Relationship Model (Reprinted Historic Data). In Embley, D. W., editor, *Handbook of Conceptual Modeling*, pages 57–84. Springer.

dos Santos, Q. C. M. (2017). CML Project.

Embley, D. W., Liddle, S. W., and Pastor, O. (2011). Conceptual-Model Programming: A Manifesto. In Embley, D. W., editor, *Handbook of Conceptual Modeling*, pages 3–16. Springer.

Gaffney, Jr., J. E. and Cruickshank, R. D. (1992). A General Economics Model of Software Reuse. In *Proceedings of the 14th International Conference on Software Engineering*, ICSE '92, pages 327–337. ACM, New York, NY, USA.

Gheraibia, Y. and Bourouis, A. (2012). Ontology and automatic code generation on modeling and simulation. In *6th SETIT. Proceedings*, pages 69–73.

Gogolla, M. and Thalheim, B. (2011). UML and OCL in Conceptual Modeling. In Embley, D. W., editor, *Handbook of Conceptual Modeling*, pages 85–122. Springer.

Greifenberg, T., Müller, K., Roth, A., Rumpe, B., Schulze, C., and Wortmann, A. (2016). Modeling Variability in Template-based Code Generators for Product Line Engineering. *CoRR*, abs/1606.02903.

Harrand, N., Fleurey, F., Morin, B., and Husa, K. E. (2016). Thingml: A language and code generation framework for heterogeneous targets. In *Proceedings of the ACM/IEEE 19th MODELS*, pages 125–135.

ISO (2011). *ISO/IEC 9899:2011 Programming languages — C*. International Organization for Standardization.

ISO (2016). IEC 9075-1: 2003 (E) Database languages — SQL Part 1: Framework (SQL/Framework).

Jensen, K. and Wirth, N. (1974). *PASCAL User Manual and Report*. Springer-Verlag.

Karagiannis, D., Buchmann, R. A., Burzynski, P., Reimer, U., and Walch, M. (2016). In *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, pages 3–30. Springer.

Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M. (1990). Telos: Representing Knowledge About Information Systems. *ACM TIS*, 8(4):325–362.

Oldevik, J., Neple, T., Grønmo, R., Aagedal, J., and Berre, A.-J. (2005). Toward Standardised Model to Text Transformations. In Hartman, A., editor, *ECMDA-FA. Proceedings*, pages 239–253. Springer.

OMG (2014a). Model Driven Architecture (MDA) Guide rev. 2.0.

OMG (2014b). Object Constraint Language (OCL), Version 2.4.

OMG (2015). Unified Modeling Language (UML), Superstructure, Version 2.5.

OMG (2016). Meta Object Facility (MOF) Core Specification, Version 2.5.1.

Parr, T. J. (2004). Enforcing Strict Model-view Separation in Template Engines. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 224–233. ACM, New York, NY, USA.

Rose, L. M., Paige, R. F., Kolovos, D. S., and Polack, F. A. C. (2008a). Constructing Models with the Human-Usable Textual Notation. In Czarnecki, K., editor, *MoDELS: 11th International Conference. Proceedings*, pages 249–263. Springer.

Rose, L. M., Paige, R. F., Kolovos, D. S., and Polack, F. A. C. (2008b). The Epsilon Generation Language. In Schieferdecker, I., editor, *ECMDA-FA: 4th European Conference. Proceedings*, pages 1–16. Springer.

Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. (2008). *EMF: eclipse modeling framework*. Pearson Education.

Thalheim, B. (2011). The Theory of Conceptual Models, the Theory of Conceptual Modelling and Foundations of Conceptual Modelling. In Embley, D. W., editor, *Handbook of Conceptual Modeling*, pages 543–577. Springer.

Tolvanen, J.-P. (2004). MetaEdit+: Domain-specific Modeling for Full Code Generation Demonstrated. In *Companion to the 19th Annual ACM SIGPLAN Conference*, OOPSLA '04, pages 39–40, New York, NY, USA. ACM.

Torgersen, M. (2007). Querying in C#: How Language Integrated Query (LINQ) Works. In *Companion to the 22Nd ACM SIGPLAN Conference*, OOPSLA '07, pages 852–853, New York, NY, USA. ACM.

Trinder, P. (1992). Comprehensions, a query notation for dbpls. In *Proceedings of DBPL3*, pages 55–68, San Francisco, USA. Morgan Kaufmann Pub. Inc.

van Emde Boas, Ghica (2004). Template programming for model-driven code generation. In *19th Annual ACM SIGPLAN Conference*.

Visic, N. and Karagiannis, D. (2014). Developing Conceptual Modeling Tools Using a DSL. In Buchmann, R., editor, *KSEM: 7th International Conference. Proceedings*, pages 162–173. Springer.

Voelter, M. (2014). *Generic Tools, Specific Languages*. PhD thesis, Delft University of Technology.

W3C (2012). OWL 2 Structural Specification and Functional-Style Syntax (Second Edition).

Wazlawick, R. S. (2014). *Object-Oriented Analysis and Design for Information Systems*. Morgan Kaufmann.

# ANEXO C – Código-Fonte do Compilador

```
==> cml-compiler/cml-bootstrapping/cml_base/README.txt

This directory is here just to ensure it is not possible to replace the cml_base module installed with the CML compiler.

If the compilation of the cml_language module fails because of the cml_base module,
it is probably because it is mistakenly trying to pick it up from this location,
as opposed to finding it in CML_MODULES_PATH.

==> cml-compiler/cml-bootstrapping/cml_language/source/associations.cml


@concept ASSOCIATION: NAMED_ELEMENT, SCOPE
{
    /association_ends = members as? ASSOCIATION_END*;

    /first_end = first(association_ends);
    /last_end = last(association_ends);

    /property_types = association_ends.associated_property.type;
    /reversed_property_types = reverse(property_types);

    /one_to_one = association_ends | all: one;
    /one_to_many = first_one_to_last_many or first_many_to_last_one;

    /first_one_to_last_many = first_end.one and last_end.many;
    /first_many_to_last_one = first_end.many and last_end.one;

    /one_property = if first_one_to_last_many then first_end.associated_property else last_end.associated_property;
    /many_property = if first_many_to_last_one then first_end.associated_property else last_end.associated_property;

    /diagnostic_id = "association " & name;
}

@concept ASSOCIATION_END: MODEL_ELEMENT
{
    concept_name: string;
    property_name: string;
    property_type: TYPE?;

    associated_concept: CONCEPT?;
    associated_property: PROPERTY?;

    /association = parent as? ASSOCIATION?;

    /associated_property_type = associated_property.type;

    /one = associated_property_type | exists: single;
    /many = associated_property_type | exists: sequence;

    /diagnostic_id = "association end " & concept_name & "." & property_name &
                     if present(property_type) then ": " & property_type.diagnostic_id as! string else "";
}

@association ASSOCIATION_END_PROPERTY
{
    ASSOCIATION_END.associated_property;
    PROPERTY.association_end;
}


==> cml-compiler/cml-bootstrapping/cml_language/source/cmlc_tasks.cml


@task cml_language_java: cmlc_java
{
    groupId = "cml";
    artifactId = "cml-language-generated";
    artifactVersion = "master-SNAPSHOT";
    packageName = "cml.language.generated";
    packagePath = "cml/language/generated";
    classTypeNamePrefix = "";
}

@task cml_language_py: cmlc_py
{
    moduleName = "cml_language";
    moduleVersion = "master-SNAPSHOT";
    classTypeNamePrefix = "CML";
}


==> cml-compiler/cml-bootstrapping/cml_language/source/concepts.cml


@abstraction CONCEPT: NAMED_ELEMENT, PROPERTY_LIST
{
    abstraction: BOOLEAN;

    generalizations: INHERITANCE*;
    specializations: INHERITANCE*;

    /ancestors = generalizations.ancestor;
    /descendants = specializations.descendant;

    /all_ancestors: CONCEPT* = distinct(concat(inherited_ancestors, ancestors));
    /all_properties: PROPERTY* = concat(properties, inherited_non_redefined_properties);

    /inherited_ancestors = distinct(ancestors.all_ancestors);
    /inherited_properties = distinct(ancestors.all_properties);
    /inherited_abstract_properties = inherited_properties | select: abstract;
    /inherited_non_redefined_properties = for inherited in inherited_properties
                                          | select: (for p in self.properties | none: p.name == inherited.name);

    /super_properties = inherited_non_redefined_properties | select: not derived;
    /non_derived_properties = properties
        | select: not derived
        | sort: p1, p2 ->
            if present(p1) and empty(p2) then +1
            else if empty(p1) and present(p2) then -1
            else 0;
    /invocation_properties = concat(super_properties, non_derived_properties);

    /slot_properties = properties | select: slot;
    /derived_properties = properties | select: derived;
    /association_properties = properties | select: present(association);

    /init_properties = all_properties | select: init;
    /non_init_properties = all_properties | select: non_init;
    /printable_properties = all_properties | select: printable;

    /property_types = all_properties.type;
    /property_concepts =
      property_types
        | select: not primitive
        | select: t -> present(concept_of(model, t))
```

```
        | collect: t -> concept_of(model, t) as! CONCEPT
        | distinct;

    /associations = model.associations | select: (association_ends | exists: associated_concept === self);

/*
    /dependencies = distinct(concat(inherited_dependencies, property_dependencies));
    /property_dependencies = distinct(concat(
                                transitive_property_concepts | collect: inherited_dependencies,
                                transitive_property_concepts | collect: name));
    /inherited_dependencies = all_ancestors | select: name | distinct;

    /transitive_property_concepts = self | recurse_depth: property_concepts;
    /property_concepts = property_types | reject: primitive | yield: concept | distinct;

    /redefined_ancestors: CONCEPT_REDEF*;
    /redefined_inherited_concrete_properties: PROPERTY*;
*/

    /diagnostic_id = "concept " & name;
}

@concept INHERITANCE
{
    ancestor: CONCEPT;
    descendant: CONCEPT;
}

@association GENERALIZATION
{
    INHERITANCE.descendant;
    CONCEPT.generalizations;
}

@association SPECIALIZATION
{
    INHERITANCE.ancestor;
    CONCEPT.specializations;
}

@concept CONCEPT_REDEF
{
    concept: CONCEPT;
    property_redefs: PROPERTY_REDEF*;
}

@concept PROPERTY_REDEF
{
    prop: PROPERTY; // shortened name because Python gets confused with a property called "property".
    redefined: BOOLEAN;
}


==> cml-compiler/cml-bootstrapping/cml_language/source/concepts_poset.cml


@function intersection(sa: CONCEPT*, sb: CONCEPT*) = sa | select: a -> (sb | exists: b -> b === a);

@function super_list(c: CONCEPT) = concat(c, c.all_ancestors);

@function sub(a: CONCEPT, b: CONCEPT) = (a === b) or (a.all_ancestors | includes: b);

@function least(s: CONCEPT*) = for a in s | select: (for b in s | all: sub(a, b));

@function least_upper_bound(a: CONCEPT, b: CONCEPT) = least(intersection(super_list(a), super_list(b)));


==> cml-compiler/cml-bootstrapping/cml_language/source/elements.cml


@abstraction ELEMENT
{
    /diagnostic_id: string;
}

@abstraction MODEL_ELEMENT: ELEMENT
{
    parent: SCOPE?;
    location: LOCATION?;

    /model: MODEL? = parent.model;
    /module: MODULE? = parent.module;
}

@abstraction NAMED_ELEMENT: MODEL_ELEMENT
{
    name: string;
}

@abstraction TYPED_ELEMENT: NAMED_ELEMENT
{
    /type: TYPE;
}

@abstraction SCOPE: MODEL_ELEMENT
{
    members: MODEL_ELEMENT*;
}

@concept LOCATION
{
    line: integer;
    column: integer;
    element: MODEL_ELEMENT;
}

@association MEMBERSHIP
{
    MODEL_ELEMENT.parent;
    SCOPE.members;
}

@association LOCALIZATION
{
    LOCATION.element;
    MODEL_ELEMENT.location;
}


==> cml-compiler/cml-bootstrapping/cml_language/source/expressions.cml


@abstraction EXPRESSION: SCOPE
{
  /kind: string;
  /type: TYPE;
```

```
  /operands = members as? EXPRESSION*;

  /matching_result_type = type;

  /boolean = type.boolean;
  /numeric = type.numeric;
  /float = type.float;

  /arithmetic = numeric or float;
  /relational = type.relational;
  /referential = type.referential;
  /primitive = type.primitive;
}

@concept LITERAL: EXPRESSION
{
  text: string;
  type: TYPE;

  /kind = "literal";
  /diagnostic_id = text;
}

@concept LET: EXPRESSION
{
  variable: string;

  /kind = "let";

  /variableExpr = first(operands) as! EXPRESSION;
  /resultExpr = last(operands) as! EXPRESSION;

  /type = resultExpr.type;

  /diagnostic_id =
    "let " & variable & " = " & variableExpr.diagnostic_id &
    " in " & resultExpr.diagnostic_id;
}

@abstraction INFIX: EXPRESSION
{
  operator: string;

  /operation: string?;

  /left = first(operands) as! EXPRESSION;
  /right = last(operands) as! EXPRESSION;

  /diagnostic_id = left.diagnostic_id & " " & operator & " " & right.diagnostic_id;

  /boolean_type = VALUE_TYPE("", primitive_type_name("boolean"));
  /undefined_type = UNDEFINED_TYPE("Incompatible operand(s) for operator '" & operator & "':\n" &
                                   "- left operand is '" & left.diagnostic_id & ": " & left.type.diagnostic_id & "'\n" &
                                   "- right operand is '" & right.diagnostic_id & ": " & right.type.diagnostic_id & "'");

  /type =
    if left.type.undefined then left.type
    else if right.type.undefined then right.type
    else inferred_type;

  /inferred_type: TYPE;
}

@concept ARITHMETIC: INFIX
{
  /kind = "arithmetic";

  /operation =
    if operator == "+" then "add" as! string?
    else if operator == "-" then "sub" as! string?
    else if operator == "*" then "mul" as! string?
    else if operator == "/" then "div" as! string?
    else if operator == "%" then "mod" as! string?
    else if operator == "^" then "exp" as! string? // type-casting necessary until conditionals are fixed to support "none"
    else none;

  /inferred_type =
    if left.arithmetic and right.arithmetic then
    (
      if subtype(left.type.name, right.type.name) then right.type
      else if subtype(right.type.name, left.type.name) then left.type
      else undefined_type
    )
    else undefined_type;
}

@concept LOGICAL: INFIX
{
  /kind = "logical";

  /operation = operator as? string?;

  /inferred_type = if left.boolean and right.boolean then boolean_type else undefined_type;
}

@concept RELATIONAL: INFIX
{
  /kind = "relational";

  /operation =
    if operator == "==" then "eq" as! string?
    else if operator == "!=" then "not_eq" as! string?
    else if operator == ">" then "gt" as! string?
    else if operator == ">=" then "gte" as! string?
    else if operator == "<" then "lt" as! string?
    else if operator == "<=" then "lte" as! string? // type-casting necessary until conditionals are fixed to support "none"
    else none;

  /inferred_type = if left.relational and right.relational then boolean_type else undefined_type;
}

@concept REFERENTIAL: INFIX
{
  /kind = "referential";

  /operation =
    if operator == "===" then "ref_eq" as! string?
    else if operator == "!==" then "not_ref_eq" as! string? // type-casting necessary until conditionals are fixed to support "none"
    else none;

  /inferred_type = if left.referential and right.referential then boolean_type else undefined_type;
}

@concept STRING_CONCAT: INFIX
{
  /kind = "string_concat";
```

```
  /operation =
    if operator == "&" then kind as! string? // type-casting necessary until conditionals are fixed to support "none"
    else none;

  /string_type = VALUE_TYPE("", primitive_type_name("string"));

  /inferred_type = if left.primitive and right.primitive then string_type else undefined_type;
}

@abstraction CONDITIONAL: EXPRESSION
{
  /kind = "conditional";

  /condExpr = first(operands) as! EXPRESSION;
  /thenExpr = first(operands | select: o -> o !== condExpr and o !== elseExpr) as! EXPRESSION;
  /elseExpr = last(operands) as! EXPRESSION;

/*
  /thenType = thenEpr.type;
  /elseType = elseExpr.type;

  /type =
    if thenType.primitive and elseType.primitive then
    (
      if subtype(thenType.name, elseType.name) then thenType
      else if subtype(elseType.name, thenType.name) then elseType
      else undefined_type
    )
    else if present(thenType.concept) and present(elseType.concept) then
    (
      if count(concept_lub) == 1 then first(concept_lub) as! TYPE
    )

  /concept_lub = lower_upper_bound(concept_of(model, thenType), concept_of(model, elseType));

  /undefined_type = UNDEFINED_TYPE(thenType.diagnostic_id & "|" & elseType.diagnostic_id);
*/
  /diagnostic_id = "if " & condExpr.diagnostic_id & " then " & thenExpr.diagnostic_id & " else " & elseExpr.diagnostic_id;
}
```

==> cml-compiler/cml-bootstrapping/cml_language/source/models.cml

```
@concept MODEL: NAMED_ELEMENT, SCOPE
{
    /name = "model";

    /model = self as? MODEL?;
    /module = first(modules);

    /modules = members as? MODULE*;
    /tasks = modules.tasks;
    /concepts = modules.concepts;
    /associations = modules.associations;

/*

    /templates = modules.templates;

    /ordered_concepts = concepts | sort: c1, c2 ->
        if (c1.dependencies | includes: c2.name) and (c2.dependencies | includes: c1.name) then (
            // If concepts depend on each other, the more general one is listed first:
            if c1.inherited_dependencies | includes: c2.name then +1
            else if c2.inherited_dependencies | includes c1.name then -1
            else 0
        )
        else if c1.dependencies | includes: c2.name then +1
        else if c2.dependencies | includes: c1.name then -1
        else 0;
*/

    /diagnostic_id = name;
}

@function concept_of(m: MODEL?, t: TYPE) = m.concepts | select: c -> c.name == t.name | first;
```

==> cml-compiler/cml-bootstrapping/cml_language/source/modules.cml

```
@concept MODULE: NAMED_ELEMENT, SCOPE
{
    /module = self as? MODULE?;

    /imports = members as? IMPORT*;
    /tasks = members as? TASK*;
    /concepts = members as? CONCEPT*;
    /associations = members as? ASSOCIATION*;

/*

    /templates = for m in members | select: m is TEMPLATE;

    /all_concepts = concat(concepts, imported_concepts);
    /all_templates = concat(templates, imported_templates);

    /imported_concepts = imported_modules | yield: concepts;
    /imported_templates = imported_modules | yield: templates;
    /imported_modules = imports | yield: imported_module;
*/

    /diagnostic_id = "module " & name;
}

@concept IMPORT: NAMED_ELEMENT
{
    imported_module: MODULE;
    first_import: boolean;

    /diagnostic_id = "import " & name;
}
```

==> cml-compiler/cml-bootstrapping/cml_language/source/primitives.cml

```
// Template functions on this file are implemented by the cml-primitives Maven module.

// Under cml.primitives.Types:
@function primitive_type_name(name: string) -> string;
@function boolean(name: string) -> boolean;
@function string(name: string) -> boolean;
@function numeric(name: string) -> boolean;
@function float(name: string) -> boolean;
@function subtype(s: string, t: string) -> boolean;
```

```
==> cml-compiler/cml-bootstrapping/cml_language/source/properties.cml


@concept PROPERTY: TYPED_ELEMENT, SCOPE
{
    derived: BOOLEAN;
    declared_type: TYPE?;
    value: EXPRESSION?;
    association_end: ASSOCIATION_END?;

    /concept = parent as? CONCEPT?;
    /association = association_end.association;

    /concrete = not abstract;
    /abstract = derived and empty(value);
    /init = present(value) and not derived;
    /non_init = empty(value) and not derived;
    /slot = not (derived or present(association_end));
    /printable = (slot and not type.sequence) or type.primitive;

    /type = if present(declared_type) then declared_type as! TYPE
            else if present(value) then value.type.inferred_type as! TYPE
            else UNDEFINED_TYPE("No type or expression defined for property: " & name);

    /named_parent = parent as? NAMED_ELEMENT?;

    /diagnostic_id = if present(named_parent)
                     then "property " & named_parent.name & "." & name & ": " & type.diagnostic_id
                     else "property " & name & ": " & type.diagnostic_id;
}

@abstraction PROPERTY_LIST: SCOPE
{
    /properties = members as? PROPERTY*;
}



==> cml-compiler/cml-bootstrapping/cml_language/source/tasks.cml


@concept TASK: NAMED_ELEMENT, PROPERTY_LIST
{
    constructor: CONSTRUCTOR?;

    /diagnostic_id = "task " & name & if present(constructor) then ": " & constructor.diagnostic_id else "";
}

@concept CONSTRUCTOR: ELEMENT
{
    name: string;

    /diagnostic_id = name;
}

==> cml-compiler/cml-bootstrapping/cml_language/source/types.cml


@abstraction TYPE: ELEMENT
{
  cardinality: string;

  /name = "";

  /min_cardinality = if cardinality == "?" or cardinality == "*" then 0 else 1;
  /max_cardinality = if cardinality == "*" or cardinality == "+" then 2140000000 else 1;

  /concept: CONCEPT? = none;
  /element_type: TYPE = self;

  /kind = if cardinality == "" then "required"
          else if cardinality == "?" then "optional"
          else if cardinality == "*" then "sequence"
          else "unknown";

  /matching_result_type = self;
  /base_type = self;

  /parameter = defined and not primitive and empty(concept);
  /defined = not undefined;
  /undefined = false;
  /something = defined and not nothing;
  /nothing = false;

  /boolean = false;
  /numeric = false;
  /float = false;
  /string = false;

  /primitive = false;
  /relational = false;
  /referential = present(concept) and not sequence;

  /single = (required or optional) and not sequence;
  /required = kind == "required";
  /optional = kind == "optional";
  /sequence = kind == "sequence";

  /inferred_cardinality = cardinality;
  /inferred_type: TYPE = self;

  /diagnostic_id = name & cardinality;
}
@concept VALUE_TYPE: TYPE
{
  name: string;

  /element_type: TYPE = VALUE_TYPE("", primitive_type_name(name));

  /boolean = boolean(name);
  /string = string(name);
  /numeric = numeric(name);
  /float = float(name);

  /primitive = boolean or numeric or float or string;
  /relational = numeric or float or string;
}
@concept REFERENCE_TYPE: TYPE
{
  name: string;

  /element_type = REFERENCE_TYPE("", name);
  /inferred_type = REFERENCE_TYPE(inferred_cardinality, name);
}
```

```
@concept UNDEFINED_TYPE: TYPE
{
  error_message: string;

  /name = "UNDEFINED";
  /cardinality = "";
  /undefined = true;
  /parent: SCOPE? = none;
  /location: LOCATION? = none;
}

==> cml-compiler/cml-bootstrapping/cml_language/templates/constructors/_java/dependencies.stg


dependencies() ::= <<
\<dependency>
    \<groupId>org.jetbrains\</groupId>
    \<artifactId>annotations\</artifactId>
    \<version>15.0\</version>
\</dependency>
\<dependency>
    \<groupId>org.jooq\</groupId>
    \<artifactId>jool\</artifactId>
    \<version>0.9.12\</version>
    \<scope>compile\</scope>
\</dependency>
\<dependency>
    \<groupId>cml\</groupId>
    \<artifactId>cml-primitives\</artifactId>
    \<version><task.artifactVersion>\</version>
\</dependency>
>>

==> cml-compiler/cml-bootstrapping/cml_language/templates/constructors/_java/pom_file.stg



import "/constructors/_java/dependencies.stg"
import "cml_base:/constructors/_java/pom_file.stg"


==> cml-compiler/cml-bootstrapping/cml_language/templates/constructors/cmlc_java/concept_file.stg


import "/lang/java/functions.stg"
import "cml_base:/constructors/cmlc_java/concept_file.stg"

==> cml-compiler/cml-bootstrapping/cml_language/templates/constructors/cmlc_java/files.stg


import "cml_base:/constructors/cmlc_java/files.stg"


==> cml-compiler/cml-bootstrapping/cml_language/templates/constructors/cmlc_py/files.stg


import "cml_base:/constructors/cmlc_py/files.stg"


==> cml-compiler/cml-bootstrapping/cml_language/templates/constructors/cmlc_py/init_file.stg


import "/lang/py/functions.stg"
import "cml_base:/constructors/cmlc_py/init_file.stg"

==> cml-compiler/cml-bootstrapping/cml_language/templates/lang/java/functions.stg


invocation_primitive_type_name(args) ::= <<
cml.primitives.Types.primitiveTypeName(<expression(args.name)>)
>>

invocation_boolean(args) ::= <<
cml.primitives.Types.boolean_(<expression(args.name)>)
>>

invocation_string(args) ::= <<
cml.primitives.Types.string(<expression(args.name)>)
>>

invocation_numeric(args) ::= <<
cml.primitives.Types.numeric(<expression(args.name)>)
>>

invocation_float(args) ::= <<
cml.primitives.Types.float_(<expression(args.name)>)
>>

invocation_subtype(args) ::= <<
cml.primitives.Types.subtype(<expression(args.s)>, <expression(args.t)>)
>>


==> cml-compiler/cml-bootstrapping/cml_language/templates/lang/py/functions.stg


invocation_primitive_type_name(args) ::= "\"\""
invocation_boolean(args) ::= "True"
invocation_string(args) ::= "True"
invocation_numeric(args) ::= "True"
invocation_float(args) ::= "True"
invocation_subtype(args) ::= "True"


==> cml-compiler/cml-bootstrapping/cml_language/tests/foundation/membership/expected/cml_language_java/cml-compiler-output.txt

model files:
- pom.xml

cml_language files:
- src/main/java/cml/language/generated/CmlLanguageFunctions.java

ASSOCIATION files:
- src/main/java/cml/language/generated/Association.java

ASSOCIATION_END files:
- src/main/java/cml/language/generated/AssociationEnd.java

CONCEPT files:
- src/main/java/cml/language/generated/Concept.java

INHERITANCE files:
- src/main/java/cml/language/generated/Inheritance.java

CONCEPT_REDEF files:
```

```
- src/main/java/cml/language/generated/ConceptRedef.java

PROPERTY_REDEF files:
- src/main/java/cml/language/generated/PropertyRedef.java

ELEMENT files:
- src/main/java/cml/language/generated/Element.java

MODEL_ELEMENT files:
- src/main/java/cml/language/generated/ModelElement.java

NAMED_ELEMENT files:
- src/main/java/cml/language/generated/NamedElement.java

TYPED_ELEMENT files:
- src/main/java/cml/language/generated/TypedElement.java

SCOPE files:
- src/main/java/cml/language/generated/Scope.java

LOCATION files:
- src/main/java/cml/language/generated/Location.java

EXPRESSION files:
- src/main/java/cml/language/generated/Expression.java

LITERAL files:
- src/main/java/cml/language/generated/Literal.java

LET files:
- src/main/java/cml/language/generated/Let.java

INFIX files:
- src/main/java/cml/language/generated/Infix.java

ARITHMETIC files:
- src/main/java/cml/language/generated/Arithmetic.java

LOGICAL files:
- src/main/java/cml/language/generated/Logical.java

RELATIONAL files:
- src/main/java/cml/language/generated/Relational.java

REFERENTIAL files:
- src/main/java/cml/language/generated/Referential.java

STRING_CONCAT files:
- src/main/java/cml/language/generated/StringConcat.java

CONDITIONAL files:
- src/main/java/cml/language/generated/Conditional.java

MODEL files:
- src/main/java/cml/language/generated/Model.java

MODULE files:
- src/main/java/cml/language/generated/Module.java

IMPORT files:
- src/main/java/cml/language/generated/Import.java

PROPERTY files:
- src/main/java/cml/language/generated/Property.java

PROPERTY_LIST files:
- src/main/java/cml/language/generated/PropertyList.java

TASK files:
- src/main/java/cml/language/generated/Task.java

CONSTRUCTOR files:
- src/main/java/cml/language/generated/Constructor.java

TYPE files:
- src/main/java/cml/language/generated/Type.java

VALUE_TYPE files:
- src/main/java/cml/language/generated/ValueType.java

REFERENCE_TYPE files:
- src/main/java/cml/language/generated/ReferenceType.java

UNDEFINED_TYPE files:
- src/main/java/cml/language/generated/UndefinedType.java

ASSOCIATION_END_PROPERTY files:
- src/main/java/cml/language/generated/AssociationEndProperty.java

GENERALIZATION files:
- src/main/java/cml/language/generated/Generalization.java

SPECIALIZATION files:
- src/main/java/cml/language/generated/Specialization.java

MEMBERSHIP files:
- src/main/java/cml/language/generated/Membership.java

LOCALIZATION files:
- src/main/java/cml/language/generated/Localization.java


==> cml-compiler/cml-bootstrapping/cml_language/tests/foundation/membership/expected/cml_language_java/ignored-list.txt

--ALL--

==> cml-compiler/cml-bootstrapping/pom.xml

<project
        xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>cml</groupId>
        <artifactId>cml-compiler</artifactId>
        <version>master-SNAPSHOT</version>
    </parent>

    <groupId>cml</groupId>
    <artifactId>cml-bootstrapping</artifactId>
    <version>master-SNAPSHOT</version>
    <packaging>pom</packaging>

    <properties>
```

```
        <cml_language.path>${project.basedir}/cml_language</cml_language.path>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-primitives</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.6.0</version>
                <executions>
                    <execution>
                        <id>cml_language_java</id>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>exec</goal>
                        </goals>
                        <configuration>
                            <executable>cml</executable>
                            <workingDirectory>${cml_language.path}</workingDirectory>
                            <arguments>
                                <argument>cml_language_java</argument>
                            </arguments>
                        </configuration>
                    </execution>
                    <execution>
                        <id>cml-language-versioned</id>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>exec</goal>
                        </goals>
                        <configuration>
                            <executable>mvn</executable>
                            <workingDirectory>${cml_language.path}/targets/cml_language_java</workingDirectory>
                            <arguments>
                                <argument>versions:set</argument>
                                <argument>-DnewVersion=${project.version}</argument>
                            </arguments>
                        </configuration>
                    </execution>
                    <execution>
                        <id>cml-language-generated</id>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>exec</goal>
                        </goals>
                        <configuration>
                            <executable>mvn</executable>
                            <workingDirectory>${cml_language.path}/targets/cml_language_java</workingDirectory>
                            <arguments>
                                <argument>clean</argument>
                                <argument>install</argument>
                            </arguments>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>


==> cml-compiler/cml-frontend/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <parent>
        <groupId>cml</groupId>
        <artifactId>cml-compiler</artifactId>
        <version>master-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>cml-frontend</artifactId>
    <version>master-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <cml.jar.name>cml-compiler</cml.jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-language</artifactId>
            <version>${project.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-generator</artifactId>
            <version>${project.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-integration</artifactId>
            <version>1.3</version>
        </dependency>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>21.0</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.maven.shared</groupId>
            <artifactId>maven-invoker</artifactId>
            <version>2.2</version>
            <scope>compile</scope>
```

```
        </dependency>
    </dependencies>

    <build>
        <finalName>${cml.jar.name}</finalName>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.0.0</version>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>cml.frontend.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>cml-frontend-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>


==> cml-compiler/cml-frontend/src/main/java/cml/frontend/Compiler.java

package cml.frontend;

import cml.generator.Generator;
import cml.io.Console;
import cml.io.FileSystem;
import cml.io.ModuleManager;
import cml.language.foundation.TempModel;
import cml.language.loader.ModelLoader;

public interface Compiler
{
    int compile(String modulePath, String targetName);

    static Compiler createCompiler()
    {
        final Console console = Console.createSystemConsole();
        return createCompiler(console);
    }

    static Compiler createCompiler(final Console console)
    {
        final FileSystem fileSystem = FileSystem.create(console);
        final ModuleManager moduleManager = ModuleManager.create(console, fileSystem);
        final ModelLoader modelLoader = ModelLoader.create(console, moduleManager);
        final Generator generator = Generator.create(console, fileSystem, moduleManager);

        return new CompilerImpl(fileSystem, moduleManager, modelLoader, generator);
    }
}

class CompilerImpl implements Compiler
{
    private static final String TARGETS_DIR = "/targets";

    private final FileSystem fileSystem;
    private final ModuleManager moduleManager;
    private final ModelLoader modelLoader;
    private final Generator generator;

    CompilerImpl(FileSystem fileSystem, ModuleManager moduleManager, ModelLoader modelLoader, Generator generator)
    {
        this.fileSystem = fileSystem;
        this.moduleManager = moduleManager;
        this.modelLoader = modelLoader;
        this.generator = generator;
    }

    @Override
    public int compile(final String modulePath, final String targetName)
    {
        final String moduleName = fileSystem.extractName(modulePath);
        final String modulesBaseDir = fileSystem.extractParentPath(modulePath);

        moduleManager.clearBaseDirs();
        moduleManager.addBaseDir(System.getenv("CML_MODULES_PATH"));
        moduleManager.addBaseDir(modulesBaseDir);

        final TempModel model = TempModel.create();
        final int exitCode = modelLoader.loadModel(model, moduleName);

        if (exitCode == 0)
        {
            return generator.generate(model, targetName, modulePath + TARGETS_DIR + "/" + targetName);
        }

        return exitCode;
    }
}


==> cml-compiler/cml-frontend/src/main/java/cml/frontend/Launcher.java

package cml.frontend;

public final class Launcher
{
    private static final String TEST_TASK_NAME = "test";
    private static final int EXIT_CODE_EXPECTED_ARGUMENTS = 51;

    private Launcher() {}

    public static void main(final String... args)
    {
        if (args.length >= 1)
        {
            final String targetName = args[0];

            if (targetName.equalsIgnoreCase(TEST_TASK_NAME))
            {
```

```
            final Tester tester = new Tester();
            final String testName = args.length > 1 ? args[1] : null;
            final String taskName = args.length > 2 ? args[2] : null;

            final int exitCode = tester.test(testName, taskName);

            System.exit(exitCode);
        }
        else
        {
            final Compiler compiler = Compiler.createCompiler();
            final String modulePath = ".";

            final int exitCode = compiler.compile(modulePath, targetName);

            System.exit(exitCode);
        }
    }
    else
    {
        System.out.println("Expected arguments. Usage: cml [<task_name> | test [<test_module>] [<task_name>]]");

        System.exit(EXIT_CODE_EXPECTED_ARGUMENTS);
    }
  }
}


==> cml-compiler/cml-frontend/src/main/java/cml/frontend/ModuleTest.java

package cml.frontend;

import cml.io.Console;
import com.google.common.io.Files;
import org.apache.maven.shared.invoker.*;
import org.codehaus.plexus.util.cli.CommandLineException;
import org.codehaus.plexus.util.cli.Commandline;
import org.codehaus.plexus.util.cli.WriterStreamConsumer;
import org.hamcrest.core.Is;
import org.jooq.lambda.Seq;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;

import java.io.*;
import java.nio.charset.Charset;
import java.util.List;
import java.util.Optional;

import static cml.frontend.Compiler.createCompiler;
import static cml.io.Console.createStringConsole;
import static java.util.Arrays.asList;
import static java.util.Collections.emptyList;
import static java.util.stream.Collectors.toList;
import static junit.framework.TestCase.assertEquals;
import static org.apache.commons.io.FileUtils.cleanDirectory;
import static org.codehaus.plexus.util.cli.CommandLineUtils.executeCommandLine;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.jooq.lambda.Seq.empty;
import static org.jooq.lambda.Seq.seq;
import static org.junit.Assert.assertTrue;

@RunWith(Parameterized.class)
public class ModuleTest
{
    private static final Charset FILE_ENCODING = Charset.forName("UTF-8");
    private static final int PROCESS_TIMEOUT_IN_SECONDS = 60;

    private static final String SOURCE_PATH = "source";
    private static final String TEMPLATES_PATH = "templates";
    private static final String TARGETS_PATH = "targets";
    private static final String TESTS_PATH = "tests";
    private static final String EXPECTED_PATH = "expected";
    private static final String CLIENTS_PATH = "clients";

    private static final String COMPILER_OUTPUT_TXT = "cml-compiler-output.txt";
    private static final String COMPILER_ERROR_TXT = "cml-compiler-errors.txt";
    private static final String CLIENT_OUTPUT_TXT = "expected-client-output.txt";
    private static final String CLIENT_JAR_SUFFIX = "-jar-with-dependencies.jar";
    private static final String IGNORED_LIST_TXT = "ignored-list.txt";
    private static final String CLIENT_PY = "client.py";
    private static final String POM_XML = "pom.xml";

    static String selectedTestName;
    static String selectedTaskName;

    @Parameterized.Parameters(name = "{0} ({1})")
    public static List<Object[]> testProperties()
    {
        return expectedDirs()
            .map(expectedDir -> new Object[] {
                testNameOf(expectedDir),
                taskNameOf(expectedDir),
                testDirOf(expectedDir),
                moduleDirOf(expectedDir),
                expectedDir
            })
            .filter(properties -> isSelectedTestModule(properties[0]))
            .filter(properties -> isSelectedTask(properties[1]))
            .collect(toList());
    }

    private static boolean isSelectedTestModule(final Object property)
    {
        return selectedTestName == null || property.equals(selectedTestName) || ((String) property).startsWith(selectedTestName + "/");
    }

    private static boolean isSelectedTask(final Object property)
    {
        return selectedTaskName == null || property.equals(selectedTaskName);
    }

    static Seq<String> selectedTestNames()
    {
        return expectedDirs().map(dir -> testNameOf(dir)).filter(testName -> isSelectedTestModule(testName));
    }

    static Seq<String> selectedTaskNames()
    {
        return expectedDirs().map(dir -> taskNameOf(dir)).filter(testName -> isSelectedTestModule(testName));
    }

    private static Seq<File> expectedDirs()
```

```
{
    return testDirs().flatMap(moduleDir -> subDirsOf(expectedPathOf(moduleDir)));
}

private static Seq<File> testDirs()
{
    return subDirsOf(TESTS_PATH)
        .flatMap(testDir -> subDirsOf(testDir.getAbsolutePath()))
        .filter(testDir -> sourceDirOf(testDir).isDirectory() || expectedDirOf(testDir).isDirectory());
}

private final String testName;
private final String taskName;
private final File testDir;
private final File moduleDir;
private final File expectedDir;
private final File targetDir;

private String compilerOutput;
private boolean testPassed;

public ModuleTest(String testName, String taskName, File testDir, File moduleDir, File expectedDir)
{
    this.testName = testName;
    this.taskName = taskName;
    this.testDir = testDir;
    this.moduleDir = moduleDir;
    this.expectedDir = expectedDir;
    this.targetDir = new File(moduleDir, TARGETS_PATH + "/" + taskName);
}

@Before
public void compileTestModule() throws Exception
{
    final Console console = createStringConsole();
    final Compiler compiler = createCompiler(console);

    compiler.compile(moduleDir.getPath(), taskName);

    compilerOutput = console.toString();
}

@After
public void cleanTargetDir() throws Exception
{
    if (testPassed & targetDir.isDirectory())
    {
        cleanDirectory(targetDir);
    }
}

@Test
public void verifyTestModule() throws IOException
{
    System.out.println("\n\nTesting " + testName + " with task " + taskName + ":");

    if (verifyCompilerOutput())
    {
        verifyTargetFiles();

        // Building generated target:
        buildMavenModule();
        checkPythonTypes();
        installPythonPackage();

        // Executing clients:
        executeJavaClient();
        executePythonClient();
    }

    System.out.print("- SUCCESS");

    testPassed = true;
}

private boolean verifyCompilerOutput() throws IOException
{
    final File expectedOutputFile = new File(expectedDir, COMPILER_OUTPUT_TXT);
    final File expectedErrorFile = new File(expectedDir, COMPILER_ERROR_TXT);

    if (expectedOutputFile.isFile())
    {
        System.out.print("- Verifying the compiler's output ...");

        assertThatOutputMatches("Compiler's output", expectedOutputFile, compilerOutput);

        System.out.println("OK");
    }
    else if (expectedErrorFile.isFile())
    {
        System.out.print("- Verifying the expected errors from compiler ...");

        assertThatOutputMatches("Compiler's output", expectedErrorFile, compilerOutput);

        System.out.println("OK");
    }
    else
    {
        System.out.println("- Ignored the compiler's output.");
    }

    return expectedOutputFile.isFile() || !expectedErrorFile.isFile(); // should continue?
}

private void verifyTargetFiles()
{
    System.out.print("- Verifying missing target files ...");

    assertThat(
        "Should have found all expected files, but missing:\n- " + missingFiles().toString("/n- "),
        missingFiles().count(), is(equalTo(0L)));

    System.out.println("OK");

    verifiedFiles().forEach(this::verifyTargetFile);

    undeclaredIgnoredFiles().forEach(ignoredFile -> System.out.println("- Ignored target file: " + relativePathOfTargetFile(ignoredFile)));
}

private void buildMavenModule()
{
    if (isMavenModule())
    {
        System.out.print("- Building Maven module: " + targetDir.getName() + " ");

        buildMavenModule(targetDir);
```

```
            System.out.println("OK");
        }
    }

    private void checkPythonTypes()
    {
        if (isPythonModule())
        {
            subDirsOf(targetDir.getAbsolutePath()).forEach(ModuleTest::checkPythonTypes);
        }
    }

    private void installPythonPackage()
    {
        if (isPythonModule())
        {
            installPythonPackage(targetDir);
        }
    }

    private void executeJavaClient()
    {
        final File javaClientDir = clientDir();

        if (new File(javaClientDir, POM_XML).isFile())
        {
            System.out.print("- Running Java client: " + clientFile(POM_XML) + " ...");

            executeJavaClient(javaClientDir);

            System.out.println("OK");
        }
    }

    private void executePythonClient()
    {
        final File pythonClient = new File(clientDir(), CLIENT_PY);

        if (pythonClient.isFile())
        {
            System.out.print("- Running Python client: " + clientFile(CLIENT_PY) + " ...");

            executePythonClient(pythonClient);

            System.out.println("OK");
        }
    }

    private File clientDir()
    {
        return new File(testDir + "/" + clientPath());
    }

    private String clientPath()
    {
        return CLIENTS_PATH + "/" + taskName;
    }

    private String clientFile(String fileName)
    {
        return clientPath() + "/" + fileName;
    }

    private void verifyTargetFile(final File expectedFile)
    {
        final String relativePath = relativePathOfExpectedFile(expectedFile);
        final File targetFile = new File(targetDir, relativePath);

        System.out.print("- Verifying target file: " + relativePath + " ...");

        assertThatOutputMatches("Target file: " + relativePath, expectedFile, targetFile);

        System.out.println("OK");
    }

    private boolean isPythonModule()
    {
        return new File(targetDir, "setup.py").isFile();
    }

    private boolean isMavenModule()
    {
        return new File(targetDir, "pom.xml").isFile();
    }

    private String relativePathOfExpectedFile(File expectedFile)
    {
        return expectedFile.getAbsolutePath().replace(expectedDir.getAbsolutePath() + "/", "");
    }

    private String relativePathOfTargetFile(File targetFile)
    {
        return targetFile.getAbsolutePath().replace(targetDir.getAbsolutePath() + "/", "");
    }

    private Seq<File> missingFiles()
    {
        return expectedFiles().filter(file -> !targetContainsExpectedFile(file));
    }

    private Seq<File> verifiedFiles()
    {
        return expectedFiles().filter(this::targetContainsExpectedFile);
    }

    private Seq<File> ignoredFiles()
    {
        return targetFiles().filter(file -> !expectedContainsTargetFile(file));
    }

    private Seq<File> declaredIgnoredFiles()
    {
        try
        {
            final String str = Files.toString(new File(expectedDir, IGNORED_LIST_TXT), FILE_ENCODING);

            if (str.trim().equals("__ALL__"))
            {
                return ignoredFiles();
            }
            else
            {
                return seq(asList(str.split("\n"))).map(relativePath -> new File(targetDir, relativePath));
            }
        }
        catch (IOException e)
        {
```

```
        return empty();
    }
}

private Seq<File> undeclaredIgnoredFiles()
{
    return ignoredFiles().removeAll(declaredIgnoredFiles());
}

private Seq<File> expectedFiles()
{
    return filesOf(expectedDir.getAbsolutePath()).filter(file -> !file.getName().equals(COMPILER_OUTPUT_TXT))
                                                 .filter(file -> !isIgnoredListFile(file));
}

private boolean isIgnoredListFile(final File file)
{
    return file.getParentFile().equals(expectedDir) && file.getName().equals(IGNORED_LIST_TXT);
}

private Seq<File> targetFiles()
{
    return filesOf(targetDir.getAbsolutePath());
}

private boolean targetContainsExpectedFile(File expectedFile)
{
    final File targetFile = fromExpectedToTargetFile(expectedFile);

    return targetFiles().contains(targetFile);
}

private boolean expectedContainsTargetFile(File targetFile)
{
    final File expectedFile = fromTargetToExpectedFile(targetFile);

    return expectedFiles().contains(expectedFile);
}

private File fromExpectedToTargetFile(File expectedFile)
{
    final String expectedFilePath = expectedFile.getAbsolutePath();
    final String expectedDirPath = expectedDir.getAbsolutePath();
    final String targetDirPath = targetDir.getAbsolutePath();

    return new File(expectedFilePath.replace(expectedDirPath, targetDirPath));
}

private File fromTargetToExpectedFile(File targetFile)
{
    final String targetFilePath = targetFile.getAbsolutePath();
    final String targetDirPath = targetDir.getAbsolutePath();
    final String expectedDirPath = expectedDir.getAbsolutePath();

    return new File(targetFilePath.replace(targetDirPath, expectedDirPath));
}

private static String testNameOf(final File expectedDir)
{
    final File moduleDir = testDirOf(expectedDir);
    return moduleDir.getParentFile().getName() + "/" + moduleDir.getName();
}

private static String taskNameOf(final File expectedDir)
{
    return expectedDir.getName();
}

private static File testDirOf(final File expectedDir)
{
    return expectedDir.getParentFile().getParentFile();
}

private static File moduleDirOf(final File expectedDir)
{
    final File testDir = testDirOf(expectedDir);

    return sourceDirOf(testDir).isDirectory() ? testDir : rootModuleDirOf(testDir);
}

private static File rootModuleDirOf(final File testDir)
{
    final File dir = testDir.getParentFile().getParentFile().getParentFile();

    assertTrue(
        "Expected module to be found in path: " + dir.getAbsolutePath(),
        sourceDirOf(dir).isDirectory() || templatesDirOf(dir).isDirectory());

    return dir;
}

private static File sourceDirOf(final File moduleDir)
{
    return new File(moduleDir, SOURCE_PATH);
}

private static File templatesDirOf(final File moduleDir)
{
    return new File(moduleDir, TEMPLATES_PATH);
}

private static File expectedDirOf(final File testDir)
{
    return new File(expectedPathOf(testDir));
}

private static String expectedPathOf(final File testDir)
{
    return testDir.getAbsolutePath() + "/" + EXPECTED_PATH;
}

private static Seq<File> subDirsOf(String basePath)
{
    final File[] subDirs = new File(basePath).listFiles(File::isDirectory);
    final List<File> subDirList = asList(subDirs == null ? new File[0] : subDirs);

    return seq(subDirList);
}

private static Seq<File> filesOf(String basePath)
{
    final File[] files = new File(basePath).listFiles(File::isFile);
    final List<File> fileList = asList(files == null ? new File[0] : files);

    final Seq<File> subFiles = subDirsOf(basePath).flatMap(subDir -> filesOf(subDir.getAbsolutePath()));

    return seq(fileList).concat(subFiles);
```

```
}

private static void executeJavaClient(File clientModuleDir)
{
    buildMavenModule(clientModuleDir);

    final File clientTargetDir = new File(clientModuleDir, "target");
    assertThat("Client target dir must exist: " + clientTargetDir, clientTargetDir.exists(), Is.is(true));

    final Optional<File> clientJarPath = filesOf(clientTargetDir.getAbsolutePath())
                                            .filter(file -> file.getAbsolutePath()
                                            .endsWith(CLIENT_JAR_SUFFIX)).findSingle();

    assertTrue("Client jar should have been found at: " + clientTargetDir, clientJarPath.isPresent());

    executeJar(clientTargetDir, clientJarPath.get().getAbsolutePath());
}

private static void executeJar(final File clientTargetDir, final String jarPath)
{
    try (final ByteArrayOutputStream outputStream = new ByteArrayOutputStream())
    {
        try
        {
            final int exitCode = executeJar(clientTargetDir.getPath(), jarPath, emptyList(), outputStream);

            final String actualClientOutput = stringOf(outputStream);
            final File expectedOutputFile = new File(clientTargetDir.getParentFile(), CLIENT_OUTPUT_TXT);

            if (expectedOutputFile.isFile())
            {
                assertThatOutputMatches("Client's output", expectedOutputFile, actualClientOutput);
            }
            else
            {
                System.out.println("\n- Ignored the Java client's output.");
            }

            assertThat(
                "Client's exit code: " + exitCode + " - output:\n---\n" + actualClientOutput + "\n---",
                exitCode, is(0));
        }
        catch (CommandLineException exception)
        {
            System.out.println();
            System.out.println("---------");
            System.out.println("Error running client: " + clientTargetDir.getName());
            System.out.println(stringOf(outputStream));

            throw new RuntimeException("CommandLineException: " + exception.getMessage(), exception);
        }
    }
    catch (IOException exception)
    {
        System.out.println();

        throw new RuntimeException("IOException: " + exception.getMessage(), exception);
    }
}

private static int executeJar(
    final String currentDirPath,
    final String jarPath,
    final List<String> args,
    final ByteArrayOutputStream outputStream) throws CommandLineException, IOException
{
    final File jarFile = new File(jarPath);
    assertThat("Jar file must exit: " + jarFile, jarFile.exists(), is(true));

    final File javaBinDir = new File(System.getProperty("java.home"), "bin");
    assertThat("Java bin dir must exit: " + javaBinDir, javaBinDir.exists(), is(true));

    final File javaExecFile = new File(javaBinDir, "java");
    assertThat("Java exec file must exit: " + javaExecFile, javaExecFile.exists(), is(true));

    final Commandline commandLine = new Commandline();
    commandLine.setWorkingDirectory(currentDirPath);
    commandLine.setExecutable(javaExecFile.getAbsolutePath());

    commandLine.createArg().setValue("-jar");
    commandLine.createArg().setValue(jarFile.getAbsolutePath());

    for (final String arg : args) commandLine.createArg().setValue(arg);

    final Writer writer = new OutputStreamWriter(outputStream);
    final WriterStreamConsumer systemOut = new WriterStreamConsumer(writer);
    final WriterStreamConsumer systemErr = new WriterStreamConsumer(writer);

    return executeCommandLine(commandLine, systemOut, systemErr, PROCESS_TIMEOUT_IN_SECONDS);
}

private static void buildMavenModule(final File moduleDir)
{
    final String m2_home = System.getenv("M2_HOME");

    assertThat(
        "In order to build the generated module, Maven should be installed and M2_HOME set.",
        new File(m2_home).isDirectory());

    System.setProperty("maven.home", m2_home);

    final InvocationRequest request = new DefaultInvocationRequest();
    request.setBaseDirectory(moduleDir);
    request.setGoals(asList("clean", "install"));
    request.setInteractive(false);

    final Console console = createStringConsole();
    final Invoker invoker = new DefaultInvoker();
    invoker.setOutputHandler(line -> {
        System.out.print(".");
        console.println(line);
    });

    try
    {
        final InvocationResult result = invoker.execute(request);

        assertThat(
            "Maven failure - exit code: " + result.getExitCode() + "\n" + console.toString(),
            result.getExitCode(), is(equalTo(0)));
    }
    catch (MavenInvocationException exception)
    {
        System.out.println();
        System.out.println("---------");
        System.out.println("Error running Maven:");
        System.out.println(console.toString());
```

```
            throw new RuntimeException("MavenInvocationException: " + exception.getMessage(), exception);
        }
    }

    private static void checkPythonTypes(final File moduleDir)
    {
        System.out.print("- Checking types of Python module: " + moduleDir.getName() + " ...");

        assertThat(
            "In order to check types of the generated module, Python 3 should be installed and PYTHON_HOME set.",
            new File(pythonCmd("python3")).isFile());

        try (ByteArrayOutputStream outputStream = new ByteArrayOutputStream())
        {
            // https://github.com/python/mypy
            final Commandline commandLine = new Commandline();
            commandLine.setExecutable(pythonCmd("python3"));
            commandLine.createArg().setValue("-m");
            commandLine.createArg().setValue("mypy");
            commandLine.createArg().setValue(moduleDir.getCanonicalPath());

            final Writer writer = new OutputStreamWriter(outputStream);
            final WriterStreamConsumer systemOut = new WriterStreamConsumer(writer);
            final WriterStreamConsumer systemErr = new WriterStreamConsumer(writer);

            try
            {
                final int exitCode = executeCommandLine(commandLine, systemOut, systemErr, PROCESS_TIMEOUT_IN_SECONDS);

                assertThat(
                    "mypy's exit code: " + exitCode + "\nmypy's output: \n---\n" + stringOf(outputStream) + "---\n",
                    exitCode, is(0));
            }
            catch (CommandLineException exception)
            {
                System.out.println("---------");
                System.out.println("Error running mypy:");
                System.out.println(stringOf(outputStream));

                throw new RuntimeException("CommandLineException: " + exception.getMessage(), exception);
            }
        }
        catch (IOException exception)
        {
            throw new RuntimeException("IOException: " + exception.getMessage(), exception);
        }

        System.out.println("OK");
    }

    private static void installPythonPackage(final File packageDir)
    {
        System.out.print("- Installing Python package: " + packageDir.getName() + " ...");

        assertThat(
            "In order to install the generated Python package, pip should be installed.",
            new File(pythonCmd("pip3")).isFile());

        try (ByteArrayOutputStream outputStream = new ByteArrayOutputStream())
        {
            // --upgrade: overwriting previous installation - https://packaging.python.org/installing
            final Commandline commandLine = new Commandline();
            commandLine.setExecutable(pythonCmd("pip3"));
            commandLine.createArg().setValue("install");
            commandLine.createArg().setValue("--upgrade");
            commandLine.createArg().setValue(packageDir.getCanonicalPath());

            final Writer writer = new OutputStreamWriter(outputStream);
            final WriterStreamConsumer systemOut = new WriterStreamConsumer(writer);
            final WriterStreamConsumer systemErr = new WriterStreamConsumer(writer);

            try
            {
                final int exitCode = executeCommandLine(commandLine, systemOut, systemErr, PROCESS_TIMEOUT_IN_SECONDS);

                assertThat(
                    "pip's exit code: " + exitCode + "\npip's output: \n---\n" + stringOf(outputStream) + "---\n",
                    exitCode, is(0));
            }
            catch (CommandLineException exception)
            {
                System.out.println("---------");
                System.out.println("Error running pip:");
                System.out.println(stringOf(outputStream));

                throw new RuntimeException("CommandLineException: " + exception.getMessage(), exception);
            }
        }
        catch (IOException exception)
        {
            throw new RuntimeException("IOException: " + exception.getMessage(), exception);
        }

        System.out.println("OK");
    }

    private static void executePythonClient(File client)
    {
        assertThat(
            "In order to run the Python client, Python 3 should be installed and PYTHON_HOME set.",
            new File(pythonCmd("python3")).isFile());

        try (ByteArrayOutputStream outputStream = new ByteArrayOutputStream())
        {
            // Executing Python module - https://www.python.org/dev/peps/pep-0338/#current-behaviour
            final Commandline commandLine = new Commandline();
            commandLine.setExecutable(pythonCmd("python3"));
            commandLine.createArg().setValue(client.getCanonicalPath());

            final Writer writer = new OutputStreamWriter(outputStream);
            final WriterStreamConsumer systemOut = new WriterStreamConsumer(writer);
            final WriterStreamConsumer systemErr = new WriterStreamConsumer(writer);

            try
            {
                int exitCode = executeCommandLine(commandLine, systemOut, systemErr, PROCESS_TIMEOUT_IN_SECONDS);

                final String actualClientOutput = stringOf(outputStream);
                final File expectedOutputFile = new File(client.getParentFile(), CLIENT_OUTPUT_TXT);

                if (expectedOutputFile.isFile())
                {
                    assertThatOutputMatches("Client's output", expectedOutputFile, actualClientOutput);
                }
                else
                {
```

```
                System.out.println("\n- Ignored the Python client's output.");
            }

            assertThat(
                "Client's exit code: " + exitCode + "\noutput: \n---\n" + actualClientOutput + "---\n",
                exitCode, is(0));
        }
        catch (CommandLineException exception)
        {
            System.out.println("---------");
            System.out.println("Error running client:");
            System.out.println(stringOf(outputStream));

            throw new RuntimeException("CommandLineException: " + exception.getMessage(), exception);
        }
    }
    catch (IOException exception)
    {
        throw new RuntimeException("IOException: " + exception.getMessage(), exception);
    }
}

private static String stringOf(final ByteArrayOutputStream outputStream)
{
    return new String(outputStream.toByteArray(), FILE_ENCODING);
}

private static String pythonCmd(String cmd)
{
    final String pythonHomeDir = System.getenv("PYTHON_HOME");
    return pythonHomeDir + "/bin/" + cmd;
}

private static void assertThatOutputMatches(
    final String reason,
    final File expectedOutputFile,
    final String actualOutput) throws IOException
{
    final String expectedOutput = Files.toString(expectedOutputFile, FILE_ENCODING);

    assertEquals(reason, expectedOutput, actualOutput);
}

private static void assertThatOutputMatches(
    final String reason,
    final File expectedOutputFile,
    final File actualOutputFile)
{
    try
    {
        final String expectedOutput = Files.toString(expectedOutputFile, FILE_ENCODING);
        final String actualOutput = Files.toString(actualOutputFile, FILE_ENCODING);

        assertEquals(reason, expectedOutput, actualOutput);
    }
    catch (IOException exception)
    {
        throw new AssertionError("IOException: " + exception.getMessage());
    }
}
}
```

==> cml-compiler/cml-frontend/src/main/java/cml/frontend/Tester.java

```
package cml.frontend;

import org.jetbrains.annotations.Nullable;
import org.junit.internal.TextListener;
import org.junit.runner.Description;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

class Tester
{
    private static final int EXIT_CODE_SUCCESS = 0;
    private static final int EXIT_CODE_TESTS_FAILED = 52;

    int test(final @Nullable String testName, final @Nullable String taskName)
    {
        final JUnitCore junit = new JUnitCore();

        junit.addListener(new TextListener(System.out)
        {
            @Override
            public void testStarted(final Description description) {}

            @Override
            protected void printHeader(final long runTime)
            {
                System.out.println();

                super.printHeader(runTime);
            }

            @Override
            protected void printFailures(final Result result)
            {
                System.out.println();
                super.printFailures(result);
            }

            @Override
            protected void printFailure(Failure failure, String prefix) {
                System.out.println();
                System.out.println(prefix + ") " + headerOf(failure));
                System.out.println();
                System.out.println(failure.getMessage());
            }

            private String headerOf(final Failure failure)
            {
                final String header = failure.getTestHeader();
                final int start = "verifyTestModule".length() + 1;
                final int end = header.indexOf(ModuleTest.class.getName()) - 2;
                return header.substring(start, end);
            }
        });

        ModuleTest.selectedTestName = testName;
        ModuleTest.selectedTaskName = taskName;

        final Result result = junit.run(ModuleTest.class);

        if (result.getRunCount() == 0)
        {
```

```
            if (testName != null && ModuleTest.selectedTestNames().count() == 0)
            {
                System.out.println("Test module not found: " + testName);
            }

            if (taskName != null && ModuleTest.selectedTaskNames().count() == 0)
            {
                System.out.println("Task not found: " + taskName);
            }
        }

        return testsExecuted(result, testName, taskName) && result.wasSuccessful() ? EXIT_CODE_SUCCESS : EXIT_CODE_TESTS_FAILED;
    }

    private boolean testsExecuted(final Result result, String testName, String taskName)
    {
        return (testName == null && taskName == null) || result.getRunCount() > 0;
    }
}
```

==> cml-compiler/cml-generator/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>cml</groupId>
        <artifactId>cml-compiler</artifactId>
        <version>master-SNAPSHOT</version>
    </parent>

    <artifactId>cml-generator</artifactId>
    <version>master-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-language</artifactId>
            <version>${project.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-templates</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-io</artifactId>
            <version>${project.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>21.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

==> cml-compiler/cml-generator/src/main/java/cml/generator/Generator.java

```
package cml.generator;

import cml.io.Console;
import cml.io.Directory;
import cml.io.FileSystem;
import cml.io.ModuleManager;
import cml.language.foundation.TempModel;
import cml.language.generated.Constructor;
import cml.language.generated.Task;
import cml.templates.TemplateGroupFile;
import cml.templates.TemplateRenderer;
import cml.templates.TemplateRepository;

import java.util.Optional;

import static cml.language.functions.ModelFunctions.targetOf;
import static cml.language.functions.ModelVisitorFunctions.visitModel;

public interface Generator
{
    int generate(TempModel model, final String targetName, final String targetDirPath);

    static Generator create(Console console, FileSystem fileSystem, ModuleManager moduleManager)
    {
        final TemplateRepository templateRepository = TemplateRepository.create(moduleManager);
        final TemplateRenderer templateRenderer = TemplateRenderer.create(console);
        final TargetFileRepository targetFileRepository = TargetFileRepository.create(templateRepository, templateRenderer);
        final TargetFileRenderer targetFileRenderer = TargetFileRenderer.create(console, fileSystem, targetFileRepository, templateRenderer);
        return new GeneratorImpl(console, fileSystem, moduleManager, targetFileRepository, targetFileRenderer);
    }
}

class GeneratorImpl implements Generator
{
    private static final int SUCCESS = 0;
    private static final int FAILURE_CONSTRUCTOR_UNKNOWN = 101;
    private static final int FAILURE_CONSTRUCTOR_UNDEFINED = 102;
    private static final int FAILURE_TASK_UNDECLARED = 103;

    private static final String NO_SOURCE_FILE_HAS_DECLARED_TASK = "no source file has declared task named: %s";
    private static final String NO_CONSTRUCTOR_DEFINED_FOR_TASK = "no constructor defined for task: %s";
    private static final String NO_TEMPLATES_FOUND_FOR_CONSTRUCTOR = "unable to find templates for constructor: %s";

    private final Console console;
    private final FileSystem fileSystem;
    private final ModuleManager moduleManager;
    private final TargetFileRepository targetFileRepository;
    private final TargetFileRenderer targetFileRenderer;

    GeneratorImpl(
        Console console,
```

```
        FileSystem fileSystem,
        ModuleManager moduleManager,
        TargetFileRepository targetFileRepository,
        TargetFileRenderer targetFileRenderer)
    {
        this.console = console;
        this.fileSystem = fileSystem;
        this.moduleManager = moduleManager;
        this.targetFileRepository = targetFileRepository;
        this.targetFileRenderer = targetFileRenderer;
    }

    @Override
    public int generate(TempModel model, final String targetName, final String targetDirPath)
    {
        TemplateGroupFile.setModuleManager(moduleManager);

        final Optional<Task> target = targetOf(model, targetName);
        if (!target.isPresent())
        {
            console.error(NO_SOURCE_FILE_HAS_DECLARED_TASK, targetName);
            return FAILURE_TASK_UNDECLARED;
        }

        final Optional<Constructor> constructor = target.get().getConstructor();
        if (!constructor.isPresent())
        {
            console.error(NO_CONSTRUCTOR_DEFINED_FOR_TASK, target.get().getName());
            return FAILURE_CONSTRUCTOR_UNDEFINED;
        }

        if (!targetFileRepository.templatesFoundFor(target.get()))
        {
            console.error(NO_TEMPLATES_FOUND_FOR_CONSTRUCTOR, constructor.get().getName());
            return FAILURE_CONSTRUCTOR_UNKNOWN;
        }

        final Optional<Directory> targetDir = fileSystem.findDirectory(targetDirPath);
        targetDir.ifPresent(fileSystem::cleanDirectory);

        final TargetGenerator targetGenerator = new TargetGenerator(targetFileRenderer, target.get(), targetDirPath);

        visitModel(model, targetGenerator);

        return SUCCESS;
    }

}
```

==> cml-compiler/cml-generator/src/main/java/cml/generator/TargetFile.java

```
package cml.generator;

import cml.templates.TemplateFile;

import java.util.Optional;

class TargetFile
{
    private static final char EXTENSION_SEPARATOR = '.';

    TargetFile(final String path, final String templateName)
    {
        this.path = path;
        this.templateName = templateName;
    }

    // Attributes:

    private final String path;

    String getPath()
    {
        return path;
    }

    String getExtension()
    {
        return path.substring(path.lastIndexOf(EXTENSION_SEPARATOR) + 1);
    }

    //---

    private final String templateName;

    String getTemplateName()
    {
        return templateName;
    }

    // Associations:

    private TemplateFile templateFile;

    Optional<TemplateFile> getTemplateFile()
    {
        return Optional.ofNullable(templateFile);
    }

    void setTemplateFile(final TemplateFile templateFile)
    {
        this.templateFile = templateFile;
    }
}
```

==> cml-compiler/cml-generator/src/main/java/cml/generator/TargetFileRenderer.java

```
package cml.generator;

import cml.io.Console;
import cml.io.FileSystem;
import cml.language.generated.Literal;
import cml.language.generated.NamedElement;
import cml.language.generated.Task;
import cml.templates.TemplateRenderer;

import java.io.File;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public interface TargetFileRenderer
{
    void renderTargetFiles(Task task, String targetDirPath, String fileType, NamedElement namedElement);
```

```
    static TargetFileRenderer create(
        Console console,
        FileSystem fileSystem,
        TargetFileRepository targetFileRepository,
        TemplateRenderer templateRenderer)
    {
        return new TargetFileRendererImpl(console, fileSystem, targetFileRepository, templateRenderer);
    }
}

class TargetFileRendererImpl implements TargetFileRenderer
{
    private static final String TASK = "task";

    private final Console console;
    private final FileSystem fileSystem;
    private final TargetFileRepository targetFileRepository;
    private final TemplateRenderer templateRenderer;

    TargetFileRendererImpl(
        Console console,
        FileSystem fileSystem,
        TargetFileRepository targetFileRepository,
        TemplateRenderer templateRenderer)
    {
        this.console = console;
        this.fileSystem = fileSystem;
        this.targetFileRepository = targetFileRepository;
        this.templateRenderer = templateRenderer;
    }

    @Override
    public void renderTargetFiles(
        final Task task,
        final String targetDirPath,
        final String modelElementType,
        final NamedElement namedElement)
    {
        final Map<String, Object> templateArgs = new HashMap<>();
        templateArgs.put(TASK, getTargetProperties(task));
        templateArgs.put(modelElementType, namedElement);

        final List<TargetFile> targetFiles = targetFileRepository.findTargetFiles(task, modelElementType, templateArgs);

        if (targetFiles.size() > 0)
        {
            if (!namedElement.getName().equals("model"))
            {
                console.println();
            }

            console.println("%s files:", namedElement.getName());

            targetFiles.forEach(targetFile -> renderTargetFile(targetFile, targetDirPath, templateArgs));
        }
    }

    private void renderTargetFile(
        final TargetFile targetFile,
        final String targetDirPath,
        final Map<String, Object> templateArgs)
    {
        console.println("- %s", targetFile.getPath());

        if (targetFile.getTemplateFile().isPresent())
        {
            final String contents = templateRenderer.renderTemplate(
                targetFile.getTemplateFile().get(),
                targetFile.getTemplateName(),
                templateArgs);

            final String path = targetDirPath + File.separatorChar + targetFile.getPath();
            fileSystem.createFile(path, contents);
        }
        else
        {
            console.println(" (not found template for: %s)", targetFile.getPath());
        }
    }

    private static Map<String, Object> getTargetProperties(final Task task)
    {
        final Map<String, Object> properties = new HashMap<>();

        //noinspection ConstantConditions
        task.getProperties()
            .stream()
            .filter(property -> property.getValue().isPresent() && property.getValue().get() instanceof Literal)
            .forEach(property -> properties.put(property.getName(), ((Literal) property.getValue().get()).getText()));

        return properties;
    }
}
```

==> cml-compiler/cml-generator/src/main/java/cml/generator/TargetFileRepository.java

```
package cml.generator;

import cml.language.features.TempModule;
import cml.language.generated.Task;
import cml.templates.TemplateFile;
import cml.templates.TemplateRenderer;
import cml.templates.TemplateRepository;

import java.util.List;
import java.util.Map;
import java.util.Optional;

import static cml.language.functions.ModelElementFunctions.moduleOf;
import static cml.language.functions.ModuleFunctions.selfOrImportedModuleOf;
import static java.util.Arrays.stream;
import static java.util.Collections.emptyList;
import static java.util.stream.Collectors.toList;

interface TargetFileRepository
{
    boolean templatesFoundFor(Task task);
    List<TargetFile> findTargetFiles(Task task, String fileType, Map<String, Object> templateArgs);

    static TargetFileRepository create(TemplateRepository templateRepository, TemplateRenderer templateRenderer)
    {
        return new TargetFileRepositoryImpl(templateRepository, templateRenderer);
    }
}
```

```
class TargetFileRepositoryImpl implements TargetFileRepository
{
    private static final String STG_EXT = ".stg";
    private static final String GROUP_FILES = "files" + STG_EXT;
    private static final String FILES_SUFFIX = "_files";
    private static final String FILE_LINE_SEPARATOR = "\\|";
    private static final String TEMPLATE_NAME_SEPARATOR = ":";

    // Collaborators:
    private final TemplateRepository templateRepository;
    private final TemplateRenderer templateRenderer;

    TargetFileRepositoryImpl(TemplateRepository templateRepository, TemplateRenderer templateRenderer)
    {
        this.templateRepository = templateRepository;
        this.templateRenderer = templateRenderer;
    }

    @Override
    public boolean templatesFoundFor(Task task)
    {
        return findFilesTemplateForTask(task).isPresent();
    }

    @Override
    public List<TargetFile> findTargetFiles(
        final Task task,
        final String modelElementType,
        final Map<String, Object> templateArgs)
    {
        final Optional<TemplateFile> fileTemplates = findFilesTemplateForTask(task);

        if (fileTemplates.isPresent() && moduleOf(task).isPresent())
        {
            final String moduleName = fileTemplates.get().getModuleName();
            final Optional<TempModule> module = selfOrImportedModuleOf(moduleOf(task).get(), moduleName);

            if (module.isPresent() && task.getConstructor().isPresent())
            {
                final String templateName = modelElementType + FILES_SUFFIX;
                final String files = templateRenderer.renderTemplate(fileTemplates.get(), templateName, templateArgs);

                if (files.trim().length() > 0)
                {
                    final String constructorName = task.getConstructor().get().getName();

                    return stream(files.split("\n"))
                        .map(line -> line.split(FILE_LINE_SEPARATOR))
                        .map(pair -> createTargetFile(module.get(), constructorName, pair[1], pair[0]))
                        .collect(toList());
                }
            }
        }

        return emptyList();
    }

    private Optional<TemplateFile> findFilesTemplateForTask(Task task)
    {
        if (moduleOf(task).isPresent() & task.getConstructor().isPresent())
        {
            final TempModule module = moduleOf(task).get();
            final String constructorName = task.getConstructor().get().getName();

            return findTemplateFile(module, constructorName, GROUP_FILES);
        }

        return Optional.empty();
    }

    private TargetFile createTargetFile(
        final TempModule module,
        String constructorName,
        final String targetFilePath,
        String templateFileName)
    {
        final String[] pair = templateFileName.split(TEMPLATE_NAME_SEPARATOR);
        if (pair.length == 2)
        {
            constructorName = pair[0];
            templateFileName = pair[1];
        }

        final TargetFile targetFile = new TargetFile(targetFilePath, templateFileName);
        final Optional<TemplateFile> templateFile = findTemplateFile(
            module, constructorName, templateFileName + STG_EXT);

        templateFile.ifPresent(targetFile::setTemplateFile);

        return targetFile;
    }

    private Optional<TemplateFile> findTemplateFile(
        final TempModule module,
        final String constructorName,
        final String templateFileName)
    {
        final Optional<TemplateFile> templateFile = templateRepository.findTemplate(
            module.getName(),
            constructorName,
            templateFileName);

        if (templateFile.isPresent())
        {
            return templateFile;
        }
        else
        {
            for (final TempModule importedModule: module.getImportedModules())
            {
                final Optional<TemplateFile> importedTemplateFile = templateRepository.findTemplate(
                    importedModule.getName(),
                    constructorName,
                    templateFileName);

                if (importedTemplateFile.isPresent())
                {
                    return importedTemplateFile;
                }
            }
        }

        return templateFile;
    }
}
```

==> cml-compiler/cml-generator/src/main/java/cml/generator/TargetGenerator.java

```
package cml.generator;

import cml.language.features.TempConcept;
import cml.language.features.TempModule;
import cml.language.foundation.TempModel;
import cml.language.generated.Association;
import cml.language.generated.NamedElement;
import cml.language.generated.Task;
import cml.language.loader.ModelVisitor;

class TargetGenerator implements ModelVisitor
{
    private static final String MODEL = "model";
    private static final String MODULE = "module";
    private static final String CONCEPT = "concept";
    private static final String ASSOCIATION = "association";

    private final TargetFileRenderer targetFileRenderer;
    private final Task task;
    private final String targetDirPath;

    TargetGenerator(
        TargetFileRenderer targetFileRenderer,
        Task task,
        String targetDirPath)
    {
        this.targetFileRenderer = targetFileRenderer;
        this.task = task;
        this.targetDirPath = targetDirPath;
    }

    @Override
    public void visit(TempModel model)
    {
        generateTargetFiles(MODEL, model);
    }

    @Override
    public void visit(final TempModule module)
    {
        generateTargetFiles(MODULE, module);
    }

    @Override
    public void visit(TempConcept concept)
    {
        generateTargetFiles(CONCEPT, concept);
    }

    @Override
    public void visit(Association association)
    {
        generateTargetFiles(ASSOCIATION, association);
    }

    private void generateTargetFiles(String namedElementType, NamedElement namedElement)
    {
        targetFileRenderer.renderTargetFiles(task, targetDirPath, namedElementType, namedElement);
    }
}
```

==> cml-compiler/cml-generator/src/test/java/templates/lang/common/FieldTest.java

```
package templates.lang.common;

import cml.language.generated.Property;
import cml.language.types.TempNamedType;
import org.junit.Test;

import java.io.IOException;

import static java.util.Collections.emptyList;

public class FieldTest extends LangTest
{
    public FieldTest(String targetLanguage)
    {
        super(targetLanguage);
    }

    @Override
    protected String getExpectedOutputPath()
    {
        return "field";
    }

    @Test
    public void field_type__optional() throws IOException
    {
        final String cardinality = "?"; // optional

        field_type(cardinality, "optional.txt");
    }

    @Test
    public void field_type__required() throws IOException
    {
        final String cardinality = ""; // required

        field_type(cardinality, "required.txt");
    }

    @Test
    public void field_type__sequence() throws IOException
    {
        final String cardinality = "*"; // sequence

        field_type(cardinality, "sequence.txt");
    }

    private void field_type(String cardinality, String expectedOutputPath) throws IOException
    {
        for (String name : commonNameFormats)
        {
            final Property property = Property.createProperty(
                name,
                null, null, emptyList(), false,
                TempNamedType.create(name, cardinality), null, null);

            testTemplateWithNamedElement("field_type", property, expectedOutputPath);
        }
    }
}
```

==> cml-compiler/cml-generator/src/test/java/templates/lang/common/GenericTest.java

```
package templates.lang.common;

import org.junit.Test;
import org.junit.experimental.theories.DataPoints;
import org.junit.experimental.theories.FromDataPoints;
import org.junit.experimental.theories.Theory;
import org.stringtemplate.v4.ST;

import java.util.ArrayList;

import static java.util.Arrays.asList;
import static java.util.Collections.emptyList;
import static java.util.Collections.singletonList;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

public class GenericTest extends TemplateTest
{
    @DataPoints("trueConditions")
    public static final Object[] trueConditions = { Boolean.TRUE, singletonList("Item") };

    @DataPoints("falseConditions")
    public static final Object[] falseConditions = { Boolean.FALSE, null, emptyList(), new ArrayList() };

    @Override
    protected String getTemplatePath()
    {
        return "lang/common/generic";
    }

    @Theory
    public void newLineIf_true(@FromDataPoints("trueConditions") Object cond)
    {
        testTemplateWithCond("newLineIf", cond, "\n");
    }

    @Theory
    public void newLineIf_false(@FromDataPoints("falseConditions") Object cond)
    {
        testTemplateWithCond("newLineIf", cond, "");
    }

    @Theory
    public void newLineIfEither_true(
        @FromDataPoints("trueConditions") Object trueCond,
        @FromDataPoints("falseConditions") Object falseCond)
    {
        testTemplateWithCond2("newLineIfEither", trueCond, falseCond, "\n");
        testTemplateWithCond2("newLineIfEither", falseCond, trueCond, "\n");
    }

    @Theory
    public void newLineIfEither_false(@FromDataPoints("falseConditions") Object cond)
    {
        testTemplateWithCond2("newLineIfEither", cond, cond, "");
    }

    @Theory
    public void newLineIf2_true(@FromDataPoints("trueConditions") Object cond)
    {
        testTemplateWithCond2("newLineIf2", cond, cond, "\n");
    }

    @Theory
    public void newLineIf2_false(@FromDataPoints("falseConditions") Object cond)
    {
        testTemplateWithCond2("newLineIf2", cond, cond, "");
    }

    @Theory
    public void newLineIf2_mixed(
        @FromDataPoints("trueConditions") Object trueCond,
        @FromDataPoints("falseConditions") Object falseCond)
    {
        testTemplateWithCond2("newLineIf2", trueCond, falseCond, "");
        testTemplateWithCond2("newLineIf2", falseCond, trueCond, "");
    }

    @Theory
    public void commaIf_true(@FromDataPoints("trueConditions") Object cond)
    {
        testTemplateWithCond("commaIf", cond, ", ");
    }

    @Theory
    public void commaIf_false(@FromDataPoints("falseConditions") Object cond)
    {
        testTemplateWithCond("commaIf", cond, "");
    }

    @Theory
    public void commaIf2_true(@FromDataPoints("trueConditions") Object cond)
    {
        testTemplateWithCond2("commaIf2", cond, cond, ", ");
    }

    @Theory
    public void commaIf2_false(@FromDataPoints("falseConditions") Object cond)
    {
        testTemplateWithCond2("commaIf2", cond, cond,"");
    }

    @Theory
    public void commaIf2_mixed(
        @FromDataPoints("trueConditions") Object trueCond,
        @FromDataPoints("falseConditions") Object falseCond)
    {
        testTemplateWithCond2("commaIf2", trueCond, falseCond,"");
        testTemplateWithCond2("commaIf2", falseCond, trueCond,"");
    }

    @Test
    public void line_list()
    {
        testLineList(emptyList(), "");
        testLineList(asList("a"), "a");
        testLineList(asList("a", "b"), "a\n\nb");
        testLineList(asList("a", null), "a");
        testLineList(asList("a", null, "c"), "a\n\nc");
        testLineList(asList(null, "b", "c"), "b\n\nc");
        testLineList(asList(null, null, "c"), "c");
        testLineList(asList(null, null, null), "");
        testLineList(asList("a", ""), "a");
```

```
        testLineList(asList("a", "", "c"), "a\n\nc");
        testLineList(asList("", "b", ""), "b");
        testLineList(asList("a", "b", "c"), "a\n\nb\n\nc");
        testLineList(asList(new ST("<first([\"a\", \"d\"])>"), "b", "c"), "a\n\nb\n\nc");
    }

    private void testTemplateWithCond(String templateName, Object cond, String expectedResult)
    {
        final ST template = getTemplate(templateName);

        template.add("cond", cond);

        final String result = template.render();

        assertThat(result, is(expectedResult));
    }

    private void testTemplateWithCond2(String templateName, Object cond1, Object cond2, String expectedResult)
    {
        final ST template = getTemplate(templateName);

        template.add("cond1", cond1);
        template.add("cond2", cond2);

        final String result = template.render();

        assertThat(result, is(expectedResult));
    }

    private void testLineList(Object list, String expectedResult)
    {
        testTemplateWithList("line_list", list, expectedResult);
    }

    private void testTemplateWithList(String templateName, Object list, String expectedResult)
    {
        final ST template = getTemplate(templateName);

        template.add("list", list);

        final String result = template.render();

        assertThat(result, is(expectedResult));
    }
}
```

==> cml-compiler/cml-generator/src/test/java/templates/lang/common/GetterTest.java

```
package templates.lang.common;

import cml.language.generated.Property;
import cml.language.generated.ValueType;
import cml.language.types.TempNamedType;
import org.junit.Test;

import java.io.IOException;

import static cml.primitives.Types.INTEGER;
import static java.util.Collections.emptyList;

public class GetterTest extends LangTest
{
    public GetterTest(String targetLanguage)
    {
        super(targetLanguage);
    }

    @Override
    protected String getExpectedOutputPath()
    {
        return "getter";
    }

    @Test
    public void getter_call__optional() throws IOException
    {
        final String cardinality = "?"; // optional

        getter_call(cardinality);
    }

    @Test
    public void getter_call__required() throws IOException
    {
        final String cardinality = null; // required

        getter_call(cardinality);
    }

    @Test
    public void getter_call__sequence() throws IOException
    {
        final String cardinality = "*"; // sequence

        getter_call(cardinality);
    }

    @Test
    public void getter_type__required() throws IOException
    {
        final String cardinality = ""; // required

        getter_type(cardinality, "required.txt");
    }

    @Test
    public void getter_type__optional() throws IOException
    {
        final String cardinality = "?"; // optional

        getter_type(cardinality, "optional.txt");
    }

    @Test
    public void getter_type__sequence() throws IOException
    {
        final String cardinality = "*"; // sequence

        getter_type(cardinality, "sequence.txt");
    }

    @Test
    public void getter_type__sequence_integer() throws IOException
    {
        testTemplateWithType("getter_type", ValueType.createValueType("*", INTEGER), "sequence_integer.txt");
    }
```

```
    @Test
    public void interface_getter__required() throws IOException
    {
        final String cardinality = ""; // required

        interface_getter(cardinality, "required.txt");
    }

    @Test
    public void interface_getter__optional() throws IOException
    {
        final String cardinality = "?"; // optional

        interface_getter(cardinality, "optional.txt");
    }

    @Test
    public void interface_getter__sequence() throws IOException
    {
        final String cardinality = "*"; // sequence

        interface_getter(cardinality, "sequence.txt");
    }

    @Test
    public void class_getter__required() throws IOException
    {
        final String cardinality = ""; // required

        class_getter(cardinality, "required.txt");
    }

    @Test
    public void class_getter__optional() throws IOException
    {
        final String cardinality = "?"; // optional

        class_getter(cardinality, "optional.txt");
    }

    @Test
    public void class_getter__sequence() throws IOException
    {
        final String cardinality = "*"; // sequence

        class_getter(cardinality, "sequence.txt");
    }

    private void getter_type(String cardinality, String expectedOutput) throws IOException
    {
        for (String name : commonNameFormats)
        {
            testTemplateWithType("getter_type", TempNamedType.create(name, cardinality), expectedOutput);
        }
    }

    private void getter_call(String cardinality) throws IOException
    {
        testTemplateWithProperty("getter_call", createProperty(cardinality), "expected.txt");
    }

    private void interface_getter(String cardinality, String expectedOutput) throws IOException
    {
        testTemplateWithProperty("interface_getter", createProperty(cardinality), expectedOutput);
    }

    private void class_getter(String cardinality, String expectedOutputFileName) throws IOException
    {
        testTemplateWithProperty("field_getter", createProperty(cardinality), expectedOutputFileName);
    }

    private static Property createProperty(String cardinality)
    {
        return Property.createProperty(
            "SomeProperty",
            null, null, emptyList(), false,
            TempNamedType.create("someType", cardinality), null, null);
    }
}


==> cml-compiler/cml-generator/src/test/java/templates/lang/common/LangTest.java

package templates.lang.common;

import cml.language.features.TempConcept;
import cml.language.generated.Association;
import cml.language.generated.NamedElement;
import cml.language.generated.Property;
import cml.language.generated.Type;
import com.google.common.io.Resources;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import org.stringtemplate.v4.ST;

import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.nio.charset.Charset;
import java.util.Collection;

import static java.lang.String.format;
import static java.util.Arrays.asList;
import static junit.framework.TestCase.assertEquals;
import static junit.framework.TestCase.assertNotNull;

@RunWith(Parameterized.class)
public abstract class LangTest extends TemplateTest
{
    private static final Charset OUTPUT_FILE_ENCODING = Charset.forName("UTF-8");
    private static final String EXPECTED_OUTPUT_PATH = "/%s/%s/%s/%s";
    private static final String LANG_GROUP_PATH = "lang/%s";
    private static final String TEMPLATES_LANG = "templates/lang";

    @Parameters
    public static Collection<String> targetLanguageExtension()
    {
        return asList("java", "py");
    }

    private final String targetLanguageExtension;

    LangTest(String targetLanguageExtension)
    {
        this.targetLanguageExtension = targetLanguageExtension;
```

```
    }

    protected abstract String getExpectedOutputPath();

    private String getTargetLanguageExtension()
    {
        return targetLanguageExtension;
    }

    @Override
    protected String getTemplatePath()
    {
        return format(LANG_GROUP_PATH, targetLanguageExtension);
    }

    void testAssociationClass(Association association, String expectedOutputPath) throws IOException
    {
        final String templateName = "association_class";
        final ST template = getTemplate(templateName);
        assertNotNull("Expected template: " + templateName, template);

        template.add("association", association);

        final String result = template.render();
        assertThatOutputMatches(expectedOutputPath + "." + getTargetLanguageExtension(), result);
    }

    void testConceptClass(TempConcept concept, String expectedOutputPath) throws IOException
    {
        final String templateName = "class";
        final ST template = getTemplate(templateName);
        assertNotNull("Expected template: " + templateName, template);

        template.add("concept", concept);

        final String result = template.render();
        assertThatOutputMatches(expectedOutputPath + "." + getTargetLanguageExtension(), result);
    }

    void testTemplateWithConcept(String templateName, TempConcept concept, String expectedOutputPath)
        throws IOException
    {
        testTemplate(templateName, "concept", concept, expectedOutputPath);
    }

    void testTemplateWithProperty(String templateName, Property property, String expectedOutputPath)
        throws IOException
    {
        testTemplate(templateName, "property", property, expectedOutputPath);
    }

    void testTemplateWithType(String templateName, Type type, String expectedOutputPath) throws IOException
    {
        testTemplate(templateName, "type", type, expectedOutputPath);
    }

    void testTemplateWithNamedElement(String templateName, NamedElement namedElement, String expectedOutputPath)
        throws IOException
    {
        testTemplate(templateName, "named_element", namedElement, expectedOutputPath);
    }

    void testTemplateWithNamedElement(String templateName, Type type, String expectedOutputPath)
        throws IOException
    {
        testTemplate(templateName, "named_element", type, expectedOutputPath);
    }

    void assertThatOutputMatches(String expectedOutputPath, String actualOutput) throws IOException
    {
        expectedOutputPath = format(
            EXPECTED_OUTPUT_PATH,
            TEMPLATES_LANG,
            targetLanguageExtension,
            getExpectedOutputPath(),
            expectedOutputPath);

        final URL expectedOutputResource = getClass().getResource(expectedOutputPath);
        assertNotNull("Expected output resource must exist: " + expectedOutputPath, expectedOutputResource);

        final String expectedOutput = Resources.toString(expectedOutputResource, OUTPUT_FILE_ENCODING);
        assertEquals(expectedOutputPath, expectedOutput, actualOutput);
    }

    private void testTemplate(String templateName, String paramName, Object paramValue, String expectedOutputPath)
        throws IOException
    {
        if (!getExpectedOutputPath().endsWith(templateName))
        {
            expectedOutputPath = templateName + File.separator + expectedOutputPath;
        }

        final ST template = getTemplate(templateName);
        assertNotNull("Expected template: " + templateName, template);

        template.add(paramName, paramValue);

        final String result = template.render();

        assertThatOutputMatches(expectedOutputPath, result);
    }
}
```

==> cml-compiler/cml-generator/src/test/java/templates/lang/common/NamedTypeTest.java

```
package templates.lang.common;

import cml.language.generated.ValueType;
import cml.language.types.TempNamedType;
import cml.templates.NameRenderer;
import org.junit.Test;
import org.stringtemplate.v4.ST;

import java.io.IOException;
import java.util.Locale;

import static cml.primitives.Types.PRIMITIVE_TYPE_NAMES;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

public class NamedTypeTest extends LangTest
{
    public NamedTypeTest(String targetLanguageExtension)
    {
        super(targetLanguageExtension);
    }
```

```
    @Override
    protected String getExpectedOutputPath()
    {
        return "type";
    }

    @Test
    public void primitive_types() throws IOException
    {
        for (String typeName : PRIMITIVE_TYPE_NAMES)
        {
            final ValueType type = ValueType.createValueType("", typeName);
            final String expectedOutputPath = NameRenderer.pascalCase(Locale.getDefault(), typeName) + ".txt";
            testTemplateWithNamedElement("type_name", type, expectedOutputPath);
        }
    }

    @Test
    public void type_name()
    {
        for (String name : commonNameFormats)
        {
            type_name(name);
        }
    }

    private void type_name(String name)
    {
        final ST template = getTemplate("type_name");

        template.add("named_element", TempNamedType.create(name, null));

        final String result = template.render();

        assertThat(result, is(pascalCase(name)));
    }

}


==> cml-compiler/cml-generator/src/test/java/templates/lang/common/TemplateTest.java

package templates.lang.common;


import cml.io.Console;
import cml.io.FileSystem;
import cml.io.ModuleManager;
import cml.templates.NameRenderer;
import cml.templates.TemplateGroupFile;
import org.junit.Before;
import org.junit.experimental.theories.Theories;
import org.junit.runner.RunWith;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroupFile;

import java.util.Collection;
import java.util.Locale;

import static java.lang.String.format;
import static java.util.Arrays.asList;
import static java.util.Collections.unmodifiableCollection;

@RunWith(Theories.class)
public abstract class TemplateTest
{
    static final Collection<String> commonNameFormats = unmodifiableCollection(asList(
        "SomeName",
        "someName",
        "some_name",
        "some-name"
    ));

    private static final String TEMPLATE_GROUP_PATH = "%s:/%s.stg";
    private static final String MODULE_NAME = "cml_base";
    private static final String CML_MODULES_BASE_DIR = "../../cml-modules";

    private STGroupFile groupFile;

    @Before
    public void setUp()
    {
        groupFile = createTemplateGroupFile(getTemplatePath());

        groupFile.registerRenderer(String.class, new NameRenderer());
    }

    ST getTemplate(String templateName)
    {
        return groupFile.getInstanceOf(templateName);
    }

    /**
     * The path to the template file without the file extension.
     * <p>
     * Used to load the template file and to load the expected output files.
     **/
    protected abstract String getTemplatePath();

    static String pascalCase(String name)
    {
        return NameRenderer.pascalCase(Locale.getDefault(), name);
    }

    static String camelCase(String name)
    {
        return NameRenderer.camelCase(Locale.getDefault(), name);
    }

    static String underscoreCase(String name)
    {
        return NameRenderer.underscoreCase(Locale.getDefault(), name);
    }

    static TemplateGroupFile createTemplateGroupFile(String templatePath)
    {
        final Console console = Console.createSystemConsole();
        final FileSystem fileSystem = FileSystem.create(console);
        final ModuleManager moduleManager = ModuleManager.create(console, fileSystem);

        moduleManager.addBaseDir(CML_MODULES_BASE_DIR);

        TemplateGroupFile.setModuleManager(moduleManager);

        return new TemplateGroupFile(format(TEMPLATE_GROUP_PATH, MODULE_NAME, templatePath));
    }
```

```
}
```

==> cml-compiler/cml-generator/src/test/resources/modules/expressions/source/main.cml

```
concept Employee
{
    number: Integer;
    name: String;
    employer: Organization;

    /employerName = employer.name;

    /selfEmployee = self;
    /alias = name;
}

concept Organization
{
    name: String;
    employees: Employee*;

    /employeeNames = employees.name;
    /employerNames = employees.employer.name;

    /employers = employees.employer;

    /employeeNumbers = employees.number;
}

association Employment
{
    Employee.employer;
    Organization.employees;
}
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/field/field_type/optional.txt

```
@Nullable SomeName
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/field/field_type/required.txt

```
SomeName
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/field/field_type/sequence.txt

```
List<SomeName>
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/field_getter/optional.txt

```
public Optional<SomeType> getSomeProperty()
{
    return Optional.ofNullable(someProperty);
}
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/field_getter/required.txt

```
public SomeType getSomeProperty()
{
    return someProperty;
}
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/field_getter/sequence.txt

```
public List<SomeType> getSomeProperty()
{
    return Collections.unmodifiableList(someProperty);
}
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/getter_call/expected.txt

```
getSomeProperty()
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/getter_type/optional.txt

```
Optional<SomeName>
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/getter_type/required.txt

```
SomeName
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/getter_type/sequence.txt

```
List<SomeName>
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/getter_type/sequence_integer.txt

```
List<Integer>
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/interface_getter/optional.txt

```
Optional<SomeType> getSomeProperty();
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/interface_getter/required.txt

```
SomeType getSomeProperty();
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/getter/interface_getter/sequence.txt

```
List<SomeType> getSomeProperty();
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/type/type_name/Boolean.txt

```
boolean
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/type/type_name/Byte.txt

```
byte
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/type/type_name/Decimal.txt

```
BigDecimal
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/type/type_name/Double.txt

```
double
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/type/type_name/Float.txt

```
float
```

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/type/type_name/Integer.txt

```
int
```

```
==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/type/type_name/Long.txt

long

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/type/type_name/Short.txt

short

==> cml-compiler/cml-generator/src/test/resources/templates/lang/java/type/type_name/String.txt

String

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/field/field_name/expected.txt

self.__some_name

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/field/field_type/optional.txt

Optional[SomeName]

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/field/field_type/required.txt

SomeName

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/field/field_type/sequence.txt

List[SomeName]

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/field_getter/optional.txt

@property
def some_property(self) -> 'Optional[SomeType]':
    return some_property

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/field_getter/required.txt

@property
def some_property(self) -> 'SomeType':
    return some_property

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/field_getter/sequence.txt

@property
def some_property(self) -> 'List[SomeType]':
    return some_property

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/getter_call/expected.txt

some_property

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/getter_type/optional.txt

Optional[SomeName]

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/getter_type/required.txt

SomeName

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/getter_type/sequence.txt

List[SomeName]

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/getter_type/sequence_integer.txt

List[int]

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/interface_getter/optional.txt

@abstractproperty
def some_property(self) -> 'Optional[SomeType]':
    pass

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/interface_getter/required.txt

@abstractproperty
def some_property(self) -> 'SomeType':
    pass

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/getter/interface_getter/sequence.txt

@abstractproperty
def some_property(self) -> 'List[SomeType]':
    pass

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/type/type_name/Boolean.txt

bool

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/type/type_name/Byte.txt

int

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/type/type_name/Decimal.txt

Decimal

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/type/type_name/Double.txt

float

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/type/type_name/Float.txt

float

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/type/type_name/Integer.txt

int

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/type/type_name/Long.txt

int

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/type/type_name/Short.txt

int

==> cml-compiler/cml-generator/src/test/resources/templates/lang/py/type/type_name/String.txt

str

==> cml-compiler/cml-io/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>cml</groupId>
        <artifactId>cml-compiler</artifactId>
        <version>master-SNAPSHOT</version>
    </parent>

    <artifactId>cml-io</artifactId>
    <version>master-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.5</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

</project>

==> cml-compiler/cml-io/src/main/java/cml/io/Console.java

package cml.io;

import static java.lang.String.format;

public interface Console
{
    default void println()
    {
        println("");
    }

    void print(String message, Object... args);
    void println(String message, Object... args);
    void info(final String message, final Object... args);
    void error(final String message, final Object... args);

    static Console createSystemConsole()
    {
        return new SystemConsole();
    }

    static Console createStringConsole()
    {
        return new StringConsole();
    }
}

class SystemConsole implements Console
{
    @Override
    public void print(final String message, final Object... args)
    {
        System.out.print(format(message, args));
    }

    @Override
    public void println(final String message, final Object... args)
    {
        System.out.println(format(message, args));
    }

    @Override
    public void info(String message, Object... args)
    {
//        System.out.println(String.format("Info: " + message, args));
    }

    @Override
    public void error(final String message, final Object... args)
    {
        System.out.println(format("Error: " + message, args));
    }
}

class StringConsole implements Console
{
    private StringBuilder str = new StringBuilder();

    @Override
    public void print(final String message, final Object... args)
    {
        str.append(format(message, args));
    }

    @Override
    public void println(final String message, final Object... args)
    {
        str.append(format(message, args)).append('\n');
    }

    @Override
    public void info(String message, Object... args)
    {
    }

    @Override
    public void error(final String message, final Object... args)
    {
        str.append(format("Error: " + message, args)).append('\n');
    }

    @Override
    public String toString()
    {
        return str.toString();
    }
}


==> cml-compiler/cml-io/src/main/java/cml/io/Directory.java

package cml.io;

import java.io.File;

public class Directory
```

```
{
    private final String path;

    Directory(final String path)
    {
        this.path = path;
    }

    public String getName()
    {
        return new File(path).getName();
    }

    public String getPath()
    {
        return path;
    }
}


==> cml-compiler/cml-io/src/main/java/cml/io/FileSystem.java

package cml.io;

import org.apache.commons.io.FileUtils;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import static java.util.Arrays.asList;
import static java.util.Collections.sort;
import static java.util.Collections.unmodifiableList;
import static org.apache.commons.io.FileUtils.forceMkdir;
import static org.jooq.lambda.Seq.seq;

public interface FileSystem
{
    Optional<Directory> findDirectory(Directory baseDir, String dirName);
    Optional<Directory> findDirectory(String path);

    List<Directory> findSubDirs(Directory baseDir);

    List<SourceFile> findSourceFiles(Directory sourceDir);
    Optional<SourceFile> findSourceFile(Directory directory, String name);

    Optional<URL> getURL(Directory directory, String path);

    String extractParentPath(String path);
    String extractName(String path);

    void createFile(String path, String content);
    void cleanDirectory(Directory directory);

    static FileSystem create(Console console)
    {
        return new FileSystemImpl(console);
    }
}

class FileSystemImpl implements FileSystem
{
    private static final String EXCEPTION_FILE_CREATION_FAILED = "Unexpected exception. File should have been created: ";
    private static final String EXCEPTION_DIRECTORY_DELETION_FAILED = "Unexpected exception. Dir should have been deleted: ";

    private static final String ERROR_BUILDING_FILE_URL = "Error building file URL: %s";

    private static final String FILE_URL_PREFIX = "file://";
    private static final String CML_FILE_EXT = "cml";

    private final Console console;

    FileSystemImpl(Console console)
    {
        this.console = console;
    }

    @Override
    public Optional<Directory> findDirectory(Directory baseDir, String dirName)
    {
        return findDirectory(baseDir.getPath() + File.separator + dirName);
    }

    @Override
    public List<Directory> findSubDirs(Directory baseDir)
    {
        final String canonicalPath = getCanonicalFile(baseDir.getPath()).getPath();
        final File[] subDirs = new File(canonicalPath).listFiles(File::isDirectory);
        final List<File> subDirList = asList(subDirs == null ? new File[0] : subDirs);

        return seq(subDirList).map(subDir -> findDirectory(subDir.getPath()))
                              .filter(Optional::isPresent)
                              .map(Optional::get)
                              .toList();
    }

    @Override
    public Optional<Directory> findDirectory(final String path)
    {
        final File file = getCanonicalFile(path);
        final Directory directory = file.isDirectory() ? new Directory(path) : null;
        return Optional.ofNullable(directory);
    }

    @Override
    public Optional<URL> getURL(Directory baseDir, String path)
    {
        if (!path.startsWith(File.separator))
        {
            path = File.separator + path;
        }

        final File file = getCanonicalFile(baseDir.getPath() + path);

        if (file.exists())
        {
            try
            {
                final URL url = new URL(FILE_URL_PREFIX + file.getPath());

                return Optional.of(url);
            }
```

```
            catch (MalformedURLException exception)
            {
                console.error(ERROR_BUILDING_FILE_URL, exception.getMessage());
            }
        }

        return Optional.empty();
    }

    @Override
    public List<SourceFile> findSourceFiles(Directory sourceDir)
    {
        final String canonicalPath = getCanonicalFile(sourceDir.getPath()).getPath();
        final File[] files = new File(canonicalPath).listFiles(File::isFile);
        final List<File> fileList = asList(files == null ? new File[0] : files);

        final List<SourceFile> sourceFiles = seq(fileList).map(file -> findSourceFile(sourceDir, file.getName()))
                                                          .filter(Optional::isPresent)
                                                          .map(Optional::get)
                                                          .filter(file -> file.getExtension().equals(CML_FILE_EXT))
                                                          .toList();

        final List<SourceFile> all = new ArrayList<>(sourceFiles);
        for (Directory subDir: findSubDirs(sourceDir))
        {
            all.addAll(findSourceFiles(subDir));
        }

        sort(all);

        return unmodifiableList(all);
    }

    @Override
    public Optional<SourceFile> findSourceFile(final Directory directory, final String name)
    {
        final File file = getCanonicalFile(directory.getPath() + File.separator + name);
        final SourceFile sourceFile = file.isFile() ? new SourceFile(file.getPath()) : null;
        return Optional.ofNullable(sourceFile);
    }

    @Override
    public String extractParentPath(String path)
    {
        final File file = getCanonicalFile(path);
        return file.getParentFile().getPath();
    }

    @Override
    public String extractName(String path)
    {
        final File file = getCanonicalFile(path);
        return file.getName();
    }

    @Override
    public void createFile(final String path, final String content)
    {
        try
        {
            final File file = new File(path);

            if (!file.getParentFile().exists())
            {
                forceMkdir(file.getParentFile());
            }

            try (final PrintWriter output = new PrintWriter(file))
            {
                output.print(content);
            }
        }
        catch (final IOException exception)
        {
            throw new RuntimeException(EXCEPTION_FILE_CREATION_FAILED + path, exception);
        }
    }

    @Override
    public void cleanDirectory(final Directory directory)
    {
        try
        {
            FileUtils.cleanDirectory(new File(directory.getPath()));
        }
        catch (final IOException exception)
        {
            throw new RuntimeException(EXCEPTION_DIRECTORY_DELETION_FAILED + directory.getPath(), exception);
        }
    }

    private static File getCanonicalFile(String path)
    {
        File file = new File(path);
        try
        {
            return file.getCanonicalFile();
        }
        catch (IOException exception)
        {
            return file.getAbsoluteFile();
        }
    }
}
```

==> cml-compiler/cml-io/src/main/java/cml/io/ModuleManager.java

```
package cml.io;

import java.io.File;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import static java.util.Collections.emptyList;

public interface ModuleManager
{
    void clearBaseDirs();
    void addBaseDir(String path);

    Optional<Directory> findModuleDir(String moduleName);

    Optional<Directory> findSourceDir(String moduleName);
    List<SourceFile> findSourceFiles(String moduleName);
```

```
    Optional<URL> findTemplateFile(String path);

    String getModuleName(String path);
    String getModuleRelativePath(String path);
    String getModulePath(String moduleName, String relativePath);

    static ModuleManager create(Console console, FileSystem fileSystem)
    {
        return new ModuleManagerImpl(console, fileSystem);
    }
}

class ModuleManagerImpl implements ModuleManager
{
    private static final String MODULE_BASE_DIR_NOT_FOUND = "module base dir not found: %s";
    private static final String UNABLE_TO_FIND_MODULE = "unable to find module: %s";

    private static final String NO_TEMPLATES_DIR = "no templates dir for module: %s";

    private static final String MODULE_NAME_SEPARATOR = ":";
    private static final String SOURCE_DIR = "source";
    private static final String TEMPLATES_DIR = "templates";

    private final List<Directory> baseDirList = new ArrayList<>();

    private final Console console;
    private final FileSystem fileSystem;

    ModuleManagerImpl(Console console, FileSystem fileSystem)
    {
        this.console = console;
        this.fileSystem = fileSystem;
    }

    @Override
    public void clearBaseDirs()
    {
        baseDirList.clear();
    }

    @Override
    public void addBaseDir(String path)
    {
        final Optional<Directory> baseDir = fileSystem.findDirectory(path);

        if (baseDir.isPresent())
        {
            baseDirList.add(baseDir.get());
        }
        else
        {
            console.error(MODULE_BASE_DIR_NOT_FOUND, path);
        }
    }

    @Override
    public Optional<Directory> findModuleDir(String moduleName)
    {
        for (Directory baseDir: baseDirList)
        {
            final Optional<Directory> moduleDir = fileSystem.findDirectory(baseDir, moduleName);

            if (moduleDir.isPresent())
            {
                return moduleDir;
            }
        }

        return Optional.empty();
    }

    public Optional<Directory> findSourceDir(String moduleName)
    {
        final Optional<Directory> moduleDir = findModuleDir(moduleName);

        if (moduleDir.isPresent())
        {
            return fileSystem.findDirectory(moduleDir.get(), SOURCE_DIR);
        }
        else
        {
            console.error(UNABLE_TO_FIND_MODULE, moduleName);

            return Optional.empty();
        }
    }

    @Override
    public List<SourceFile> findSourceFiles(String moduleName)
    {
        final Optional<Directory> sourceDir = findSourceDir(moduleName);

        if (sourceDir.isPresent())
        {
            return fileSystem.findSourceFiles(sourceDir.get());
        }
        else
        {
            console.error(UNABLE_TO_FIND_MODULE, moduleName);

            return emptyList();
        }
    }

    @Override
    public Optional<URL> findTemplateFile(String path)
    {
        final Optional<Directory> moduleDir = findModuleDir(getModuleName(path));

        if (moduleDir.isPresent())
        {
            final Optional<Directory> templatesDir = fileSystem.findDirectory(moduleDir.get(), TEMPLATES_DIR);

            if (templatesDir.isPresent())
            {
                return fileSystem.getURL(templatesDir.get(), getModuleRelativePath(path));
            }
            else
            {
                console.info(NO_TEMPLATES_DIR, getModuleName(path));
            }
        }
        else
        {
            console.error(UNABLE_TO_FIND_MODULE, getModuleName(path));
        }
```

```
        return Optional.empty();
    }

    @Override
    public String getModuleName(String path)
    {
        final String[] pair = path.split(MODULE_NAME_SEPARATOR);

        return pair.length == 2 ? pair[0] : "unspecified-module";
    }

    @Override
    public String getModuleRelativePath(String path)
    {
        final String[] pair = path.split(MODULE_NAME_SEPARATOR);

        return pair.length == 2 ? pair[1] : path;
    }

    @Override
    public String getModulePath(String moduleName, String relativePath)
    {
        if (!relativePath.startsWith(File.separator))
        {
            relativePath = File.separator + relativePath;
        }

        return moduleName + MODULE_NAME_SEPARATOR + relativePath;
    }
}
```

==> cml-compiler/cml-io/src/main/java/cml/io/SourceFile.java

```
package cml.io;

public class SourceFile implements Comparable<SourceFile>
{
    private static final char EXTENSION_SEPARATOR = '.';

    private final String path;

    SourceFile(final String path)
    {
        this.path = path;
    }

    public String getPath()
    {
        return path;
    }

    String getExtension()
    {
        return path.substring(path.lastIndexOf(EXTENSION_SEPARATOR) + 1);
    }

    @Override
    public int compareTo(final SourceFile other)
    {
        return path.compareTo(other.path);
    }
}
```

==> cml-compiler/cml-language/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>cml</groupId>
        <artifactId>cml-compiler</artifactId>
        <version>master-SNAPSHOT</version>
    </parent>

    <artifactId>cml-language</artifactId>
    <version>master-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-language-generated</artifactId>
            <version>master-SNAPSHOT</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-io</artifactId>
            <version>${project.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.antlr</groupId>
            <artifactId>antlr4-runtime</artifactId>
            <version>4.6</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>15.0</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-templates</artifactId>
            <version>${project.version}</version>
            <scope>test</scope>
```

```
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.antlr</groupId>
                <artifactId>antlr4-maven-plugin</artifactId>
                <version>4.6</version>
                <executions>
                    <execution>
                        <id>antlr</id>
                        <goals>
                            <goal>antlr4</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

```
==> cml-compiler/cml-language/src/main/antlr4/cml/language/grammar/CML.g4

grammar CML;

@header
{
import org.jooq.lambda.*;
import cml.language.features.*;
import cml.language.expressions.*;
import cml.language.types.*;
import cml.language.foundation.*;
import cml.language.generated.*;
}

import CompilationUnits, Ignored;



==> cml-compiler/cml-language/src/main/antlr4/imports/Annotations.g4

grammar Annotations;

import Names;

annotationList:
  '[' ( annotationDeclaration (',' annotationDeclaration)* )? ']';

annotationDeclaration:
  NAME;



==> cml-compiler/cml-language/src/main/antlr4/imports/Associations.g4

grammar Associations;

import Names, Types;

associationDeclaration
  returns [Association association]:
  ASSOCIATION NAME
  '{' (associationEndDeclaration ';')* '}';

associationEndDeclaration
  returns [Association association]:
  conceptName=NAME '.' propertyName=NAME
  (':' typeDeclaration)?;



==> cml-compiler/cml-language/src/main/antlr4/imports/CompilationUnits.g4

grammar CompilationUnits;

import Modules, Concepts, Associations, Tasks, Templates, Functions;

compilationUnit:
  declarations*;

declarations
  : moduleDeclaration
  | conceptDeclaration
  | associationDeclaration
  | taskDeclaration
  | templateDeclaration
  | functionDeclaration;



==> cml-compiler/cml-language/src/main/antlr4/imports/Concepts.g4

grammar Concepts;

import Names, Properties;

conceptDeclaration returns [TempConcept concept]:
  (ABSTRACTION | CONCEPT) NAME
  (':' generalizations)?
  (';' | propertyList);

generalizations:
  NAME (',' NAME)*;



==> cml-compiler/cml-language/src/main/antlr4/imports/Expressions.g4

grammar Expressions;

import Literals, Types;

expression returns [Expression expr]
  : literalExpression
  | pathExpression
  | lambdaExpression
  | invocationExpression
  | comprehensionExpression

  // Grouping
  | '(' inner=expression ')'
```

```
  // Arithmetic Expressions:
  | operator=('+' | '-') expression
  | <assoc=right> expression operator='^' expression
  | expression operator=('*' | '/' | '%') expression
  | expression operator=('+' | '-') expression

  // String Concatenation:
  | expression operator='&' expression

  // Relational Expressions:
  | expression operator=('<' | '<=' | '>' | '>=') expression
  | expression operator=('==' | '!=') expression

  // Referential Expressions:
  | expression operator=('===' | '!==') expression

  // Type-Checking:
  | expression operator=(IS | ISNT) type=typeDeclaration

  // Type-Casting:
  | expression operator=(AS | ASB | ASQ) type=typeDeclaration

  // Logical Expressions:
  | operator=NOT expression
  | expression operator=AND expression
  | expression operator=OR expression
  | expression operator=XOR expression
  | expression operator=IMPLIES expression

  // Conditional Expressions:
  | IF cond=expression
    THEN then=expression ELSE else_=expression
  | then=expression conj=(GIVEN | UNLESS) cond=expression
  | then=expression (ORQ | XORQ) else_=expression

  // Let Expressions:
  | LET var=NAME '=' varExpr=expression IN resultExpr=expression;

pathExpression returns [Path path]:
  NAME ('.' NAME)*;

lambdaExpression returns[Lambda lambda]:
  '{' lambdaParameterList? expression '}';

lambdaParameterList returns[Seq<String> params]:
  NAME (',' NAME)* '->';

invocationExpression returns [Invocation invocation]:
  NAME '(' expression (',' expression)* ')' lambdaExpression?;

comprehensionExpression returns [Comprehension comprehension]:
  (pathExpression |
   FOR enumeratorDeclaration (',' enumeratorDeclaration)*)
  queryStatement+;

enumeratorDeclaration returns [Enumerator enumerator]:
  var=NAME IN pathExpression;

queryStatement returns [Query query]:
  '|' (NAME | keywordExpression+);

keywordExpression returns [Keyword keyword]:
  NAME ':' lambdaParameterList? expression;
```

==> cml-compiler/cml-language/src/main/antlr4/imports/Functions.g4

```
grammar Functions;

import Names, Types, Annotations, Expressions;

functionDeclaration returns [Function function]:
  annotationList?
  FUNCTION name=NAME
  typeParams=typeParameterList?
  params=functionParameterList
  ('->' resultType=typeDeclaration)? ('=' expression)? ';';

functionParameterList returns [Seq<FunctionParameter> params]:
  '(' functionParameterDeclaration? (',' functionParameterDeclaration)* ')';

functionParameterDeclaration returns [FunctionParameter param]:
 name=NAME ':' type=typeDeclaration;
```

==> cml-compiler/cml-language/src/main/antlr4/imports/Ignored.g4

```
lexer grammar Ignored;

// import: none

// Ignoring Whitespace:

WS:
  ( ' ' | '\t' | '\f' | '\n' | '\r' )+ -> skip;


// Ignoring Comments:

COMMENT:
  (('//' | '--') .*? '\n' | '/*' .*? '*/' ) -> skip;
```

==> cml-compiler/cml-language/src/main/antlr4/imports/Literals.g4

```
grammar Literals;

import Names;

literalExpression returns [Literal literal]:
  BOOLEAN | STRING |
  INTEGER | LONG | SHORT | BYTE | DECIMAL |
  FLOAT | DOUBLE;

STRING:
  '"' (ESC | . )*? '"';

fragment ESC: '\\'[btnr"\\];

INTEGER:
  ('0'..'9')+;

LONG:
```

```
  ('0'..'9')+ 'l';

SHORT:
  ('0'..'9')+ 's';

BYTE:
  ('0'..'9')+ 'b';

DECIMAL:
  ('0'..'9')* '.' ('0'..'9')+;

FLOAT:
  ('0'..'9')* '.' ('0'..'9')+ 'f';

DOUBLE:
  ('0'..'9')* '.' ('0'..'9')+ 'd';


==> cml-compiler/cml-language/src/main/antlr4/imports/Modules.g4

grammar Modules;

import Names;

moduleDeclaration returns [TempModule module]:
  MODULE NAME '{' importDeclaration* '}';

importDeclaration returns [Import _import]:
  IMPORT NAME ';';

==> cml-compiler/cml-language/src/main/antlr4/imports/Names.g4

lexer grammar Names;

// import: none

// All reserved words must be declared before NAME.
// Otherwise, they are recognized as a NAME instead.

FOR: 'for';
IN: 'in';
AS: 'as';
ASB: 'as!';
ASQ: 'as?';
IS: 'is';
ISNT: 'isnt';
IF: 'if';
THEN: 'then';
ELSE: 'else';
GIVEN: 'given';
UNLESS: 'unless';
LET: 'let';
ORQ: 'or?';
XORQ: 'xor?';
BOOLEAN: 'true' | 'false';
AND: 'and';
OR: 'or';
XOR: 'xor';
IMPLIES: 'implies';
NOT: 'not';
TEMPLATE: '@template';
FUNCTION: '@function';
ABSTRACTION: '@abstraction';
CONCEPT: '@concept';
TASK: '@task';
ASSOCIATION: '@association';
MODULE: '@module';
IMPORT: '@import';

NAME:
  ('A'..'Z' | 'a'..'z')
  ( 'A'..'Z' | 'a'..'z' | '0'..'9' | '-' )*;



==> cml-compiler/cml-language/src/main/antlr4/imports/Properties.g4

grammar Properties;

import Names, Types, Expressions;

propertyList:
  '{' (propertyDeclaration ';')* '}';

propertyDeclaration returns [Property property]:
 DERIVED? NAME (':' typeDeclaration)? ('=' expression)?;

DERIVED: '/';

==> cml-compiler/cml-language/src/main/antlr4/imports/Tasks.g4

grammar Tasks;

import Names, Properties;

taskDeclaration returns [Task task]:
  TASK NAME
  constructorDeclaration?
  (';' | propertyList);

constructorDeclaration: ':' NAME;



==> cml-compiler/cml-language/src/main/antlr4/imports/Templates.g4

grammar Templates;

import Names, Functions;

templateDeclaration returns [Template template]:
  TEMPLATE
  functionDeclaration;




==> cml-compiler/cml-language/src/main/antlr4/imports/Types.g4

grammar Types;

import Names;

typeDeclaration returns [Type type]
  : name=NAME cardinality? // named type
  | tuple=tupleTypeDeclaration // tuple type
```

```
  | params=tupleTypeDeclaration '->' result=typeDeclaration // function type
  | '(' inner=typeDeclaration ')';

cardinality:
  ('?' | '*');

tupleTypeDeclaration returns [TupleType type]:
  '(' ( tupleTypeElementDeclaration (',' tupleTypeElementDeclaration)* )? ')' cardinality?;

tupleTypeElementDeclaration returns [TupleTypeElement element]:
  (name=NAME ':')? type=typeDeclaration;

typeParameterList returns [Seq<TypeParameter> params]:
  '<' typeParameter (',' typeParameter)* '>';

typeParameter returns [TypeParameter param]:
  name=NAME;
```

==> cml-compiler/cml-language/src/main/java/cml/language/expressions/Comprehension.java

```
package cml.language.expressions;

import cml.language.generated.Expression;
import cml.language.types.TempNamedType;
import org.jetbrains.annotations.Nullable;
import org.jooq.lambda.Seq;

import java.util.List;
import java.util.Optional;
import java.util.stream.Stream;

import static java.util.Collections.emptyList;
import static java.util.Collections.unmodifiableList;
import static java.util.stream.Collectors.toList;
import static org.jooq.lambda.Seq.seq;

public class Comprehension extends ExpressionBase
{
    private final @Nullable Path path;
    private final List<Enumerator> enumerators;
    private final List<Query> queries;

    public Comprehension(
        @SuppressWarnings("NullableProblems") final Path path,
        final Stream<Query> queries)
    {
        this.path = path;
        this.enumerators = emptyList();
        this.queries = queries.collect(toList());
    }

    public Comprehension(
        final Stream<Enumerator> enumerators,
        final Stream<Query> queries)
    {
        this.path = null;
        this.enumerators = enumerators.collect(toList());
        this.queries = queries.collect(toList());
    }

    public Optional<Path> getPath()
    {
        return Optional.ofNullable(path);
    }

    public List<Enumerator> getEnumerators()
    {
        return unmodifiableList(enumerators);
    }

    public List<Query> getQueries()
    {
        return unmodifiableList(queries);
    }

    public List<Expression> getExpressions()
    {
        return seq(enumerators).map(Enumerator::getPath)
                               .collect(toList());
    }

    public Seq<String> getEnumeratorVariablesForQuery(final Query query)
    {
        return seq(enumerators).map(Enumerator::getVariable)
                               .concat(query.getExpressionParams());
    }

    @Override
    public String getKind()
    {
        return "comprehension";
    }

    @Override
    public TempNamedType getType()
    {
        throw new UnsupportedOperationException("Unexpected type inference of a comprehension. Should have been transformed into an invocation.");
    }

    @Override
    public String getDiagnosticId()
    {
        throw new UnsupportedOperationException("Unexpected diagnostic of a comprehension. Should have been transformed into an invocation.");
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/expressions/Enumerator.java

```
package cml.language.expressions;

public class Enumerator
{
    private final String variable;
    private final Path path;

    public Enumerator(String variable, Path path)
    {
        this.variable = variable;
        this.path = path;
    }
```

```
    public String getVariable()
    {
        return variable;
    }

    public Path getPath()
    {
        return path;
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/expressions/ExpressionBase.java

package cml.language.expressions;

import cml.language.generated.*;

import java.util.List;
import java.util.Optional;

import static java.util.Collections.emptyList;
import static org.jooq.lambda.Seq.seq;

public abstract class ExpressionBase implements Expression
{
    protected final Expression expression;

    ExpressionBase()
    {
        this(null, emptyList());
    }

    ExpressionBase(Scope scope)
    {
        this(scope, emptyList());
    }

    ExpressionBase(List<Expression> operands)
    {
        this(null, operands);
    }

    ExpressionBase(Scope parent, List<Expression> operands)
    {
        final Element element = Element.extendElement(this);
        final ModelElement modelElement = ModelElement.extendModelElement(this, element, parent, null);
        final Scope scope = Scope.extendScope(this, element, modelElement, seq(operands).map(s -> (ModelElement)s).toList());

        expression = Expression.extendExpression(this, element, modelElement, scope);
    }

    @Override
    public Optional<Location> getLocation()
    {
        return expression.getLocation();
    }

    @Override
    public Optional<Scope> getParent()
    {
        return expression.getParent();
    }

    @Override
    public Optional<Model> getModel()
    {
        return expression.getModel();
    }

    @Override
    public Optional<Module> getModule()
    {
        return expression.getModule();
    }

    @Override
    public List<ModelElement> getMembers()
    {
        return expression.getMembers();
    }

    @Override
    public String getKind()
    {
        return expression.getKind();
    }

    @Override
    public Type getType()
    {
        return expression.getType();
    }

    @Override
    public Type getMatchingResultType()
    {
        return expression.getMatchingResultType();
    }

    @Override
    public boolean isBoolean()
    {
        return expression.isBoolean();
    }

    @Override
    public boolean isNumeric()
    {
        return expression.isNumeric();
    }

    @Override
    public boolean isFloat()
    {
        return expression.isFloat();
    }

    @Override
    public boolean isArithmetic()
    {
        return expression.isArithmetic();
    }

    @Override
    public boolean isRelational()
    {
```

```
        return expression.isRelational();
    }

    @Override
    public boolean isReferential()
    {
        return expression.isReferential();
    }

    @Override
    public boolean isPrimitive()
    {
        return expression.isPrimitive();
    }

    @Override
    public List<Expression> getOperands()
    {
        return expression.getOperands();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/expressions/InvalidExpression.java

```
package cml.language.expressions;

import cml.language.generated.Type;
import org.jetbrains.annotations.Nullable;

import java.util.Optional;

import static cml.language.generated.UndefinedType.createUndefinedType;
import static java.lang.String.format;
import static java.util.Optional.ofNullable;

public class InvalidExpression extends ExpressionBase
{
    private final @Nullable String text;

    public InvalidExpression(final @Nullable String text)
    {
        this.text = text;
    }

    public Optional<String> getText()
    {
        return ofNullable(text);
    }

    @Override
    public String getKind()
    {
        return "invalid";
    }

    @Override
    public Type getType()
    {
        return createUndefinedType(format("Unable to infer type of invalid expression: '%s'", text));
    }

    @Override
    public String getDiagnosticId()
    {
        return text;
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/expressions/Invocation.java

```
package cml.language.expressions;

import cml.language.features.Function;
import cml.language.features.FunctionParameter;
import cml.language.features.TempModule;
import cml.language.generated.Concept;
import cml.language.generated.Expression;
import cml.language.generated.NamedElement;
import cml.language.generated.Type;
import cml.language.types.*;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.jooq.lambda.Seq;
import org.jooq.lambda.tuple.Tuple2;

import java.util.*;

import static cml.language.functions.ModelElementFunctions.moduleOf;
import static cml.language.functions.TypeFunctions.*;
import static cml.language.generated.NamedElement.extendNamedElement;
import static cml.language.generated.UndefinedType.createUndefinedType;
import static java.lang.String.format;
import static java.util.Collections.*;
import static java.util.stream.Collectors.toMap;
import static org.jooq.lambda.Seq.seq;

public interface Invocation extends Expression, NamedElement
{
    String MESSAGE_UNABLE_TO_FIND_FUNCTION_OF_INVOCATION = "Unable to find function of invocation: ";
    String MESSAGE_SHOULD_MATCH_NUMBER_OF_PARAMS_IN_FUNCTION = "Number of arguments in invocation should match the number of parameters in functio
    String MESSAGE_SHOULD_MATCH_PARAMETER_TYPE_IN_FUNCTION = "Argument type should match parameter type in function: ";

    List<Expression> getArguments();
    Map<String, Expression> getNamedArguments();

    default List<FunctionParameter> getParameters()
    {
        if (getFunction().isPresent())
        {
            return getFunction().get().getParameters();
        }
        else if (getConcept().isPresent())
        {
            final Concept concept = getConcept().get();

            return seq(concept.getInvocationProperties())
                .map(p -> new FunctionParameter(p.getName(), p.getType()))
                .toList();
        }
        else
        {
            return emptyList();
        }
    }
```

```
default Map<FunctionParameter, Expression> getParameterizedArguments()
{
    return seq(getParameters())
        .zip(getArguments())
        .collect(toMap(Tuple2::v1, Tuple2::v2));
}

default Map<FunctionType, Lambda> getTypedLambdaArguments()
{
    return seq(getParameterizedArguments()).filter(t -> t.v1.getType() instanceof FunctionType)
                                           .filter(t -> t.v2 instanceof Lambda)
                                           .map(t -> new Tuple2<>((FunctionType) t.v1.getType(), (Lambda) t.v2))
                                           .collect(toMap(Tuple2::v1, Tuple2::v2));
}

default Map<Type, Lambda> getUntypedParameterlessLambdaArguments()
{
    return seq(getParameterizedArguments()).filter(t -> !(t.v1.getType() instanceof FunctionType))
                                           .filter(t -> t.v2 instanceof Lambda)
                                           .map(t -> new Tuple2<>(t.v1.getType(), (Lambda) t.v2))
                                           .filter(t -> t.v2.getParameters().count() == 0)
                                           .collect(toMap(Tuple2::v1, Tuple2::v2));
}

Optional<Function> getFunction();
void setFunction(@NotNull Function function);

Optional<Concept> getConcept();
void setConcept(@NotNull Concept concept);

default Type getInferredType()
{
    if (getFunction().isPresent())
    {
        final Type resultType = getFunction().get().getType();

        return getMatchingTypeOf(resultType);
    }
    else if (getConcept().isPresent())
    {
        final Concept concept = getConcept().get();
        final TempNamedType namedType = TempNamedType.create(concept.getName());

        namedType.setConcept(concept);

        return namedType;
    }
    else
    {
        return createUndefinedType(MESSAGE_UNABLE_TO_FIND_FUNCTION_OF_INVOCATION + getName());
    }
}

default Type getMatchingTypeOf(final Type type)
{
    if (getParameters().size() == getArguments().size())
    {
        if (type instanceof TupleType)
        {
            final TupleType tupleType = (TupleType) type;
            final Seq<TupleTypeElement> matchingElements = tupleType.getElements()
                                                                    .map(e -> new TupleTypeElement(getMatchingTypeOf(e.getType()), e.getName().

            return new TupleType(matchingElements);
        }
        else if (type.isParameter())
        {
            int paramIndex = getParamIndexOfMatchingType(getParameters(), type);

            if (paramIndex < getArguments().size())
            {
                Type paramType = getArguments().get(paramIndex).getMatchingResultType();

                if (paramType.isUndefined())
                {
                    paramIndex = getParamIndexOfMatchingType(getParameters(), type, paramIndex);

                    if (paramIndex < getArguments().size())
                    {
                        paramType = getArguments().get(paramIndex).getMatchingResultType();
                    }
                }

                if (paramType instanceof TupleType && type instanceof MemberType)
                {
                    final TupleType tupleType = (TupleType) paramType;
                    final MemberType memberType = (MemberType) type;

                    paramType = tupleType.getElementTypes()
                                         .get(memberType.getParamIndex())
                                         .orElse(createUndefinedType(MESSAGE_SHOULD_MATCH_PARAMETER_TYPE_IN_FUNCTION + getName()));
                }

                return withCardinality(paramType, type.getCardinality());
            }
            else
            {
                return createUndefinedType(MESSAGE_SHOULD_MATCH_PARAMETER_TYPE_IN_FUNCTION + getName());
            }
        }
        else
        {
            return type;
        }
    }
    else
    {
        return createUndefinedType(MESSAGE_SHOULD_MATCH_NUMBER_OF_PARAMS_IN_FUNCTION + getName());
    }
}

default void createScopeFor(Lambda lambda)
{
    assert lambda.getFunctionType().isPresent() || lambda.getParameters().count() == 0;
    assert !lambda.isInnerExpressionInSomeScope();

    final Optional<Type> scopeType = lambda.getExpectedScopeType();

    if (scopeType.isPresent())
    {
        final Type matchingType = getMatchingTypeOf(scopeType.get());

        assert matchingType.getConcept().isPresent(): "Expected concept but found '" + matchingType.getDiagnosticId() + "' for lambda: " + lamb

        final Optional<Concept> concept = matchingType.getConcept();

        assert concept.isPresent();
```

```
            new LambdaScope(concept.get(), lambda);
        }
        else
        {
            final Optional<TempModule> module = moduleOf(this);

            assert module.isPresent();

            final LambdaScope lambdaScope = new LambdaScope(this, lambda);

            if (lambda.getFunctionType().isPresent())
            {
                lambda.getTypedParameters()
                      .forEach((name, type) -> lambdaScope.addParameter(name, getMatchingTypeOf(type)));
            }
        }
    }

    default boolean typeMatches(final FunctionParameter param, final Expression argument)
    {
        final Type paramType = param.getMatchingResultType();
        final Type argumentType = argument.getMatchingResultType();

        return !argumentType.isUndefined() && isAssignableFrom(getMatchingTypeOf(paramType), argumentType);
    }

    static Invocation create(String name, List<Expression> arguments)
    {
        return new InvocationImpl(name, arguments);
    }

    static Invocation create(String name, LinkedHashMap<String, Expression> namedArguments)
    {
        return new ParameterizedInvocation(name, namedArguments);
    }
}

class InvocationImpl extends ExpressionBase implements Invocation
{
    private final NamedElement namedElement;

    private final List<Expression> arguments;

    private @Nullable Function function;
    private @Nullable Concept concept;

    InvocationImpl(String name, List<Expression> arguments)
    {
        super(seq(arguments).toList());

        namedElement = extendNamedElement(this, expression, expression, name);

        this.arguments = new ArrayList<>(arguments);
    }

    @Override
    public List<Expression> getArguments()
    {
        return unmodifiableList(arguments);
    }

    @Override
    public Map<String, Expression> getNamedArguments()
    {
        return seq(getParameters())
            .zip(getArguments())
            .collect(toMap(t -> t.v1().getName(), Tuple2::v2));
    }

    @Override
    public Optional<Function> getFunction()
    {
        return Optional.ofNullable(function);
    }

    @Override
    public void setFunction(@NotNull Function function)
    {
        assert this.function == null;

        this.function = function;
    }

    @Override
    public Optional<Concept> getConcept()
    {
        return Optional.ofNullable(concept);
    }

    @Override
    public void setConcept(@NotNull final Concept concept)
    {
        assert this.concept == null;

        this.concept = concept;
    }

    @Override
    public String getKind()
    {
        return "invocation";
    }

    @Override
    public Type getType()
    {
        return getInferredType();
    }

    @Override
    public Type getMatchingResultType()
    {
        return expression.getMatchingResultType();
    }

    @Override
    public String getName()
    {
        return namedElement.getName();
    }

    @Override
    public String getDiagnosticId()
    {
        return format("%s(%s) -> %s", getName(), seq(arguments).map(a -> a.getDiagnosticId()).toString(", "), getType().getDiagnosticId());
    }
```

```
}

class ParameterizedInvocation extends ExpressionBase implements Invocation
{
    private final NamedElement namedElement;

    private final LinkedHashMap<String, Expression> namedArguments;

    private @Nullable Function function;
    private @Nullable Concept concept;

    ParameterizedInvocation(String name, LinkedHashMap<String, Expression> namedArguments)
    {
        super(seq(namedArguments.values()).toList());

        namedElement = extendNamedElement(this, expression, expression, name);

        this.namedArguments = new LinkedHashMap<>(namedArguments);
    }

    @Override
    public List<Expression> getArguments()
    {
        return new ArrayList<>(namedArguments.values());
    }

    @Override
    public Map<String, Expression> getNamedArguments()
    {
        return unmodifiableMap(namedArguments);
    }

    @Override
    public Optional<Function> getFunction()
    {
        return Optional.ofNullable(function);
    }

    @Override
    public void setFunction(@NotNull Function function)
    {
        assert this.function == null;

        this.function = function;
    }

    @Override
    public Optional<Concept> getConcept()
    {
        return Optional.ofNullable(concept);
    }

    @Override
    public void setConcept(@NotNull final Concept concept)
    {
        this.concept = concept;
    }

    @Override
    public String getKind()
    {
        return "invocation";
    }

    @Override
    public Type getType()
    {
        return getInferredType();
    }

    @Override
    public Type getMatchingResultType()
    {
        return expression.getMatchingResultType();
    }

    @Override
    public String getName()
    {
        return namedElement.getName();
    }

    @Override
    public String getDiagnosticId()
    {
        final Seq<String> namedArguments = seq(getNamedArguments()).map(t -> format("%s: %s", t.v1, t.v2.getDiagnosticId()));

        return format("%s(%s) -> %s", getName(), namedArguments.toString(", "), getType().getDiagnosticId());
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/expressions/Keyword.java

package cml.language.expressions;

import cml.language.generated.Expression;
import org.jooq.lambda.Seq;

import java.util.List;

import static java.lang.String.format;
import static org.jooq.lambda.Seq.seq;

public class Keyword
{
    private final String name;
    private final List<String> parameters;
    private final Expression expression;

    public Keyword(final String name, final Seq<String> parameters, final Expression expression)
    {
        this.name = name;
        this.parameters = parameters.toList();
        this.expression = expression;
    }

    public String getName()
    {
        return name;
    }

    public Seq<String> getParameters()
    {
        return seq(parameters);
    }
```

```
    public Expression getExpression()
    {
        return expression;
    }

    public Expression getLambdaExpression()
    {
        return new Lambda(seq(parameters), expression);
    }

    @Override
    public String toString()
    {
        return parameters.isEmpty() ?
            format("%s: %s }", getName(), expression) :
            format("{ %s: %s -> %s }", getName(), seq(parameters).toString(", "), expression);
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/expressions/Lambda.java

package cml.language.expressions;

import cml.language.generated.Expression;
import cml.language.generated.Type;
import cml.language.types.FunctionType;
import cml.language.types.MemberType;
import org.jetbrains.annotations.Nullable;
import org.jooq.lambda.Seq;
import org.jooq.lambda.tuple.Tuple2;

import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.OptionalLong;
import java.util.function.Function;

import static cml.language.functions.ScopeFunctions.typeOfVariableNamed;
import static cml.language.generated.UndefinedType.createUndefinedType;
import static java.lang.String.format;
import static java.util.Collections.emptyMap;
import static java.util.Optional.*;
import static java.util.stream.Collectors.toMap;
import static org.jooq.lambda.Seq.seq;

public class Lambda extends ExpressionBase
{
    private final List<String> parameters;
    private final Expression innerExpression;

    private @Nullable FunctionType functionType;

    public Lambda(final Seq<String> parameters, final Expression innerExpression)
    {
        this.parameters = parameters.toList();
        this.innerExpression = innerExpression;
    }

    public Seq<String> getParameters()
    {
        return seq(parameters);
    }

    public Expression getInnerExpression()
    {
        return innerExpression;
    }

    @Override
    public List<Expression> getOperands()
    {
        return Seq.of(innerExpression).toList();
    }

    public Optional<FunctionType> getFunctionType()
    {
        return ofNullable(functionType);
    }

    public void setFunctionType(@Nullable FunctionType functionType)
    {
        assert this.functionType == null;

        this.functionType = functionType;
    }

    public Map<String, Type> getTypedParameters()
    {
        if (functionType == null)
        {
            return emptyMap();
        }
        if (parameters.size() == getParamTypeCount())
        {
            return getParameters().zip(getParamTypes())
                                  .collect(toMap(Tuple2::v1, Tuple2::v2));
        }
        else if (getParamTypeCount() == 1)
        {
            final Function<String, MemberType> mapping = paramName ->
            {
                final OptionalLong paramIndex = getParameters().indexOf(paramName);

                assert paramIndex.isPresent();

                return new MemberType(functionType.getSingleParamType(), paramName, paramIndex.getAsLong());
            };

            return getParameters().zip(getParameters().map(mapping))
                                  .collect(toMap(Tuple2::v1, Tuple2::v2));
        }
        else
        {
            return getTypeDefinedParams().zip(getParamTypes())
                                         .concat(getUntypedParams())
                                         .collect(toMap(Tuple2::v1, Tuple2::v2));
        }
    }

    public Seq<Tuple2<String, Type>> getUntypedParams()
    {
        return getTypeUndefinedParams().zip(getTypeUndefinedParams()
                                       .map(p -> createUndefinedType("Unable to infer type of parameter: " + p)));
    }

    public Seq<String> getTypeDefinedParams()
```

```
    {
        return getParameters().limit(getParamTypeCount());
    }

    public Seq<String> getTypeUndefinedParams()
    {
        return getParameters().skip(getParamTypeCount());
    }

    public Seq<Type> getParamTypes()
    {
        assert functionType != null;

        return functionType.getParamTypes();
    }

    public long getParamTypeCount()
    {
        return getParamTypes().count();
    }

    public Optional<Type> getExpectedScopeType()
    {
        if (parameters.isEmpty() && functionType != null && functionType.isSingleParam())
        {
            return of(functionType.getSingleParamType());
        }
        else
        {
            return empty();
        }
    }

    @Override
    public String getKind()
    {
        return "lambda";
    }

    @Override
    public Type getType()
    {
        return functionType == null ? createUndefinedType("Function type not specified for: " + getDiagnosticId()) : functionType;
    }

    @Override
    public Type getMatchingResultType()
    {
        return (Type) innerExpression.getType();
    }

    public boolean isInnerExpressionInSomeScope()
    {
        return innerExpression.getParent().isPresent();
    }

    @Override
    public String getDiagnosticId()
    {
        return parameters.isEmpty() ?
            format("{ %s }", innerExpression.getDiagnosticId()) :
            format("{ %s -> %s }", seq(parameters).map(this::stringOf).toString(", "), innerExpression.getDiagnosticId())
            + " - inferred result type: " + getMatchingResultType().getDiagnosticId();
    }

    private String stringOf(final String parameter)
    {
        final Optional<Type> actualType = typeOfVariableNamed(parameter, innerExpression);
        final Type formalType = getTypedParameters().get(parameter);

        return actualType.map(t -> parameter + ": " + t.getDiagnosticId())
                         .orElseGet(() -> formalType == null ? parameter : formalType.getDiagnosticId());
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/expressions/LambdaScope.java

```
package cml.language.expressions;

import cml.language.foundation.ScopeBase;
import cml.language.generated.Scope;
import cml.language.generated.Type;

import java.util.HashMap;
import java.util.Map;

import static java.util.Collections.singletonList;
import static java.util.Collections.unmodifiableMap;

public class LambdaScope extends ScopeBase
{
    private final Map<String, Type> parameters = new HashMap<>();

    public LambdaScope(Scope parent, Lambda lambda)
    {
        super(parent, singletonList(lambda.getInnerExpression()));
    }

    public void addParameter(final String name, final Type type)
    {
        parameters.put(name, type);
    }

    public Map<String, Type> getParameters()
    {
        return unmodifiableMap(parameters);
    }

    @Override
    public String getDiagnosticId()
    {
        return parameters.toString();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/expressions/Path.java

```
package cml.language.expressions;

import cml.language.generated.Let;
import cml.language.generated.Scope;
import cml.language.generated.Type;
import cml.language.types.TempNamedType;
import org.jetbrains.annotations.Nullable;
```

```
import java.util.LinkedList;
import java.util.List;
import java.util.Optional;

import static cml.language.functions.ModelElementFunctions.selfTypeOf;
import static cml.language.functions.ScopeFunctions.scopeOfType;
import static cml.language.functions.ScopeFunctions.typeOfVariableNamed;
import static cml.language.functions.TypeFunctions.withCardinality;
import static cml.language.generated.UndefinedType.createUndefinedType;
import static java.util.Collections.singletonList;
import static java.util.Collections.unmodifiableList;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;
import static org.jooq.lambda.Seq.seq;

public class Path extends ExpressionBase
{
    private final @Nullable Path base;
    private final String name;

    public static Path create(List<String> names)
    {
        Path path = null;

        for (String name: names)
        {
            final Path base = path;

            path = new Path(base, name);
        }

        return path;
    }

    public Path(@Nullable Path base, String name)
    {
        super(singletonList(base));

        this.base = base;
        this.name = name;
    }

    public List<String> getNames()
    {
        List<String> names = new LinkedList<>();
        names.add(getName());

        Optional<Path> path = getBase();
        while (path.isPresent())
        {
            names.add(0, path.get().getName());

            path = path.get().getBase();
        }

        return unmodifiableList(names);
    }

    public List<String> getMemberNames()
    {
        assert getNames().size() >= 1: "Path must have at least one name in order to determine its member names.";

        return getNames().stream().skip(1).collect(toList());
    }

    public boolean isVariable()
    {
        if (isFirst())
        {
            Optional<LambdaScope> lambdaScope = getLambdaScope(getParent());

            while (lambdaScope.isPresent())
            {
                if (lambdaScope.get().getParameters().containsKey(getName()))
                {
                    return true;
                }

                lambdaScope = getLambdaScope(lambdaScope.get().getParent());
            }

            Optional<Let> letScope = getLetScope(getParent());

            while (letScope.isPresent())
            {
                if (letScope.get().getVariable().equals(getName()))
                {
                    return true;
                }

                letScope = getLetScope(letScope.get().getParent());
            }
        }

        return false;
    }

    @SuppressWarnings("OptionalUsedAsFieldOrParameterType")
    private Optional<LambdaScope> getLambdaScope(Optional<Scope> parent)
    {
        while (parent.isPresent() && !(parent.get() instanceof LambdaScope))
        {
            parent = parent.get().getParent();
        }

        return seq(parent).cast(LambdaScope.class).findFirst();
    }

    private Optional<Let> getLetScope(Optional<Scope> parent)
    {
        while (parent.isPresent() && !(parent.get() instanceof Let))
        {
            parent = parent.get().getParent();
        }

        return seq(parent).cast(Let.class).findFirst();
    }

    public boolean isFirst()
    {
        return !getBase().isPresent();
    }

    public boolean isLast()
    {
        return !getParent().isPresent() || !(getParent().get() instanceof Path);
```

```
}

@Override
public Type getType()
{
    assert getNames().size() >= 1: "In order to be able to determine its type, path must have at least one name.";
    assert getParent().isPresent(): "In order to be able to determine its type, path must be bound to a scope: " + getNames() + " " + getLocati

    Scope scope = getParent().get();

    if (isSelf()) return selfTypeOf(scope);

    if (isNone()) return TempNamedType.NOTHING;

    final String variableName = getNames().get(0);
    final Optional<Type> variableType = typeOfVariableNamed(variableName, scope);

    StringBuilder intermediatePath = new StringBuilder(variableName);

    if (variableType.isPresent())
    {
        Type type = variableType.get();

        for (final String memberName: getMemberNames())
        {
            intermediatePath.append(".").append(memberName);

            final Optional<Scope> optionalScope = scopeOfType(type, scope);

            if (optionalScope.isPresent())
            {
                scope = optionalScope.get();

                final Optional<Type> memberType = typeOfVariableNamed(memberName, scope);

                if (memberType.isPresent())
                {
                    if (memberType.get().isUndefined())
                    {
                        return memberType.get();
                    }
                    else
                    {
                        final String cardinality = memberType.get().getCardinality();

                        type = withCardinality(
                            memberType.get(),
                            type.isSequence() ? "*" : (memberType.get().isRequired() && type.isOptional() ? "?" : cardinality));
                    }
                }
                else
                {
                    return createUndefinedType("Unable to find type of member: " + intermediatePath);
                }
            }
            else
            {
                return createUndefinedType("Unable to find type: " + type.getDiagnosticId());
            }
        }

        return type;
    }
    else
    {
        return createUndefinedType("Unable to find type of variable: " + intermediatePath);
    }
}

public Type getOriginalType()
{
    assert getNames().size() >= 1: "In order to be able to determine its type, path must have at least one name.";
    assert getParent().isPresent(): "In order to be able to determine its type, path must be bound to a scope: " + getNames() + " " + getLocati

    Scope scope = getParent().get();

    if (isSelf()) return selfTypeOf(scope);

    final String variableName = getNames().get(0);
    final Optional<Type> variableType = typeOfVariableNamed(variableName, scope);

    StringBuilder intermediatePath = new StringBuilder(variableName);

    if (variableType.isPresent())
    {
        Type type = variableType.get();

        for (final String memberName: getMemberNames())
        {
            intermediatePath.append(".").append(memberName);

            final Optional<Scope> optionalScope = scopeOfType(type, scope);

            if (optionalScope.isPresent())
            {
                scope = optionalScope.get();

                final Optional<Type> memberType = typeOfVariableNamed(memberName, scope);

                if (memberType.isPresent())
                {
                    type = memberType.get();
                }
                else
                {
                    return createUndefinedType("Unable to find type of member: " + intermediatePath);
                }
            }
            else
            {
                return createUndefinedType("Unable to find type: " + type.getDiagnosticId());
            }
        }

        return type;
    }
    else
    {
        return createUndefinedType("Unable to find type of variable: " + intermediatePath);
    }
}

public boolean isSelf()
{
    return getNames().size() == 1 && getNames().get(0).equals("self");
}
```

```
    public boolean isNone()
    {
        return getNames().size() == 1 && getNames().get(0).equals("none");
    }

    public Optional<Path> getBase()
    {
        return Optional.ofNullable(base);
    }

    public String getName()
    {
        return name;
    }

    @Override
    public String getKind()
    {
        return "path";
    }

    @Override
    public String getDiagnosticId()
    {
        return getNames().stream().collect(joining("."));
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/expressions/Query.java

```
package cml.language.expressions;

import cml.language.generated.Expression;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;
import org.jooq.lambda.Seq;

import java.util.List;
import java.util.Optional;
import java.util.stream.Stream;

import static java.util.Collections.emptyList;
import static java.util.Collections.unmodifiableList;
import static java.util.stream.Collectors.toList;
import static org.jooq.lambda.Seq.empty;
import static org.jooq.lambda.Seq.seq;

public class Query
{
    private final @Nullable String name;
    private final List<Keyword> keywords;

    public Query(final @NotNull String name)
    {
        this.name = name;
        this.keywords = emptyList();
    }

    public Query(final Stream<Keyword> keywords)
    {
        this.name = null;
        this.keywords = keywords.collect(toList());
    }

    public List<Keyword> getKeywords()
    {
        return unmodifiableList(keywords);
    }

    public String getInvocationName()
    {
        final Optional<Keyword> first = seq(keywords).findFirst();

        assert name != null || first.isPresent();

        return first.map(Keyword::getName).orElse(name);
    }

    public Optional<Expression> getExpression()
    {
        final Optional<Keyword> first = seq(keywords).findFirst();

        return first.map(Keyword::getExpression);
    }

    public Seq<String> getExpressionParams()
    {
        final Optional<Keyword> first = seq(keywords).findFirst();

        return first.map(Keyword::getParameters).orElse(empty());
    }

    public List<Keyword> getExtraKeywords()
    {
        return seq(keywords).skip(1).toList();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/expressions/TempConditional.java

```
package cml.language.expressions;

import cml.language.generated.Concept;
import cml.language.generated.Conditional;
import cml.language.generated.Expression;
import cml.language.generated.Type;
import cml.language.types.TempNamedType;

import java.util.Optional;

import static cml.language.functions.ModelFunctions.conceptOf;
import static cml.language.functions.TypeFunctions.firstGeneralizationAssignableFrom;
import static cml.language.functions.TypeFunctions.isAssignableFrom;
import static cml.language.generated.Conditional.extendConditional;
import static java.util.Arrays.asList;

public class TempConditional extends ExpressionBase implements Conditional
{
    private final Conditional conditional;

    public TempConditional(Expression cond, Expression then, Expression else_)
    {
        super(asList(cond, then, else_));
```

```
        this.conditional = extendConditional(this, expression, expression, expression, expression);
    }

    @Override
    public String getKind()
    {
        return conditional.getKind();
    }

    @Override
    public Expression getCondExpr()
    {
        return conditional.getCondExpr();
    }

    @Override
    public Expression getThenExpr()
    {
        return conditional.getThenExpr();
    }

    @Override
    public Expression getElseExpr()
    {
        return conditional.getElseExpr();
    }

    @Override
    public Type getType()
    {
        final Type thenType = getThenExpr().getType();
        final Type elseType = getElseExpr().getType();

        if (isAssignableFrom(thenType, elseType))
        {
            return thenType;
        }
        else if (isAssignableFrom(elseType, thenType))
        {
            return elseType;
        }
        else
        {
            final Optional<Concept> concept1 = getModel().flatMap(m -> conceptOf(m, thenType.getName()));
            final Optional<Concept> concept2 = getModel().flatMap(m -> conceptOf(m, elseType.getName()));

            if (concept1.isPresent() && concept2.isPresent())
            {
                final Optional<Type> thenTypeGeneralization = firstGeneralizationAssignableFrom(thenType, elseType);

                if (thenTypeGeneralization.isPresent())
                {
                    return thenTypeGeneralization.get();
                }
                else
                {
                    final Optional<Type> elseTypeGeneralization = firstGeneralizationAssignableFrom(thenType, elseType);

                    if (elseTypeGeneralization.isPresent())
                    {
                        return elseTypeGeneralization.get();
                    }
                }
            }
        }

        return TempNamedType.create(thenType.getDiagnosticId() + "|" + elseType.getDiagnosticId());
    }

    @Override
    public String getDiagnosticId()
    {
        return conditional.getDiagnosticId();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/expressions/TypeCast.java

```
package cml.language.expressions;

import cml.language.generated.Expression;
import cml.language.generated.Type;

import static cml.language.functions.TypeFunctions.isCastAllowed;
import static cml.language.generated.UndefinedType.createUndefinedType;
import static java.lang.String.format;
import static java.util.Collections.singletonList;

public class TypeCast extends ExpressionBase
{
    public static final String ASB = "asb";
    public static final String ASQ = "asq";

    private final Expression expr;
    private final String operator;
    private final Type castType;

    public TypeCast(Expression expr, final String operator, final Type castType)
    {
        super(singletonList(expr));
        this.expr = expr;
        this.operator = operator;
        this.castType = castType;
    }

    public Expression getExpr()
    {
        return expr;
    }

    public String getOperator()
    {
        return operator;
    }

    public Type getCastType()
    {
        return castType;
    }

    @Override
    public String getKind()
    {
        return "type_cast";
    }

    @Override
```

```
    public Type getType()
    {
        final Type exprType = expr.getType();

        if (isCastAllowed(operator, exprType, castType))
        {
            return castType;
        }
        else
        {
            return createUndefinedType(
                format(
                    "%s:\n- left operand is '%s: %s'\n- right operand is '%s'",
                    diagnosticMessage(operator, exprType, castType),
                    expr.getDiagnosticId(),
                    exprType.getDiagnosticId(),
                    castType.getDiagnosticId()));
        }
    }

    private static String diagnosticMessage(String operator, Type exprType, Type castType)
    {
        if (operator.equals(ASQ) && castType.isRequired())
        {
            return "Cannot use 'as?' to cast to required-cardinality type";
        }
        else if (operator.equals(ASB) && exprType.isSequence() && castType.isRequired())
        {
            return "Cannot cast from sequence type to required-cardinality type";
        }
        else if (exprType.isSequence() && castType.isOptional())
        {
            return "Cannot cast from sequence type to optional-cardinality type";
        }
        else
        {
            return format("Incompatible operand(s) for operator '%s'",
                          operator.replace('q', '?').replace('b', '!'));
        }
    }

    @Override
    public String getDiagnosticId()
    {
        return expr.getDiagnosticId() + (operator.equals(ASB) ? "as!" : "as?") + castType.getDiagnosticId();
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/expressions/TypeCheck.java

package cml.language.expressions;

import cml.language.generated.Expression;
import cml.language.generated.Type;
import cml.language.types.TempNamedType;

import static cml.language.functions.TypeFunctions.isAssignableFrom;
import static cml.language.generated.UndefinedType.createUndefinedType;
import static java.lang.String.format;
import static java.util.Collections.singletonList;

public class TypeCheck extends ExpressionBase
{
    private final Expression expr;
    private final String operator;
    private final Type checkedType;

    public TypeCheck(Expression expr, final String operator, final Type checkedType)
    {
        super(singletonList(expr));
        this.expr = expr;
        this.operator = operator;
        this.checkedType = checkedType;
    }

    public Expression getExpr()
    {
        return expr;
    }

    public String getOperator()
    {
        return operator;
    }

    public Type getCheckedType()
    {
        return checkedType;
    }

    @Override
    public String getKind()
    {
        return "type_check";
    }

    @Override
    public Type getType()
    {
        final Type exprType = expr.getType();

        if (exprType.isReferential() && checkedType.isReferential() && isAssignableFrom(exprType, checkedType))
        {
            return TempNamedType.BOOLEAN;
        }
        else
        {
            return createUndefinedType(
                format(
                    "Incompatible operand(s) for operator '%s':\n- left operand is '%s: %s'\n- right operand is '%s'",
                    getOperator(),
                    expr.getDiagnosticId(),
                    exprType.getDiagnosticId(),
                    checkedType.getDiagnosticId()));
        }
    }

    @Override
    public String getDiagnosticId()
    {
        return format("%s is %s", expr.getDiagnosticId(), checkedType.getDiagnosticId());
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/expressions/Unary.java
```

```
package cml.language.expressions;

import cml.language.generated.Expression;
import cml.language.generated.Type;
import cml.language.types.TempNamedType;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

import static cml.language.generated.UndefinedType.createUndefinedType;
import static java.lang.String.format;
import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
import static java.util.Collections.unmodifiableCollection;

public class Unary extends ExpressionBase
{
    private final Collection<String> LOGIC_OPERATORS = unmodifiableCollection(singletonList(
        "not"
    ));

    private final Collection<String> NUMERIC_OPERATORS = unmodifiableCollection(asList(
        "+", "-"
    ));

    private static Map<String, String> OPERATIONS =
        new HashMap<String, String>()
        {{
            put("+", "unary_add");
            put("-", "unary_sub");
            put("not", "not");
        }};

    private final String operator;
    private final Expression subExpr;

    public Unary(String operator, Expression subExpr)
    {
        super(singletonList(subExpr));

        this.operator = operator;
        this.subExpr = subExpr;
    }

    public String getOperator()
    {
        return operator;
    }

    public Optional<String> getOperation()
    {
        final String operation = OPERATIONS.get(getOperator());

        return Optional.ofNullable(operation);
    }

    public Expression getSubExpr()
    {
        return subExpr;
    }

    @Override
    public String getKind()
    {
        return "unary";
    }

    @Override
    public Type getType()
    {
        final Type subExprType = subExpr.getType();

        if (subExprType.isUndefined())
        {
            return subExprType;
        }
        else if (LOGIC_OPERATORS.contains(operator) && subExprType.isBoolean())
        {
            return TempNamedType.BOOLEAN;
        }
        else if (NUMERIC_OPERATORS.contains(operator) && (subExprType.isNumeric() || subExprType.isFloat()))
        {
            return subExprType;
        }
        else
        {
            return createUndefinedType(
                format(
                    "Incompatible operand '%s' of type '%s' for operator '%s'.",
                    getSubExpr().getDiagnosticId(),
                    subExprType.getDiagnosticId(),
                    getOperator()));
        }
    }

    @Override
    public String getDiagnosticId()
    {
        return format("%s %s ", getOperator(), getSubExpr());
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/features/Function.java

```
package cml.language.features;

import cml.language.foundation.ScopeBase;
import cml.language.generated.Expression;
import cml.language.generated.Type;
import cml.language.types.TypeParameter;
import org.jetbrains.annotations.Nullable;
import org.jooq.lambda.Seq;

import java.util.List;
import java.util.Optional;
import java.util.stream.Stream;

import static cml.language.generated.UndefinedType.createUndefinedType;
import static java.lang.String.format;
import static java.util.Collections.unmodifiableList;
import static org.jooq.lambda.Seq.*;

public class Function extends ScopeBase
```

```
{
    private String name;
    private Type type;
    private List<TypeParameter> typeParams;

    public Function(final String name, final Type type, final Stream<FunctionParameter> parameters)
    {
        this(null, name, type, parameters);
    }

    public Function(@Nullable final TempModule module, final String name, final Type type, final Stream<FunctionParameter> parameters)
    {
        this(module, name, type, parameters, empty(), null);
    }

    public Function(@Nullable final TempModule module, final String name, final Type type, final Stream<FunctionParameter> parameters, final Seq<Ty
    {
        super(module, seq(concat(seq(parameters), seq(Optional.ofNullable(expression)))).toList());

        this.name = name;
        this.type = type;
        this.typeParams = typeParams.toList();
    }

    public String getName()
    {
        return name;
    }

    public Type getType()
    {
        return type == null && getExpression().isPresent() ?
            getExpression().get().getType() :
            (type == null ? createUndefinedType("Unable to find type of function: " + getDiagnosticId()) : type);
    }

    public List<FunctionParameter> getParameters()
    {
        return seq(getMembers()).filter(m -> m instanceof FunctionParameter)
                                .cast(FunctionParameter.class)
                                .toList();
    }

    public List<TypeParameter> getTypeParams()
    {
        return unmodifiableList(typeParams);
    }

    public Optional<Expression> getExpression()
    {
        return seq(getMembers()).filter(m -> m instanceof Expression)
                                .cast(Expression.class)
                                .findFirst();
    }

    @Override
    public String getDiagnosticId()
    {
        return format(
            "function %s(%s) -> %s", getName(),
            seq(getParameters()).toString(", "),
            getType().getDiagnosticId());
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/features/FunctionParameter.java

package cml.language.features;

import cml.language.generated.Type;
import cml.language.types.TypedElementBase;

import static java.lang.String.format;

public class FunctionParameter extends TypedElementBase
{
    public FunctionParameter(final String name, final Type type)
    {
        super(name, type);
    }

    public Type getMatchingResultType()
    {
        return getType().getMatchingResultType();
    }

    @Override
    public String getDiagnosticId()
    {
        return format("%s: %s", getName(), getType().getDiagnosticId());
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/features/TempConcept.java

package cml.language.features;

import cml.language.generated.*;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.stream.Stream;

import static cml.language.functions.ConceptFunctions.redefinedAncestors;
import static cml.language.functions.ConceptFunctions.redefinedInheritedConcreteProperties;
import static cml.language.generated.Concept.extendConcept;
import static cml.language.generated.Element.extendElement;
import static cml.language.generated.ModelElement.extendModelElement;
import static cml.language.generated.NamedElement.extendNamedElement;
import static cml.language.generated.PropertyList.extendPropertyList;
import static cml.language.generated.Scope.extendScope;
import static java.util.Collections.emptyList;
import static java.util.stream.Collectors.toList;
import static java.util.stream.Stream.concat;
import static org.jooq.lambda.Seq.seq;

public interface TempConcept extends Concept, PropertyList
{
    default List<String> getDependencies()
    {
        return concat(
            getGeneralizationDependencies().stream(),
            getPropertyDependencies().stream())
            .distinct()
```

```
            .collect(toList());
    }

    default List<String> getGeneralizationDependencies()
    {
        return getAllAncestors().stream()
                                .map(Concept::getName)
                                .filter(name -> !name.equals(getName()))
                                .distinct()
                                .collect(toList());
    }

    default List<String> getPropertyDependencies()
    {
        final Stream<String> generalizations = getTransitivePropertyConcepts().stream()
                                                                               .flatMap(concept -> concept.getGeneralizationDependencies().stream());

        final Stream<String> directDependencies = getTransitivePropertyConcepts().stream()
                                                                                 .map(TempConcept::getName);

        return concat(generalizations, directDependencies)
            .filter(name -> !name.equals(getName()))
            .distinct()
            .collect(toList());
    }

    default List<TempConcept> getTransitivePropertyConcepts()
    {
        final List<TempConcept> concepts = new ArrayList<>();

        seq(getPropertyConcepts()).map(c -> (TempConcept)c).forEach(c -> c.appendToPropertyConcepts(concepts));

        return concepts;
    }

    default void appendToPropertyConcepts(List<TempConcept> concepts)
    {
        if (!concepts.contains(this))
        {
            concepts.add(this);

            seq(getPropertyConcepts()).map(c -> (TempConcept)c).forEach(c -> c.appendToPropertyConcepts(concepts));
        }
    }

    @SuppressWarnings("unused")
    default List<ConceptRedef> getRedefinedAncestors()
    {
        return redefinedAncestors(this, this);
    }

    @SuppressWarnings("unused")
    default List<Property> getRedefinedInheritedConcreteProperties()
    {
        return redefinedInheritedConcreteProperties(this);
    }

    default List<TempConcept> getAllGeneralizations()
    {
        final List<TempConcept> generalizations = new ArrayList<>();

        getAncestors().forEach(c -> ((TempConcept) c).appendToGeneralizations(generalizations));

        return generalizations;
    }

    default void appendToGeneralizations(List<TempConcept> generalizations)
    {
        if (!generalizations.contains(this))
        {
            generalizations.add(this);

            getAncestors().forEach(c -> ((TempConcept) c).appendToGeneralizations(generalizations));
        }
    }

    static TempConcept create(TempModule module, String name)
    {
        return create(module, name, false, emptyList(), null);
    }

    static TempConcept create(TempModule module, String name, List<Property> propertyList)
    {
        return new ConceptImpl(module, name, false, propertyList, null);
    }

    static TempConcept create(TempModule module, String name, boolean _abstract, List<Property> propertyList, Location location)
    {
        return new ConceptImpl(module, name, _abstract, propertyList, location);
    }
}

class ConceptImpl implements TempConcept
{
    private final Concept concept;

    ConceptImpl(TempModule module, String name, boolean abstraction, final List<Property> properties, Location location)
    {
        final Element element = extendElement(this);
        final ModelElement modelElement = extendModelElement(this, element, module, location);
        final NamedElement namedElement = extendNamedElement(this, element, modelElement, name);
        final Scope scope = extendScope(this, element, modelElement, seq(properties).map(p -> (ModelElement)p).toList());
        final PropertyList propertyList = extendPropertyList(this, element, modelElement, scope);
        this.concept = extendConcept(this, element, modelElement, scope, namedElement, propertyList, abstraction, emptyList(), emptyList());
    }

    @Override
    public Optional<Location> getLocation()
    {
        return concept.getLocation();
    }

    @Override
    public Optional<Scope> getParent()
    {
        return concept.getParent();
    }

    @Override
    public Optional<Model> getModel()
    {
        return concept.getModel();
    }

    @Override
    public Optional<Module> getModule()
    {
```

```
    return concept.getModule();
}

@Override
public String getName()
{
    return concept.getName();
}

@Override
public List<ModelElement> getMembers()
{
    return concept.getMembers();
}

@Override
public List<Property> getProperties()
{
    return concept.getProperties();
}

@Override
public boolean isAbstraction()
{
    return concept.isAbstraction();
}

@Override
public List<Property> getDerivedProperties()
{
    return concept.getDerivedProperties();
}

@Override
public List<Property> getAssociationProperties()
{
    return concept.getAssociationProperties();
}

@Override
public List<Type> getPropertyTypes()
{
    return concept.getPropertyTypes();
}

@Override
public List<Association> getAssociations()
{
    return concept.getAssociations();
}

@Override
public List<Property> getInitProperties()
{
    return concept.getInitProperties();
}

@Override
public List<Property> getNonInitProperties()
{
    return concept.getNonInitProperties();
}

@Override
public List<Property> getPrintableProperties()
{
    return concept.getPrintableProperties();
}

@Override
public List<Concept> getPropertyConcepts()
{
    return concept.getPropertyConcepts();
}

@Override
public List<Property> getNonDerivedProperties()
{
    return concept.getNonDerivedProperties();
}

@Override
public List<Property> getInvocationProperties()
{
    return concept.getInvocationProperties();
}

@Override
public List<Property> getSlotProperties()
{
    return concept.getSlotProperties();
}

@Override
public List<Inheritance> getGeneralizations()
{
    return concept.getGeneralizations();
}

@Override
public List<Inheritance> getSpecializations()
{
    return concept.getSpecializations();
}

@Override
public List<Concept> getAncestors()
{
    return concept.getAncestors();
}

@Override
public List<Concept> getDescendants()
{
    return concept.getDescendants();
}

@Override
public List<Concept> getAllAncestors()
{
    return concept.getAllAncestors();
}

@Override
public List<Property> getAllProperties()
{
    return concept.getAllProperties();
```

```
    }

    @Override
    public List<Concept> getInheritedAncestors()
    {
        return concept.getInheritedAncestors();
    }

    @Override
    public List<Property> getInheritedProperties()
    {
        return concept.getInheritedProperties();
    }

    @Override
    public List<Property> getInheritedAbstractProperties()
    {
        return concept.getInheritedAbstractProperties();
    }

    @Override
    public List<Property> getInheritedNonRedefinedProperties()
    {
        return concept.getInheritedNonRedefinedProperties();
    }

    @Override
    public List<Property> getSuperProperties()
    {
        return concept.getSuperProperties();
    }

    @Override
    public String getDiagnosticId()
    {
        return concept.getDiagnosticId();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/features/Template.java

```
package cml.language.features;

import cml.language.foundation.NamedElementBase;

public class Template extends NamedElementBase
{
    private final Function function;

    public Template(TempModule module, Function function)
    {
        super(module, function.getName());

        this.function = function;
    }

    public Function getFunction()
    {
        return function;
    }

    @Override
    public String getDiagnosticId()
    {
        return function.getDiagnosticId();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/features/TempModule.java

```
package cml.language.features;

import cml.language.foundation.TempModel;
import cml.language.generated.*;

import java.util.List;
import java.util.Optional;

import static cml.language.generated.Element.extendElement;
import static cml.language.generated.ModelElement.extendModelElement;
import static cml.language.generated.Module.extendModule;
import static cml.language.generated.NamedElement.extendNamedElement;
import static cml.language.generated.Scope.extendScope;
import static java.util.Collections.emptyList;
import static java.util.stream.Collectors.toList;
import static java.util.stream.Stream.concat;

public interface TempModule extends NamedElement, Scope, Module
{
    default List<TempModule> getImportedModules()
    {
        return getImports().stream()
                           .map(Import::getImportedModule)
                           .map(m -> (TempModule)m)
                           .collect(toList());
    }

    default List<TempConcept> getImportedConcepts()
    {
        return getImportedModules()
                    .stream()
                    .flatMap(m -> m.getConcepts().stream())
                    .map(c -> (TempConcept)c)
                    .collect(toList());
    }

    default List<TempConcept> getAllConcepts()
    {
        return concat(
            getConcepts().stream(),
            getImportedConcepts().stream())
            .map(c -> (TempConcept)c)
            .collect(toList());
    }

    default List<Template> getTemplates()
    {
        return getMembers().stream()
                           .filter(e -> e instanceof Template)
                           .map(e -> (Template)e)
                           .collect(toList());
    }

    default List<Function> getFunctions()
    {
```

```
        return getMembers().stream()
                           .filter(e -> e instanceof Function)
                           .map(e -> (Function)e)
                           .collect(toList());
    }

    default List<Function> getDefinedFunctions()
    {
        return getMembers().stream()
                           .filter(e -> e instanceof Function)
                           .map(e -> (Function)e)
                           .filter(f -> f.getExpression().isPresent())
                           .collect(toList());
    }

    static TempModule create(TempModel model, String name)
    {
        return new TempModuleImpl(model, name);
    }
}

class TempModuleImpl implements TempModule
{
    private final Module module;

    TempModuleImpl(TempModel model, String name)
    {
        final Element element = extendElement(this);
        final ModelElement modelElement = extendModelElement(this, element, model, null);
        final NamedElement namedElement = extendNamedElement(this, element, modelElement, name);
        final Scope scope = extendScope(this, element, modelElement, emptyList());

        this.module = extendModule(this, element, modelElement, namedElement, scope);
    }

    @Override
    public Optional<Location> getLocation()
    {
        return module.getLocation();
    }

    @Override
    public Optional<Scope> getParent()
    {
        return module.getParent();
    }

    @Override
    public Optional<Model> getModel()
    {
        return module.getModel();
    }

    @Override
    public String getName()
    {
        return module.getName();
    }

    @Override
    public List<ModelElement> getMembers()
    {
        return module.getMembers();
    }

    @Override
    public Optional<Module> getModule()
    {
        return module.getModule();
    }

    @Override
    public List<Import> getImports()
    {
        return module.getImports();
    }

    @Override
    public List<Task> getTasks()
    {
        return module.getTasks();
    }

    @Override
    public List<Concept> getConcepts()
    {
        return module.getConcepts();
    }

    @Override
    public List<Association> getAssociations()
    {
        return module.getAssociations();
    }

    @Override
    public String getDiagnosticId()
    {
        return module.getDiagnosticId();
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/foundation/Diagnostic.java

package cml.language.foundation;

import cml.language.generated.Element;
import org.jetbrains.annotations.Nullable;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import static java.util.Collections.emptyList;

public class Diagnostic
{
    private final String code;
    private final Element element;
    private final @Nullable String message;
    private final List<? extends Element> participants;

    public Diagnostic(String code, Element element)
    {
        this(code, element, emptyList());
    }
```

```
    public Diagnostic(String code, Element element, @Nullable String message)
    {
        this(code, element, message, emptyList());
    }

    public Diagnostic(String code, Element element, List<? extends Element> participants)
    {
        this(code, element, null, participants);
    }

    @SuppressWarnings("WeakerAccess")
    public Diagnostic(String code, Element element, @Nullable String message, List<? extends Element> participants)
    {
        this.code = code;
        this.element = element;
        this.message = message;
        this.participants = new ArrayList<>(participants);
    }

    public String getCode()
    {
        return code;
    }

    public Element getElement()
    {
        return element;
    }

    public Optional<String> getMessage()
    {
        return Optional.ofNullable(message);
    }

    public List<? extends Element> getParticipants()
    {
        return participants;
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/foundation/Invariant.java

```
package cml.language.foundation;

import cml.language.generated.ModelElement;

public interface Invariant<T extends ModelElement>
{
    boolean evaluate(T self);
    Diagnostic createDiagnostic(T self);

    default boolean isCritical()
    {
        return false;
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/foundation/InvariantValidator.java

```
package cml.language.foundation;

import cml.language.generated.ModelElement;

import java.util.List;

@FunctionalInterface
public interface InvariantValidator<T extends ModelElement>
{
    List<Invariant<T>> getInvariants();

    default void validate(T self, List<Diagnostic> diagnostics)
    {
        for (Invariant<T> invariant: getInvariants())
        {
            if (!invariant.evaluate(self))
            {
                diagnostics.add(invariant.createDiagnostic(self));

                if (invariant.isCritical()) break;
            }
        }
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/foundation/ModelElementBase.java

```
package cml.language.foundation;

import cml.language.generated.*;
import org.jetbrains.annotations.Nullable;

import java.util.Optional;

import static cml.language.generated.Element.extendElement;
import static cml.language.generated.ModelElement.extendModelElement;

public abstract class ModelElementBase implements ModelElement
{
    protected final Element element;
    protected final ModelElement modelElement;

    protected ModelElementBase()
    {
        this(null);
    }

    protected ModelElementBase(@Nullable Scope parent)
    {
        element = extendElement(this);
        modelElement = extendModelElement(this, element, parent, null);
    }

    @Override
    public Optional<Location> getLocation()
    {
        return modelElement.getLocation();
    }

    @Override
    public Optional<Scope> getParent()
    {
        return modelElement.getParent();
    }
```

```
    @Override
    public Optional<Model> getModel()
    {
        return modelElement.getModel();
    }

    @Override
    public Optional<Module> getModule()
    {
        return modelElement.getModule();
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/foundation/NamedElementBase.java

package cml.language.foundation;

import cml.language.generated.NamedElement;
import cml.language.generated.Scope;
import org.jetbrains.annotations.Nullable;

import static cml.language.generated.Element.extendElement;
import static cml.language.generated.NamedElement.extendNamedElement;

public abstract class NamedElementBase extends ModelElementBase implements NamedElement
{
    private final NamedElement namedElement;

    public NamedElementBase(String name)
    {
        this(null, name);
    }

    public NamedElementBase(@Nullable Scope parent, String name)
    {
        super(parent);

        namedElement = extendNamedElement(this, extendElement(this), modelElement, name);
    }

    @Override
    public String getName()
    {
        return namedElement.getName();
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/foundation/Pair.java

package cml.language.foundation;

import cml.language.generated.ModelElement;

import java.util.Objects;

import static java.util.Objects.hash;

public class Pair<T extends ModelElement>
{
    private final T left;
    private final T right;

    public Pair(T left, T right)
    {
        this.left = left;
        this.right = right;
    }

    public T getLeft()
    {
        return left;
    }

    public T getRight()
    {
        return right;
    }

    @Override
    public boolean equals(Object o)
    {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Pair<?> pair = (Pair<?>) o;
        return (Objects.equals(left, pair.left) &&
                Objects.equals(right, pair.right)) ||
               (Objects.equals(left, pair.right) &&
                Objects.equals(right, pair.left));
    }

    @Override
    public int hashCode()
    {
        final int h1 = left == null ? 0 : left.hashCode();
        final int h2 = right == null ? 0 : right.hashCode();

        return h1 > h2 ? hash(h2, h1) : hash(h1, h2);
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/foundation/Parameter.java

package cml.language.foundation;

import cml.language.generated.*;
import cml.language.types.TempNamedType;
import cml.language.types.TypedElementBase;
import org.jetbrains.annotations.Nullable;

import java.util.Optional;

import static cml.language.functions.ModelElementFunctions.siblingNamed;
import static cml.language.generated.ModelElement.extendModelElement;
import static cml.language.generated.NamedElement.extendNamedElement;
import static java.util.Optional.empty;

public interface Parameter extends TypedElement
{
    Optional<String> getScopeName();

    default Optional<Parameter> getParameterScope()
    {
        if (getScopeName().isPresent())
```

```
        {
            return siblingNamed(getScopeName().get(), this, Parameter.class);
        }
        else
        {
            return empty();
        }
    }

    static Parameter create(String name, TempNamedType type, @Nullable String scopeName)
    {
        return new ParameterImpl(name, type, scopeName);
    }
}

class ParameterImpl extends TypedElementBase implements Parameter
{
    private final @Nullable String scopeName;

    ParameterImpl(String name, TempNamedType type, @Nullable String scopeName)
    {
        super(name, type);

        this.scopeName = scopeName;
    }

    @Override
    public Optional<String> getScopeName()
    {
        return Optional.ofNullable(scopeName);
    }

    @Override
    public String getDiagnosticId()
    {
        return getType().getDiagnosticId();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/foundation/ScopeBase.java

```
package cml.language.foundation;

import cml.language.generated.ModelElement;
import cml.language.generated.Scope;
import org.jetbrains.annotations.Nullable;

import java.util.List;

public abstract class ScopeBase extends ModelElementBase implements Scope
{
    protected final Scope scope;

    public ScopeBase(@Nullable Scope parent, List<ModelElement> members)
    {
        super(parent);

        this.scope = Scope.extendScope(this, element, modelElement, members);
    }

    @Override
    public List<ModelElement> getMembers()
    {
        return scope.getMembers();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/foundation/TempModel.java

```
package cml.language.foundation;

import cml.language.features.Function;
import cml.language.features.TempConcept;
import cml.language.features.TempModule;
import cml.language.features.Template;
import cml.language.generated.*;

import java.util.Comparator;
import java.util.List;
import java.util.Optional;

import static cml.language.generated.Element.extendElement;
import static cml.language.generated.ModelElement.extendModelElement;
import static cml.language.generated.NamedElement.extendNamedElement;
import static cml.language.generated.Scope.extendScope;
import static java.util.Collections.emptyList;
import static java.util.stream.Collectors.toList;

public interface TempModel extends NamedElement, Scope, Model
{
    default List<TempConcept> getOrderedConcepts()
    {
        return getConcepts().stream()
                            .map(c -> (TempConcept)c)
                            .sorted(byDependencyOrder())
                            .collect(toList());
    }

    static Comparator<TempConcept> byDependencyOrder()
    {
        return (TempConcept c1, TempConcept c2) -> {
            try
            {
                final boolean c1_gt_c2 = c1.getDependencies().contains(c2.getName());
                final boolean c2_gt_c1 = c2.getDependencies().contains(c1.getName());

                if (c1_gt_c2 && c2_gt_c1)
                {
                    // If concepts depend on each other, the more general one is listed first:
                    if (c1.getGeneralizationDependencies().contains(c2.getName())) return +1;
                    else if (c2.getGeneralizationDependencies().contains(c1.getName())) return -1;
                    else return 0;
                }
                else if (c1_gt_c2) return +1;
                else if (c2_gt_c1) return -1;
                else return 0;
            }
            catch (Throwable e)
            {
                e.printStackTrace();
                return 0;
            }
        };
    }
```

```
    default List<Template> getTemplates()
    {
        return getModules().stream()
                           .map(m -> (TempModule)m)
                           .flatMap(m -> m.getTemplates().stream())
                           .collect(toList());
    }

    default List<Function> getFunctions()
    {
        return getModules().stream()
                           .map(m -> (TempModule)m)
                           .flatMap(m -> m.getFunctions().stream())
                           .collect(toList());
    }

    default List<Function> getDefinedFunctions()
    {
        return getModules().stream()
                           .map(m -> (TempModule)m)
                           .flatMap(m -> m.getDefinedFunctions().stream())
                           .collect(toList());
    }

    static TempModel create()
    {
        return new ModelImpl();
    }
}

class ModelImpl implements TempModel
{
    private final Model model;

    ModelImpl()
    {
        final Element element = extendElement(this);
        final ModelElement modelElement = extendModelElement(this, element, null, null);
        final NamedElement namedElement = extendNamedElement(this, element, modelElement, "");
        final Scope scope = extendScope(this, element, modelElement, emptyList());

        this.model = Model.extendModel(this, element, modelElement, namedElement, scope);
    }

    @Override
    public Optional<Location> getLocation()
    {
        return model.getLocation();
    }

    @Override
    public Optional<Scope> getParent()
    {
        return model.getParent();
    }

    @Override
    public List<ModelElement> getMembers()
    {
        return model.getMembers();
    }

    @Override
    public Optional<Model> getModel()
    {
        return model.getModel();
    }

    @Override
    public Optional<Module> getModule()
    {
        return model.getModule();
    }

    @Override
    public List<Module> getModules()
    {
        return model.getModules();
    }

    @Override
    public List<Task> getTasks()
    {
        return model.getTasks();
    }

    @Override
    public List<Concept> getConcepts()
    {
        return model.getConcepts();
    }

    @Override
    public List<Association> getAssociations()
    {
        return model.getAssociations();
    }

    @Override
    public String getName()
    {
        return model.getName();
    }

    @Override
    public String getDiagnosticId()
    {
        return model.getDiagnosticId();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/functions/ConceptFunctions.java

```
package cml.language.functions;

import cml.language.features.TempConcept;
import cml.language.foundation.Pair;
import cml.language.generated.ConceptRedef;
import cml.language.generated.Property;
import cml.language.generated.PropertyRedef;

import java.util.List;
import java.util.Optional;
import java.util.function.Function;
import java.util.stream.Stream;
```

```
import static cml.language.generated.ConceptRedef.createConceptRedef;
import static cml.language.generated.PropertyRedef.createPropertyRedef;
import static java.util.stream.Collectors.toList;
import static java.util.stream.Stream.concat;
import static org.jooq.lambda.Seq.seq;

@SuppressWarnings("WeakerAccess")
public class ConceptFunctions
{
    public static List<Property> redefinedInheritedConcreteProperties(TempConcept concept)
    {
        return concept.getInheritedProperties()
                      .stream()
                      .filter(Property::isConcrete)
                      .map(p -> propertyOf(concept, p.getName()).orElse(p))
                      .collect(toList());
    }

    public static Optional<Property> propertyOf(TempConcept concept, String propertyName)
    {
        return concept.getProperties().stream()
                      .filter(p -> p.getName().equals(propertyName))
                      .findFirst();
    }

    public static Function<TempConcept, ConceptRedef> conceptRedefined(TempConcept concept, TempConcept redefBase)
    {
        return c -> {
            final List<PropertyRedef> propertyRedefs = c.getNonDerivedProperties()
                                                        .stream()
                                                        .map(p -> propertyOf(concept, p.getName()).orElse(p))
                                                        .filter(p -> p.getConcept().isPresent())
                                                        .map(p -> createPropertyRedef(p, p.getConcept().get() == redefBase))
                                                        .collect(toList());

            return createConceptRedef(c, propertyRedefs);
        };
    }

    public static Function<ConceptRedef, ConceptRedef> overridden(List<ConceptRedef> redefinitions)
    {
        return conceptRedef -> {
            final Optional<ConceptRedef> override = redefinitions.stream()
                                                                 .filter(o -> o.getConcept() == conceptRedef.getConcept())
                                                                 .findFirst();

            if (override.isPresent())
            {
                final List<PropertyRedef> propertyRedefs = conceptRedef.getPropertyRedefs()
                                                                       .stream()
                                                                       .map(p -> propertyRedefOf(override.get().getPropertyRedefs(), p).orElse(p))
                                                                       .collect(toList());

                return createConceptRedef(conceptRedef.getConcept(), propertyRedefs);
            }
            else
            {
                return conceptRedef;
            }
        };
    }

    public static Optional<PropertyRedef> propertyRedefOf(List<PropertyRedef> propertyRedefs, PropertyRedef propertyRedef)
    {
        return propertyRedefs.stream()
                             .filter(p -> p.getProp().getName().equals(propertyRedef.getProp().getName()))
                             .filter(p -> p.isRedefined())
                             .findFirst();
    }

    public static List<ConceptRedef> redefinedAncestors(TempConcept concept)
    {
        return redefinedAncestors(concept, concept);
    }

    public static List<ConceptRedef> redefinedAncestors(TempConcept concept, TempConcept redefBase)
    {
        final List<ConceptRedef> redefinitions = concept.getAllAncestors()
                                                        .stream()
                                                        .map(c -> (TempConcept)c)
                                                        .map(conceptRedefined(concept, redefBase))
                                                        .collect(toList());

        final Stream<ConceptRedef> inheritedRedefinitions = concept.getAncestors()
                                                                   .stream()
                                                                   .flatMap(c -> redefinedAncestors((TempConcept) c, redefBase)
                                                                   .stream())
                                                                   .map(overridden(redefinitions));

        return seq(concat(inheritedRedefinitions, redefinitions.stream()))
            .distinct(r -> r.getConcept())
            .collect(toList());
    }

    public static List<Pair<TempConcept>> generalizationPairs(TempConcept concept)
    {
        return concept.getAncestors().stream().flatMap(
            c1 -> concept.getAncestors().stream()
                         .filter(c2 -> c1 != c2)
                         .map(c2 -> new Pair<>((TempConcept)c1, (TempConcept)c2))
        )
                         .distinct()
                         .collect(toList());
    }

    public static List<Pair<Property>> generalizationPropertyPairs(TempConcept concept)
    {
        return generalizationPairs(concept).stream().flatMap(pair ->
            pair.getLeft().getAllProperties().stream().flatMap(p1 ->
                pair.getRight()
                    .getAllProperties()
                    .stream()
                    .filter(p2 -> p1 != p2)
                    .filter(p2 -> p1.getName().equals(p2.getName()))
                    .map(p2 -> new Pair<>(p1, p2))
            )
        )
        .distinct()
        .collect(toList());
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/functions/ModelElementFunctions.java

```
package cml.language.functions;
```

```
import cml.language.features.TempConcept;
import cml.language.features.TempModule;
import cml.language.generated.Import;
import cml.language.generated.ModelElement;
import cml.language.types.TempNamedType;

import java.util.Optional;

import static cml.language.functions.ScopeFunctions.memberNamed;
import static java.util.Optional.empty;

@SuppressWarnings("WeakerAccess")
public class ModelElementFunctions
{
    public static Optional<TempModule> moduleOf(ModelElement element)
    {
        if (element instanceof Import)
        {
            final Import _import = (Import) element;
            final TempModule module = (TempModule) _import.getImportedModule();

            return Optional.of(module);
        }
        else if (element instanceof TempModule)
        {
            final TempModule module = (TempModule) element;

            return Optional.of(module);
        }
        else
        {
            //noinspection Convert2MethodRef
            return element.getParent().flatMap(s -> moduleOf(s));
        }
    }

    public static TempNamedType selfTypeOf(ModelElement element)
    {
        if (element instanceof TempConcept)
        {
            final TempConcept concept = (TempConcept) element;
            final TempNamedType namedType = TempNamedType.create(concept.getName());

            namedType.setConcept(concept);

            return namedType;
        }
        else
        {
            assert element.getParent().isPresent(): "Parent scope required in order to determine self's type.";

            return selfTypeOf(element.getParent().get());
        }
    }

    public static <T> Optional<T> siblingNamed(String name, ModelElement element, Class<T> clazz)
    {
        if (element.getParent().isPresent())
        {
            return memberNamed(name, element.getParent().get(), clazz);
        }
        else
        {
            return empty();
        }
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/functions/ModelFunctions.java

```
package cml.language.functions;

import cml.language.features.Function;
import cml.language.features.TempModule;
import cml.language.features.Template;
import cml.language.foundation.TempModel;
import cml.language.generated.*;

import java.util.Optional;

@SuppressWarnings("unused")
public class ModelFunctions
{
    public static Optional<TempModule> moduleOf(TempModel model, String name)
    {
        for (final Module module : model.getModules())
        {
            if (module.getName().equals(name))
            {
                return Optional.of((TempModule) module);
            }
        }

        return Optional.empty();
    }

    public static Optional<Concept> conceptOf(Model model, String name)
    {
        return model.getConcepts()
                    .stream()
                    .filter(concept -> concept.getName().equals(name))
                    .findFirst();
    }

    public static Optional<Association> associationOf(TempModel model, String name)
    {
        return model.getAssociations()
                    .stream()
                    .filter(association -> association.getName().equals(name))
                    .findFirst();
    }

    public static Optional<Task> targetOf(TempModel model, String name)
    {
        return model.getTasks()
                    .stream()
                    .filter(target -> target.getName().equals(name))
                    .findFirst();
    }

    public static Optional<Template> templateOf(TempModel model, String name)
    {
        return model.getTemplates()
                    .stream()
                    .filter(t -> t.getName().equals(name))
```

```
                .findFirst();
    }

    public static Optional<Function> functionOf(TempModel model, String name)
    {
        return model.getFunctions()
                .stream()
                .filter(t -> t.getName().equals(name))
                .findFirst();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/functions/ModelVisitorFunctions.java

```
package cml.language.functions;

import cml.language.features.Function;
import cml.language.features.TempConcept;
import cml.language.features.TempModule;
import cml.language.foundation.TempModel;
import cml.language.generated.Association;
import cml.language.generated.Expression;
import cml.language.generated.Property;
import cml.language.loader.ModelVisitor;

import static org.jooq.lambda.Seq.seq;

@SuppressWarnings("WeakerAccess")
public class ModelVisitorFunctions
{
    public static void visitModel(TempModel model, ModelVisitor visitor)
    {
        visitor.visit(model);

        model.getModules().forEach(m -> visitModule((TempModule) m, visitor));

        model.getConcepts().forEach(c -> visitConcept((TempConcept) c, visitor));

        model.getAssociations().forEach(a -> visitAssociation(a, visitor));
    }

    public static void visitModule(TempModule module, ModelVisitor visitor)
    {
        visitor.visit(module);

        module.getDefinedFunctions().forEach(f -> visitFunction(f, visitor));
    }

    public static void visitFunction(Function function, ModelVisitor visitor)
    {
        visitor.visit(function);

        function.getExpression().ifPresent(expression -> visitExpression(expression, visitor));
    }

    public static void visitConcept(TempConcept concept, ModelVisitor visitor)
    {
        visitor.visit(concept);

        seq(concept.getProperties()).forEach(p -> visitProperty(p, visitor));
    }

    public static void visitAssociation(Association association, ModelVisitor visitor)
    {
        visitor.visit(association);

        association.getAssociationEnds().forEach(visitor::visit);
    }

    public static void visitProperty(Property property, ModelVisitor visitor)
    {
        visitor.visit(property);

        property.getValue().ifPresent(expression -> visitExpression(expression, visitor));
    }

    public static void visitExpression(Expression expression, ModelVisitor visitor)
    {
        visitor.visit(expression);

        expression.getOperands().forEach(e -> visitExpression(e, visitor));
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/functions/ModuleFunctions.java

```
package cml.language.functions;

import cml.language.features.TempConcept;
import cml.language.features.TempModule;
import cml.language.foundation.TempModel;
import cml.language.generated.Import;
import cml.language.generated.Location;

import java.util.Optional;

import static cml.language.functions.ModelFunctions.moduleOf;
import static cml.language.generated.Import.createImport;

@SuppressWarnings("unused")
public class ModuleFunctions
{
    public static Optional<TempModule> importedModuleOf(TempModule module, String name)
    {
        for (final TempModule m: module.getImportedModules())
        {
            if (m.getName().equals(name))
            {
                return Optional.of(m);
            }
        }

        return Optional.empty();
    }

    public static Optional<TempModule> selfOrImportedModuleOf(TempModule module, String name)
    {
        if (module.getName().equals(name))
        {
            return Optional.of(module);
        }

        return importedModuleOf(module, name);
    }
```

```
    public static Optional<TempConcept> conceptOf(TempModule module, String name)
    {
        return module.getAllConcepts()
                     .stream()
                     .filter(concept -> concept.getName().equals(name))
                     .findFirst();
    }

    public static Import createImportOfModule(final String moduleName, final Location location, final TempModule importingModule)
    {
        assert importingModule.getModel().isPresent();

        final TempModel model = (TempModel) importingModule.getModel().get();
        final Optional<TempModule> existingModule = moduleOf(model, moduleName);
        final boolean firstImport = !existingModule.isPresent();
        final TempModule importedModule = firstImport ? TempModule.create(model, moduleName) : existingModule.get();

        return createImport(moduleName, importingModule, location, importedModule, firstImport);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/functions/ScopeFunctions.java

```
package cml.language.functions;

import cml.language.expressions.LambdaScope;
import cml.language.expressions.Path;
import cml.language.features.TempConcept;
import cml.language.generated.*;
import cml.language.types.TempNamedType;

import java.util.List;
import java.util.Optional;

import static cml.language.functions.ModelElementFunctions.selfTypeOf;
import static java.util.Optional.empty;
import static java.util.Optional.ofNullable;
import static java.util.stream.Collectors.toList;

@SuppressWarnings("WeakerAccess")
public class ScopeFunctions
{
    public static <T> List<T> membersOf(Scope scope, Class<T> clazz)
    {
        //noinspection unchecked
        return scope.getMembers()
                    .stream()
                    .filter(e -> clazz.isAssignableFrom(e.getClass()))
                    .map(e -> (T)e)
                    .collect(toList());
    }

    public static <T> Optional<T> memberNamed(String name, Scope scope, Class<T> clazz)
    {
        //noinspection unchecked
        return membersOf(scope, NamedElement.class)
            .stream()
            .filter(e -> name.equals(e.getName()))
            .filter(e -> clazz.isAssignableFrom(e.getClass()))
            .map(e -> (T)e)
            .findFirst();
    }

    public static Optional<Type> typeOfMemberNamed(String name, Scope scope)
    {
        final Optional<TypedElement> typedElement = memberNamed(name, scope, TypedElement.class);

        return typedElement.map(e -> e.getType());
    }

    public static <T> Optional<T> elementNamed(String name, Scope scope, Class<T> clazz)
    {
        final Optional<T> member = memberNamed(name, scope, clazz);

        if (member.isPresent())
        {
            return member;
        }
        else if (scope.getParent().isPresent())
        {
            return elementNamed(name, scope.getParent().get(), clazz);
        }

        return empty();
    }

    public static Optional<Scope> scopeOfType(Type type, Scope scope)
    {
        if (type instanceof TempNamedType)
        {
            final TempNamedType namedType = (TempNamedType)type;
            return elementNamed(namedType.getName(), scope, Scope.class);
        }

        return empty();
    }

    public static Optional<Type> typeOfVariableNamed(String name, Scope scope)
    {
        if (name.equals("self"))
        {
            return ofNullable(selfTypeOf(scope));
        }
        else if (scope instanceof Let)
        {
            final Let let = (Let) scope;

            if (let.getVariable().equals(name))
            {
                return Optional.of(let.getVariableExpr().getType());
            }
        }
        else if (scope instanceof LambdaScope)
        {
            final LambdaScope lambdaScope = (LambdaScope) scope;
            final Optional<Type> type = ofNullable(lambdaScope.getParameters().get(name));

            if (type.isPresent()) return type;
        }
        else if (scope instanceof Path)
        {
            final Path path = (Path) scope;
            final Optional<Scope> typeScope = scopeOfType(path.getType(), scope);
            final Optional<Type> type = typeScope.flatMap(s -> typeOfVariableNamed(name, s));

            if (type.isPresent()) return type;
```

```
        }
        else if (scope instanceof TempConcept)
        {
            final Optional<Type> memberType = typeOfMemberNamed(name, scope);
            if (memberType.isPresent()) return memberType;

            final TempConcept concept = (TempConcept) scope;

            for (Concept ancestor: concept.getAncestors())
            {
                final Optional<Type> type = typeOfVariableNamed(name, ancestor);

                if (type.isPresent()) return type;
            }
        }
        else
        {
            final Optional<Type> memberType = typeOfMemberNamed(name, scope);
            if (memberType.isPresent()) return memberType;
        }

        return scope.getParent().flatMap(s -> typeOfVariableNamed(name, s));
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/functions/TypeFunctions.java

package cml.language.functions;

import cml.language.expressions.TypeCast;
import cml.language.features.FunctionParameter;
import cml.language.features.TempConcept;
import cml.language.generated.Type;
import cml.language.generated.ValueType;
import cml.language.types.FunctionType;
import cml.language.types.MemberType;
import cml.language.types.TempNamedType;
import cml.language.types.TupleType;

import java.util.List;
import java.util.Objects;
import java.util.Optional;

import static cml.language.functions.ModelElementFunctions.selfTypeOf;
import static cml.language.generated.ValueType.createValueType;
import static cml.primitives.Types.subtype;
import static org.jooq.lambda.Seq.seq;
import static org.jooq.lambda.Seq.zip;

@SuppressWarnings("WeakerAccess")
public class TypeFunctions
{
    public static Type withCardinality(Type type, String cardinality)
    {
        if (type instanceof ValueType)
        {
            return createValueType(cardinality, type.getName());
        }
        else if (type instanceof TempNamedType)
        {
            final TempNamedType namedType = (TempNamedType) type;
            final TempNamedType newType = TempNamedType.create(namedType.getName(), cardinality);

            namedType.getConcept().map(c -> (TempConcept)c).ifPresent(newType::setConcept);

            return newType;
        }
        else if (type instanceof TupleType)
        {
            final TupleType tupleType = (TupleType) type;
            return new TupleType(seq(tupleType.getElements()), cardinality);
        }
        else if (type.isUndefined())
        {
            return type;
        }
        else
        {
            throw new UnsupportedOperationException("withCardinality(" + type + ", " + cardinality + ")");
        }
    }

    public static boolean isElementTypeAssignableFrom(Type leftType, Type rightType)
    {
        assert leftType.isRequired();
        assert rightType.isRequired();

        if (leftType instanceof ValueType && rightType instanceof ValueType)
        {
            return subtype(rightType.getName(), leftType.getName());
        }
        else if (leftType instanceof TempNamedType)
        {
            final TempNamedType leftNamedType = (TempNamedType) leftType;

            if (rightType instanceof TempNamedType)
            {
                final TempNamedType rightNamedType = (TempNamedType) rightType;

                if (isNameEquals(leftNamedType, rightNamedType))
                {
                    return true;
                }
                else if (leftNamedType.getConcept().isPresent() && rightNamedType.getConcept().isPresent())
                {
                    return seq(rightNamedType.getConcept()).map(c -> (TempConcept)c)
                                                           .flatMap(c -> c.getAllGeneralizations().stream())
                                                           .anyMatch(c -> isElementTypeAssignableFrom(leftNamedType, selfTypeOf(c)));
                }
            }

            return false;
        }
        else if (leftType instanceof TupleType)
        {
            final TupleType leftTupleType = (TupleType) leftType;

            if (rightType instanceof TupleType)
            {
                final TupleType rightTupleType = (TupleType)rightType;

                return zip(leftTupleType.getElements(), rightTupleType.getElements()).allMatch(t -> t.v1.isAssignableFrom(t.v2));
            }

            return false;
        }
```

```
    else if (leftType instanceof FunctionType)
    {
        final FunctionType leftFunctionType = (FunctionType) leftType;

        if (rightType instanceof FunctionType)
        {
            final FunctionType rightFunctionType = (FunctionType)rightType;

            return isAssignableFrom(leftFunctionType.getParams(), rightFunctionType.getParams()) &&
                   isAssignableFrom(leftFunctionType.getResult(), rightFunctionType.getResult());
        }

        return false;
    }
    else if (leftType instanceof MemberType)
    {
        final MemberType leftMemberType = (MemberType) leftType;

        return isElementTypeAssignableFrom(leftMemberType.getBaseType(), rightType);
    }
    else
    {
        return false;
    }
}

public static boolean isCardinalityAssignableFrom(Type thisType, Type thatType)
{
    return Objects.equals(thisType.getCardinality(), thatType.getCardinality()) ||
           (thisType.isOptional() && thatType.isRequired()) || (thisType.isSequence());
}

public static boolean isAssignableFrom(Type thisType, Type thatType)
{
    if (thatType.isNothing())
    {
        return thisType.isSomething() && !thisType.isRequired();
    }
    else
    {
        return isElementTypeAssignableFrom(thisType.getElementType(), thatType.getElementType()) &&
            isCardinalityAssignableFrom(thisType, thatType);
    }
}

public static boolean isEqualTo(Type thisType, Type thatType)
{
    if (thisType instanceof ValueType && thatType instanceof ValueType)
    {
        return isNameEquals(thisType, thatType) && isCardinalityEquals(thisType, thatType);
    }
    else if (thisType instanceof TempNamedType && thatType instanceof TempNamedType)
    {
        return isNameEquals(thisType, thatType) && isCardinalityEquals(thisType, thatType);
    }

    return false;
}

public static boolean isNameEquals(final Type thisType, final Type thatType)
{
    assert thisType.getName() != null;
    assert thatType.getName() != null;

    if (thisType.isPrimitive() && thatType.isPrimitive())
    {
        return thisType.getName().equalsIgnoreCase(thatType.getName());
    }
    else
    {
        return thisType.getName().equals(thatType.getName());
    }
}

public static boolean isCardinalityEquals(final Type thisType, final Type thatType)
{
    return Objects.equals(thisType.getCardinality(), thatType.getCardinality());
}

@SuppressWarnings("SimplifiableIfStatement")
public static boolean isCastAllowed(String operator, Type exprType, Type castType)
{
    if (operator.equals(TypeCast.ASQ) && castType.isRequired())
    {
        return false;
    }
    else if (operator.equals(TypeCast.ASB) && castType.isRequired())
    {
        return (isElementTypeAssignableFrom(exprType.getElementType(), castType.getElementType()) ||
                isElementTypeAssignableFrom(castType.getElementType(), exprType.getElementType()))
               && exprType.isSingle();
    }
    else
    {
        return isGeneralizationAssignableFrom(exprType, castType);
    }
}

public static boolean isGeneralizationAssignableFrom(final Type exprType, final Type castType)
{
    return isElementGeneralizationAssignableFrom(exprType.getElementType(), castType.getElementType()) &&
           isCardinalityAssignableFrom(castType, exprType);
}

@SuppressWarnings("SimplifiableIfStatement")
public static boolean isElementGeneralizationAssignableFrom(final Type thisElementType, final Type thatElementType)
{
    if (isElementTypeAssignableFrom(thisElementType, thatElementType)) return true;

    return seq(thisElementType.getConcept()).map(c -> (TempConcept)c)
                                            .flatMap(c -> c.getAllGeneralizations().stream())
                                            .anyMatch(c -> isElementTypeAssignableFrom(selfTypeOf(c), thatElementType));
}

public static Optional<Type> firstGeneralizationAssignableFrom(final Type leftType, final Type rightType)
{
    final Type leftElementType = leftType.getElementType();
    final Type rightElementType = rightType.getElementType();

    final Optional<TempConcept> match = seq(leftElementType.getConcept()).map(c -> (TempConcept)c)
                                                                          .flatMap(c -> c.getAllGeneralizations().stream())
                                                                          .findFirst(c -> isElementTypeAssignableFrom(selfTypeOf(c), rightElemen

    return match.map(c -> withCardinality(selfTypeOf(c), leftType.getCardinality()));
}

public static int getParamIndexOfMatchingType(List<FunctionParameter> parameters, Type type)
```

```
    {
        return getParamIndexOfMatchingType(parameters, type, -1);
    }

    public static int getParamIndexOfMatchingType(List<FunctionParameter> parameters, Type type, int skipIndex)
    {
        assert type.isParameter(): "Must be called only when type is a parameter.";

        int index = 0;
        for (FunctionParameter parameter: parameters)
        {
            if (isElementTypeAssignableFrom(parameter.getMatchingResultType().getElementType(), type.getBaseType().getElementType()))
            {
                if (index != skipIndex) break;
            }
            index++;
        }

        return index;
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/AbstractPropertyInAbstractConcept.java

```
package cml.language.invariants;

import cml.language.features.TempConcept;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Property;

import static java.util.Collections.emptyList;

public class AbstractPropertyInAbstractConcept implements Invariant<Property>
{
    @Override
    public boolean evaluate(Property self)
    {
        if (self.getParent().isPresent() && self.getParent().get() instanceof TempConcept)
        {
            final TempConcept concept = (TempConcept) self.getParent().get();

            return self.isConcrete() || concept.isAbstraction();
        }
        else
        {
            return self.isConcrete();
        }
    }

    @Override
    public Diagnostic createDiagnostic(Property self)
    {
        return new Diagnostic("abstract_property_in_abstract_concept", self, emptyList());
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/AbstractPropertyRedefinition.java

```
package cml.language.invariants;

import cml.language.features.TempConcept;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Property;

import java.util.List;
import java.util.function.Predicate;

import static java.util.stream.Collectors.toList;
import static org.jooq.lambda.Seq.seq;

public class AbstractPropertyRedefinition implements Invariant<TempConcept>
{
    @Override
    public boolean evaluate(TempConcept self)
    {
        return self.isAbstraction() || seq(self.getInheritedAbstractProperties()).allMatch(abstractPropertyRedefinedIn(self));
    }

    private Predicate<Property> abstractPropertyRedefinedIn(TempConcept self)
    {
        return p1 -> self.getProperties()
                         .stream()
                         .filter(p2 -> p1.getName().equals(p2.getName()))
                         .filter(Property::isConcrete)
                         .count() > 0;
    }

    @Override
    public Diagnostic createDiagnostic(TempConcept self)
    {
        final List<Property> abstractProperties = seq(self.getInheritedAbstractProperties())
            .filter(abstractPropertyRedefinedIn(self).negate())
            .collect(toList());

        return new Diagnostic("abstract_property_redefinition", self, abstractProperties);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/AssociationEndPropertyFoundInModel.java

```
package cml.language.invariants;

import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.AssociationEnd;
import cml.language.generated.ModelElement;

import java.util.List;
import java.util.Optional;

import static java.util.stream.Collectors.toList;
import static java.util.stream.Stream.concat;
import static java.util.stream.Stream.of;

public class AssociationEndPropertyFoundInModel implements Invariant<AssociationEnd>
{
    @Override
    public boolean evaluate(AssociationEnd self)
    {
        return self.getAssociatedConcept().isPresent() && self.getAssociatedProperty().isPresent();
    }

    @Override
```

```
    public Diagnostic createDiagnostic(AssociationEnd self)
    {
        @SuppressWarnings("ConstantConditions")
        final List<ModelElement> participants = concat(of(self.getAssociatedConcept()), of(self.getAssociatedProperty()))
            .filter(Optional::isPresent)
            .map(e -> (ModelElement)e.get())
            .collect(toList());

        return new Diagnostic("association_end_property_found_in_model", self, participants);
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/invariants/AssociationEndTypeMatchesPropertyType.java

package cml.language.invariants;

import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.AssociationEnd;

import static cml.language.functions.TypeFunctions.isEqualTo;
import static java.util.Collections.singletonList;

public class AssociationEndTypeMatchesPropertyType implements Invariant<AssociationEnd>
{
    @Override
    public boolean evaluate(AssociationEnd self)
    {
        return !self.getPropertyType().isPresent() ||
            !self.getAssociatedProperty().isPresent() ||
            isEqualTo(self.getPropertyType().get(), self.getAssociatedProperty().get().getType());
    }

    @Override
    public Diagnostic createDiagnostic(AssociationEnd self)
    {
        assert self.getAssociatedProperty().isPresent();

        return new Diagnostic(
            "association_end_type_matches_property_type",
            self,
            singletonList(self.getAssociatedProperty().get()));
    }
}

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/AssociationEndTypesMustMatch.java

package cml.language.invariants;

import cml.language.features.TempConcept;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Association;
import cml.language.generated.AssociationEnd;
import cml.language.generated.Property;

import java.util.Optional;

import static cml.language.functions.ModelElementFunctions.selfTypeOf;
import static cml.language.functions.TypeFunctions.isEqualTo;

public class AssociationEndTypesMustMatch implements Invariant<Association>
{
    @Override
    public boolean evaluate(Association self)
    {
        if (self.getAssociationEnds().size() != 2)
        {
            return true;
        }

        final Optional<AssociationEnd> first = self.getAssociationEnds().stream().findFirst();
        final Optional<AssociationEnd> last = self.getAssociationEnds().stream().reduce((previous, next) -> next);

        if (!first.isPresent() || !last.isPresent())
        {
            return true;
        }

        final AssociationEnd end1 = first.get();
        final AssociationEnd end2 = last.get();

        if (!end1.getAssociatedConcept().isPresent() || !end1.getAssociatedProperty().isPresent() ||
            !end2.getAssociatedConcept().isPresent() || !end2.getAssociatedProperty().isPresent())
        {
            return true;
        }

        final TempConcept firstConcept = end1.getAssociatedConcept().map(c -> (TempConcept) c).get();
        final TempConcept secondConcept = end2.getAssociatedConcept().map(c -> (TempConcept) c).get();
        final Property firstProperty = end1.getAssociatedProperty().get();
        final Property secondProperty = end2.getAssociatedProperty().get();

        return typesMatch(firstConcept, secondProperty) && typesMatch(secondConcept, firstProperty);
    }

    private static boolean typesMatch(TempConcept concept, Property property)
    {
        return isEqualTo(property.getType().getElementType(), selfTypeOf(concept));
    }

    @Override
    public Diagnostic createDiagnostic(Association self)
    {
        return new Diagnostic("association_end_types_must_match", self, self.getAssociationEnds());
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/invariants/AssociationMustHaveTwoAssociationEnds.java

package cml.language.invariants;

import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Association;

public class AssociationMustHaveTwoAssociationEnds implements Invariant<Association>
{
    @Override
    public boolean evaluate(Association self)
    {
        return self.getAssociationEnds().size() == 2;
    }

    @Override
```

```
    public Diagnostic createDiagnostic(Association self)
    {
        return new Diagnostic("association_must_have_two_association_ends", self, self.getAssociationEnds());
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/CompatibleGeneralizations.java

```
package cml.language.invariants;

import cml.language.features.TempConcept;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Property;

import java.util.List;
import java.util.stream.Stream;

import static cml.language.functions.ConceptFunctions.generalizationPropertyPairs;
import static cml.language.functions.TypeFunctions.isAssignableFrom;
import static cml.language.functions.TypeFunctions.isEqualTo;
import static java.util.stream.Collectors.toList;

public class CompatibleGeneralizations implements Invariant<TempConcept>
{
    @Override
    public boolean evaluate(TempConcept self)
    {
        return generalizationPropertyPairs(self)
                .stream()
                .allMatch(pair -> isEqualTo(pair.getLeft().getType(), pair.getRight().getType()));
    }

    @Override
    public Diagnostic createDiagnostic(TempConcept self)
    {
        final List<Property> conflictingProperties = generalizationPropertyPairs(self)
                                                        .stream()
                                                        .filter(pair -> !isEqualTo(pair.getLeft().getType(), pair.getRight().getType()))
                                                        .flatMap(pair -> Stream.of(pair.getLeft(), pair.getRight()))
                                                        .collect(toList());

        return new Diagnostic("compatible_generalizations", self, conflictingProperties);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/ConflictRedefinition.java

```
package cml.language.invariants;

import cml.language.features.TempConcept;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.foundation.Pair;
import cml.language.generated.Property;

import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Stream;

import static cml.language.functions.ConceptFunctions.generalizationPropertyPairs;
import static cml.language.functions.TypeFunctions.isAssignableFrom;
import static java.util.stream.Collectors.toList;

public class ConflictRedefinition implements Invariant<TempConcept>
{
    @Override
    public boolean evaluate(TempConcept self)
    {
        return getConflictingPropertyPairs(self)
            .map(Pair::getLeft)
            .allMatch(propertyRedefinedIn(self));
    }

    private Stream<Pair<Property>> getConflictingPropertyPairs(TempConcept self)
    {
        return generalizationPropertyPairs(self)
                .stream()
                .filter(pair -> isAssignableFrom(pair.getLeft().getType(), pair.getRight().getType()))
                .filter(pair -> pair.getLeft().isDerived() ||
                    pair.getLeft().getValue().isPresent() ||
                    pair.getRight().isDerived() ||
                    pair.getRight().getValue().isPresent());
    }

    private Predicate<Property> propertyRedefinedIn(TempConcept self)
    {
        return p1 -> self.getProperties()
                        .stream()
                        .anyMatch(p2 -> p1.getName().equals(p2.getName()));
    }

    @Override
    public Diagnostic createDiagnostic(TempConcept self)
    {
        final List<Property> conflictingProperties = getConflictingPropertyPairs(self)
            .flatMap(pair -> Stream.of(pair.getLeft(), pair.getRight()))
            .filter(propertyRedefinedIn(self).negate())
            .collect(toList());

        return new Diagnostic("conflict_redefinition", self, conflictingProperties);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/ExpressionInvariant.java

```
package cml.language.invariants;

import cml.language.expressions.Invocation;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Expression;

public class ExpressionInvariant implements Invariant<Expression>
{
    private final InvocationInvariant invocationInvariant = new InvocationInvariant();

    @Override
    public boolean isCritical()
    {
        return true;
    }

    @Override
    public boolean evaluate(final Expression self)
```

```
    {
        for (final Expression expr: self.getOperands())
        {
            final boolean pass = evaluateInvariantsOf(expr);

            if (!pass) return false;
        }

        return evaluateInvariantsOf(self);
    }

    @Override
    public Diagnostic createDiagnostic(final Expression self)
    {
        for (final Expression expr: self.getOperands())
        {
            final boolean pass = evaluateInvariantsOf(expr);

            if (!pass) return createDiagnosticOf(expr);
        }

        return createDiagnosticOf(self);
    }

    @SuppressWarnings("SimplifiableIfStatement")
    private boolean evaluateInvariantsOf(final Expression expr)
    {
        if (expr instanceof Invocation) return invocationInvariant.evaluate((Invocation) expr);
        else return true;
    }

    private Diagnostic createDiagnosticOf(final Expression expr)
    {
        if (expr instanceof Invocation) return invocationInvariant.createDiagnostic((Invocation) expr);
        else throw new IllegalArgumentException("Unexpected diagnostic for expression: " + expr);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/GeneralizationCompatibleRedefinition.java

```
package cml.language.invariants;

import cml.language.features.TempConcept;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Property;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static cml.language.functions.TypeFunctions.isAssignableFrom;

public class GeneralizationCompatibleRedefinition implements Invariant<Property>
{
    @Override
    public boolean evaluate(Property self)
    {
        return getRedefinedProperties(self).allMatch(p -> isAssignableFrom(p.getType(), self.getType()));
    }

    @Override
    public Diagnostic createDiagnostic(Property self)
    {
        final List<Property> conflictingProperties = getRedefinedProperties(self)
            .filter(p -> !isAssignableFrom(p.getType(), self.getType()))
            .collect(Collectors.toList());

        return new Diagnostic("generalization_compatible_redefinition", self, conflictingProperties);
    }

    private Stream<Property> getRedefinedProperties(Property self)
    {
        if (self.getParent().isPresent() && self.getParent().get() instanceof TempConcept)
        {
            final TempConcept concept = (TempConcept) self.getParent().get();

            return concept.getInheritedProperties().stream()
                          .filter(p -> p.getName().equals(self.getName()));
        }
        else
        {
            return Stream.empty();
        }
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/InvocationInvariant.java

```
package cml.language.invariants;

import cml.language.expressions.Invocation;
import cml.language.features.Function;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.ModelElement;
import org.jooq.lambda.tuple.Tuple2;

import java.util.List;

import static java.util.Collections.emptyList;
import static org.jooq.lambda.Seq.seq;

public class InvocationInvariant implements Invariant<Invocation>
{
    private static final String MESSAGE_UNABLE_TO_FIND_FUNCTION_OF_INVOCATION = "Unable to find function of invocation: ";
    private static final String MESSAGE_SHOULD_MATCH_NUMBER_OF_PARAMS_IN_FUNCTION = "Number of arguments in invocation should match the number of
    private static final String MESSAGE_SHOULD_MATCH_PARAMETER_TYPE_IN_FUNCTION = "Argument type should match parameter type in function: ";
    private static final String MESSAGE_SHOULD_MATCH_NUMBER_OF_PROPS_IN_CONCEPT = "Number of arguments in invocation should match the number of pr
    private static final String MESSAGE_SHOULD_MATCH_PARAMETER_TYPE_IN_CONCEPT_PROPERTY = "Argument type should match parameter type in concept pr

    @Override
    public boolean evaluate(final Invocation self)
    {
        if (self.getParameters().size() == self.getArguments().size())
        {
            return seq(self.getParameterizedArguments()).allMatch(t -> self.typeMatches(t.v1, t.v2));
        }

        return false;
    }

    @Override
    public Diagnostic createDiagnostic(final Invocation self)
    {
```

```
        return new Diagnostic(
            "matching_function_for_invocation",
            self,
            getDiagnosticMessage(self),
            getDiagnosticParticipants(self));
    }

    private String getDiagnosticMessage(final Invocation self)
    {
        if (self.getFunction().isPresent())
        {
            final Function function = self.getFunction().get();

            if (function.getParameters().size() == self.getArguments().size())
            {
                return MESSAGE_SHOULD_MATCH_PARAMETER_TYPE_IN_FUNCTION + self.getName();
            }
            else
            {
                return MESSAGE_SHOULD_MATCH_NUMBER_OF_PARAMS_IN_FUNCTION + self.getName();
            }
        }
        else if (self.getConcept().isPresent())
        {
            if (self.getParameters().size() == self.getArguments().size())
            {
                return MESSAGE_SHOULD_MATCH_PARAMETER_TYPE_IN_CONCEPT_PROPERTY + self.getName();
            }
            else
            {
                return MESSAGE_SHOULD_MATCH_NUMBER_OF_PROPS_IN_CONCEPT + self.getName();
            }
        }

        return MESSAGE_UNABLE_TO_FIND_FUNCTION_OF_INVOCATION + self.getName();
    }

    private List<? extends ModelElement> getDiagnosticParticipants(final Invocation self)
    {
        if (self.getFunction().isPresent())
        {
            final Function function = self.getFunction().get();

            if (function.getParameters().size() == self.getArguments().size())
            {
                return seq(self.getParameterizedArguments()).filter(t -> !self.typeMatches(t.v1, t.v2))
                                                            .flatMap(Tuple2::toSeq)
                                                            .map(e -> (ModelElement)e)
                                                            .toList();
            }
        }

        return emptyList();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/NotOwnGeneralization.java

```
package cml.language.invariants;

import cml.language.features.TempConcept;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;

public class NotOwnGeneralization implements Invariant<TempConcept>
{
    @Override
    public boolean evaluate(TempConcept self)
    {
        return !self.getAllGeneralizations().contains(self);
    }

    @Override
    public Diagnostic createDiagnostic(TempConcept self)
    {
        return new Diagnostic("not_own_generalization", self);
    }

    @Override
    public boolean isCritical()
    {
        return true;
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/PropertyTypeAssignableFromExpressionType.java

```
package cml.language.invariants;

import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Property;

import static cml.language.functions.TypeFunctions.isAssignableFrom;
import static java.lang.String.format;

public class PropertyTypeAssignableFromExpressionType implements Invariant<Property>
{
    @Override
    public boolean evaluate(Property self)
    {
        return !(self.getDeclaredType().isPresent() && self.getValue().isPresent()) ||
            isAssignableFrom(self.getDeclaredType().get(), self.getValue().get().getType());
    }

    @Override
    public Diagnostic createDiagnostic(Property self)
    {
        assert self.getDeclaredType().isPresent();
        assert self.getValue().isPresent();

        return new Diagnostic(
            "property_type_assignable_from_expression_type",
            self,
            format(
                "Declared type is %s but type inferred from expression is %s.",
                self.getDeclaredType().get().getDiagnosticId(),
                self.getValue().get().getType().getDiagnosticId()));
    }

}
```

==> cml-compiler/cml-language/src/main/java/cml/language/invariants/PropertyTypeSpecifiedOrInferred.java

```
package cml.language.invariants;

import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Property;
import cml.language.generated.UndefinedType;

public class PropertyTypeSpecifiedOrInferred implements Invariant<Property>
{
    @Override
    public boolean evaluate(Property self)
    {
        return self.getType().isDefined();
    }

    @Override
    public Diagnostic createDiagnostic(Property self)
    {
        assert self.getType() instanceof UndefinedType;

        final UndefinedType undefinedType = (UndefinedType) self.getType();

        return new Diagnostic(
            "property_type_specified_or_inferred",
            self,
            undefinedType.getErrorMessage());
    }

}


==> cml-compiler/cml-language/src/main/java/cml/language/invariants/UniquePropertyName.java

package cml.language.invariants;

import cml.language.features.TempConcept;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.Invariant;
import cml.language.generated.Property;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class UniquePropertyName implements Invariant<Property>
{
    @Override
    public boolean evaluate(Property self)
    {
        return getConflictingProperties(self).count() == 0;
    }

    @Override
    public Diagnostic createDiagnostic(Property self)
    {
        final List<Property> participants = getConflictingProperties(self).collect(Collectors.toList());

        return new Diagnostic("unique_property_name", self, participants);
    }

    private Stream<Property> getConflictingProperties(Property self)
    {
        if (self.getParent().isPresent() && self.getParent().get() instanceof TempConcept)
        {
            final TempConcept concept = (TempConcept) self.getParent().get();

            return concept.getProperties()
                          .stream()
                          .filter(p -> p != self && p.getName().equals(self.getName()));
        }
        else
        {
            return Stream.empty();
        }
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/loader/FunctionLinker.java

package cml.language.loader;

import cml.language.expressions.Invocation;
import cml.language.foundation.TempModel;
import cml.language.generated.Expression;

import static cml.language.functions.ModelFunctions.*;
import static org.jooq.lambda.Seq.seq;

public class FunctionLinker implements ModelVisitor
{
    private TempModel model;

    @Override
    public void visit(final TempModel model)
    {
        this.model = model;
    }

    @Override
    public void visit(final Expression expression)
    {
        if (expression instanceof Invocation)
        {
            final Invocation invocation = (Invocation) expression;

            if (!invocation.getFunction().isPresent() && !invocation.getConcept().isPresent())
            {
                functionOf(model, invocation.getName()).ifPresent(f -> invocation.setFunction(f));

                if (!invocation.getFunction().isPresent())
                {
                    templateOf(model, invocation.getName()).ifPresent(t -> invocation.setFunction(t.getFunction()));
                }

                if (invocation.getFunction().isPresent())
                {
                    seq(invocation.getTypedLambdaArguments()).filter(t -> !t.v2.getFunctionType().isPresent())
                                                             .forEach(t -> t.v2.setFunctionType(t.v1));
                }
            }

            if (!invocation.getFunction().isPresent() && !invocation.getConcept().isPresent())
            {
                conceptOf(model, invocation.getName()).ifPresent(c -> invocation.setConcept(c));
            }
        }
```

```
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/loader/LambdaScopeLinker.java

package cml.language.loader;

import cml.language.expressions.Invocation;
import cml.language.generated.Expression;

public class LambdaScopeLinker implements ModelVisitor
{
    @Override
    public void visit(final Expression expression)
    {
        if (expression instanceof Invocation)
        {
            final Invocation invocation = (Invocation) expression;

            // First, lets arguments link scope of its own lambdas.
            // E.g.: select(recurse(self, { children }), { ranking > 10 })

            invocation.getArguments().forEach(this::visit);

            if (invocation.getFunction().isPresent())
            {
                invocation.getTypedLambdaArguments().forEach(
                    (functionType, lambda) ->
                    {
                        if (lambda.getFunctionType().isPresent() && !lambda.isInnerExpressionInSomeScope())
                        {
                            invocation.createScopeFor(lambda);
                        }
                    });

                invocation.getUntypedParameterlessLambdaArguments().forEach(
                    (functionType, lambda) ->
                    {
                        if (!lambda.isInnerExpressionInSomeScope())
                        {
                            invocation.createScopeFor(lambda);
                        }
                    });
            }
        }
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/loader/ModelAugmentationException.java

package cml.language.loader;

class ModelAugmentationException extends ModelLoadingException
{
    ModelAugmentationException(String message, Object... args)
    {
        super(message, args);
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/loader/ModelAugmenter.java

package cml.language.loader;

import cml.language.features.TempConcept;
import cml.language.features.TempModule;
import cml.language.generated.*;
import cml.language.grammar.CMLBaseListener;
import cml.language.grammar.CMLParser;
import cml.language.grammar.CMLParser.ConceptDeclarationContext;
import cml.language.types.TempNamedType;
import org.antlr.v4.runtime.ParserRuleContext;
import org.antlr.v4.runtime.Token;
import org.antlr.v4.runtime.tree.ParseTree;
import org.jetbrains.annotations.Nullable;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

import static cml.language.functions.ModuleFunctions.conceptOf;
import static cml.language.generated.AssociationEnd.createAssociationEnd;
import static cml.language.generated.Inheritance.createInheritance;
import static org.jooq.lambda.Seq.seq;

class ModelAugmenter extends CMLBaseListener
{
    private static final String NO_CONCEPT_NAME_PROVIDED_FOR_ASSOCIATION_END = "No concept name provided for association end.";
    private static final String NO_PROPERTY_NAME_PROVIDED_FOR_ASSOCIATION_END = "No property name provided for association end.";

    private final TempModule module;

    ModelAugmenter(TempModule module)
    {
        this.module = module;
    }

    @Override
    public void enterConceptDeclaration(ConceptDeclarationContext ctx)
    {
        if (ctx.generalizations() != null)
        {
            final List<String> ancestorNames = ctx.generalizations().NAME()
                                                  .stream()
                                                  .map(ParseTree::getText)
                                                  .collect(Collectors.toList());

            final List<TempConcept> foundAncestors = ancestorNames.stream()
                                                                  .map(name -> conceptOf(module, name))
                                                                  .filter(Optional::isPresent)
                                                                  .map(Optional::get)
                                                                  .collect(Collectors.toList());

            final List<String> foundAncestorNames = foundAncestors.stream()
                                                                  .map(NamedElement::getName)
                                                                  .collect(Collectors.toList());

            foundAncestors.forEach(ancestor -> createInheritance(ancestor, ctx.concept));

            final List<String> missingAncestorNames = ancestorNames.stream()
                                                                   .filter(name -> !foundAncestorNames.contains(name))
                                                                   .collect(Collectors.toList());

            if (missingAncestorNames.size() > 0)
```

```
            {
                final String missingAncestors = missingAncestorNames.toString();
                throw new ModelAugmentationException(
                    "Unable to find ancestors: %s",
                    missingAncestors.substring(1, missingAncestors.length() - 1));
            }
        }
    }

    @Override
    public void enterAssociationEndDeclaration(CMLParser.AssociationEndDeclarationContext ctx)
    {
        if (ctx.conceptName == null)
        {
            throw new ModelSynthesisException(NO_CONCEPT_NAME_PROVIDED_FOR_ASSOCIATION_END);
        }

        if (ctx.propertyName == null)
        {
            throw new ModelSynthesisException(NO_PROPERTY_NAME_PROVIDED_FOR_ASSOCIATION_END);
        }

        final Association association = ctx.association;
        final String conceptName = ctx.conceptName.getText();
        final String propertyName = ctx.propertyName.getText();
        final @Nullable Type propertyType = (ctx.typeDeclaration() == null) ? null : ctx.typeDeclaration().type;
        final Optional<TempConcept> concept = conceptOf(module, conceptName);
        final Optional<Property> property = seq(concept).flatMap(c -> c.getAllProperties().stream())
                                                        .filter(p -> p.getName().equals(propertyName))
                                                        .findFirst();

        createAssociationEnd(association, locationOf(ctx), conceptName, propertyName, propertyType, concept.orElse(null), property.orElse(null));
    }

    @Override
    public void enterTypeDeclaration(CMLParser.TypeDeclarationContext ctx)
    {
        final Type type = ctx.type;

        if (type != null && type instanceof TempNamedType && !type.isPrimitive())
        {
            final TempNamedType namedType = (TempNamedType)type;

            conceptOf(module, namedType.getName()).ifPresent(namedType::setConcept);
        }
    }

    private Location locationOf(ParserRuleContext ctx)
    {
        return createLocation(ctx, null);
    }

    private Location createLocation(ParserRuleContext ctx, ModelElement element)
    {
        final Token token = ctx.getStart();

        return Location.createLocation(token.getLine(), token.getCharPositionInLine() + 1, element);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/loader/ModelLoader.java

```
package cml.language.loader;

import cml.io.Console;
import cml.io.Directory;
import cml.io.ModuleManager;
import cml.io.SourceFile;
import cml.language.features.TempModule;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.TempModel;
import cml.language.generated.Element;
import cml.language.generated.Import;
import cml.language.generated.Location;
import cml.language.generated.ModelElement;
import cml.language.grammar.CMLLexer;
import cml.language.grammar.CMLParser;
import cml.language.grammar.CMLParser.CompilationUnitContext;
import org.antlr.v4.runtime.ANTLRInputStream;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.tree.ParseTreeWalker;
import org.jetbrains.annotations.Nullable;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import static cml.language.functions.ModelVisitorFunctions.visitModel;
import static cml.language.functions.ModuleFunctions.createImportOfModule;
import static cml.language.functions.ModuleFunctions.importedModuleOf;

public interface ModelLoader
{
    int loadModel(TempModel model, String moduleName);

    static ModelLoader create(Console console, ModuleManager moduleManager)
    {
        return new ModelLoaderImpl(console, moduleManager);
    }
}

class ModelLoaderImpl implements ModelLoader
{
    private static final String CML_BASE_MODULE = "cml_base";

    private static final int SUCCESS = 0;
    private static final int FAILURE_SOURCE_FILE_NOT_FOUND = 2;
    private static final int FAILURE_FAILED_LOADING_MODEL = 3;
    private static final int FAILURE_MODEL_VALIDATION = 4;

    private static final String NO_SOURCE_FILES_IN_MODULE = "no source files in module: %s";
    private static final String NO_SOURCE_DIR_IN_MODULE = "no source dir in module: %s";

    private final Console console;
    private final ModuleManager moduleManager;

    ModelLoaderImpl(final Console console, final ModuleManager moduleManager)
    {
        this.console = console;
        this.moduleManager = moduleManager;
    }

    @Override
    public int loadModel(TempModel model, String moduleName)
```

```
{
    try
    {
        final int exitCode = loadModule(model, moduleName, null);

        if (exitCode == SUCCESS)
        {
            linkFunctions(model);
            linkLambdaScope(model);

            return validateModel(model);
        }

        return exitCode;
    }
    catch (final Throwable exception)
    {
        if (exception.getMessage() == null)
        {
            console.error("Unable to parse source files.");
        }
        else
        {
            console.error(exception.getMessage());
        }

        if (!(exception instanceof ModelLoadingException))
        {
            exception.printStackTrace(System.err);
        }

        return FAILURE_FAILED_LOADING_MODEL;
    }
}

private int loadModule(TempModel model, String moduleName, @Nullable Import _import) throws IOException
{
    TempModule module;

    if (_import == null)
    {
        module = TempModule.create(model, moduleName);
    }
    else if (_import.isFirstImport())
    {
        module = (TempModule) _import.getImportedModule();
    }
    else
    {
        return SUCCESS;
    }

    final Optional<Directory> sourceDir = moduleManager.findSourceDir(moduleName);
    if (sourceDir.isPresent())
    {
        final List<SourceFile> sourceFiles = moduleManager.findSourceFiles(moduleName);
        if (sourceFiles.isEmpty())
        {
            console.error(NO_SOURCE_FILES_IN_MODULE, moduleName);
            return FAILURE_SOURCE_FILE_NOT_FOUND;
        }

        final List<CompilationUnitContext> compilationUnitContexts = new ArrayList<>();
        for (SourceFile sourceFile: sourceFiles)
        {
            final CompilationUnitContext compilationUnitContext = parse(sourceFile);
            synthesizeModule(module, compilationUnitContext);
            compilationUnitContexts.add(compilationUnitContext);
        }

        addBaseModule(module);

        for (Import i: module.getImports())
        {
            int exitCode = loadModule(model, i.getName(), i);

            if (exitCode != SUCCESS)
            {
                return exitCode;
            }
        }

        for (CompilationUnitContext compilationUnitContext: compilationUnitContexts)
        {
            augmentModule(module, compilationUnitContext);
        }
    }
    else
    {
        console.info(NO_SOURCE_DIR_IN_MODULE, moduleName);
    }

    return SUCCESS;
}

private void addBaseModule(TempModule module)
{
    if (!module.getName().equals(CML_BASE_MODULE) && !importedModuleOf(module, CML_BASE_MODULE).isPresent())
    {
        createImportOfModule(CML_BASE_MODULE, null, module);
    }
}

private void synthesizeModule(TempModule module, CompilationUnitContext compilationUnitContext)
{
    final ParseTreeWalker walker = new ParseTreeWalker();
    final ModelSynthesizer modelSynthesizer = new ModelSynthesizer(module);

    walker.walk(modelSynthesizer, compilationUnitContext);
}

private void augmentModule(TempModule module, CompilationUnitContext compilationUnitContext)
{
    final ParseTreeWalker walker = new ParseTreeWalker();
    final ModelAugmenter modelAugmenter = new ModelAugmenter(module);

    walker.walk(modelAugmenter, compilationUnitContext);
}

private void linkFunctions(final TempModel model)
{
    visitModel(model, new FunctionLinker());
}

private void linkLambdaScope(final TempModel model)
{
    visitModel(model, new LambdaScopeLinker());
```

```
    }

    private int validateModel(TempModel model)
    {
        final ModelValidator modelValidator = new ModelValidator();

        visitModel(model, modelValidator);

        if (modelValidator.getDiagnostics().size() == 0)
        {
            return SUCCESS;
        }
        else
        {
            for (Diagnostic diagnostic: modelValidator.getDiagnostics())
            {
                console.print(
                    "\nFailed validation: required %s: in %s",
                    diagnostic.getCode(),
                    diagnostic.getElement().getDiagnosticId());

                printLocation(diagnostic.getElement());

                if (diagnostic.getMessage().isPresent())
                {
                    console.println(diagnostic.getMessage().get());
                }

                for(Element element: diagnostic.getParticipants())
                {
                    console.print("- %s", element.getDiagnosticId());

                    printLocation(element);
                }
            }

            return FAILURE_MODEL_VALIDATION;
        }
    }

    private void printLocation(Element element)
    {
        if (element instanceof ModelElement)
        {
            final ModelElement modelElement = (ModelElement) element;

            if (modelElement.getLocation().isPresent())
            {
                final Location location = modelElement.getLocation().get();

                console.print(" (%d:%d)", location.getLine(), location.getColumn());
            }
        }

        console.println("");
    }

    private CompilationUnitContext parse(SourceFile sourceFile) throws IOException
    {
        try (final FileInputStream fileInputStream = new FileInputStream(sourceFile.getPath()))
        {
            final ANTLRInputStream input = new ANTLRInputStream(fileInputStream);
            final CMLLexer lexer = new CMLLexer(input);
            final CommonTokenStream tokens = new CommonTokenStream(lexer);
            final CMLParser parser = new CMLParser(tokens);
            final SyntaxErrorListener syntaxErrorListener = new SyntaxErrorListener(console);

            parser.getErrorListeners().clear();
            parser.addErrorListener(syntaxErrorListener);

            return parser.compilationUnit();
        }
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/loader/ModelLoadingException.java

```
package cml.language.loader;

import static java.lang.String.format;

class ModelLoadingException extends RuntimeException
{
    ModelLoadingException(String message, Object... args)
    {
        super(format(message, args));
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/loader/ModelSynthesisException.java

```
package cml.language.loader;

class ModelSynthesisException extends ModelLoadingException
{
    ModelSynthesisException(String message, Object... args)
    {
        super(message, args);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/loader/ModelSynthesizer.java

```
package cml.language.loader;

import cml.language.expressions.*;
import cml.language.features.*;
import cml.language.generated.*;
import cml.language.grammar.CMLBaseListener;
import cml.language.grammar.CMLParser;
import cml.language.grammar.CMLParser.*;
import cml.language.types.*;
import org.antlr.v4.runtime.ParserRuleContext;
import org.antlr.v4.runtime.Token;
import org.antlr.v4.runtime.tree.ParseTree;
import org.jooq.lambda.Seq;

import java.util.List;
import java.util.Optional;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.stream.Stream;

import static cml.language.functions.ModuleFunctions.createImportOfModule;
import static cml.language.generated.Arithmetic.createArithmetic;
```

```
import static cml.language.generated.Association.createAssociation;
import static cml.language.generated.Constructor.createConstructor;
import static cml.language.generated.Logical.createLogical;
import static cml.language.generated.Property.createProperty;
import static cml.language.generated.Referential.createReferential;
import static cml.language.generated.Relational.createRelational;
import static cml.language.generated.StringConcat.createStringConcat;
import static cml.language.generated.ValueType.createValueType;
import static cml.language.transforms.InvocationTransforms.invocationOf;
import static cml.primitives.Types.BOOLEAN;
import static cml.primitives.Types.BYTE;
import static cml.primitives.Types.DECIMAL;
import static cml.primitives.Types.DOUBLE;
import static cml.primitives.Types.FLOAT;
import static cml.primitives.Types.INTEGER;
import static cml.primitives.Types.LONG;
import static cml.primitives.Types.SHORT;
import static cml.primitives.Types.STRING;
import static cml.primitives.Types.*;
import static java.lang.String.format;
import static java.util.Arrays.asList;
import static java.util.Collections.emptyList;
import static java.util.Collections.singletonList;
import static java.util.Optional.ofNullable;
import static java.util.stream.Collectors.toList;
import static org.jooq.lambda.Seq.empty;
import static org.jooq.lambda.Seq.seq;

class ModelSynthesizer extends CMLBaseListener
{
    private static final String QUOTE = "\"";
    private static final String INVALID_MODULE_NAME = "Module declaration name (%s) should match the module's directory name: %s";
    private static final String NO_NAME_PROVIDED_FOR_CONCEPT = "No name provided for concept.";
    private static final String NO_NAME_PROVIDED_FOR_ASSOCIATION = "No name provided for association.";
    private static final String NO_NAME_PROVIDED_FOR_PROPERTY = "No name provided for property.";
    private static final String NO_NAME_PROVIDED_FOR_TARGET = "No name provided for task.";

    private final TempModule module;

    ModelSynthesizer(TempModule module)
    {
        this.module = module;
    }

    @Override
    public void exitModuleDeclaration(ModuleDeclarationContext ctx)
    {
        final String name = ctx.NAME().getText();

        if (!name.equals(module.getName()))
        {
            throw new ModelSynthesisException(format(INVALID_MODULE_NAME, name, module.getName()));
        }
    }

    @Override
    public void exitImportDeclaration(ImportDeclarationContext ctx)
    {
        final String moduleName = ctx.NAME().getText();
        final Location location = locationOf(ctx);

        ctx._import = createImportOfModule(moduleName, location, module);
    }

    @Override
    public void exitConceptDeclaration(ConceptDeclarationContext ctx)
    {
        if (ctx.NAME() == null)
        {
            throw new ModelSynthesisException(NO_NAME_PROVIDED_FOR_CONCEPT);
        }

        final String name = ctx.NAME().getText();
        final boolean _abstract = ctx.ABSTRACTION() != null;
        final List<Property> propertyList = seq(ctx.propertyList() == null ? empty() : ctx.propertyList().propertyDeclaration())
            .map(node -> node.property)
            .toList();

        ctx.concept = TempConcept.create(module, name, _abstract, propertyList, locationOf(ctx));
    }

    @Override
    public void exitAssociationDeclaration(AssociationDeclarationContext ctx)
    {
        if (ctx.NAME() == null)
        {
            throw new ModelSynthesisException(NO_NAME_PROVIDED_FOR_ASSOCIATION);
        }

        final String name = ctx.NAME().getText();

        ctx.association = createAssociation(name, module, locationOf(ctx), emptyList());

        seq(ctx.associationEndDeclaration() == null ? empty() : ctx.associationEndDeclaration())
            .forEach(node -> node.association = ctx.association);
    }

    @Override
    public void exitTaskDeclaration(TaskDeclarationContext ctx)
    {
        if (ctx.NAME() == null)
        {
            throw new ModelSynthesisException(NO_NAME_PROVIDED_FOR_TARGET);
        }

        final String name = ctx.NAME().getText();
        final Constructor constructor = ctx.constructorDeclaration() == null ? null : createConstructor(ctx.constructorDeclaration().NAME().getText
        final List<ModelElement> propertyList = seq(ctx.propertyList() == null ? empty() : ctx.propertyList().propertyDeclaration())
            .map(node -> (ModelElement) node.property)
            .toList();

        ctx.task = Task.createTask(name, module, locationOf(ctx), propertyList, constructor);
    }

    @Override
    public void exitTemplateDeclaration(final TemplateDeclarationContext ctx)
    {
        if (!ctx.functionDeclaration().function.getParent().isPresent())
        {
            ctx.template = new Template(module, ctx.functionDeclaration().function);
        }
    }

    @Override
    public void exitPropertyDeclaration(PropertyDeclarationContext ctx)
    {
        if (ctx.NAME() == null)
```

```
    {
        throw new ModelSynthesisException(NO_NAME_PROVIDED_FOR_PROPERTY);
    }

    final String name = ctx.NAME().getText();
    final Type type = (ctx.typeDeclaration() == null) ? null : ctx.typeDeclaration().type;
    final Expression value = (ctx.expression() == null) ? null : ctx.expression().expr;

    ctx.property = createProperty(name, null, locationOf(ctx), singletonList(value), ctx.DERIVED() != null, type, value, null);
}

@Override
public void exitTypeDeclaration(TypeDeclarationContext ctx)
{
    if (ctx.NAME() == null)
    {
        if (ctx.params != null) ctx.type = new FunctionType(ctx.params.type, ctx.result.type);
        else if (ctx.tuple != null) ctx.type = ctx.tuple.type;
        else if (ctx.inner != null) ctx.type = ctx.inner.type;
    }
    else
    {
        final String name = ctx.NAME().getText();
        final String cardinality = (ctx.cardinality() == null) ? "" : ctx.cardinality().getText();

        if (primitive(name))
        {
            ctx.type = createValueType(cardinality, primitiveTypeName(name));
        }
        else
        {
            ctx.type = TempNamedType.create(name, cardinality);
        }
    }
}

@Override
public void exitTupleTypeDeclaration(final TupleTypeDeclarationContext ctx)
{
    final Seq<TupleTypeElement> elements = seq(ctx.tupleTypeElementDeclaration()).map(c -> c.element);
    final String cardinality = (ctx.cardinality() == null) ? "" : ctx.cardinality().getText();

    ctx.type = new TupleType(elements, cardinality);
}

@Override
public void exitTupleTypeElementDeclaration(final TupleTypeElementDeclarationContext ctx)
{
    final Type type = ctx.type.type;
    final Optional<String> name = ofNullable(ctx.name).map(Token::getText);

    ctx.element = new TupleTypeElement(type, name.orElse(null));
}

@Override
public void exitFunctionDeclaration(final FunctionDeclarationContext ctx)
{
    final String name = ctx.name.getText();
    final Type type = ctx.resultType == null ? null : ctx.resultType.type;
    final Stream<FunctionParameter> params = ctx.functionParameterList().params;

    if (ctx.parent instanceof TemplateDeclarationContext)
    {
        ctx.function = new Function(name, type, params);
    }
    else
    {
        final Seq<TypeParameter> typeParams = ctx.typeParameterList() == null ? empty() : ctx.typeParameterList().params;
        final Expression expr = ctx.expression() == null ? null : ctx.expression().expr;

        ctx.function = new Function(module, name, type, params, typeParams, expr);
    }
}

@Override
public void exitFunctionParameterList(final FunctionParameterListContext ctx)
{
    ctx.params = seq(ctx.functionParameterDeclaration()).map(c -> c.param);
}

@Override
public void exitFunctionParameterDeclaration(final FunctionParameterDeclarationContext ctx)
{
    final String name = ctx.name.getText();

    ctx.param = new FunctionParameter(name, ctx.type.type);
}

@Override
public void exitTypeParameterList(final TypeParameterListContext ctx)
{
    ctx.params = seq(ctx.typeParameter()).map(c -> c.param);
}

@Override
public void exitTypeParameter(final TypeParameterContext ctx)
{
    final String name = ctx.name.getText();

    ctx.param = new TypeParameter(name);
}

@Override
public void exitExpression(ExpressionContext ctx)
{
    if (ctx.literalExpression() != null) ctx.expr = ctx.literalExpression().literal;
    else if (ctx.pathExpression() != null) ctx.expr = ctx.pathExpression().path;
    else if (ctx.lambdaExpression() != null) ctx.expr = ctx.lambdaExpression().lambda;
    else if (ctx.invocationExpression() != null) ctx.expr = ctx.invocationExpression().invocation;
    else if (ctx.comprehensionExpression() != null) ctx.expr = invocationOf(ctx.comprehensionExpression().comprehension);
    else if (ctx.operator != null && ctx.type != null) ctx.expr = createTypeExpression(ctx);
    else if (ctx.operator != null && ctx.expression().size() == 1) ctx.expr = createUnary(ctx);
    else if (ctx.operator != null && ctx.expression().size() == 2) ctx.expr = createInfix(ctx);
    else if (ctx.cond != null) ctx.expr = conditionalExpressionOf(ctx);
    else if (ctx.varExpr != null) ctx.expr = letExpressionOf(ctx);
    else if (ctx.inner != null) ctx.expr = ctx.inner.expr;

    createLocation(ctx, ctx.expr);
}

private Expression createTypeExpression(final ExpressionContext ctx)
{
    assert ctx.expression().size() == 1;

    final Expression expr = ctx.expression().get(0).expr;
    final String operator = ctx.operator.getText().replace('?', 'q').replace('!', 'b');
```

```
    if (operator.equals(TypeCast.ASB) || operator.equals(TypeCast.ASQ))
    {
        final Type castType = ctx.type.type;
        return new TypeCast(expr, operator, castType);
    }
    else
    {
        final Type checkedType = ctx.type.type;
        return new TypeCheck(expr, operator, checkedType);
    }
}

private Unary createUnary(ExpressionContext ctx)
{
    final String operator = ctx.operator.getText();
    final Expression expr = ctx.expression().get(0).expr;

    return new Unary(operator, expr);
}

private Infix createInfix(ExpressionContext ctx)
{
    final String operator = ctx.operator.getText();
    final Expression left = ctx.expression().get(0).expr;
    final Expression right = ctx.expression().get(1).expr;

    if (arithmeticOperator(operator))
    {
        return createArithmetic(operator, asList(left, right), null, locationOf(ctx));
    }
    else if (logicalOperator(operator))
    {
        return createLogical(operator, asList(left, right), null, locationOf(ctx));
    }
    else if (relationalOperator(operator))
    {
        return createRelational(operator, asList(left, right), null, locationOf(ctx));
    }
    else if (referentialOperator(operator))
    {
        return createReferential(operator, asList(left, right), null, locationOf(ctx));
    }
    else if (stringOperator(operator))
    {
        return createStringConcat(operator, asList(left, right), null, locationOf(ctx));
    }
    else
    {
        throw new IllegalStateException("Unexpected operator accepted by parser: " + operator + locationOf(ctx).toString());
    }
}

private TempConditional conditionalExpressionOf(final ExpressionContext ctx)
{
    final Expression cond = ctx.cond.expr;
    final Expression then = ctx.then.expr;
    final Expression else_ = ctx.else_.expr;

    return new TempConditional(cond, then, else_);
}

private Expression letExpressionOf(final ExpressionContext ctx)
{
    final Expression variableExpr = ctx.varExpr.expr;
    final Expression resultExpr = ctx.resultExpr.expr;
    final String variable = ctx.var.getText();

    return Let.createLet(asList(variableExpr, resultExpr), null, locationOf(ctx), variable);
}

@Override
public void exitPathExpression(PathExpressionContext ctx)
{
    final List<String> pathNames = ctx.NAME()
                                      .stream()
                                      .map(ParseTree::getText)
                                      .collect(toList());

    ctx.path = Path.create(pathNames);
}

@Override
public void exitLiteralExpression(LiteralExpressionContext ctx)
{
    final String text = getText(ctx);

    if (text != null)
    {
        final ValueType type = createValueType("", primitiveTypeNameOf(ctx));
        ctx.literal = Literal.createLiteral(emptyList(), null, locationOf(ctx), text, type);
    }
}

@Override
public void exitLambdaExpression(final LambdaExpressionContext ctx)
{
    final Seq<String> parameters = ctx.lambdaParameterList() == null ? empty() : ctx.lambdaParameterList().params;
    final Expression expr = ctx.expression().expr;

    ctx.lambda = new Lambda(parameters, expr);
}

@Override
public void exitLambdaParameterList(final LambdaParameterListContext ctx)
{
    ctx.params = seq(ctx.NAME()).map(ParseTree::getText);
}

@Override
public void exitInvocationExpression(InvocationExpressionContext ctx)
{
    final String name = ctx.NAME().getText();
    final Optional<Lambda> lambda = ofNullable(ctx.lambdaExpression()).map(c -> c.lambda);
    final List<Expression> arguments = expressionsOf(ctx).concat(seq(lambda)).toList();

    ctx.invocation = Invocation.create(name, arguments);
}

private Seq<Expression> expressionsOf(final InvocationExpressionContext ctx)
{
    final AtomicInteger index = new AtomicInteger(2);
    return seq(ctx.expression()).map(e -> {
        final int i = index.incrementAndGet();
        return e.expr == null ? new InvalidExpression(ctx.getChild(i).getText()) : e.expr;
    });
}
```

```
    @Override
    public void exitComprehensionExpression(ComprehensionExpressionContext ctx)
    {
        final Path path = ctx.pathExpression() == null ? null : ctx.pathExpression().path;
        final Stream<Enumerator> enumerators = ctx.enumeratorDeclaration()
                                                  .stream()
                                                  .map(e -> e.enumerator);
        final Stream<Query> queries = ctx.queryStatement()
                                         .stream().map(q -> q.query);

        if (path == null)
        {
            ctx.comprehension = new Comprehension(seq(enumerators), seq(queries));
        }
        else
        {
            ctx.comprehension = new Comprehension(path, seq(queries));
        }
    }

    @Override
    public void exitEnumeratorDeclaration(EnumeratorDeclarationContext ctx)
    {
        final String variable = ctx.var.getText();
        final Path path = ctx.pathExpression().path;

        ctx.enumerator = new Enumerator(variable, path);
    }

    @Override
    public void exitQueryStatement(final QueryStatementContext ctx)
    {
        if (ctx.NAME() == null)
        {
            final Stream<Keyword> keywords = ctx.keywordExpression()
                                                .stream()
                                                .map(k -> k.keyword);

            ctx.query = new Query(keywords);
        }
        else
        {
            final String name = ctx.NAME().getText();

            ctx.query = new Query(name);
        }
    }

    @Override
    public void exitKeywordExpression(final KeywordExpressionContext ctx)
    {
        final String name = ctx.NAME().getText();
        final Seq<String> parameters = ctx.lambdaParameterList() == null ? empty() : ctx.lambdaParameterList().params;
        final Expression expression = ctx.expression().expr;

        ctx.keyword = new Keyword(name, parameters, expression);
    }

    private static String getText(LiteralExpressionContext ctx)
    {
        if (ctx.BOOLEAN() != null) return ctx.BOOLEAN().getText();
        else if (ctx.STRING() != null) return unwrap(ctx.STRING().getText());
        else if (ctx.INTEGER() != null) return ctx.INTEGER().getText();
        else if (ctx.DECIMAL() != null) return ctx.DECIMAL().getText();
        else if (ctx.LONG() != null) return extractSuffix(ctx.LONG().getText());
        else if (ctx.SHORT() != null) return extractSuffix(ctx.SHORT().getText());
        else if (ctx.BYTE() != null) return extractSuffix(ctx.BYTE().getText());
        else if (ctx.FLOAT() != null) return extractSuffix(ctx.FLOAT().getText());
        else if (ctx.DOUBLE() != null) return extractSuffix(ctx.DOUBLE().getText());
        else return null;
    }

    private static String extractSuffix(String text)
    {
        return text.substring(0, text.length() - 1);
    }

    private static String primitiveTypeNameOf(LiteralExpressionContext ctx)
    {
        if (ctx.BOOLEAN() != null) return BOOLEAN;
        else if (ctx.STRING() != null) return STRING;
        else if (ctx.INTEGER() != null) return INTEGER;
        else if (ctx.LONG() != null) return LONG;
        else if (ctx.SHORT() != null) return SHORT;
        else if (ctx.BYTE() != null) return BYTE;
        else if (ctx.DECIMAL() != null) return DECIMAL;
        else if (ctx.FLOAT() != null) return FLOAT;
        else if (ctx.DOUBLE() != null) return DOUBLE;
        else return null;
    }

    private static String unwrap(String text)
    {
        if (text.startsWith(QUOTE))
        {
            text = text.substring(1);
        }

        if (text.endsWith(QUOTE))
        {
            text = text.substring(0, text.length() - 1);
        }

        return text;
    }

    private Location locationOf(ParserRuleContext ctx)
    {
        return createLocation(ctx, null);
    }

    private Location createLocation(ParserRuleContext ctx, ModelElement element)
    {
        final Token token = ctx.getStart();

        return Location.createLocation(token.getLine(), token.getCharPositionInLine() + 1, element);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/loader/ModelValidator.java

```
package cml.language.loader;

import cml.language.features.TempConcept;
import cml.language.foundation.Diagnostic;
import cml.language.foundation.InvariantValidator;
```

```
import cml.language.generated.Association;
import cml.language.generated.AssociationEnd;
import cml.language.generated.Expression;
import cml.language.generated.Property;
import cml.language.invariants.*;

import java.util.ArrayList;
import java.util.List;

import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;

public class ModelValidator implements ModelVisitor
{
    private final InvariantValidator<TempConcept> conceptInvariantValidator = () -> asList(
        new NotOwnGeneralization(),
        new CompatibleGeneralizations(),
        new ConflictRedefinition(),
        new AbstractPropertyRedefinition()
    );

    private final InvariantValidator<Property> propertyInvariantValidator = () -> asList(
        new UniquePropertyName(),
        new PropertyTypeSpecifiedOrInferred(),
        new PropertyTypeAssignableFromExpressionType(),
        new GeneralizationCompatibleRedefinition(),
        new AbstractPropertyInAbstractConcept()
    );

    private final InvariantValidator<Association> associationInvariantValidator = () -> asList(
        new AssociationMustHaveTwoAssociationEnds(),
        new AssociationEndTypesMustMatch()
    );

    private final InvariantValidator<AssociationEnd> associationEndInvariantValidator = () -> asList(
        new AssociationEndPropertyFoundInModel(),
        new AssociationEndTypeMatchesPropertyType()
    );

    private final InvariantValidator<Expression> expressionInvariantValidator = () -> singletonList(new ExpressionInvariant());

    private final List<Diagnostic> diagnostics = new ArrayList<>();

    List<Diagnostic> getDiagnostics()
    {
        return diagnostics;
    }

    @Override
    public void visit(TempConcept concept)
    {
        conceptInvariantValidator.validate(concept, diagnostics);
    }

    @Override
    public void visit(Property property)
    {
        propertyInvariantValidator.validate(property, diagnostics);
    }

    @Override
    public void visit(Association association)
    {
        associationInvariantValidator.validate(association, diagnostics);
    }

    @Override
    public void visit(AssociationEnd associationEnd)
    {
        associationEndInvariantValidator.validate(associationEnd, diagnostics);
    }

    @Override
    public void visit(final Expression expression)
    {
        expressionInvariantValidator.validate(expression, diagnostics);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/loader/ModelVisitor.java

```
package cml.language.loader;

import cml.language.features.Function;
import cml.language.features.TempConcept;
import cml.language.features.TempModule;
import cml.language.foundation.TempModel;
import cml.language.generated.Association;
import cml.language.generated.AssociationEnd;
import cml.language.generated.Expression;
import cml.language.generated.Property;

@SuppressWarnings("unused")
public interface ModelVisitor
{
    default void visit(TempModel model) {}
    default void visit(TempModule module) {}
    default void visit(Function function) {}
    default void visit(TempConcept concept) {}
    default void visit(Property property) {}
    default void visit(Association association) {}
    default void visit(AssociationEnd associationEnd) {}
    default void visit(Expression expression) {}
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/loader/SyntaxErrorListener.java

```
package cml.language.loader;

import cml.io.Console;
import org.antlr.v4.runtime.BaseErrorListener;
import org.antlr.v4.runtime.RecognitionException;
import org.antlr.v4.runtime.Recognizer;

public class SyntaxErrorListener extends BaseErrorListener
{
    private final Console console;

    SyntaxErrorListener(final Console console)
    {
        this.console = console;
    }

    @Override
    public void syntaxError(Recognizer<?, ?> recognizer, Object offendingSymbol, int line, int charPositionInLine, String defaultErrorMessage, Rec
```

```
    {
        console.println("Syntax Error: %s (%s:%s)", defaultErrorMessage, line, charPositionInLine + 1);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/transforms/InvocationTransforms.java

```
package cml.language.transforms;

import cml.language.expressions.Comprehension;
import cml.language.expressions.Invocation;
import cml.language.expressions.Lambda;
import cml.language.expressions.Query;
import cml.language.generated.Expression;

import java.util.LinkedHashMap;
import java.util.List;
import java.util.Optional;

import static org.jooq.lambda.Seq.seq;

public class InvocationTransforms
{
    public static Invocation invocationOf(final Comprehension comprehension)
    {
        final List<Query> reversedQueries = seq(comprehension.getQueries()).reverse().toList();

        return invocationOf(comprehension, reversedQueries);
    }

    private static Invocation invocationOf(final Comprehension comprehension, final List<Query> queries)
    {
        final Optional<Query> first = seq(queries).findFirst();
        assert first.isPresent();

        final Query query = first.get();
        final String name = query.getInvocationName();
        final LinkedHashMap<String, Expression> arguments = new LinkedHashMap<>();

        final List<Query> rest = seq(queries).skip(1).toList();

        if (rest.isEmpty())
        {
            if (comprehension.getPath().isPresent())
            {
                arguments.put("seq", comprehension.getPath().get());
            }
            else if (comprehension.getEnumerators().size() == 1)
            {
                seq(comprehension.getEnumerators()).findFirst()
                                                   .ifPresent(e -> arguments.put("seq", e.getPath()));
            }
            else
            {
                arguments.put("seq", Invocation.create("cross_join", comprehension.getExpressions()));
            }
        }
        else
        {
            arguments.put("seq", invocationOf(comprehension, rest));
        }

        query.getExpression().ifPresent(expr -> arguments.put(
            "expr", new Lambda(comprehension.getEnumeratorVariablesForQuery(query), expr)));

        query.getExtraKeywords().forEach(k -> arguments.put(k.getName(), k.getLambdaExpression()));

        return Invocation.create(name, arguments);
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/types/BaseType.java

```
package cml.language.types;

import cml.language.generated.Concept;
import cml.language.generated.Type;

import java.util.Optional;

import static cml.language.generated.Element.extendElement;

public abstract class BaseType implements Type
{
    private final Type type;

    public BaseType()
    {
        this("");
    }

    public BaseType(String cardinality)
    {
        this.type = Type.extendType(this, extendElement(this), cardinality);
    }

    @Override
    public String getCardinality()
    {
        return type.getCardinality();
    }

    @Override
    public String getName()
    {
        return type.getName();
    }

    @Override
    public int getMinCardinality()
    {
        return type.getMinCardinality();
    }

    @Override
    public int getMaxCardinality()
    {
        return type.getMaxCardinality();
    }

    @Override
    public Optional<Concept> getConcept()
    {
        return Optional.empty();
    }
```

```
@Override
public String getKind()
{
    return type.getKind();
}

@Override
public Type getMatchingResultType()
{
    return type.getMatchingResultType();
}

@Override
public Type getBaseType()
{
    return type.getBaseType();
}

@Override
public boolean isParameter()
{
    return type.isParameter();
}

@Override
public boolean isDefined()
{
    return type.isDefined();
}

@Override
public boolean isUndefined()
{
    return type.isUndefined();
}

@Override
public boolean isSomething()
{
    return type.isSomething();
}

@Override
public boolean isNothing()
{
    return type.isNothing();
}

@Override
public boolean isRelational()
{
    return type.isRelational();
}

@Override
public boolean isReferential()
{
    return type.isReferential();
}

@Override
public boolean isPrimitive()
{
    return type.isPrimitive();
}

@Override
public boolean isNumeric()
{
    return type.isNumeric();
}

@Override
public boolean isFloat()
{
    return type.isFloat();
}

@Override
public boolean isBoolean()
{
    return type.isBoolean();
}

@Override
public Type getElementType()
{
    return type.getElementType();
}

@Override
public boolean isString()
{
    return type.isString();
}

@Override
public boolean isSingle()
{
    return type.isSingle();
}

@Override
public boolean isRequired()
{
    return type.isRequired();
}

@Override
public boolean isOptional()
{
    return type.isOptional();
}

@Override
public boolean isSequence()
{
    return type.isSequence();
}

@Override
public String getInferredCardinality()
{
    return type.getInferredCardinality();
}

@Override
```

```
    public Type getInferredType()
    {
        return type.getInferredType();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/types/FunctionType.java

```
package cml.language.types;

import cml.language.generated.Type;
import org.jooq.lambda.Seq;

import java.util.Optional;

import static java.lang.String.format;

public class FunctionType extends BaseType
{
    private final TupleType params;
    private final Type result;

    public FunctionType(final TupleType params, final Type result)
    {
        this.params = params;
        this.result = result;
    }

    public TupleType getParams()
    {
        return params;
    }

    public Type getResult()
    {
        return result;
    }

    public Seq<Type> getParamTypes()
    {
        return params.getElements().map(TupleTypeElement::getType);
    }

    public boolean isSingleParam()
    {
        return getParams().getElements().count() == 1;
    }

    public Type getSingleParamType()
    {
        assert isSingleParam();

        final Optional<TupleTypeElement> single = getParams().getElements().findSingle();

        assert single.isPresent();

        return single.get().getType();
    }

    public Type getMatchingResultType()
    {
        return getResult();
    }

    @Override
    public boolean isString()
    {
        return false;
    }

    @Override
    public Type getElementType()
    {
        assert isRequired();

        return this;
    }

    @Override
    public String getKind()
    {
        return "function";
    }

    @Override
    public String getDiagnosticId()
    {
        return format("%s -> %s", params.getDiagnosticId(), result.getDiagnosticId());
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/types/MemberType.java

```
package cml.language.types;

import cml.language.generated.Type;

import static java.lang.String.format;

public class MemberType extends BaseType
{
    private final Type baseType;
    private final String name;
    private final long paramIndex;

    public MemberType(final Type baseType, final String name, final long paramIndex)
    {
        this.baseType = baseType;
        this.name = name;
        this.paramIndex = paramIndex;
    }

    public Type getBaseType()
    {
        return baseType;
    }

    public String getName()
    {
        return name;
    }

    public long getParamIndex()
    {
        return paramIndex;
```

```
    }

    @Override
    public Type getElementType()
    {
        return this;
    }

    @Override
    public String getDiagnosticId()
    {
        return format("%s.%s", baseType, getName());
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/types/TempNamedType.java

package cml.language.types;

import cml.language.features.TempConcept;
import cml.language.generated.Concept;
import cml.language.generated.Element;
import cml.language.generated.Type;
import cml.language.generated.ValueType;
import cml.primitives.Types;
import org.jetbrains.annotations.NotNull;

import java.util.Optional;

import static cml.language.functions.TypeFunctions.withCardinality;
import static cml.language.generated.Element.extendElement;
import static cml.language.generated.Type.extendType;

public interface TempNamedType extends Type
{
    TempNamedType NOTHING = TempNamedType.create("NOTHING", "?");
    ValueType BOOLEAN = ValueType.createValueType("", Types.BOOLEAN);
    ValueType STRING = ValueType.createValueType("", Types.STRING);

    String getName();

    @Override
    default Type getInferredType()
    {
        return withCardinality(this, getInferredCardinality());
    }

    default boolean isNothing()
    {
        return getName().toUpperCase().equals(NOTHING.getName());
    }

    @Override
    default TempNamedType getElementType()
    {
        final TempNamedType elementType = TempNamedType.create(getName());

        getConcept().map(c -> (TempConcept)c).ifPresent(elementType::setConcept);

        return elementType;
    }

    void setConcept(@NotNull Concept concept);

    static TempNamedType create(String name)
    {
        return new NamedTypeImpl(name, "");
    }

    static TempNamedType create(String name, String cardinality)
    {
        return new NamedTypeImpl(name, cardinality);
    }
}

class NamedTypeImpl implements TempNamedType
{
    private final Type type;

    private final String name;

    private Concept concept;

    NamedTypeImpl(String name, String cardinality)
    {
        final Element element = extendElement(this);

        this.type = extendType(this, element, cardinality);

        this.name = name;
    }

    @Override
    public String getKind()
    {
        return type.getKind();
    }

    @Override
    public Type getMatchingResultType()
    {
        return type.getMatchingResultType();
    }

    @Override
    public Type getBaseType()
    {
        return type.getBaseType();
    }

    @Override
    public boolean isParameter()
    {
        return type.isParameter();
    }

    @Override
    public boolean isDefined()
    {
        return type.isDefined();
    }

    @Override
    public boolean isUndefined()
    {
        return type.isUndefined();
```

```
    }

    @Override
    public boolean isSomething()
    {
        return type.isSomething();
    }

    @Override
    public boolean isBoolean()
    {
        return type.isBoolean();
    }

    @Override
    public boolean isNumeric()
    {
        return type.isNumeric();
    }

    @Override
    public boolean isFloat()
    {
        return type.isFloat();
    }

    @Override
    public boolean isString()
    {
        return type.isString();
    }

    @Override
    public boolean isPrimitive()
    {
        return type.isPrimitive();
    }

    @Override
    public boolean isRelational()
    {
        return isNumeric() || isFloat() || isString();
    }

    @Override
    public boolean isReferential()
    {
        return type.isReferential();
    }

    @Override
    public boolean isSingle()
    {
        return type.isSingle();
    }

    @Override
    public boolean isRequired()
    {
        return type.isRequired();
    }

    @Override
    public boolean isOptional()
    {
        return type.isOptional();
    }

    @Override
    public boolean isSequence()
    {
        return type.isSequence();
    }

    @Override
    public String getInferredCardinality()
    {
        return type.getInferredCardinality();
    }

    @Override
    public String getName()
    {
        return name;
    }

    @Override
    public String getCardinality()
    {
        return type.getCardinality();
    }

    @Override
    public int getMinCardinality()
    {
        return type.getMinCardinality();
    }

    @Override
    public int getMaxCardinality()
    {
        return type.getMaxCardinality();
    }

    @Override
    public Optional<Concept> getConcept()
    {
        return Optional.ofNullable(concept);
    }

    public void setConcept(@NotNull Concept concept)
    {
        assert this.concept == null;

        this.concept = concept;
    }

    @Override
    public String getDiagnosticId()
    {
        return isUndefined() ? getName() : getName() + getCardinality();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/types/TupleType.java

```
package cml.language.types;

import cml.language.generated.Type;
import org.jooq.lambda.Seq;

import java.util.List;

import static cml.language.functions.TypeFunctions.withCardinality;
import static java.lang.String.format;
import static org.jooq.lambda.Seq.seq;

public class TupleType extends BaseType
{
    private final List<TupleTypeElement> elements;

    public TupleType(final Seq<TupleTypeElement> elements)
    {
        this(elements, "");
    }

    public TupleType(final Seq<TupleTypeElement> elements, String cardinality)
    {
        super(cardinality);

        this.elements = elements.toList();
    }

    public Seq<TupleTypeElement> getElements()
    {
        return seq(elements);
    }

    public Seq<Type> getElementTypes()
    {
        return seq(elements).map(TupleTypeElement::getType);
    }

    @Override
    public Type getElementType()
    {
        return withCardinality(this, "");
    }

    @Override
    public String getDiagnosticId()
    {
        return format("(%s)", seq(elements).map(e -> e.getDiagnosticId()).toString(",")) + getCardinality();
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/types/TupleTypeElement.java

```
package cml.language.types;

import cml.language.foundation.ModelElementBase;
import cml.language.functions.TypeFunctions;
import cml.language.generated.Type;
import org.jetbrains.annotations.Nullable;

import java.util.Optional;

import static java.lang.String.format;
import static java.util.Optional.ofNullable;

public class TupleTypeElement extends ModelElementBase
{
    private final Type type;
    private final @Nullable String name;

    public TupleTypeElement(final Type type, final @Nullable String name)
    {
        this.type = type;
        this.name = name;
    }

    public Type getType()
    {
        return type;
    }

    public Optional<String> getName()
    {
        return ofNullable(name);
    }

    public boolean isAssignableFrom(final TupleTypeElement that)
    {
        return TypeFunctions.isAssignableFrom(this.type, that.type) &&
                (name != null ? name.equals(that.name) : that.name == null);
    }

    @Override
    public String getDiagnosticId()
    {
        return name == null ? type.getDiagnosticId() : format("%s: %s", name, type.getDiagnosticId());
    }
}
```

==> cml-compiler/cml-language/src/main/java/cml/language/types/TypedElementBase.java

```
package cml.language.types;

import cml.language.foundation.NamedElementBase;
import cml.language.generated.Scope;
import cml.language.generated.Type;
import cml.language.generated.TypedElement;
import org.jetbrains.annotations.Nullable;

public abstract class TypedElementBase extends NamedElementBase implements TypedElement
{
    private final Type type;

    public TypedElementBase(final String name, final Type type)
    {
        this(null, name, type);
    }

    public TypedElementBase(@Nullable final Scope parent, final String name, final Type type)
    {
        super(parent, name);

        this.type = type;
    }
```

```
    @Override
    public Type getType()
    {
        return type;
    }
}


==> cml-compiler/cml-language/src/main/java/cml/language/types/TypeParameter.java

package cml.language.types;

public class TypeParameter
{
    private final String name;

    public TypeParameter(final String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }
}


==> cml-compiler/cml-language/src/test/java/cml/language/ExpressionTest.java

package cml.language;

import cml.io.Console;
import cml.io.FileSystem;
import cml.io.ModuleManager;
import cml.language.features.TempConcept;
import cml.language.features.TempModule;
import cml.language.foundation.TempModel;
import cml.language.generated.Expression;
import cml.language.generated.Property;
import cml.language.generated.Type;
import cml.language.generated.UndefinedType;
import cml.language.loader.ModelLoader;
import cml.templates.ModelAdaptor;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroupFile;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import java.util.Optional;
import java.util.Properties;

import static cml.language.functions.ModelFunctions.moduleOf;
import static cml.language.functions.ModuleFunctions.conceptOf;
import static java.util.Arrays.stream;
import static java.util.stream.Collectors.toList;
import static org.jooq.lambda.Seq.seq;
import static org.junit.Assert.*;

@RunWith(Parameterized.class)
public class ExpressionTest
{
    private static final String BASE_PATH = "./src/test/resources/cml/language/ExpressionTest";
    private static final String ENCODING = "UTF-8";
    private static final char START_CHAR = '<';
    private static final char STOP_CHAR = '>';

    @Parameterized.Parameters(name = "{0}")
    public static List<Object[]> modulePaths()
    {
        final File file = new File(BASE_PATH);
        final File[] files = file.listFiles(File::isDirectory);

        return stream(files == null ? new File[0] : files)
                .map(f -> new Object[] { f.getName(), f })
                .collect(toList());
    }

    private final File moduleDir;
    private final FileSystem fileSystem;
    private final ModuleManager moduleManager;
    private final ModelLoader modelLoader;
    private final STGroupFile groupFile;

    public ExpressionTest(@SuppressWarnings("unused") String moduleName, File moduleDir)
    {
        final Console console = Console.createSystemConsole();

        this.moduleDir = moduleDir;
        this.fileSystem = FileSystem.create(console);
        this.moduleManager = ModuleManager.create(console, fileSystem);
        this.modelLoader = ModelLoader.create(console, moduleManager);
        this.groupFile = getOclTemplateGroup();
    }

    @Test
    public void expressionOCL() throws Exception
    {
        final TempConcept concept = loadExpressions();
        final Properties expectedOCL = loadProperties("expected_ocl.properties");

        for (final Property property: seq(concept.getProperties()))
        {
            assertExpectedOCL(expectedOCL, property);
        }
    }

    @Test
    public void expectedType() throws Exception
    {
        final TempConcept concept = loadExpressions();
        final Properties expectedType = loadProperties("expected_type.properties");

        for (final Property property: seq(concept.getProperties()))
        {
            assertExpectedType(expectedType, property);
        }
    }

    private TempConcept loadExpressions()
    {
        final String modulesBaseDir = fileSystem.extractParentPath(moduleDir.getPath());
```

```
        moduleManager.clearBaseDirs();
        moduleManager.addBaseDir(modulesBaseDir);

        final TempModel model = TempModel.create();
        modelLoader.loadModel(model, moduleDir.getName());

        final String moduleName = moduleDir.getName();
        final Optional<TempModule> module = moduleOf(model, moduleName);
        assertTrue("Module should be found: " + moduleName, module.isPresent());

        final Optional<TempConcept> concept = conceptOf(module.get(), "Expressions");
        assertTrue("The Expressions concept should be found in module: " + moduleName, concept.isPresent());

        return concept.get();
    }

    private Properties loadProperties(String fileName) throws IOException
    {
        final Properties properties = new Properties();
        final File propertiesFile = new File(moduleDir, fileName);

        try (final FileInputStream fileInputStream = new FileInputStream(propertiesFile))
        {
            properties.load(fileInputStream);
        }

        return properties;
    }

    private static STGroupFile getOclTemplateGroup()
    {
        final STGroupFile groupFile = new STGroupFile(
            BASE_PATH + File.separator + "ocl.stg",
            ENCODING, START_CHAR, STOP_CHAR);

        groupFile.registerModelAdaptor(Object.class, new ModelAdaptor());

        return groupFile;
    }

    private void assertExpectedOCL(Properties expectedOCL, Property property)
    {
        final String expectedOCLExpression = expectedOCL.getProperty(property.getName());

        if (expectedOCLExpression == null)
        {
            assertFalse(
                "Expected non-init property or missing property in expected_ocl.properties file: " + property.getName(),
                property.getValue().isPresent());
        }
        else
        {
            assertTrue("Expected value for property: " + property.getName(), property.getValue().isPresent());

            final ST oclTemplate = groupFile.getInstanceOf("ocl");
            oclTemplate.add("expr", property.getValue().get());

            assertEquals(
                "Property should match OCL: " + property.getName(),
                expectedOCLExpression,
                oclTemplate.render());
        }
    }

    private void assertExpectedType(Properties expectedTypes, Property property)
    {
        final String expectedType = expectedTypes.getProperty(property.getName());

        if (expectedType == null)
        {
            assertFalse(
                "Expected non-init property or missing property in expected_type.properties file: " + property.getName(),
                property.getValue().isPresent());
        }
        else
        {
            assertTrue("Expected type for property: " + property.getName(), property.getValue().isPresent());

            final Expression value = property.getValue().orElse(null);
            assertNotNull("Should have init for property: " + property.getName(), value);

            final Type type = value.getType();
            assertNotNull("Should have computed type for property: " + property.getName(), type);

            if (type instanceof UndefinedType)
            {
                final UndefinedType undefinedType = (UndefinedType) type;

                fail("Type Error of property '" + property.getName() + "': " + undefinedType.getErrorMessage());
            }

            assertEquals(
                "Property should match expected type: " + property.getName(),
                expectedType,
                type.getDiagnosticId());
        }
    }
}


==> cml-compiler/cml-language/src/test/java/cml/language/foundation/PairTest.java

package cml.language.foundation;

import cml.language.generated.Property;
import cml.language.types.TempNamedType;
import org.junit.Test;

import static cml.language.generated.Property.createProperty;
import static java.util.Collections.emptyList;
import static org.junit.Assert.assertEquals;

public class PairTest
{
    @Test
    public void equals_hashCode() throws Exception
    {
        final Property p1 = createProperty( "p1", null, null, emptyList(), false, TempNamedType.create("Integer"), null, null);
        final Property p2 = createProperty( "p2", null, null, emptyList(), false, TempNamedType.create("Decimal"), null, null);

        final Pair<Property> pair1 = new Pair<>(p1, p2);
        final Pair<Property> pair2 = new Pair<>(p2, p1);

        assertEquals(pair1, pair2);
        assertEquals(pair1.hashCode(), pair2.hashCode());
    }
```

```
}
```

==> cml-compiler/cml-language/src/test/java/cml/language/ModelLoaderTest.java

```
package cml.language;

import cml.io.Console;
import cml.io.FileSystem;
import cml.io.ModuleManager;
import cml.language.features.TempConcept;
import cml.language.features.TempModule;
import cml.language.foundation.TempModel;
import cml.language.generated.*;
import cml.language.loader.ModelLoader;
import cml.language.types.TempNamedType;
import org.jetbrains.annotations.Nullable;
import org.junit.Before;
import org.junit.Test;

import java.util.Optional;

import static cml.language.functions.ConceptFunctions.propertyOf;
import static cml.language.functions.ModelFunctions.associationOf;
import static cml.language.functions.TypeFunctions.isEqualTo;
import static junit.framework.TestCase.assertNotNull;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.CoreMatchers.sameInstance;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.junit.Assert.*;

public class ModelLoaderTest
{
    private static final String BASE_PATH = "src/test/resources/cml/language/ModelLoader/";

    private FileSystem fileSystem;
    private ModuleManager moduleManager;
    private ModelLoader modelLoader;

    @Before
    public void setUp()
    {
        final Console systemConsole = Console.createSystemConsole();

        fileSystem = FileSystem.create(systemConsole);
        moduleManager = ModuleManager.create(systemConsole, fileSystem);
        modelLoader = ModelLoader.create(systemConsole, moduleManager);
    }

    @Test
    public void module()
    {
        final String moduleName = "module_name";
        final TempModule module = loadModule(moduleName);
        final Concept concept = module.getConcepts().get(0);

        assertThat(module.getName(), is(moduleName));
        assertThat(concept.getName(), is("SomeConcept"));

        final String anotherModuleName = "another_module";
        final Import _import = module.getImports().get(0);
        final TempModule anotherModule = (TempModule) _import.getImportedModule();
        final Concept anotherConcept = anotherModule.getConcepts().get(0);

        assertThat(_import.getName(), is(anotherModuleName));
        assertThat(anotherModule.getName(), is(anotherModuleName));
        assertThat(anotherConcept.getName(),is("AnotherConcept"));
        assertThat(concept.getAncestors().get(0), is(sameInstance(anotherConcept)));
    }

    @Test
    public void invalid_module_name()
    {
        final String moduleName = "invalid_module_name";
        final String modulePath = BASE_PATH + moduleName;
        final String modulesBaseDir = fileSystem.extractParentPath(modulePath);

        moduleManager.clearBaseDirs();
        moduleManager.addBaseDir(modulesBaseDir);

        final TempModel model = TempModel.create();
        final int result = modelLoader.loadModel(model, moduleName);

        assertThat(result, is(3));
    }

    @Test
    public void concrete_concept()
    {
        final TempConcept concept = loadConcept("concrete_concept");

        assertThat(concept.getName(), is("ModelElement"));
        assertFalse("Concept should be concrete.", concept.isAbstraction());
    }

    @Test
    public void derived_property()
    {
        final TempConcept concept = loadConcept("derived_property");

        assertThat(concept.getName(), is("SomeConcept"));
        assertTrue("derivedProperty should be derived.", propertyOf(concept, "derivedProperty").get().isDerived());
        assertFalse("nonDerivedProperty should not be derived.", propertyOf(concept,"nonDerivedProperty").get().isDerived());
    }

    @Test
    public void abstract_concept()
    {
        final TempConcept concept = loadConcept("abstract_concept");

        assertThat(concept.getName(), is("ModelElement"));
        assertTrue("Concept should be abstract.", concept.isAbstraction());
    }

    @Test
    public void expressions()
    {
        final TempConcept concept = loadConcept("expressions");

        assertThat(concept.getName(), is("Expressions"));

        assertPropertyFound(concept, "str", "SomeString");
        assertPropertyFound(concept, "int", "123");
        assertPropertyFound(concept, "dec", "123.456");
    }

    @Test
    public void associations()
```

```
    {
        final Association employment = loadAssociation("Employment");
        assertAssociationEndFound(employment, "Employee", "employer");
        assertAssociationEndFound(employment, "Organization", "employees");

        final Association vehicleOwnership = loadAssociation("VehicleOwnership");
        assertAssociationEndFound(vehicleOwnership, "Vehicle", "owner", TempNamedType.create("Organization"));
        assertAssociationEndFound(vehicleOwnership, "Organization", "fleet", TempNamedType.create("Vehicle", "*"));
    }

    private TempModule loadModule(String sourceFileName)
    {
        return (TempModule) loadModel(sourceFileName).getModules().get(0);
    }

    private TempConcept loadConcept(String sourceFileName)
    {
        return (TempConcept) loadModel(sourceFileName).getConcepts().get(0);
    }

    private Association loadAssociation(String name)
    {
        final Optional<Association> association = associationOf(loadModel("associations"), name);

        assert association.isPresent();

        return association.get();
    }

    private TempModel loadModel(String moduleName)
    {
        final String modulePath = BASE_PATH + moduleName;
        final String modulesBaseDir = fileSystem.extractParentPath(modulePath);

        moduleManager.clearBaseDirs();
        moduleManager.addBaseDir(modulesBaseDir);

        final TempModel model = TempModel.create();

        modelLoader.loadModel(model, moduleName);

        return model;
    }

    private void assertPropertyFound(TempConcept concept, String propertyName, String propertyValue)
    {
        final Property str = propertyOf(concept, propertyName).orElse(null);
        assertNotNull(propertyName, str);

        final Literal literal = (Literal)str.getValue().orElse(null);
        assertNotNull(propertyName, literal);

        assertThat(propertyName, literal.getText(), is(propertyValue));
    }

    private void assertAssociationEndFound(Association association, String conceptName, String propertyName)
    {
        assertAssociationEndFound(association, conceptName, propertyName, null);
    }

    private void assertAssociationEndFound(
        Association association,
        String conceptName, String propertyName,
        @Nullable TempNamedType expectedType)
    {
        final AssociationEnd associationEnd = association.getAssociationEnds().stream()
                                                         .filter(associationEnd1 -> associationEnd1.getConceptName().equals(conceptName))
                                                         .filter(associationEnd2 -> associationEnd2.getPropertyName().equals(propertyName))
                                                         .findFirst().orElse(null);

        assertNotNull(conceptName + "." + propertyName, associationEnd);
        assertTrue(conceptName, associationEnd.getAssociatedConcept().isPresent());
        assertTrue(conceptName + "." + propertyName, associationEnd.getAssociatedProperty().isPresent());

        if (expectedType == null)
        {
            assertFalse("Did not expect type for: " + conceptName + "." + propertyName, associationEnd.getPropertyType().isPresent());
        }
        else
        {
            assertTrue("Did expect type for: " + conceptName + "." + propertyName, associationEnd.getPropertyType().isPresent());
            assertTrue("Expected matching type for: " + conceptName + "." + propertyName, isEqualTo(expectedType, associationEnd.getPropertyType().
        }

        final TempConcept concept = associationEnd.getAssociatedConcept().map(c -> (TempConcept) c).get();
        assertEquals(conceptName, concept.getName(), conceptName);

        final Property property = associationEnd.getAssociatedProperty().get();
        assertEquals(conceptName + "." + propertyName, property.getName(), propertyName);

        assertTrue(conceptName + "." + propertyName, concept.getMembers().contains(property));
    }
}
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/comprehension/expected_ocl.properties

```
# suppress inspection "UnusedProperty" for whole file

path_ancestors = class.ancestors

select_query = top_items->select((name = "this"))
select_query_nested = top_items->select((name = "this"))->select((qty > 10))
select_query_nested_nested = top_items->select((name = "this"))->select((qty > 10))->select((number <= 10))
select_query_embedded = top_orders->select(items->exists((qty > 10)))->select((total > 100))

select_invocation_nested = top_items->select((name = "this"))->select((qty > 10))
select_invocation_nested_nested = top_items->select((name = "this"))->select((qty > 10))->select((number <= 10))
select_invocation_intermediate = top_orders->select(items->exists((qty > 10)))->select((total > 100))

reject_query = top_items->reject((name = "this"))

yield_query = top_orders->collect(items)->flatten()
yield_query_unique = top_orders->collect(items.product)->flatten()->asSet()

recurse_ancestors = class->closure(ancestors)->flatten()
recurse_unique = class->closure(ancestors)->flatten()->asSet()
recurse_children = children->closure(children)->flatten()->select((ranking > 10))
recurse_self = self->closure(children)->flatten()->select((ranking > 10))

reduce_query =  top_orders->iterate(order; sum = 0 | (sum + order.total))
reduce_enumeration =  top_orders->iterate(order; sum = 0 | (sum + order.total))

enumeration = top_orders->select(order | (order.total > 100))
enumeration_yield = top_orders->collect(order | order.items)->flatten()
enumeration_select = top_orders->select(order | (order.total > 1000))->collect(order | order.items)->flatten()

enumeration_embedded = top_orders->collect(order | order)->flatten()->select((total > 100))
```

```
enumeration_embedded_select = top_orders->select(order | (order.total > 100))->collect(items)->flatten()

cross_join_enumeration = cross_join(top_orders, gold_customers)->select(order, gold_customer | (order.customer = gold_customer))->collect(order, go
cross_join_nested = cross_join(top_orders, gold_customers)->select(order, gold_customer | ((order.customer = gold_customer) and (order.total > 1000


==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/comprehension/expected_type.properties

# suppress inspection "UnusedProperty" for whole file

path_ancestors = Class*

select_query = Item*
select_query_nested = Item*
select_query_nested_nested = Item*
select_query_embedded = Order*

select_invocation_nested = Item*
select_invocation_nested_nested = Item*
select_invocation_intermediate = Order*

reject_query = Item*

yield_query = Item*
yield_query_unique = Product*

recurse_ancestors = Class*
recurse_unique = Class*
recurse_children = Class*
recurse_self = Class*

reduce_query = decimal
reduce_enumeration = decimal

enumeration = Order*
enumeration_yield = Item*
enumeration_select = Item*

enumeration_embedded = Order*
enumeration_embedded_select = Item*

cross_join_enumeration = Order*
cross_join_nested = string*


==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/comprehension/source/main.cml


@template @function select<T>(seq: T*, expr: (T) -> BOOLEAN) -> T*;

@template @function reject<T>(seq: T*, expr: (T) -> BOOLEAN) -> T*;

@template @function yield<T>(seq: T*, expr: (T) -> R*) -> R*;

@template @function distinct<T>(seq: T*) -> T*;

@template @function exists<T>(seq: T*, expr: (T) -> BOOLEAN) -> BOOLEAN;

@template @function recurse<T,S>(seq: T*, expr: (T) -> S*) -> S*;

@template @function reduce<T,S>(seq: T*, expr: (T, S) -> S, from: () -> S) -> S;

@template @function cross_join<T,S>(seq1: T*, seq2: S*) -> (T, S)*;

@concept Expressions
{
    top_items: Item*;
    class: Class;
    children: Class*;
    top_orders: Order*;
    gold_customers: Customer*;

    path_ancestors = class.ancestors;

    select_query = top_items | select: name == "this";
    select_query_nested = top_items | select: name == "this" | select: qty > 10;
    select_query_nested_nested = top_items | select: name == "this" | select: qty > 10 | select: number <= 10;

    select_query_embedded = top_orders | select: (items | exists: qty > 10) | select: total > 100;

    select_invocation_nested = select(top_items | select: name == "this",  { qty > 10 });
    select_invocation_nested_nested = select(select(top_items | select: name == "this",  { qty > 10 }), { number <= 10 });
    select_invocation_intermediate = top_orders | select: exists(items, { qty > 10 }) | select: total > 100;

    reject_query = top_items | reject: name == "this";

    yield_query = top_orders | yield: items;
    yield_query_unique = top_orders | yield: items.product | distinct;

    recurse_ancestors = class | recurse: ancestors;
    recurse_unique = class | recurse: ancestors | distinct;
    recurse_children = children | recurse: children | select: ranking > 10; // equivalent to: self | recurse children | ...
    recurse_self = self | recurse: children | select: ranking > 10;

    reduce_query = top_orders | reduce: order, sum -> sum + order.total from: 0;
    reduce_enumeration = for order in top_orders | reduce: sum -> sum + order.total from: 0;

    enumeration = for order in top_orders | select: order.total > 100;
    enumeration_yield = for order in top_orders | yield: order.items;
    enumeration_select = for order in top_orders | select: order.total > 1000 | yield: order.items;

    enumeration_embedded = select(for order in top_orders | yield: order, { total > 100 });
    enumeration_embedded_select = yield(for order in top_orders | select: order.total > 100, { items });


    cross_join_enumeration = for order in top_orders, gold_customer in gold_customers
                               | select: order.customer === gold_customer
                               | yield: order;
    cross_join_nested = yield(for order in top_orders, gold_customer in gold_customers
                                | select: order.customer === gold_customer and order.total > 1000,
                              { order, gold_customer -> order.customer.name });
}

@concept Order
{
    customer: Customer;
    items: Item*;
    total: DECIMAL;
}

@concept Item
{
    number: INTEGER;
    name: STRING;
    product: Product;
    qty: INTEGER;
}
```

```
@concept Customer
{
    name: STRING;
}

@concept Product;

@concept Class
{
    ancestors: Class*;
    children: Class*;
    ranking: INTEGER;
}
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/if_then_else/expected_ocl.properties

```
# suppress inspection "UnusedProperty" for whole file

if_then_else = (if a then b else c endif)

low_precedence = (1 + (if ((a < 0) and (b = 3)) then 2 else (3 * (- 4)) endif))
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/if_then_else/expected_type.properties

```
# suppress inspection "UnusedProperty" for whole file

if_then_else = integer|string

low_precedence = integer
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/if_then_else/source/main.cml

```
@concept Expressions
{
    b: INTEGER;
    c: STRING;

    if_then_else = if a then b else c;

    low_precedence = 1 + if a < 0 and b == 3 then 2 else 3 * -4;
}
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/infix/expected_ocl.properties

```
# suppress inspection "UnusedProperty" for whole file

addition = (1 + 2)
subtraction = (2 - 1)
multiplication = (2 * 3)
division = (6 / 2)
modulus = (5 % 2)

addition_left_assoc = ((1 + 2) + 3)
subtraction_left_assoc = ((3 - 2) - 1)
multiplication_left_assoc = ((2 * 3) * 1)
division_left_assoc = ((12 / 6) / 2)

multiplication_before_addition = (1 + (2 * 3))
division_before_addition = (1 + (2 / 3))

division_before_subtraction = (3 - (1 / 2))
multiplication_before_subtraction = (3 - (1 * 2))

multiplication_division_left_assoc = ((3 * 4) / 2)
division_multiplication_left_assoc = ((3 / 4) * 2)

addition_subtraction_left_assoc = ((3 - 4) + 2)
subtraction_addition_left_assoc = ((3 + 4) - 2)

exponentiation = (2 ^ 3)
exponentiation_right_assoc = (2 ^ (3 ^ 4))

exponentiation_precedence = ((2 ^ 3) + ((3 ^ 4) * (4 ^ (5 ^ 6))))

equality = (3 = 3)
inequality = (2 <> 3)
less_than = (2 < 3)
less_or_equal_than = (2 <= 3)
greater_than = (3 > 2)
greater_or_equal_than = (3 >= 2)

and_expr = (q and p)
or_expr = (q or p)
xor_expr = (q xor p)
implies_expr = (p implies q)

precedence_and_before_or = (p or (q and r))
precedence_or_before_and = ((p and q) or r)

precedence_or_before_xor = ((p or q) xor r)
precedence_xor_before_or = (p xor (q or r))

precedence_xor_before_implies = ((p xor q) implies r)
precedence_implies_before_xor = (p implies (q xor r))

boolean_lower_precedence = ((((a > (1 + 1)) and (b >= (4 / 2))) or ((c <= (3 ^ 2)) and (d = (3 * (- 2))))) xor ((not e) and (not f)))

numeric = (2.0 * 3)
floating_point = (2.0f * 3.0d)

str_concat = (((str1 + "some") + str2) + "else")
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/infix/expected_type.properties

```
# suppress inspection "UnusedProperty" for whole file

addition = integer
subtraction = integer
multiplication = integer
division = integer
modulus = integer

addition_left_assoc = integer
subtraction_left_assoc = integer
multiplication_left_assoc = integer
division_left_assoc = integer

multiplication_before_addition = integer
division_before_addition = integer

division_before_subtraction = integer
```

```
multiplication_before_subtraction = integer

multiplication_division_left_assoc = integer
division_multiplication_left_assoc = integer

addition_subtraction_left_assoc = integer
subtraction_addition_left_assoc = integer

exponentiation = integer
exponentiation_right_assoc = integer

exponentiation_precedence = integer

equality = boolean
inequality = boolean
less_than = boolean
less_or_equal_than = boolean
greater_than = boolean
greater_or_equal_than = boolean

and_expr = boolean
or_expr = boolean
xor_expr = boolean
implies_expr = boolean

precedence_and_before_or = boolean
precedence_or_before_and = boolean

precedence_or_before_xor = boolean
precedence_xor_before_or = boolean

precedence_xor_before_implies = boolean
precedence_implies_before_xor = boolean

boolean_lower_precedence = boolean

numeric = decimal
floating_point = double

str_concat = string
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/infix/source/main.cml

```
@concept Expressions
{
    // Arithmetic Operators:
    addition = 1 + 2;
    subtraction = 2 - 1;
    multiplication = 2 * 3;
    division = 6 / 2;
    modulus = 5 % 2;

    // Left Associative:
    addition_left_assoc = 1 + 2 + 3;
    subtraction_left_assoc = 3 - 2 - 1;
    multiplication_left_assoc = 2 * 3 * 1;
    division_left_assoc = 12 / 6 / 2;

    // Precedence of multiplication and division over addition:
    multiplication_before_addition = 1 + 2 * 3;
    division_before_addition = 1 + 2 / 3;

    // Precedence of multiplication and division over subtraction:
    division_before_subtraction = 3 - 1 / 2;
    multiplication_before_subtraction = 3 - 1 * 2;

    // Same precedence for multiplication and division:
    multiplication_division_left_assoc = 3 * 4 / 2;
    division_multiplication_left_assoc = 3 / 4 * 2;

    // Same precedence for addition and subtraction:
    addition_subtraction_left_assoc = 3 - 4 + 2;
    subtraction_addition_left_assoc = 3 + 4 - 2;

    // Exponentiation:
    exponentiation = 2 ^ 3;
    exponentiation_right_assoc = 2 ^ 3 ^ 4;
    exponentiation_precedence = 2^3+3^4*4^5^6;

    // Relational Operators:
    equality = 3 == 3;
    inequality = 2 != 3;
    less_than = 2 < 3;
    less_or_equal_than = 2 <= 3;
    greater_than = 3 > 2;
    greater_or_equal_than = 3 >= 2;

    // Boolean Operators:
    q: boolean;
    p: boolean;
    and_expr = q and p;
    or_expr = q or p;
    xor_expr = q xor p;
    implies_expr = p implies q;

    // "and" always takes precedence over "or", and "or" over "xor":
    r: boolean;
    precedence_and_before_or = p or q and r;
    precedence_or_before_and = p and q or r;

    precedence_or_before_xor = p or q xor r;
    precedence_xor_before_or = p xor q or r;

    precedence_xor_before_implies = p xor q implies r;
    precedence_implies_before_xor = p implies q xor r;

    // Arithmetic and relational operators always take precedence over boolean ones:
    a: integer;
    b: integer;
    c: integer;
    d: integer;
    e: boolean;
    f: boolean;
    boolean_lower_precedence = a > 1+1 and b >= 4/2 or c <= 3^2 and d == 3*-2 xor not e and not f;

    // Numeric Types:
    numeric = 2.0 * 3;

    // Binary Floating-Point:
    floating_point = 2.0f * 3.0d;

    // String concatenation:
    str1: string;
    str2: string;
    str_concat = str1 & "some" & str2 & "else";
}
```

```
==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/invocation/expected_ocl.properties

# suppress inspection "UnusedProperty" for whole file

path_ancestors = class.ancestors

exists_invocation = top_items->exists((name = "this"))
select_invocation = top_items->select((name = "this"))
reject_invocation = top_items->reject((name = "this"))

select_invocation_embedded = top_orders->select(items->exists((qty > 10)))->select((total > 100))

yield_invocation = top_orders->collect(items)->flatten()
yield_invocation_distinct = top_orders->collect(items.product)->flatten()->asSet()

recurse_ancestors = class->closure(ancestors)->flatten()
recurse_distinct = class->closure(ancestors)->flatten()->asSet()
recurse_children = children->closure(children)->flatten()->select((ranking > 10))
recurse_self = self->closure(children)->flatten()
recurse_self_nested = self->closure(children)->flatten()->select((ranking > 10))

reduce_invocation = top_orders->iterate(order; sum = 0.0 | (sum + order.total))


==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/invocation/expected_type.properties

# suppress inspection "UnusedProperty" for whole file

path_ancestors = Class*

exists_invocation = boolean
select_invocation = Item*
reject_invocation = Item*

select_invocation_embedded = Order*

yield_invocation = Item*
yield_invocation_distinct = Product*

recurse_ancestors = Class*
recurse_distinct = Class*
recurse_children = Class*
recurse_self = Class*
recurse_self_nested = Class*

reduce_invocation = decimal


==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/invocation/source/main.cml


@template @function exists<T>(seq: T*, expr: (T) -> Boolean) -> BOOLEAN;

@template @function select<T>(seq: T*, expr: (T) -> Boolean) -> T*;
@template @function reject<T>(seq: T*, expr: (T) -> Boolean) -> T*;

@template @function yield<T>(seq: T*, expr: (T) -> R*) -> R*;

@template @function distinct<T>(seq: T*) -> T*;

@template @function recurse<T,S>(seq: T*, expr: (T) -> S*) -> S*;

@template @function reduce<T,S>(seq: T*, expr: (T, S) -> S, from: () -> S) -> S;

@concept Expressions
{
    top_items: Item*;
    class: Class;
    children: Class*;
    top_orders: Order*;

    path_ancestors = class.ancestors;

    exists_invocation = exists(top_items, { name == "this" });
    select_invocation = select(top_items, { name == "this" });
    reject_invocation = reject(top_items) { name == "this" }; // trailing lambdas

    select_invocation_embedded = select(select(top_orders, { exists(items, { qty > 10 }) }), { total > 100 });

    yield_invocation = yield(top_orders, { items });
    yield_invocation_distinct = distinct(yield(top_orders, { items.product }));

    recurse_ancestors = recurse(class, { ancestors });
    recurse_distinct = distinct(recurse(class, { ancestors }));

    recurse_children = select(recurse(children, { children }), { ranking > 10 }); // equivalent to: self | recurse children | ...
    recurse_self = recurse(self, { children });
    recurse_self_nested = select(recurse(self, { children }), { ranking > 10});

    reduce_invocation = reduce(top_orders, { order, sum -> sum + order.total }, { 0.0 });
}

@concept Order
{
    customer: Customer;
    items: Item*;
    total: DECIMAL;
}

@concept Item
{
    name: STRING;
    product: Product;
    qty: INTEGER;
}

@concept Customer
{
    name: STRING;
}

@concept Product;

@concept Class
{
    ancestors: Class*;
    children: Class*;
    ranking: INTEGER;
}


==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/literals/expected_ocl.properties
```

```
# suppress inspection "UnusedProperty" for whole file

LiteralTrueBoolean = true
LiteralFalseBoolean = false
LiteralStringInit = "\\tSome \\"String\\"\\n"
LiteralIntegerInit = 123
LiteralDecimalInit = 123.456
LiteralDecimalInit2 = .456
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/literals/expected_type.properties

```
# suppress inspection "UnusedProperty" for whole file

LiteralTrueBoolean = boolean
LiteralFalseBoolean = boolean
LiteralStringInit = string
LiteralIntegerInit = integer
LiteralDecimalInit = decimal
LiteralDecimalInit2 = decimal
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/literals/source/main.cml

```
@concept Expressions
{
    LiteralTrueBoolean = true;
    LiteralFalseBoolean = false;
    LiteralStringInit = "\tSome \"String\"\n";
    LiteralIntegerInit = 123;
    LiteralDecimalInit = 123.456;
    LiteralDecimalInit2 = .456;
}
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/ocl.stg

```
ocl(expr) ::= <<
<({<expr.kind>_ocl})(expr)>
>>

invocation_ocl(invocation) ::= <<
<({<invocation.name>})(invocation.namedArguments)>
>>

lambda_ocl(lambda) ::= <<
<if(lambda.parameters)><lambda.parameters; separator=", "> | <endif><\\>
<ocl(lambda.innerExpression)>
>>

exists(args) ::= <<
<collection_invocation(args, "exists", "")>
>>

select(args) ::= <<
<collection_invocation(args, "select", "")>
>>

reject(args) ::= <<
<collection_invocation(args, "reject", "")>
>>

yield(args) ::= <<
<collection_invocation(args, "collect", "->flatten()")>
>>

recurse(args) ::= <<
<collection_invocation(args, "closure", "->flatten()")>
>>

reduce(args) ::= <<
<ocl(args.seq)>->iterate(<args.expr.parameters; separator="; "> = <ocl(args.from.innerExpression)> | <ocl(args.expr.innerExpression)>)
>>

distinct(args) ::= <<
<ocl(args.seq)>->asSet()
>>

flatten(expr) ::= <<
(<expr> -> flatten())
>>

cross_join(args) ::= <<
cross_join(<ocl(args.seq1)>, <ocl(args.seq2)>)
>>

collection_invocation(args, operation, appendix) ::= <<
<ocl(args.seq)>-><operation>(<ocl(args.expr)>)<appendix>
>>

path_ocl(path) ::= <<
<path.names; separator=".">
>>

unary_ocl(unary) ::= <<
(<expr_operator(unary)> <ocl(unary.subExpr)>)
>>

arithmetic_ocl(expr) ::= <<
<infix_ocl(expr)>
>>

logical_ocl(expr) ::= <<
<infix_ocl(expr)>
>>

relational_ocl(expr) ::= <<
<infix_ocl(expr)>
>>

referential_ocl(expr) ::= <<
<infix_ocl(expr)>
>>

string_concat_ocl(expr) ::= <<
<infix_ocl(expr)>
>>

infix_ocl(infix) ::= <<
(<ocl(infix.left)> <expr_operator(infix)> <ocl(infix.right)>)
>>

expr_operator(expr) ::= <<
<if(expr.operation)><\\>
<({operator_<expr.operation>})()><\\>
<else><\\>
<expr.operator><\\>
<endif>
```

```
>>

operator_unary_add() ::= "+"
operator_unary_sub() ::= "-"

operator_add() ::= "+"
operator_sub() ::= "-"
operator_mul() ::= "*"
operator_div() ::= "/"
operator_mod() ::= "%"
operator_exp() ::= "^"

operator_string_concat() ::= "+"

operator_eq() ::= "="
operator_not_eq() ::= "\<>"
operator_ref_eq() ::= "="
operator_not_ref_eq() ::= "\<>"
operator_gt() ::= ">"
operator_gte() ::= ">="
operator_lt() ::= "\<"
operator_lte() ::= "\<="

operator_and() ::= "and"
operator_or() ::= "or"
operator_xor() ::= "xor"
operator_implies() ::= "implies"
operator_not() ::= "not"

conditional_ocl(conditional) ::= <<
(if <ocl(conditional.condExpr)> then <ocl(conditional.thenExpr)> else <ocl(conditional.elseExpr)> endif)
>>

literal_ocl(literal) ::= <<
<({literal_<literal.type.name>})(literal.text)>
>>

literal_boolean(text) ::= <<
<text>
>>

literal_string(text) ::= <<
"<text>"
>>

literal_integer(text) ::= <<
<text>
>>

literal_decimal(text) ::= <<
<text>
>>

literal_float(text) ::= <<
<text>f
>>

literal_double(text) ::= <<
<text>d
>>


==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/parenthesis/expected_ocl.properties

addition_before_multiplication = ((1 + 2) * 3)
addition_before_division = ((1 + 2) / 3)


==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/parenthesis/expected_type.properties

addition_before_multiplication = integer
addition_before_division = integer


==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/parenthesis/source/main.cml

@concept Expressions
{
    addition_before_multiplication = (1 + 2) * 3;
    addition_before_division = (1 + 2) / 3;
}

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/path/expected_ocl.properties

# suppress inspection "UnusedProperty" for whole file

self_var = self
single_var = foo
path_var = somePath.bar
path_var2 = somePath.oneMorePath.etc
path_var3 = somePathList.oneMorePath.etc
derived_value = value.flag
super_derived_value = super_value.flag
derived_some_path = self.somePath


==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/path/expected_type.properties

# suppress inspection "UnusedProperty" for whole file

self_var = Expressions
single_var = string
path_var = integer
path_var2 = decimal
path_var3 = decimal*
derived_value = boolean
super_derived_value = boolean
derived_some_path = SomeConcept

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/path/source/main.cml


@concept Expressions: AbstractConcept
{
    foo: STRING;
    somePath: SomeConcept;
    somePathList: SomeConcept*;

    self_var = self;
    derived_some_path = self.somePath;
    single_var = foo;
    path_var = somePath.bar;
    path_var2 = somePath.oneMorePath.etc;
    path_var3 = somePathList.oneMorePath.etc;

    /derived_value = value.flag;
```

```
    /super_derived_value = super_value.flag;
}

@concept SomeConcept
{
    bar: INTEGER;
    oneMorePath: AnotherConcept;
}

@concept AnotherConcept
{
    etc: DECIMAL;
    flag: BOOLEAN;
}

@abstraction AbstractConcept: SuperAbstractConcept
{
    value: AnotherConcept;
}

@abstraction SuperAbstractConcept
{
    super_value: AnotherConcept;
}
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/unary/expected_ocl.properties

```
# suppress inspection "UnusedProperty" for whole file

plus = (+ 1)
minus = (- 2)
not_expr = (not p)

unary_before_infix = (((+ 1) * (- 2)) + ((- 3) / (+ 4)))
unary_before_infix2 = ((1 * (- 2)) - ((- 3) / (+ 4)))
unary_before_infix3 = ((1 - ((- 2) * (- 3))) + (+ 4))
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/unary/expected_type.properties

```
# suppress inspection "UnusedProperty" for whole file

plus = integer
minus = integer
not_expr = boolean

unary_before_infix = integer
unary_before_infix2 = integer
unary_before_infix3 = integer
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ExpressionTest/unary/source/main.cml

```
@concept Expressions
{
    p: BOOLEAN;

    // Unary operators:
    plus = +1;
    minus = -2;
    not_expr = not p;

    // Unary Precedence:
    unary_before_infix = +1 * -2 + -3 / +4;
    unary_before_infix2 = 1 * -2 - -3 / +4;
    unary_before_infix3 = 1 - -2 * -3 + +4;
}
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ModelLoader/abstract_concept/source/main.cml

```
@abstraction ModelElement;
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ModelLoader/another_module/source/main.cml

```
// This module is imported by: module_name

@concept AnotherConcept;
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ModelLoader/associations/source/main.cml

```
@concept Vehicle
{
    plate: String;
    driver: Employee?;
    owner: Organization;
}

@concept Employee
{
    name: String;
    employer: Organization;
}

@concept Organization
{
    name: String;
    employees: Employee*;
    fleet: Vehicle*;
    drivers = fleet.driver;
}

@association Employment
{
    Employee.employer;
    Organization.employees;
}

@association VehicleOwnership
{
    Vehicle.owner: Organization;
    Organization.fleet: Vehicle*;
}
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ModelLoader/concrete_concept/source/main.cml

```
@concept ModelElement;
```

==> cml-compiler/cml-language/src/test/resources/cml/language/ModelLoader/derived_property/source/main.cml

```
@concept SomeConcept
{
    /derivedProperty: String = "A";
    nonDerivedProperty: String = "A";
}
```

```
==> cml-compiler/cml-language/src/test/resources/cml/language/ModelLoader/expressions/source/main.cml

@concept Expressions
{
    str = "SomeString";
    int = 123;
    dec = 123.456;
}

==> cml-compiler/cml-language/src/test/resources/cml/language/ModelLoader/invalid_module_name/source/main.cml

@module incorrect_module_name
{
}


==> cml-compiler/cml-language/src/test/resources/cml/language/ModelLoader/module_name/source/main.cml

@module module_name
{
    @import another_module;
}

@concept SomeConcept: AnotherConcept;

==> cml-compiler/cml-package/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <parent>
        <groupId>cml</groupId>
        <artifactId>cml-compiler</artifactId>
        <version>master-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>cml-package</artifactId>
    <version>master-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-frontend</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.0.0</version>
                <configuration>
                  <descriptors>
                    <descriptor>src/main/assembly/distribution.xml</descriptor>
                  </descriptors>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>


==> cml-compiler/cml-package/src/main/assembly/distribution.xml

<assembly xmlns="http://maven.apache.org/ASSEMBLY/2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/ASSEMBLY/2.0.0 http://maven.apache.org/xsd/assembly-2.0.0.xsd">
  <id>distribution</id>
  <formats>
    <format>zip</format>
  </formats>
  <fileSets>
    <fileSet>
      <directory>${basedir}/src/main/resources</directory>
      <includes>
        <include>**/*</include>
        <include>**/*.*</include>
      </includes>
      <outputDirectory></outputDirectory>
    </fileSet>
    <fileSet>
      <directory>${basedir}/../../cml-modules</directory>
      <includes>
        <include>**/*.*</include>
      </includes>
      <outputDirectory>cml-modules</outputDirectory>
    </fileSet>
  </fileSets>
  <files>
    <file>
      <source>${basedir}/../cml-frontend/target/cml-compiler-jar-with-dependencies.jar</source>
      <outputDirectory>lib</outputDirectory>
      <destName>cml-compiler.jar</destName>
    </file>
  </files>
</assembly>


==> cml-compiler/cml-package/src/main/resources/bin/cml

#!/bin/bash

if [ -z "$CML_BASE_DIR" ]; then
    export CML_BASE_DIR="##PREFIX##"
fi

export CML_MODULES_PATH="$CML_BASE_DIR/cml-modules"

java -jar "$CML_BASE_DIR/lib/cml-compiler.jar" "$@"


==> cml-compiler/cml-package/src/main/resources/LICENSE.txt

                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
```

and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

==> cml-compiler/cml-primitives/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>cml</groupId>
        <artifactId>cml-compiler</artifactId>
        <version>master-SNAPSHOT</version>
    </parent>

    <artifactId>cml-primitives</artifactId>
    <version>master-SNAPSHOT</version>
    <packaging>jar</packaging>

</project>
```

==> cml-compiler/cml-primitives/src/main/java/cml/primitives/Types.java

```
package cml.primitives;

import java.util.Collection;
import java.util.List;

import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
import static java.util.Collections.unmodifiableCollection;
import static java.util.Collections.unmodifiableList;

@SuppressWarnings({"unused", "WeakerAccess"})
public class Types
{
    public static final String BOOLEAN = "boolean";
    public static final String STRING = "string";
    public static final String INTEGER = "integer";
    public static final String LONG = "long";
    public static final String SHORT = "short";
    public static final String BYTE =  "byte";
    public static final String DECIMAL = "decimal";
    public static final String FLOAT = "float";
    public static final String DOUBLE = "double";

    public static final Collection<String> PRIMITIVE_TYPE_NAMES = unmodifiableCollection(asList(
        BOOLEAN, INTEGER, DECIMAL, STRING, // "REGEX", // main primitive types
        BYTE, SHORT, LONG, FLOAT, DOUBLE // "CHAR" // remaining primitive types
    ));

    public static final List<String> NUMERIC_TYPE_NAMES = unmodifiableList(asList(
        BYTE, SHORT, INTEGER, LONG, DECIMAL // from narrower to wider
    ));

    public static final List<String> BINARY_FLOATING_POINT_TYPE_NAMES = unmodifiableList(asList(
        FLOAT, DOUBLE // from narrower to wider
    ));

    public static final List<String> ARITHMETIC_OPERATORS = unmodifiableList(asList(
```

```
        "+", "-", "*", "/", "%", "^"
    ));

    private static final Collection<String> LOGICAL_OPERATORS = unmodifiableCollection(asList(
        "and", "or", "xor", "implies"
    ));

    private static final Collection<String> RELATIONAL_OPERATORS = unmodifiableCollection(asList(
        "==", "!=", ">", ">=", "<", "<="
    ));

    private static final Collection<String> REFERENTIAL_OPERATORS = unmodifiableCollection(asList(
        "===", "!=="
    ));

    private static final Collection<String> STRING_OPERATORS = unmodifiableCollection(singletonList("&"));

    public static String primitiveTypeName(final String name)
    {
        return name.toLowerCase();
    }

    public static boolean primitive(String typeName)
    {
        return PRIMITIVE_TYPE_NAMES.contains(primitiveTypeName(typeName));
    }

    public static boolean numeric(String typeName)
    {
        return NUMERIC_TYPE_NAMES.contains(primitiveTypeName(typeName));
    }

    public static boolean float_(String typeName)
    {
        return BINARY_FLOATING_POINT_TYPE_NAMES.contains(primitiveTypeName(typeName));
    }

    public static boolean boolean_(String typeName)
    {
        return primitiveTypeName(typeName).equals(BOOLEAN);
    }

    public static boolean string(String typeName)
    {
        return primitiveTypeName(typeName).equals(STRING);
    }

    // Used by template: invocation_subtype(args)
    @SuppressWarnings("SimplifiableIfStatement")
    public static boolean subtype(String s, String t)
    {
        if (numeric(s) && numeric(t))
        {
            return numericSubType(s, t);
        }
        else if (float_(s) && float_(t))
        {
            return binaryFloatingPointSubtype(s, t);
        }
        else if (primitive(s) && primitive(t))
        {
            return primitiveTypeName(s).equals(primitiveTypeName(t));
        }
        else
        {
            return false;
        }
    }

    public static boolean numericSubType(String s, String t)
    {
        if (numeric(s) && numeric(t))
        {
            final int i = NUMERIC_TYPE_NAMES.indexOf(primitiveTypeName(s));
            final int j = NUMERIC_TYPE_NAMES.indexOf(primitiveTypeName(t));

            return i <= j;
        }

        return false;
    }

    public static boolean binaryFloatingPointSubtype(String s, String t)
    {
        if (float_(s) && float_(t))
        {
            final int i = BINARY_FLOATING_POINT_TYPE_NAMES.indexOf(primitiveTypeName(s));
            final int j = BINARY_FLOATING_POINT_TYPE_NAMES.indexOf(primitiveTypeName(t));

            return i <= j;
        }

        return false;
    }

    public static boolean arithmeticOperator(String operator)
    {
        return ARITHMETIC_OPERATORS.contains(operator);
    }

    public static boolean logicalOperator(String operator)
    {
        return LOGICAL_OPERATORS.contains(operator);
    }

    public static boolean relationalOperator(String operator)
    {
        return RELATIONAL_OPERATORS.contains(operator);
    }

    public static boolean referentialOperator(String operator)
    {
        return REFERENTIAL_OPERATORS.contains(operator);
    }

    public static boolean stringOperator(String operator)
    {
        return STRING_OPERATORS.contains(operator);
    }
}
```

==> cml-compiler/cml-templates/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>cml</groupId>
        <artifactId>cml-compiler</artifactId>
        <version>master-SNAPSHOT</version>
    </parent>

    <artifactId>cml-templates</artifactId>
    <version>master-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>cml</groupId>
            <artifactId>cml-io</artifactId>
            <version>${project.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.antlr</groupId>
            <artifactId>ST4</artifactId>
            <version>4.0.8</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>15.0</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
            <version>2.6</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

</project>
```

==> cml-compiler/cml-templates/src/main/java/cml/templates/ModelAdaptor.java

```
package cml.templates;

import org.jooq.lambda.Seq;
import org.stringtemplate.v4.Interpreter;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.misc.ObjectModelAdaptor;
import org.stringtemplate.v4.misc.STNoSuchPropertyException;

import java.util.Optional;

public class ModelAdaptor extends ObjectModelAdaptor
{
    @Override
    public synchronized Object getProperty(Interpreter interp, ST self, Object o, Object property, String propertyName)
        throws STNoSuchPropertyException
    {
        final Object value = super.getProperty(interp, self, o, property, propertyName);

        if (value instanceof Optional)
        {
            @SuppressWarnings("rawtypes")
            final Optional optionalValue = (Optional) value;

            //noinspection unchecked
            return optionalValue.orElse(null);
        }
        else if (value instanceof Seq)
        {
            @SuppressWarnings("rawtypes")
            final Seq seq = (Seq) value;

            //noinspection unchecked
            return seq.toList();
        }
        else
        {
            return value;
        }
    }
}
```

==> cml-compiler/cml-templates/src/main/java/cml/templates/NameRenderer.java

```
package cml.templates;

import org.stringtemplate.v4.StringRenderer;

import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.concurrent.atomic.AtomicInteger;

import static java.util.Arrays.stream;
import static java.util.Collections.synchronizedMap;
import static java.util.stream.Collectors.toList;
import static org.apache.commons.lang.StringUtils.splitByCharacterTypeCamelCase;

public class NameRenderer extends StringRenderer
{
    private static final String PASCAL_CASE = "pascal-case";
    private static final String CAMEL_CASE = "camel-case";
    private static final String UNDERSCORE_CASE = "underscore-case";
    private static final String LOWER_CASE = "lower-case";
    private static final String UPPER_CASE = "upper-case";
    private static final String INDEXED = "indexed";
    private static final String INC_INDEXED = "inc-indexed";

    private static final String UNDERSCORE = "_";
    private static final String DASH = "-";
    private static final String DOT = ".";

    private final Map<String, AtomicInteger> indices = synchronizedMap(new HashMap<>());

    @Override
```

```
    public String toString(Object o, String formatString, Locale locale)
    {
        if (o == null)
        {
            return "";
        }
        else if (PASCAL_CASE.equals(formatString))
        {
            return pascalCase(locale, o.toString());
        }
        else if (CAMEL_CASE.equals(formatString))
        {
            return camelCase(locale, o.toString());
        }
        else if (UNDERSCORE_CASE.equals(formatString))
        {
            return underscoreCase(locale, o.toString());
        }
        else if (LOWER_CASE.equals(formatString))
        {
            return o.toString().toLowerCase(locale);
        }
        else if (UPPER_CASE.equals(formatString))
        {
            return o.toString().toUpperCase(locale);
        }
        else if (INC_INDEXED.equals(formatString))
        {
            return new_indexed(o.toString());
        }
        else if (INDEXED.equals(formatString))
        {
            return indexed(o.toString());
        }
        else
        {
            return super.toString(o, formatString, locale);
        }
    }

    private String new_indexed(final String text)
    {
        indices.computeIfAbsent(text, key -> new AtomicInteger(0));

        return text + indices.get(text).incrementAndGet();
    }

    private String indexed(final String text)
    {
        indices.computeIfAbsent(text, key -> new AtomicInteger(0));

        return text + indices.get(text);
    }

    public static String pascalCase(Locale locale, String str)
    {
        final List<String> words = words(str);
        final StringBuilder builder = new StringBuilder(str.length() + words.size());

        for (String p : words)
        {
            builder.append(capitalized(locale, p));
        }

        return builder.toString();
    }

    public static String camelCase(Locale locale, String str)
    {
        final List<String> words = words(str);
        final StringBuilder builder = new StringBuilder(str.length() + words.size());

        builder.append(words.get(0).toLowerCase(locale));
        words.remove(0);

        for (String p : words)
        {
            builder.append(capitalized(locale, p));
        }

        return builder.toString();
    }

    public static String underscoreCase(Locale locale, String str)
    {
        final List<String> words = words(str);
        final StringBuilder builder = new StringBuilder(str.length() + words.size());

        for (String p : words)
        {
            builder.append(p.toLowerCase(locale)).append(UNDERSCORE);
        }

        return builder.substring(0, builder.length() - 1);
    }

    private static List<String> words(String str)
    {
        return stream(splitByCharacterTypeCamelCase(str))
            .map(String::trim)
            .filter(word -> !isSeparator(word))
            .collect(toList());
    }

    private static boolean isSeparator(String word)
    {
        return word.equals(UNDERSCORE) || word.equals(DASH) || word.equals(DOT);
    }

    private static String capitalized(Locale locale, String word)
    {
        final StringBuilder builder = new StringBuilder(word.length());

        builder.append(word.substring(0, 1).toUpperCase(locale));
        builder.append(word.substring(1).toLowerCase(locale));

        return builder.toString();
    }
}


==> cml-compiler/cml-templates/src/main/java/cml/templates/TemplateFile.java

package cml.templates;

public class TemplateFile
{
```

```
    private final String moduleName;
    private final String path;

    TemplateFile(String moduleName, final String path)
    {
        this.moduleName = moduleName;
        this.path = path;
    }

    public String getModuleName()
    {
        return moduleName;
    }

    public String getPath()
    {
        return path;
    }
}
```

==> cml-compiler/cml-templates/src/main/java/cml/templates/TemplateGroupFile.java

```
package cml.templates;

import cml.io.ModuleManager;
import org.antlr.runtime.RecognitionException;
import org.antlr.runtime.Token;
import org.stringtemplate.v4.STGroup;
import org.stringtemplate.v4.STGroupFile;
import org.stringtemplate.v4.misc.ErrorManager;
import org.stringtemplate.v4.misc.Misc;
import org.stringtemplate.v4.misc.STLexerMessage;

import java.net.URL;
import java.util.Optional;

public class TemplateGroupFile extends STGroupFile
{
    private static final String ENCODING = "UTF-8";
    private static final char START_CHAR = '<';
    private static final char STOP_CHAR = '>';

    private static ModuleManager moduleManager;

    public static void setModuleManager(ModuleManager moduleManager)
    {
        TemplateGroupFile.moduleManager = moduleManager;
    }

    public TemplateGroupFile(String path)
    {
        super(path, ENCODING, START_CHAR, STOP_CHAR);

        errMgr = new ErrorManager()
        {
            @Override
            public void lexerError(final String srcName, final String msg, final Token templateToken, final RecognitionException e)
            {
                // Overriding to display the full srcName:
                listener.compileTimeError(new STLexerMessage(srcName, msg, templateToken, e));
            }
        };

        registerModelAdaptor(Object.class, new ModelAdaptor());
        registerRenderer(String.class, new NameRenderer());
    }

    @Override
    public URL getURL(String path)
    {
        final Optional<URL> templateFile = moduleManager.findTemplateFile(path);
        return templateFile.orElse(null);
    }

    @Override
    public void importTemplates(Token fileNameToken)
    {
        final STGroup g = new TemplateGroupFile(importFilePathOf(fileNameToken));

        g.setListener(getListener());

        importTemplates(g, true);
    }

    private String importFilePathOf(final Token fileNameToken)
    {
        final String filePath = Misc.strip(fileNameToken.getText(), 1);

        if (filePath.contains(":/"))
        {
            return filePath;
        }
        else
        {
            final String moduleName = moduleManager.getModuleName(getFileName());
            return moduleManager.getModulePath(moduleName, filePath);
        }
    }
}
```

==> cml-compiler/cml-templates/src/main/java/cml/templates/TemplateRenderer.java

```
package cml.templates;

import cml.io.Console;
import org.stringtemplate.v4.ST;
import org.stringtemplate.v4.STGroupFile;

import java.util.Map;
import java.util.Map.Entry;

public interface TemplateRenderer
{
    String renderTemplate(TemplateFile templateFile, String templateName, Map<String, Object> args);

    static TemplateRenderer create(Console console)
    {
        return new TemplateRendererImpl(console);
    }
}

class TemplateRendererImpl implements TemplateRenderer
{
    private static final String UNABLE_TO_LOAD_TEMPLATE = "Unable to load template named '%s' from file: %s";
```

```
    private final Console console;

    TemplateRendererImpl(Console console)
    {
        this.console = console;
    }

    @Override
    public String renderTemplate(TemplateFile templateFile, String templateName, Map<String, Object> args)
    {
        final STGroupFile groupFile = new TemplateGroupFile(templateFile.getPath());

        final ST template = groupFile.getInstanceOf(templateName);

        if (template == null)
        {
            console.info(UNABLE_TO_LOAD_TEMPLATE, templateName, templateFile.getPath());

            return "";
        }

        for (final Entry<String, Object> entry : args.entrySet())
        {
            template.add(entry.getKey(), entry.getValue());
        }

        return template.render();
    }
}



==> cml-compiler/cml-templates/src/main/java/cml/templates/TemplateRepository.java

package cml.templates;

import cml.io.ModuleManager;

import java.net.URL;
import java.util.Optional;

import static java.lang.String.format;

public interface TemplateRepository
{
    Optional<TemplateFile> findTemplate(String moduleName, String constructorName, String fileName);

    static TemplateRepository create(ModuleManager moduleManager)
    {
        return new TemplateRepositoryImpl(moduleManager);
    }
}

class TemplateRepositoryImpl implements TemplateRepository
{
    private static final String CONSTRUCTOR_PATH = "%s:/constructors/%s/%s";

    private final ModuleManager moduleManager;

    TemplateRepositoryImpl(ModuleManager moduleManager)
    {
        this.moduleManager = moduleManager;
    }

    @Override
    public Optional<TemplateFile> findTemplate(
        final String moduleName,
        final String constructorName,
        final String templateFileName)
    {
        final String path = format(CONSTRUCTOR_PATH, moduleName, constructorName, templateFileName);
        final Optional<URL> url = moduleManager.findTemplateFile(path);

        return url.isPresent() ? Optional.of(new TemplateFile(moduleName, path)) : Optional.empty();
    }

}



==> cml-compiler/pom.xml

<project
        xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml</groupId>
    <artifactId>cml-compiler</artifactId>
    <version>master-SNAPSHOT</version>
    <packaging>pom</packaging>

    <modules>
        <module>cml-primitives</module>
        <module>cml-bootstrapping</module>
        <module>cml-io</module>
        <module>cml-templates</module>
        <module>cml-language</module>
        <module>cml-generator</module>
        <module>cml-frontend</module>
        <module>cml-package</module>
    </modules>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>


==> cml-compiler/release

#!/usr/bin/env bash

git diff-index --quiet HEAD --
```

```
if [ $? -ne 0 ]; then
    echo "Please commit changes before doing a release."
    exit
fi

curl -s "https://api.travis-ci.org/cmlang/cml-compiler.svg?branch=master" | grep pass > /dev/null
if [ $? -ne 0 ]; then
    echo "Please make sure you have a successful build before doing a release."
    exit
fi

if [ -z "$1" ]; then
    echo Please specify the release version as the first argument.
    echo Optionally, also specify the snapshot version to follow the release as the second argument.
    exit 1
fi

CML_SNAPSHOT_VERSION="$2"

if [ -z "$CML_SNAPSHOT_VERSION" ]; then
    CML_SNAPSHOT_VERSION="master"
fi

CML_VERSION="$1"

CML_YEAR=$(echo "${CML_VERSION}" | cut -s -d'.' -f 1)
if [ -z "$CML_YEAR" ]; then
    echo "Missing year in: $CML_VERSION"
    exit 1
fi

CML_MONTH=$(echo "${CML_VERSION}" | cut -s -d'.' -f 2)
if [ -z "$CML_MONTH" ]; then
    echo "Missing month in: $CML_VERSION"
    exit 1
fi

CML_DAY=$(echo "${CML_VERSION}" | cut -s -d'.' -f 3 | cut -s -d'-' -f 1)
if [ -z "$CML_DAY" ]; then
    echo "Missing day or channel in: $CML_VERSION"
    exit 1
fi

CML_CHANNEL=$(echo "${CML_VERSION}" | cut -s -d'-' -f 2)
if [ -z "$CML_CHANNEL" ]; then
    echo "Missing channel in: $CML_VERSION"
    exit 1
fi

CML_PKG_NAME="cml-compiler-${CML_VERSION}.zip"
CML_RELEASE_DIR="cml-releases/cml-compiler"
CML_ROOT="../.."

if [ ! -d "${CML_ROOT}/${CML_RELEASE_DIR}" ]; then
    echo
    echo ERROR: Unable to find local copy of: https://github.com/cmlang/${CML_RELEASE_DIR}
    echo
    exit 1
fi

if [ ! -d "${CML_ROOT}/homebrew-cml" ]; then
    echo
    echo ERROR: Unable to find local copy of: https://github.com/cmlang/homebrew-cml
    echo
    exit 1
fi

if [ -f "${CML_ROOT}/${CML_RELEASE_DIR}/${CML_PKG_NAME}" ]; then
    echo
    echo WARNING: Version ${CML_VERSION} already released at: https://github.com/cmlang/${CML_RELEASE_DIR}
    echo
    exit 1
fi

CML_RELEASE_NOTES="${CML_PKG_NAME}.notes.md"

if [ ! -f "${CML_ROOT}/${CML_RELEASE_DIR}/${CML_RELEASE_NOTES}" ]; then
    echo
    echo REMINDER: Release notes required before releasing version ${CML_VERSION}:
    echo - Please write the release notes in: ${CML_RELEASE_NOTES}
    echo - Once written, please commit the .zip.notes.md file to the cml-releases repository under the cml-compiler dir.
    echo
    echo After that, you should be able to run the release command again.
    echo
    exit 1
fi

git pull --tags origin master
if [ $? -ne 0 ]; then exit; fi

mvn clean install
if [ $? -ne 0 ]; then exit; fi

mvn versions:set -DnewVersion=${CML_VERSION}
if [ $? -ne 0 ]; then exit; fi

mvn clean install
if [ $? -ne 0 ]; then exit; fi

cd cml-package
if [ $? -ne 0 ]; then exit; fi

mvn clean assembly:single
if [ $? -ne 0 ]; then exit; fi

cd target
if [ $? -ne 0 ]; then exit; fi

CML_PKG_SOURCE_NAME="cml-package-${CML_VERSION}-distribution.zip"
CML_PKG_SHA=$(shasum -a 256 ${CML_PKG_SOURCE_NAME} | awk '{print $1}')
if [ $? -ne 0 ]; then exit; fi

CML_ROOT="../../../.."
cp "${CML_PKG_SOURCE_NAME}" "${CML_ROOT}/${CML_RELEASE_DIR}/${CML_PKG_NAME}"
if [ $? -ne 0 ]; then exit; fi

cd "${CML_ROOT}/${CML_RELEASE_DIR}"
if [ $? -ne 0 ]; then exit; fi

git diff-index --quiet HEAD --
if [ $? -ne 0 ]; then
    echo "Uncommitted changes in the release dir."
    exit
fi

git pull --tags origin master
if [ $? -ne 0 ]; then exit; fi
```

```
CML_PKG_SHA_TXT="${CML_PKG_NAME}.sha256.txt"
echo "${CML_PKG_SHA}" >> "${CML_PKG_SHA_TXT}"
if [ $? -ne 0 ]; then exit; fi

git add "${CML_PKG_NAME}"
if [ $? -ne 0 ]; then exit; fi

git add "${CML_PKG_SHA_TXT}"
if [ $? -ne 0 ]; then exit; fi

git add "${CML_RELEASE_NOTES}"
if [ $? -ne 0 ]; then exit; fi

git commit -m "Committing version ${CML_VERSION} of CML Compiler."
if [ $? -ne 0 ]; then exit; fi

git tag -m "CML Compiler ${CML_VERSION}" -a ${CML_VERSION}
if [ $? -ne 0 ]; then exit; fi

git push --tags origin master
if [ $? -ne 0 ]; then exit; fi

CML_ROOT="../.."
cd "${CML_ROOT}/homebrew-cml"
if [ $? -ne 0 ]; then exit; fi

git diff-index --quiet HEAD --
if [ $? -ne 0 ]; then
    echo "Uncommitted changes in the Homebrew dir."
    exit
fi

git pull --tags origin master
if [ $? -ne 0 ]; then exit; fi

CML_FORMULA="cml-compiler.rb"
CML_TAP_URL="https://raw.githubusercontent.com/cmlang/cml-releases/master/cml-compiler"
cat <<EOF > ${CML_FORMULA}

class CmlCompiler < Formula
  VERSION = "${CML_VERSION}"

  desc "The CML Compiler"
  homepage "https://github.com/cmlang"
  url "${CML_TAP_URL}/cml-compiler-#{VERSION}.zip"
  sha256 "${CML_PKG_SHA}"

  def install
    inreplace "bin/cml", "##PREFIX##", "#{prefix}/libexec"
    libexec.install Dir["*"]
    bin.install_symlink libexec/"bin/cml"
  end

  test do
    assert_equal "Version: #{VERSION}", \`cml --version\`.strip
  end
end
EOF
if [ $? -ne 0 ]; then exit; fi

CML_MINOR_VERSION="${CML_YEAR}.${CML_MONTH}.${CML_DAY}"
CML_MINOR_VERSION_JOINED="${CML_YEAR}${CML_MONTH}${CML_DAY}"
CML_MINOR_FORMULA="cml-compiler@${CML_MINOR_VERSION}.rb"
sed "s/CmlCompiler/CmlCompilerAT${CML_MINOR_VERSION_JOINED}/g" < ${CML_FORMULA} > "${CML_MINOR_FORMULA}"
if [ $? -ne 0 ]; then exit; fi

CML_MAJOR_FORMULA="cml-compiler@${CML_YEAR}.rb"
sed "s/CmlCompiler/CmlCompilerAT${CML_YEAR}/g" < ${CML_FORMULA} > "${CML_MAJOR_FORMULA}"
if [ $? -ne 0 ]; then exit; fi

CML_CHANNEL_CLASS=$(echo ${CML_CHANNEL} | sed -e 's/^a/A/g' -e 's/^b/B/g' -e 's/^s/S/g')
CML_CHANNEL_FORMULA="cml-compiler-${CML_CHANNEL}.rb"
sed "s/CmlCompiler/CmlCompiler${CML_CHANNEL_CLASS}/g" < ${CML_FORMULA} > "${CML_CHANNEL_FORMULA}"
if [ $? -ne 0 ]; then exit; fi

git add ${CML_FORMULA}
if [ $? -ne 0 ]; then exit; fi

git add ${CML_MAJOR_FORMULA}
if [ $? -ne 0 ]; then exit; fi

git add ${CML_MINOR_FORMULA}
if [ $? -ne 0 ]; then exit; fi

git add ${CML_CHANNEL_FORMULA}
if [ $? -ne 0 ]; then exit; fi

git commit -m "Committing version ${CML_VERSION} of CML Compiler."
if [ $? -ne 0 ]; then exit; fi

git tag -m "CML Compiler ${CML_VERSION}" -a ${CML_VERSION}
if [ $? -ne 0 ]; then exit; fi

git push --tags origin master
if [ $? -ne 0 ]; then exit; fi

CML_ROOT=".."
cd "${CML_ROOT}/cml-compiler"
if [ $? -ne 0 ]; then exit; fi

git add -A .
if [ $? -ne 0 ]; then exit; fi

git commit -m "Committing version: ${CML_VERSION}"
if [ $? -ne 0 ]; then exit; fi

git tag -m "Release ${CML_VERSION}" -a ${CML_VERSION}
if [ $? -ne 0 ]; then exit; fi

git push --tags origin master
if [ $? -ne 0 ]; then exit; fi

cd cml-compiler
snapshot-version ${CML_SNAPSHOT_VERSION}
if [ $? -ne 0 ]; then exit; fi

git push --tags origin master
if [ $? -ne 0 ]; then exit; fi

echo
echo Version ${CML_VERSION} released.
echo
echo Check release with:
echo \$ brew tap cmlang\/cml
echo \$ brew upgrade cml-compiler
echo \$ cd \<module\/path\>
```

```
echo \$ cml \<task_name\>
echo


==> cml-compiler/snapshot-version

#!/usr/bin/env bash

if [ -z "$1" ]; then
    echo Please specify the snapshot version as the only argument.
    exit 1
fi

CML_VERSION="$1-SNAPSHOT"
mvn versions:set -DnewVersion=${CML_VERSION}
if [ $? -ne 0 ]; then exit; fi

git add -A .
if [ $? -ne 0 ]; then exit; fi

git commit -m "Committing snapshot version: ${CML_VERSION}"
if [ $? -ne 0 ]; then exit; fi



==> cml-modules/cml_base/source/functions.cml


@function empty<T>(seq: T*) -> boolean;
@function present<T>(seq: T*) -> boolean;
//@function present<T>(seq: T*) -> boolean = not empty(seq);

@function first<T>(seq: T*) -> T?;
@function last<T>(seq: T*) -> T?;

@function exists<T>(seq: T*, expr: (T) -> boolean) -> boolean;
@function all<T>(seq: T*, expr: (T) -> boolean) -> boolean;
@function none<T>(seq: T*, expr: (T) -> boolean) -> boolean;

@function includes<T>(seq: T*, expr: T) -> boolean;
@function excludes<T>(seq: T*, expr: T) -> boolean;

@function select<T>(seq: T*, expr: (T) -> Boolean) -> T*;
@function reject<T>(seq: T*, expr: (T) -> Boolean) -> T*;

@function collect<T>(seq: T*, expr: (T) -> R*) -> R*;

@function distinct<T>(seq: T*) -> T*;
@function reverse<T>(seq: T*) -> T*;

@function sort<T>(seq: T*, expr: (T, T) -> integer) -> T*;
@function compare<T>(expr1: T, expr2: T) -> integer;

@function concat<T>(seq1: T*, seq2: T*) -> T*;

@function count<T>(seq: T*) -> long;

@function cross_join<T,S>(seq1: T*, seq2: S*) -> (T, S)*;


==> cml-modules/cml_base/templates/constructors/_java/files.stg


model_files(task, model) ::= <<
_java:pom_file|pom.xml
>>

module_files(task, module) ::= <<
<if(module.definedFunctions)>
_java:functions_file|src/main/java/<task.packagePath>/<module.name; format="pascal-case">Functions.java
<endif>
>>

concept_files(task, concept) ::= <<
concept_file|src/main/java/<task.packagePath>/<concept.name; format="pascal-case">.java
>>

association_files(task, association) ::= <<
association_file|src/main/java/<task.packagePath>/<association.name; format="pascal-case">.java
>>

==> cml-modules/cml_base/templates/constructors/_java/functions_file.stg


import "/lang/java.stg"

functions_file(task, module) ::= <<
package <task.packageName>;

<common_imports(task, module.model)>
public<\ ><\\>
<structure(
    keyword="class",
    header=functions_class_name(module),
    base_list=true,
    sections = [functions(module)]
)>
>>

functions(module) ::= <<
<module.definedFunctions:static_function_declaration();separator="\n\n">
>>

static_function_declaration(function) ::= <<
public<\ ><\\>
<static_operation(
    name=function_name(function),
    params=function.parameters,
    result_type=getter_type(function.type),
    type_params=function.typeParams,
    statements=function_expression(function)
)>
>>

function_name(function) ::= <<
<function.name; format="camel-case">
>>

function_expression(function) ::= <<
<return(function_result(function))>
>>

function_result(function) ::= <<
<({expression_<function.expression.type.kind>_to_<function.type.kind>})(function.expression)>
>>
```

```
path_root_call_self_field(path) ::= <<
<if(path.type.optional)>Optional.ofNullable(<endif><\\>
<field_name(path)><\\>
<if(path.type.optional)>)<endif>
>>


==> cml-modules/cml_base/templates/constructors/_java/pom_file.stg


pom_file(task, model) ::= <<
<project_start()>

    \<modelVersion>4.0.0\</modelVersion>

    <module_def(task)>

    \<properties>
        \<project.build.sourceEncoding>UTF-8\</project.build.sourceEncoding>
    \</properties>

    \<dependencies>
        <dependencies()>
    \</dependencies>

    \<build>
        <plugins()>
    \</build>

<project_end()>
>>

project_start() ::= <<
\<project
    <pom_ns()>
    <xsi()>
    <ns_loc()>
>>

pom_ns() ::= <<
xmlns="http://maven.apache.org/POM/4.0.0"
>>

xsi() ::= <<
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>>

ns_loc() ::= <<
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
>>

project_end() ::= <<
\</project>
>>

module_def(task) ::= <<
\<groupId><task.groupId>\</groupId>
\<artifactId><task.artifactId>\</artifactId>
\<version><task.artifactVersion>\</version>
\<packaging>jar\</packaging>
>>

dependencies() ::= <<
\<dependency>
    \<groupId>org.jetbrains\</groupId>
    \<artifactId>annotations\</artifactId>
    \<version>15.0\</version>
\</dependency>
\<dependency>
    \<groupId>org.jooq\</groupId>
    \<artifactId>jool\</artifactId>
    \<version>0.9.12\</version>
    \<scope>compile\</scope>
\</dependency>
>>

plugins() ::= <<
\<plugins>
    \<plugin>
        \<groupId>org.apache.maven.plugins\</groupId>
        \<artifactId>maven-compiler-plugin\</artifactId>
        \<version>3.1\</version>
        \<configuration>
            \<source>1.8\</source>
            \<target>1.8\</target>
        \</configuration>
    \</plugin>
    \<plugin>
      \<groupId>org.apache.maven.plugins\</groupId>
      \<artifactId>maven-source-plugin\</artifactId>
      \<version>2.4\</version>
      \<executions>
        \<execution>
          \<id>attach-sources\</id>
          \<goals>
            \<goal>jar\</goal>
          \</goals>
        \</execution>
      \</executions>
    \</plugin>
\</plugins>
>>

==> cml-modules/cml_base/templates/constructors/_py/files.stg


model_files(task, model) ::= <<
_py:setup_file|setup.py
init_file|<python_file_name(task.moduleName)>/__init__.py
>>

python_file_name(name) ::= <<
<name; format="underscore-case">
>>



==> cml-modules/cml_base/templates/constructors/_py/init_file.stg


init_file(task, model) ::= <<
from typing import *
from abc import *
from decimal import *

import functools, itertools
```

```
<if(model.associations)>
<model.associations:py_association_class();separator="\n\n\n">


<endif>
<model.orderedConcepts:py_concept_class();separator="\n\n\n">
>>


==> cml-modules/cml_base/templates/constructors/_py/setup_file.stg


setup_file(task, model) ::= <<
"""A setuptools based setup module.

See:
https://packaging.python.org/en/latest/distributing.html
https://github.com/pypa/sampleproject
"""

# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
    name='<task.moduleName; format="underscore-case">',

    # Versions should comply with PEP440.  For a discussion on single-sourcing
    # the version across setup.py and the project code, see
    # https://packaging.python.org/en/latest/single_source_version.html
    version='<task.moduleVersion>',

    description='A sample Python project',

    # The project's main homepage.
    url='https://github.com/pypa/sampleproject',

    # Author details
    author='The Python Packaging Authority',
    author_email='pypa-dev@googlegroups.com',

    # Choose your license
    license='MIT',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',

        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: MIT License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3.6.1',
    ],

    # What does your project relate to?
    keywords='sample setuptools development',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests']),

    # Alternatively, if you want to distribute just a my_module.py, uncomment
    # this:
    #   py_modules=["my_module"],

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    # install_requires=['peppercorn'],

    # List additional groups of dependencies here (e.g. development
    # dependencies). You can install these using the following syntax,
    # for example:
    # $ pip install -e .[dev,test]
    extras_require={
        'dev': ['check-manifest'],
        'test': ['coverage'],
    },

    # If there are data files included in your packages that need to be
    # installed, specify them here.  If using Python 2.6 or less, then these
    # have to be included in MANIFEST.in as well.
    # package_data={
    #     'sample': ['package_data.dat'],
    #},

    # Although 'package_data' is the preferred approach, in some case you may
    # need to place data files outside of your packages. See:
    # http://docs.python.org/3.4/distutils/setupscript.html#installing-additional-files # noqa
    # In this case, 'data_file' will be installed into '\<sys.prefix>/my_data'
    #data_files=[('my_data', ['data/data_file'])],

    # To provide executable scripts, use entry points in preference to the
    # "scripts" keyword. Entry points provide cross-platform support and allow
    # pip to create the appropriate form of executable for the target platform.
    #entry_points={
    #    'console_scripts': [
    #        'sample=sample:main',
    #    ],
    #},
)
>>

==> cml-modules/cml_base/templates/constructors/cmlc_java/association_file.stg


import "/design/cmlc_java.stg"

association_file(task, association) ::= <<
package <task.packageName>;

import java.util.*;
```

```
import java.math.*;
import org.jetbrains.annotations.*;

<association_class(association, "Impl")>
>>


==> cml-modules/cml_base/templates/constructors/cmlc_java/concept_file.stg


import "/design/cmlc_java.stg"

concept_file(task, concept) ::= <<
package <task.packageName>;

<common_imports(task, concept.model)>
public <interface(concept, "Impl")>

<class2(concept, "Impl")>
>>


==> cml-modules/cml_base/templates/constructors/cmlc_java/files.stg


import "/constructors/_java/files.stg"

==> cml-modules/cml_base/templates/constructors/cmlc_py/files.stg


import "/constructors/_py/files.stg"



==> cml-modules/cml_base/templates/constructors/cmlc_py/init_file.stg


import "/design/cmlc_py.stg"

import "/constructors/_py/init_file.stg"

py_concept_class(concept) ::= <<
<interface(concept, "Impl")>


<class2(concept, "Impl")>
>>

py_association_class(association) ::= <<
<association_class(association, "")>
>>



==> cml-modules/cml_base/templates/constructors/poj/association_file.stg


import "/design/poj.stg"

association_file(task, association) ::= <<
package <task.packageName>;

import java.util.*;
import java.math.*;
import org.jetbrains.annotations.*;

<association_class(association, "")>
>>

==> cml-modules/cml_base/templates/constructors/poj/concept_file.stg


import "/design/poj.stg"

concept_file(task, concept) ::= <<
package <task.packageName>;

<common_imports(task, concept.model)>
public <class(concept)>
>>



==> cml-modules/cml_base/templates/constructors/poj/files.stg


import "/constructors/_java/files.stg"

==> cml-modules/cml_base/templates/constructors/pop/files.stg


import "/constructors/_py/files.stg"



==> cml-modules/cml_base/templates/constructors/pop/init_file.stg


import "/design/pop.stg"

import "/constructors/_py/init_file.stg"

py_concept_class(concept) ::= <<
<class(concept)>
>>

py_association_class(association) ::= <<
<association_class(association, "")>
>>



==> cml-modules/cml_base/templates/design/cmlc/actual_self.stg

// design/cmlc

actual_self() ::= "actual_self"

actual_self_type(concept) ::= "<field_type_optional(concept)>"

actual_self_decl(concept) ::= "<param_decl(actual_self(), actual_self_type(concept))>"

actual_self_field() ::= <<
<field_access(actual_self())>
```

```
>>


==> cml-modules/cml_base/templates/design/cmlc/class.stg

// design/cmlc

class_self_field(concept) ::= <<
<field_decl(actual_self_decl(concept))>
>>

extend_invocation(conceptRedef) ::= <<
<field_access(conceptRedef.concept)> = <\\>
<type_name(conceptRedef.concept)>.<interface_extend_name(type_name(conceptRedef.concept))><\\>
(<actual_self_field()><extend_invocation_extra_args(conceptRedef)>)<statement_terminator()>
>>

extend_invocation_extra_args(conceptRedef) ::= <<
<commaIfEither(conceptRedef.concept.allAncestors, conceptRedef.propertyRedefs)><\\>
<conceptRedef.concept.allAncestors:field_access(); separator=", "><\\>
<commaIf2(conceptRedef.concept.allAncestors, conceptRedef.propertyRedefs)><\\>
<conceptRedef.propertyRedefs:extend_invocation_arg(); separator=", ">
>>



==> cml-modules/cml_base/templates/design/cmlc/invocation.stg


invocation_concept(concept, args) ::= <<
<type_name(concept)>.<interface_create_name(type_name(concept))>(<args:nullable_expr(); separator=", ">)
>>



==> cml-modules/cml_base/templates/design/cmlc.stg


import "/design/cmlc/actual_self.stg"
import "/design/cmlc/class.stg"
import "/design/cmlc/invocation.stg"


==> cml-modules/cml_base/templates/design/cmlc_java/class.stg

// constructors/cmlc_java

class_primary_constructor(inherited_properties, super_properties, ancestors) ::= <<
<class_constructor(
    [actual_self_decl(concept), concept.superProperties, concept.nonDerivedProperties],
    class_primary_constructor_statements()
)>
>>

class_primary_constructor_statements() ::= <<
<actual_self_init()>

<if(concept.redefinedAncestors)><\\>
<concept.redefinedAncestors:extend_invocation();separator="\n">
<endif><\\>
<[concept.slotProperties]:field_init();separator="\n"><\\>
<association_link_calls(concept)>
>>

class_extension_constructor(concept) ::= <<
<if(concept.allAncestors)><\\>
<class_constructor(
    [actual_self_decl(concept), concept.allAncestors, concept.nonDerivedProperties],
    class_extension_constructor_statements()
)><\\>
<endif>
>>

class_extension_constructor_statements() ::= <<
<actual_self_init()>

<[concept.allAncestors, concept.slotProperties]:field_init();separator="\n"><\\>
<association_link_calls(concept)>
>>

actual_self_init() ::= <<
<actual_self_field()> = <actual_self()> == null ? <self()> : <actual_self()>;
>>

extend_invocation_arg(propertyRedef) ::= <<
<field_or_default_value(propertyRedef.prop)>
>>



==> cml-modules/cml_base/templates/design/cmlc_java/interface.stg

// constructors/cmlc_java

interface_primary_create_method_statements(concept) ::= <<
<return(
    new_call(
        interface_impl_name(),
        [null_value(), concept.superProperties, concept.nonDerivedProperties]
    )
)>
>>

interface_secondary_create_method(concept) ::= <<
<if(concept.initProperties && class_name_suffix)><\\>
<static_operation(
    name=interface_create_name(interface_name()),
    params=[concept.nonInitProperties],
    result_type=interface_name(),
    type_params=[],
    statements=return(init_call(interface_create_name(interface_name()), [concept.superProperties, concept.nonDerivedProperties]))
)><\\>
<endif>
>>


==> cml-modules/cml_base/templates/design/cmlc_java.stg


import "/design/cmlc_java/interface.stg"
import "/design/cmlc_java/class.stg"

import "/design/cmlc.stg"
import "/design/java.stg"
import "/design/common.stg"

import "/lang/java.stg"
```

```
==> cml-modules/cml_base/templates/design/cmlc_py/class.stg

// constructors/cmlc_py

class_primary_constructor(inherited_properties, super_properties, ancestors) ::= <<
<if(concept.abstraction || !concept.superProperties)><\\>
<class_constructor(
    [actual_self_decl(concept), ancestors:optional_param_decl(), concept.nonDerivedProperties],
    class_primary_constructor_statements()
)><\\>
<else><\\>
<class_constructor(
    [actual_self_decl(concept), ancestors:optional_param_decl(), concept.nonDerivedProperties, class_primary_constructor_keyword_args(concept)],
    class_primary_constructor_statements()
)><\\>
<endif>
>>

class_primary_constructor_keyword_args(concept) ::= "**kwargs"

class_primary_constructor_statements() ::= <<
<if(super_properties)><\\>
<concept.nonDerivedProperties:field_init();separator="\n"><\\>
<else><\\>
<actual_self_init()>

<if(concept.redefinedAncestors)><\\>
<concept.redefinedAncestors:field_init_or_extend_invocation();separator="\n">
<else><\\>
<ancestors:field_init();separator="\n">
<endif><\\>
<[concept.slotProperties]:field_init();separator="\n"><\\>
<association_link_calls(concept)><\\>
<endif>
>>

field_init_or_extend_invocation(conceptRedef) ::= <<
<if(concept.abstraction)><\\>
<field_init(conceptRedef.concept)><\\>
<else><\\>
if <field_name(conceptRedef.concept)> is None:
<indent()><extend_invocation(conceptRedef)>
else:
<indent()><field_init(conceptRedef.concept)><\\>
<endif>
>>

actual_self_init() ::= <<
if <actual_self()> is None:
<indent()><actual_self_field()> = <self()>  # type: <actual_self_type(concept)>
else:
<indent()><actual_self_field()> = <actual_self()>
>>

extend_invocation_arg(propertyRedef) ::= <<
<if(propertyRedef.prop.derived)><\\>
<field_default_value(propertyRedef.prop.type)><\\>
<elseif(propertyRedef.redefined)><\\>
<field_name(propertyRedef.prop)><\\>
<else><\\>
kwargs['<field_name(propertyRedef.prop)>']<\\>
<endif>
>>


==> cml-modules/cml_base/templates/design/cmlc_py/interface.stg

// constructors/cmlc_py

interface_primary_create_method_statements(concept) ::= <<
<return(
    new_call(
        interface_impl_name(),
        [null_value(), concept.allAncestors:{item|<null_value()>}, concept.nonDerivedProperties, concept.superProperties:keyword_arg()]
    )
)>
>>

keyword_arg(property) ::= "<field_name(property)>=<field_name(property)>"



==> cml-modules/cml_base/templates/design/cmlc_py.stg


import "/design/cmlc_py/interface.stg"
import "/design/cmlc_py/class.stg"

import "/design/cmlc.stg"
import "/design/py.stg"
import "/design/common.stg"

import "/lang/py.stg"



==> cml-modules/cml_base/templates/design/common/association.stg


association_class(association, class_name_suffix) ::= <<
<modifier("public")><\\>
<structure(
    keyword="class",
    header=association_name(association),
    base_list=true,
    sections = [
        association_singleton(association),
        association_new(association),
        association_init(association),
        association_fields(association),
        association_constructor(association),
        association_link_methods(association),
        association_getters(association)
    ]
)>
>>

association_init(association) ::= <<
<static_operation(
    name="init",
    params=[association_init_param()],
    result_type=void(),
    type_params=[],
```

```
    statements=association_init_statements(association)
)>
>>

association_init_statements(association) ::= <<
<association.associationEnds:association_init_statement(association);separator="\n">
>>

association_fields(association) ::= <<
<association.associationEnds:association_field_decl();separator="\n">
>>

association_link_methods(association) ::= <<
<if(association.oneToMany)><\\>
<association_one_to_many_link_method(association)>

<endif><\\>
<association_link_method(association)><\\>
<if(association.oneToOne)><\\>


<association_link_method_reversed(association)><\\>
<endif>
>>

association_one_to_many_link_method(association) ::= <<
<modifier("synchronized")><\\>
<instance_operation(
    name="link_many",
    params=[association.oneProperty, association.manyProperty],
    result_type=void(),
    statements=association_one_to_many_link_method_statements(association)
)>
>>

association_link_method(association) ::= <<
<modifier("synchronized")><\\>
<instance_operation(
    name="link",
    params=association.propertyTypes,
    result_type=void(),
    statements=association_link_method_statements(association.associationEnds)
)>
>>

association_link_method_reversed(association) ::= <<
<modifier("synchronized")><\\>
<instance_operation(
    name="link",
    params=association.reversedPropertyTypes,
    result_type=void(),
    statements=association_link_method_statements(association.associationEnds)
)>
>>

association_link_method_statements(association_ends) ::= <<
<association_ends:association_link_statement(); separator="\n\n">
>>

association_link_statement(association_end) ::= <<
<if(association_end.associatedProperty.type.sequence)><\\>
<association_link_statement_seq(association_end)><\\>
<else><\\>
<association_link_statement_non_seq(association_end)><\\>
<endif>
>>

association_link_seq(association_end) ::= <<
<association_link_target(association_end)><association_link_seq_suffix()>
>>

association_link_target(association_end) ::= <<
<field_name(association_end.associatedProperty.type)>
>>

association_link_source(association_end) ::= <<
<field_name(association_end.associatedConcept)>
>>

association_link_field(association_end) ::= <<
<field_access(association_end.associatedProperty)>
>>

association_getters(association) ::= <<
<association.associationEnds:association_getter(); separator="\n\n">
>>

association_getter(association_end) ::= <<
<modifier("synchronized")><\\>
<instance_operation(
    name=association_getter_name(association_end.associatedProperty),
    params=[association_end.associatedConcept],
    result_type=association_getter_result_type(association_end),
    statements=association_getter_statements(association_end)
)>
>>

association_getter_result_type(association_end) ::= <<
<if(association_end.associatedProperty.type.sequence)><\\>
<getter_type_sequence(association_end.associatedProperty.type)><\\>
<else><\\>
<getter_type_optional(association_end.associatedProperty.type)><\\>
<endif>
>>

association_getter_statements(association_end) ::= <<
<if(association_end.associatedProperty.type.sequence)><\\>
<association_getter_statements_seq(association_end)><\\>
<else><\\>
<association_getter_statements_non_seq(association_end)><\\>
<endif>
>>

association_link_calls(concept) ::= <<
<if(concept.associationProperties)>

<[concept.associationProperties]:association_link_call();separator="\n"><\\>
<endif>
>>

association_link_call(property) ::= <<
<if(property.type.sequence)><\\>
<association_link_call_seq(property)><\\>
<else><\\>
<association_link_call_non_seq(property)><\\>
<endif>
>>
```

```
association_class_getters(properties) ::= <<
<properties:association_class_getter();separator="\n\n">
>>

association_class_getter(property) ::= <<
<class_getter_annotations(property)><\\>
<modifier("public")><\\>
<instance_operation(
    name=getter_name(property),
    params=[],
    result_type=getter_type(property.type),
    statements=return(association_class_getter_expr(property))
)>
>>



==> cml-modules/cml_base/templates/design/common.stg


import "/design/common/association.stg"


==> cml-modules/cml_base/templates/design/java/association.stg


association_singleton(association) ::= <<
private static final <association_name(association)> singleton = new <association_name(association)>();
>>

association_new(association) ::= ""

association_init_param() ::= "Class\<?> cls"

association_init_statement(association_end, association) ::= <<
if (<type_name(association_end.associatedConcept)>.class.isAssignableFrom(cls))
{
<indent()><type_name(association_end.associatedConcept)><class_name_suffix>.set<type_name(association)>(singleton);
}
>>

association_name(association) ::= <<
<type_name(association)>
>>

association_field_decl(association_end) ::= <<
<if(association_end.associatedProperty.type.single)><\\>
private final Map\<<field_type(association_end.associatedConcept)>, <field_type(association_end.associatedProperty)>\> <field_name(association_end.
<else><\\>
private final Map\<<field_type(association_end.associatedConcept)>, <association_link_seq_type(association_end)>\> <field_name(association_end.asso
<endif>
>>

association_constructor(association) ::= ""

association_link_statement_seq(association_end) ::= <<
final <association_link_seq_type(association_end)> <association_link_seq(association_end)> = <association_link_field(association_end)>.computeIfAbs
if (!<association_link_seq(association_end)>.contains(<association_link_target(association_end)>))
{
<indent()><association_link_seq(association_end)>.add(<association_link_target(association_end)>);
}
>>

association_link_seq_type(association_end) ::= <<
Set\<<type_name(association_end.associatedProperty.type)>\>
>>

association_link_statement_non_seq(association_end) ::= <<
<association_link_field(association_end)>.put(<association_link_source(association_end)>, <association_link_target(association_end)>);
>>

association_link_seq_suffix() ::= "List"

association_getter_name(property) ::= <<
<property.name; format="camel-case">Of
>>

association_getter_statements_seq(association_end) ::= <<
final <association_link_seq_type(association_end)> <association_link_seq(association_end)> = <association_link_field(association_end)>.get(<associa

return (<association_link_seq(association_end)> == null) ? Collections.emptyList() : new ArrayList\<>(<association_link_seq(association_end)>);
>>

association_getter_statements_non_seq(association_end) ::= <<
return Optional.ofNullable(<association_link_field(association_end)>.get(<association_link_source(association_end)>));
>>

class_association_field(association) ::= <<
private static <field_type(association)> <field_name(association)>;
>>

association_setter_name(association) ::= <<
set<association_name(association)>
>>

association_setter_param(association) ::= <<
<association_name(association)> association
>>

association_setter_statements(association) ::= <<
<field_name(association)> = association;
>>

association_static_initializer_statement(association, concept) ::= <<
<association_name(association)>.init(<field_type(concept)>.class);
>>

association_one_to_many_link_method_statements(association) ::= <<
for (<type_name(association.manyProperty.type)> <field_name(association.manyProperty.type)>: <field_name(association.manyProperty)>) link(<field_nar
>>

association_link_call_seq(property) ::= <<
<field_name(property.association)>.linkMany(<actual_self_field()>, <field_name(property)>);
>>

association_link_call_non_seq(property) ::= <<
<field_name(property.association)>.link(<field_name(property)>, <actual_self_field()>);
>>

association_class_getter_expr(property) ::= <<
<field_name(property.association)>.<field_name(property)>Of(<actual_self()>)<\\>
<if(property.type.required)><\\>
.get()<\\>
<endif>
>>
```

```
==> cml-modules/cml_base/templates/design/java.stg


import "/design/java/association.stg"


==> cml-modules/cml_base/templates/design/plain/actual_self.stg

// design/plain

actual_self() ::= "<self()>"

actual_self_field() ::= "<self()>"


==> cml-modules/cml_base/templates/design/plain/class.stg

// design/plain

class_primary_constructor(inherited_properties, super_properties, ancestors) ::= <<
<if(ancestors || concept.nonDerivedProperties)><\\>
<modifier("public")><\\>
<class_constructor(
    [ancestors:optional_param_decl(), super_properties, concept.nonDerivedProperties],
    class_primary_constructor_statements()
)><\\>
<endif>
>>

class_primary_constructor_statements() ::= <<
<super_constructor(inherited_properties)><\\>
<[concept.slotProperties]:field_init();separator="\n"><\\>
<association_link_calls(concept)>
>>



==> cml-modules/cml_base/templates/design/plain/invocation.stg


invocation_concept(concept, args) ::= <<
<new_call(type_name(concept), args:nullable_expr())>
>>



==> cml-modules/cml_base/templates/design/plain.stg


import "/design/plain/actual_self.stg"
import "/design/plain/class.stg"
import "/design/plain/invocation.stg"


==> cml-modules/cml_base/templates/design/poj/class.stg

// constructors/poj

class_secondary_constructor(non_init_properties, super_properties) ::= <<
<if(!interfaces && concept.initProperties)><\\>
<modifier("public")><\\>
<class_constructor(non_init_properties, class_secondary_constructor_statements(concept, super_properties))><\\>
<endif>
>>

class_secondary_constructor_statements(concept, super_properties) ::= <<
<this_constructor_call([super_properties, concept.nonDerivedProperties])>
>>



==> cml-modules/cml_base/templates/design/poj.stg


import "/design/poj/class.stg"

import "/design/plain.stg"
import "/design/java.stg"
import "/design/common.stg"

import "/lang/java.stg"



==> cml-modules/cml_base/templates/design/pop.stg


import "/design/plain.stg"
import "/design/py.stg"
import "/design/common.stg"

import "/lang/py.stg"


==> cml-modules/cml_base/templates/design/py/association.stg


association_link_methods(association) ::= <<
<if(association.oneToMany)><\\>
<association_one_to_many_link_method(association)>

<endif><\\>
<association_link_method(association)>
>>

association_singleton(association) ::= <<
_singleton = None
>>

association_new(association) ::= <<
def __new__(cls) -> '<association_name(association)>':
<indent()>if cls._singleton is None:
<indent()><indent()>cls._singleton = super(<association_name(association)>, cls).__new__(cls)
<indent()>return cls._singleton
>>

association_name(association) ::= <<
_<type_name(association)>
>>

association_init(association) ::= ""

association_field_decl(association_ends) ::= ""

association_constructor(association) ::= <<
```

```
<instance_operation(
    name=constructor_name(type_name(association)),
    params=[],
    result_type=[],
    statements=association_constructor_statements(association)
)>
>>

association_constructor_statements(association) ::= <<
<association.associationEnds:association_field_init(); separator="\n">
>>

association_field_init(association_end) ::= <<
<field_access(association_end.associatedProperty)> = {}  # type: Dict[<field_type(association_end.associatedConcept)>, <field_type(association_end.
>>

association_link_statement_seq(association_end) ::= <<
if <association_link_source(association_end)> in <association_link_field(association_end)>:
<indent()><association_link_seq(association_end)> = <association_link_field(association_end)>[<association_link_source(association_end)>]
else:
<indent()><association_link_seq(association_end)> = [<association_link_target(association_end)>]
if not (<association_link_target(association_end)> in <association_link_seq(association_end)>):
<indent()><association_link_seq(association_end)>.append(<association_link_target(association_end)>)
<association_link_field(association_end)>[<association_link_source(association_end)>] = <association_link_seq(association_end)>
>>

association_link_statement_non_seq(association_end) ::= <<
<association_link_field(association_end)>[<association_link_source(association_end)>] = <association_link_target(association_end)>
>>

association_link_seq_suffix() ::= "_list"

association_getter_name(property) ::= <<
<property.name; format="underscore-case">_of
>>

association_getter_statements_non_seq(association_end) ::= <<
if <association_link_source(association_end)> in <association_link_field(association_end)>:
<indent()>return <association_link_field(association_end)>[<association_link_source(association_end)>]
else:
<indent()>return None
>>

association_getter_statements_seq(association_end) ::= <<
if <association_link_source(association_end)> in <association_link_field(association_end)>:
<indent()><association_link_seq(association_end)> = <association_link_field(association_end)>[<association_link_source(association_end)>]
else:
<indent()><association_link_seq(association_end)> = []
return list(<association_link_seq(association_end)>)
>>

class_association_field(association) ::= <<
_<field_name(association)> = <association_name(association)>()
>>

class_association_setter(association) ::= ""

association_one_to_many_link_method_statements(association) ::= <<
for <field_name(association.manyProperty.type)> in <field_name(association.manyProperty)>: self.link(<field_name(association.oneProperty.type)>=<fi
>>

association_link_call_seq(property) ::= <<
self._<field_name(property.association)>.link_many(<actual_self_field()>, <field_name(property)>)
>>

association_link_call_non_seq(property) ::= <<
self._<field_name(property.association)>.link(<field_name(property.type)>=<field_name(property)>, <field_name(property.concept)>=<actual_self_field
>>

association_class_getter_expr(property) ::= <<
self._<field_name(property.association)>.<field_name(property)>_of(<actual_self_field()>)
>>



==> cml-modules/cml_base/templates/design/py.stg


import "/design/py/association.stg"


==> cml-modules/cml_base/templates/lang/common/block.stg


block_header(keyword, content) ::= <<
<({<keyword>:keyword})()><content><block_header_terminator()>
>>

block_header_terminator() ::= ""


==> cml-modules/cml_base/templates/lang/common/call.stg


import "/lang/common/literal.stg"
import "/lang/common/type.stg"
import "/lang/common/statement.stg"

call(name, args) ::= <<
<name>(<call_arg_list(args)>)
>>

call_arg_list(args) ::= <<
<args:call_arg(); separator=", ">
>>

call_arg(arg) ::= <<
<if(arg.name)><\\>
<field_name(arg)><\\>
<else><\\>
<arg><\\>
<endif>
>>

new_call(class_name, args) ::= <<
<call({<new_keyword()><class_name>}, args)>
>>

init_call(name, args) ::= <<
<name>(<args:field_or_init_value(); separator=", ">)
>>

super_constructor_call(properties) ::= <<
<super_constructor_name()>(<properties:field_or_default_value(); separator=", ")><statement_terminator()>
>>
```

```
this_constructor_call(properties) ::= <<
<this_constructor_name()>(<properties:field_or_init_value(); separator=", ">)<statement_terminator()>
>>


==> cml-modules/cml_base/templates/lang/common/class.stg


import "/lang/common/generic.stg"
import "/lang/common/getter.stg"
import "/lang/common/to_string.stg"
import "/lang/common/interface.stg"

class(concept) ::= <<
<class7(
     concept=concept,
     class_name_suffix="",
     abstract=concept.abstraction,
     ancestors=concept.ancestors,
     interfaces=[],
     field_ancestors=[],
     delegated_properties=[],
     super_properties=concept.superProperties,
     inherited_properties=concept.redefinedInheritedConcreteProperties
)>
>>

class2(concept, class_name_suffix) ::= <<
<class7(
     concept=concept,
     class_name_suffix=class_name_suffix,
     abstract=false,
     ancestors=[],
     interfaces=[concept],
     field_ancestors=concept.allAncestors,
     delegated_properties=concept.inheritedNonRedefinedProperties,
     super_properties=[],
     inherited_properties=[]
)>
>>

class7(
     concept, class_name_suffix,
     abstract,
     ancestors, interfaces,
     field_ancestors,
     delegated_properties, super_properties, inherited_properties
) ::= <<
<class_abstract()><\\>
<structure(
     keyword="class",
     header=class_header(),
     base_list=true,
     sections = [
          class_association_fields(concept),
          class_self_field(concept),
          class_fields(field_ancestors, concept.slotProperties),
          class_secondary_constructor(concept.nonInitProperties, super_properties),
          class_primary_constructor(inherited_properties, super_properties, field_ancestors),
          class_extension_constructor(concept),
          class_getters(),
          class_to_string(),
          class_association_setters(concept),
          class_static_initializer(concept)
     ]
)>
>>

class_abstract() ::= <<
<if(abstract)><abstract_keyword()><endif>
>>

class_header() ::= <<
<class_name()><ancestor_list(abstract, ancestors, interfaces)>
>>

class_name() ::= <<
<type_name(concept)><class_name_suffix>
>>

class_self_field(concept) ::= ""

class_fields(field_ancestors, properties) ::= <<
<ancestor_fields(field_ancestors)><\\>
<emptyLineIf2(field_ancestors, properties)><\\>
<property_fields(properties)>
>>

class_getters() ::= <<
<line_list([
     field_getters(concept.slotProperties),
     association_class_getters(concept.associationProperties),
     derived_getters(concept.derivedProperties),
     delegated_getters(delegated_properties)
])>
>>

ancestor_fields(ancestors) ::= <<
<ancestors:field_decl();separator="\n">
>>

property_fields(properties) ::= <<
<properties:field_decl();separator="\n">
>>

field_getters(properties) ::= <<
<properties:field_getter();separator="\n\n">
>>

derived_getters(properties) ::= <<
<properties:derived_getter();separator="\n\n">
>>

delegated_getters(properties) ::= <<
<properties:delegate_getter();separator="\n\n">
>>

super_constructor(inherited_properties) ::= <<
<if(inherited_properties)><\\>
<super_constructor_call(inherited_properties)>
<endif>
>>

class_secondary_constructor(non_init_properties, super_properties) ::= ""

class_extension_constructor(concept) ::= ""
```

```
class_to_string() ::= <<
<to_string(concept)>
>>

class_constructor(params, statements) ::= <<
<instance_operation(
    name=constructor_name(class_name()),
    params=params,
    result_type=[],
    statements=statements
)>
>>

field_getter(property) ::= <<
<class_getter_annotations(property)><\\>
<modifier("public")><\\>
<instance_operation(
    name=getter_name(property),
    params=[],
    result_type=getter_type(property.type),
    statements=return(getter_field_value(property))
)>
>>

derived_getter(property) ::= <<
<if(property.value)><\\>
<derived_concrete_getter(property, derived_getter_expression(property.value))><\\>
<elseif(abstract)><\\>
<derived_abstract_getter(property)><\\>
<else><\\>
<derived_concrete_getter(property, field_or_default_value(property))><\\>
<endif>
>>

derived_getter_expression(expr) ::= <<
<if(expr.type.sequence)><\\>
<to_list(expression(expr))><\\>
<else><\\>
<expression(expr)><\\>
<endif>
>>

derived_concrete_getter(property, expression) ::= <<
<class_getter_annotations(property)><\\>
<modifier("public")><\\>
<instance_operation(
    name=getter_name(property),
    params=[],
    result_type=getter_type(property.type),
    statements=return(expression)
)>
>>

derived_abstract_getter(property) ::= <<
<modifier("public")><\\>
<abstract_keyword()><\\>
<interface_getter(property)>
>>

delegate_getter(property) ::= <<
<class_getter_annotations(property)><\\>
<modifier("public")><\\>
<instance_operation(
    name={<getter_name(property)>},
    params=[],
    result_type=getter_type(property.type),
    statements=return({<field_access(property.parent)>.<getter_call(property)>})
)>
>>

class_association_fields(concept) ::= <<
<concept.associations:class_association_field();separator="\n">
>>

class_association_field(association) ::= <<
<concept.associations:class_association_field();separator="\n\n">
>>

class_association_setters(concept) ::= <<
<concept.associations:class_association_setter();separator="\n\n">
>>

class_association_setter(association) ::= <<
<static_operation(
    name=association_setter_name(association),
    params=[association_setter_param(association)],
    result_type=void(),
    type_params=[],
    statements=association_setter_statements(association)
)>
>>

class_static_initializer(concept) ::= <<
<if(concept.associations)><\\>
<static_initializer(class_static_initializer_statements(concept))><\\>
<endif>
>>

class_static_initializer_statements(concept) ::= <<
<concept.associations:association_static_initializer_statement(concept);separator="\n">
>>

==> cml-modules/cml_base/templates/lang/common/expression.stg


expression(expr) ::= <<
<({expression_<expr.kind>})(expr)>
>>

expression_path(path) ::= <<
<path_expr(path)>
>>

expression_unary(expr) ::= <<
<if(expr.operation)><\\>
(<({operation_<expr.operation>})(expr.subExpr)>)<\\>
<else><\\>
<expression_operator(expr)> <expression(expr.subExpr)>
<endif>
>>

operation_unary_add(expr) ::= "+<expression(expr)>"
operation_unary_sub(expr) ::= "-<expression(expr)>"

expression_arithmetic(expr) ::= <<
<expression_infix(expr)>
>>
```

```
expression_logical(expr) ::= <<
<expression_infix(expr)>
>>

expression_relational(expr) ::= <<
<expression_infix(expr)>
>>

expression_referential(expr) ::= <<
<expression_infix(expr)>
>>

expression_string_concat(expr) ::= <<
<expression_infix(expr)>
>>

expression_infix(expr) ::= <<
<if(expr.operation)><\\>
(<({operation_<expr.operation>})(expr.left, expr.right)>)<\\>
<else><\\>
(<nullable_expr(expr.left)> <expr.operator> <nullable_expr(expr.right)>)<\\>
<endif>
>>

operation_ref_eq(left, right) ::= <<
<({operation_ref_eq_<left.type.kind>_<right.type.kind>})(left, right)>
>>

operation_ref_eq_required_optional(left, right) ::= "<operation_ref_eq_required_required(left, right)>"
operation_ref_eq_optional_required(left, right) ::= "<operation_ref_eq_required_optional(right, left)>"
operation_ref_eq_optional_optional(left, right) ::= "<operation_ref_eq_optional_required(left, right)>"

operation_not_ref_eq(left, right) ::= <<
<({operation_not_ref_eq_<left.type.kind>_<right.type.kind>})(left, right)>
>>

operation_not_ref_eq_required_optional(left, right) ::= "<operation_not_ref_eq_required_required(left, right)>"
operation_not_ref_eq_optional_required(left, right) ::= "<operation_not_ref_eq_required_optional(right, left)>"
operation_not_ref_eq_optional_optional(left, right) ::= "<operation_not_ref_eq_optional_required(left, right)>"

expression_literal(literal) ::= <<
<({literal_expr_<literal.type.name; format="lower-case">})(literal.text)>
>>

expression_invocation(invocation) ::= <<
<if(invocation.function)><\\>
<if(invocation.function.expression)><\\>
<call(operation_name(invocation.name), invocation.function.parameters:expression_arg(invocation.namedArguments))><\\>
<elseif(invocation.namedArguments.seq)><\\>
<({invocation_<invocation.namedArguments.seq.type.kind>_<invocation.name>})(invocation.namedArguments)><\\>
<else><\\>
<({invocation_<invocation.name>})(invocation.namedArguments)><\\>
<endif><\\>
<elseif(invocation.concept)><\\>
<invocation_concept(invocation.concept, invocation.arguments)><\\>
<else><\\>
<endif>
>>

expression_arg(param, args) ::= <<
<({arg_<args.(param.name).type.kind>_to_<param.type.kind>})(args.(param.name))>
>>

arg_required_to_required(arg) ::= <<
<expression(arg)>
>>

arg_required_to_optional(arg) ::= <<
Optional.of(<expression(arg)>)
>>

arg_required_to_sequence(arg) ::= <<
Seq.of(<expression(arg)>).toList()
>>

arg_optional_to_optional(arg) ::= <<
<expression(arg)>.orElse(<field_default_value(expr.type)>)
>>

arg_optional_to_sequence(arg) ::= <<
seq(<expression(arg)>).toList()
>>

arg_sequence_to_sequence(arg) ::= <<
seq(<expression(arg)>).toList()
>>

expression_type_check(type_check) ::= <<
<({type_check_<type_check.operator>})(type_check.expr, type_check.checkedType)>
>>

expression_type_cast(type_cast) ::= <<
<({type_cast_<type_cast.operator>_from_<type_cast.expr.type.kind>_to_<type_cast.castType.kind>})(type_cast.expr, type_cast.castType)>
>>


==> cml-modules/cml_base/templates/lang/common/field.stg


import "/lang/common/type.stg"

field_type(named_element) ::= <<
<if(named_element.type)><\\>
<({field_type_<named_element.type.kind>})(named_element.type)><\\>
<else><\\>
<type_name(named_element)><\\>
<endif>
>>

field_type_required(type) ::= <<
<type_name(type)>
>>

field_or_default_value(property) ::= <<
<if(property.derived)><\\>
<field_default_value(property.type)><\\>
<else><\\>
<field_name(property)><\\>
<endif>
>>

field_or_init_value(property) ::= <<
<if(property.derived)><\\>
<field_default_value(property.type)><\\>
<elseif(property.value)><\\>
<literal_expr(property.value)><\\>
```

```
<else><\\>
<field_name(property)><\\>
<endif>
>>

field_default_value(type) ::= <<
<if(type.numeric)><integral_zero()><\\>
<elseif(type.float)><float_zero()><\\>
<elseif(type.boolean)><boolean_false()><\\>
<else><null_value()><endif>
>>

field_access(element) ::= <<
<field_access_prefix><if(element.name)><field_name(element)><else><element><endif>
>>


==> cml-modules/cml_base/templates/lang/common/generic.stg


newLineIf(cond) ::= <<
<if(cond)><\n><endif>
>>

emptyLineIf(cond1) ::= <<
<if(cond1)><\n><\n><endif>
>>

newLineIf2(cond1, cond2) ::= <<
<if(cond1 && cond2)><\n><endif>
>>

emptyLineIf2(cond1, cond2) ::= <<
<if(cond1 && cond2)><\n><\n><endif>
>>

newLineIfEither(cond1, cond2) ::= <<
<if(cond1 || cond2)><\n><endif>
>>

emptyLineIfEither(cond1, cond2) ::= <<
<if(cond1 || cond2)><\n><\n><endif>
>>

commaIf(cond) ::= <<
<if(cond)>, <endif>
>>

commaIf2(cond1, cond2) ::= <<
<if(cond1 && cond2)>, <endif>
>>

commaIfEither(cond1, cond2) ::= <<
<if(cond1 || cond2)>, <endif>
>>

line_list(list) ::= <<
<if(list)><\\>
<trim_template(all_lines(list))><\\>
<endif>
>>

trim_template(template) ::= <<
<trim((template))>
>>

all_lines(list) ::= <<
<first_line(list)><line_list(rest(list))>
>>

first_line(list) ::= <<
<first(list)><\n><\n>
>>



==> cml-modules/cml_base/templates/lang/common/getter.stg


import "/lang/common/field.stg"

getter_type(type) ::= <<
<({getter_type_<type.kind>})(type)>
>>

getter_field_value(property) ::= <<
<({getter_field_value_<property.type.kind>})(property)>
>>


==> cml-modules/cml_base/templates/lang/common/indentation.stg


indent() ::= "    "

==> cml-modules/cml_base/templates/lang/common/interface.stg


import "/lang/common/generic.stg"
import "/lang/common/operation.stg"
import "/lang/common/getter.stg"
import "/lang/common/call.stg"

interface(concept, class_name_suffix) ::= <<
<structure(
    keyword="interface",
    header=interface_header(concept),
    base_list=concept.allProperties,
    sections = [
        interface_getters(concept),
        interface_secondary_create_method(concept),
        interface_primary_create_method(concept),
        interface_extend_method()
    ]
)>
>>

interface_header(concept) ::= <<
<type_name(concept)><ancestor_list(true, concept.ancestors, [])>
>>

interface_getters(concept) ::= <<
<concept.properties:interface_getter();separator="\n\n">
>>

interface_secondary_create_method(concept) ::= ""
```

```
interface_primary_create_method(concept) ::= <<
<if(!concept.abstraction && class_name_suffix)><\\>
<static_operation(
     name=interface_create_name(interface_name()),
     params=[concept.superProperties, concept.nonDerivedProperties],
     result_type=interface_name(),
     type_params=[],
    statements=interface_primary_create_method_statements(concept)
)><\\>
<endif>
>>

interface_primary_create_method_statements(concept) ::= ""

interface_extend_method() ::= <<
<if(class_name_suffix)><\\>
<interface_extend_method_with_params_args(
     [actual_self_decl(concept), concept.allAncestors, concept.nonDerivedProperties],
     [actual_self(), concept.allAncestors, concept.nonDerivedProperties]
)><\\>
<endif>
>>

interface_extend_method_with_params_args(params, args) ::= <<
<static_operation(
     name=interface_extend_name(interface_name()),
     params=params,
     result_type=interface_name(),
     type_params=[],
     statements=return(new_call(interface_impl_name(), args))
)>
>>

interface_create_name(name) ::= <<
<operation_name(({create<name>}))>
>>

interface_extend_name(name) ::= <<
<operation_name(({extend<name>}))>
>>

interface_getter(property) ::= <<
<interface_getter_annotations(property)><\\>
<interface_operation(
     name=getter_name(property),
     params=[],
     result_type=getter_type(property.type)
)>
>>

interface_operation(name, params, result_type) ::= <<
<instance_operation_header(name, params, result_type)><\\>
<statement_terminator()><\\>
<interface_operation_body()>
>>

interface_operation_body() ::= ""

interface_name() ::= <<
<type_name(concept)>
>>

interface_impl_name() ::= <<
<type_name(concept)><class_name_suffix>
>>

==> cml-modules/cml_base/templates/lang/common/keyword.stg


modifier(m) ::= ""

abstract_keyword() ::= ""

new_keyword() ::= ""

==> cml-modules/cml_base/templates/lang/common/lambda.stg


lambda ::= [
     "parameters": [],
     "innerExpression": false
]

lambda_var ::= [
     "name": "item"
]

lambda_new_var_name() ::= <<
<lambda_var.name; format="inc-indexed">
>>

lambda_var_name() ::= <<
<lambda_var.name; format="indexed">
>>


==> cml-modules/cml_base/templates/lang/common/literal.stg


literal_expr(literal) ::= <<
<({literal_expr_<literal.type.name; format="underscore-case">})(literal.text)>
>>


==> cml-modules/cml_base/templates/lang/common/operation.stg


instance_operation(name, params, result_type, statements) ::= <<
<instance_operation_header(name, params, result_type)><\\>
<block(statements)>
>>

instance_operation_header(name, params, result_type) ::= <<
<block_header("operation", operation_header_content(true, operation_name(name), params, result_type))>
>>

static_operation(name, params, result_type, type_params, statements) ::= <<
<static_operation_header(name, params, result_type, type_params)><\\>
<block(statements)>
>>

static_operation_header(name, params, result_type, type_params) ::= <<
<static_operation_modifier()><\\>
<if(type_params)>\<<type_params:type_name();separator=", ">\><\ ><endif><\\>
<block_header("operation", operation_header_content(false, operation_name(name), params, result_type))>
>>
```

```
static_initializer(statements) ::= <<
static<block(statements)>
>>

optional_param_decl(param) ::= <<
<param_decl(field_name(param),field_type_optional(param))>
>>
```

==> cml-modules/cml_base/templates/lang/common/statement.stg

```
statement_terminator() ::= ""
```

==> cml-modules/cml_base/templates/lang/common/structure.stg

```
import "/lang/common/block.stg"

structure(keyword, header, base_list, sections) ::= <<
<structure_(field_access_prefix=field_access_prefix(), ...)>
>>

structure_(field_access_prefix, keyword, header, base_list, sections) ::= <<
<block_header(keyword, header)><\\>
<if(base_list)><structure_preamble()><block(line_list(sections))><else><block_empty()><endif>
>>

structure_preamble() ::= ""
```

==> cml-modules/cml_base/templates/lang/common/to_string.stg

```
to_string(concept) ::= <<
<modifier("public")><\\>
<instance_operation(
    name=to_string_name(),
    params=[],
    result_type=primitive_type_name("String"),
    statements=return(to_string_expr(concept))
)>
>>

to_string_fields(properties, sep) ::= <<
<properties:to_string_field();separator={<sep>}>
>>

to_string_field_value(property) ::= <<
<({to_string_field_value_<property.type.kind>})(property)>
>>
```

==> cml-modules/cml_base/templates/lang/common/type.stg

```
type_name(named_element) ::= <<
<prefix_type_name(named_element, "")>
>>

boxed_type_name(named_element) ::= <<
<prefix_type_name(named_element, "boxed_")>
>>

prefix_type_name(named_element, prefix) ::= <<
<if(named_element.primitive)><\\>
<prefix_primitive_type_name(named_element.name, prefix)><\\>
<elseif(named_element.name)><\\>
<class_type_name(named_element.name)><\\>
<else><\\>
<class_type_name(named_element)><\\>
<endif>
>>

primitive_type_name(element_name) ::= <<
<prefix_primitive_type_name(element_name, "")>
>>

boxed_primitive_type_name(element_name) ::= <<
<prefix_primitive_type_name(element_name, "boxed_")>
>>

prefix_primitive_type_name(element_name, prefix) ::= <<
<(primitive_type_template_name(element_name, prefix))()>
>>

primitive_type_template_name(element_name, prefix) ::= <<
<prefix>primitive_type_<element_name; format="underscore-case">
>>

boxed_primitive_type_boolean() ::= "<primitive_type_boolean()>"
boxed_primitive_type_integer() ::= "<primitive_type_integer()>"
boxed_primitive_type_decimal() ::= "<primitive_type_decimal()>"
boxed_primitive_type_string() ::= "<primitive_type_string()>"
boxed_primitive_type_regex() ::= "<primitive_type_regex()>"
boxed_primitive_type_byte() ::= "<primitive_type_byte()>"
boxed_primitive_type_short() ::= "<primitive_type_short()>"
boxed_primitive_type_long() ::= "<primitive_type_long()>"
boxed_primitive_type_float() ::= "<primitive_type_float()>"
boxed_primitive_type_double() ::= "<primitive_type_double()>"
boxed_primitive_type_char() ::= "<primitive_type_char()>"
```

==> cml-modules/cml_base/templates/lang/common.stg

```
import "/lang/common/generic.stg"
import "/lang/common/indentation.stg"
import "/lang/common/block.stg"
import "/lang/common/keyword.stg"
import "/lang/common/lambda.stg"
import "/lang/common/literal.stg"
import "/lang/common/type.stg"
import "/lang/common/field.stg"
import "/lang/common/operation.stg"
import "/lang/common/getter.stg"
import "/lang/common/to_string.stg"
import "/lang/common/structure.stg"
import "/lang/common/interface.stg"
import "/lang/common/class.stg"
import "/lang/common/statement.stg"
import "/lang/common/call.stg"
import "/lang/common/expression.stg"
```

```
==> cml-modules/cml_base/templates/lang/java/expression.stg


nullable_expr(expr) ::= <<
<if(expr.type.nothing)><\\>
<null_value()><\\>
<elseif(expr.type.optional)><\\>
<expression(expr)>.orElse(<field_default_value(expr.type)>)<\\>
<else><\\>
<expression(expr)><\\>
<endif>
>>

expression_required_to_required(arg) ::= <<
<expression(arg)>
>>

expression_required_to_optional(arg) ::= <<
Optional.of(<expression(arg)>)
>>

expression_required_to_sequence(arg) ::= <<
Seq.of(<expression(arg)>).toList()
>>

expression_optional_to_optional(arg) ::= <<
<expression(arg)>
>>

expression_optional_to_sequence(arg) ::= <<
seq(<expression(arg)>).toList()
>>

expression_sequence_to_sequence(arg) ::= <<
seq(<expression(arg)>).toList()
>>

expression_let(let) ::= <<
<expression_let_("", let)>
>>

expression_let_(field_access_prefix, let) ::= <<
new Supplier\<<boxed_type_name(let.type)>\>() {
<indent()>public <boxed_type_name(let.type)> get()
<indent()>{
<indent()><indent()>final <field_type(let.variableExpr)> <let.variable; format="camel-case"> = <expression(let.variableExpr)>;
<indent()><indent()><return(expression(let.resultExpr))>
<indent()>}
}.get()
>>




==> cml-modules/cml_base/templates/lang/java/imports.stg


common_imports(task, model) ::= <<
import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;
<if(model.definedFunctions)><\\>

<model.modules:functions_import(task); separator="\n"><\\>
<endif>
>>

functions_import(module, task) ::= <<
<if(module.definedFunctions)>import static <task.packageName>.<functions_class_name(module)>.*;<endif>
>>

functions_class_name(module) ::= <<
<type_name(module)>Functions
>>

==> cml-modules/cml_base/templates/lang/java/invocation.stg


invocation_sequence_empty(args) ::= <<
<expression(args.seq)>.isEmpty()
>>

invocation_optional_empty(args) ::= <<
!<expression(args.seq)>.isPresent()
>>

invocation_required_empty(args) ::= <<
<expression(args.seq)> == null
>>

invocation_sequence_present(args) ::= <<
!<expression(args.seq)>.isEmpty()
>>

invocation_optional_present(args) ::= <<
<expression(args.seq)>.isPresent()
>>

invocation_required_present(args) ::= <<
<expression(args.seq)> != null
>>

invocation_sequence_first(args) ::= <<
seq(<expression(args.seq)>).findFirst()
>>

invocation_optional_first(args) ::= <<
seq(<expression(args.seq)>).findFirst()
>>

invocation_required_first(args) ::= <<
Seq.of(<expression(args.seq)>).findFirst()
>>
```

```
invocation_sequence_last(args) ::= <<
seq(<expression(args.seq)>).findLast()
>>

invocation_optional_last(args) ::= <<
seq(<expression(args.seq)>).findLast()
>>

invocation_required_last(args) ::= <<
Seq.of(<expression(args.seq)>).findLast()
>>

invocation_sequence_exists(args) ::= <<
seq(<expression(args.seq)>).anyMatch(<expression(args.expr)>)
>>

invocation_optional_exists(args) ::= <<
seq(<expression(args.seq)>).anyMatch(<expression(args.expr)>)
>>

invocation_required_exists(args) ::= <<
Seq.of(<expression(args.seq)>).anyMatch(<expression(args.expr)>)
>>

invocation_sequence_all(args) ::= <<
seq(<expression(args.seq)>).allMatch(<expression(args.expr)>)
>>

invocation_optional_all(args) ::= <<
seq(<expression(args.seq)>).allMatch(<expression(args.expr)>)
>>

invocation_required_all(args) ::= <<
Seq.of(<expression(args.seq)>).allMatch(<expression(args.expr)>)
>>

invocation_sequence_none(args) ::= <<
seq(<expression(args.seq)>).noneMatch(<expression(args.expr)>)
>>

invocation_optional_none(args) ::= <<
seq(<expression(args.seq)>).noneMatch(<expression(args.expr)>)
>>

invocation_required_none(args) ::= <<
Seq.of(<expression(args.seq)>).noneMatch(<expression(args.expr)>)
>>

invocation_sequence_includes(args) ::= <<
seq(<expression(args.seq)>).contains(<expression(args.expr.innerExpression)>)
>>

invocation_optional_includes(args) ::= <<
seq(<expression(args.seq)>).contains(<expression(args.expr.innerExpression)>)
>>

invocation_required_includes(args) ::= <<
Seq.of(<expression(args.seq)>).contains(<expression(args.expr.innerExpression)>)
>>

invocation_sequence_excludes(args) ::= <<
!seq(<expression(args.seq)>).contains(<expression(args.expr.innerExpression)>)
>>

invocation_optional_excludes(args) ::= <<
!seq(<expression(args.seq)>).contains(<expression(args.expr.innerExpression)>)
>>

invocation_required_excludes(args) ::= <<
!Seq.of(<expression(args.seq)>).contains(<expression(args.expr.innerExpression)>)
>>

invocation_sequence_select(args) ::= <<
seq(<expression(args.seq)>).filter(<expression(args.expr)>)
>>

invocation_optional_select(args) ::= <<
seq(<expression(args.seq)>).filter(<expression(args.expr)>)
>>

invocation_required_select(args) ::= <<
Seq.of(<expression(args.seq)>).filter(<expression(args.expr)>)
>>

invocation_sequence_reject(args) ::= <<
seq(<expression(args.seq)>).removeAll(seq(<expression(args.seq)>).filter(<expression(args.expr)>))
>>

invocation_optional_reject(args) ::= <<
seq(<expression(args.seq)>).removeAll(seq(<expression(args.seq)>).filter(<expression(args.expr)>))
>>

invocation_required_reject(args) ::= <<
Seq.of(<expression(args.seq)>).removeAll(Seq.of(<expression(args.seq)>).filter(<expression(args.expr)>))
>>

invocation_sequence_collect(args) ::= <<
seq(<expression(args.seq)>).map(<expression(args.expr)>)
>>

invocation_optional_collect(args) ::= <<
seq(<expression(args.seq)>).map(<expression(args.expr)>)
>>

invocation_required_collect(args) ::= <<
Seq.of(<expression(args.seq)>).map(<expression(args.expr)>)
>>

invocation_sequence_sort(args) ::= <<
seq(<expression(args.seq)>).sorted(<expression(args.expr)>)
>>

invocation_compare(args) ::= <<
(<expression(args.expr1)> \< <expression(args.expr2)>) ? -1 : ((<expression(args.expr2)> \< <expression(args.expr1)>) ? +1 : 0)
>>

invocation_sequence_reverse(args) ::= <<
seq(<expression(args.seq)>).reverse()
>>

invocation_concat(args) ::= <<
concat(<arg_to_seq(args.seq1)>, <arg_to_seq(args.seq2)>)
>>

invocation_sequence_count(args) ::= <<
seq(<expression(args.seq)>).count()
```

```
>>

invocation_sequence_distinct(args) ::= <<
seq(<expression(args.seq)>).distinct()
>>

invocation_cross_join(args) ::= <<
crossJoin(<expression(args.seq1)>, <expression(args.seq2)>)
>>

arg_to_seq(arg) ::= <<
<({expression_<arg.type.kind>_to_sequence})(arg)>
>>



==> cml-modules/cml_base/templates/lang/java/lambda.stg


expression_lambda(lambda) ::= <<
<if(lambda.parameters)>(<lambda.parameters; separator=", ", format="camel-case">)<else><lambda_new_var_name()><endif><\\>
<\ >-><\ ><expression(lambda.innerExpression)>
>>

==> cml-modules/cml_base/templates/lang/java/type.stg


type_check_is(expr, type) ::= <<
<expression(expr)> instanceof <type_name(type)>
>>

type_check_isnt(expr, type) ::= <<
!(<expression(expr)> instanceof <type_name(type)>)
>>

type_cast_asb_from_required_to_required(expr, type) ::= <<
(<type_name(type)>)<expression(expr)>
>>

type_cast_asb_from_required_to_optional(expr, type) ::= <<
<type_cast_asb_from_required_to_sequence(expr, type)>.findFirst()
>>

type_cast_asb_from_required_to_sequence(expr, type) ::= <<
<from_required(expr)>.<cast(type)>
>>

type_cast_asb_from_optional_to_required(expr, type) ::= <<
<type_cast_asb_from_optional_to_sequence(expr, type)>.findFirst().get()
>>

type_cast_asb_from_optional_to_optional(expr, type) ::= <<
<type_cast_asb_from_optional_to_sequence(expr, type)>.findFirst()
>>

type_cast_asb_from_optional_to_sequence(expr, type) ::= <<
<from_optional(expr)>.<cast(type)>
>>

type_cast_asb_from_sequence_to_sequence(expr, type) ::= <<
<from_sequence(expr)>.<cast(type)>
>>

type_cast_asq_from_required_to_optional(expr, type) ::= <<
<type_cast_asq_from_required_to_sequence(expr, type)>.findFirst()
>>

type_cast_asq_from_required_to_sequence(expr, type) ::= <<
<from_required(expr)>.<of_type(type)>
>>

type_cast_asq_from_optional_to_optional(expr, type) ::= <<
<type_cast_asq_from_optional_to_sequence(expr, type)>.findFirst()
>>

type_cast_asq_from_optional_to_sequence(expr, type) ::= <<
<from_optional(expr)>.<of_type(type)>
>>

type_cast_asq_from_sequence_to_sequence(expr, type) ::= <<
<from_sequence(expr)>.<of_type(type)>
>>

from_required(expr) ::= <<
Seq.of(<expression(expr)>)
>>

from_optional(expr) ::= <<
seq(<expression(expr)>)
>>

from_sequence(expr) ::= <<
seq(<expression(expr)>)
>>

of_type(type) ::= <<
ofType(<type_name(type)>.class)
>>

cast(type) ::= <<
cast(<type_name(type)>.class)
>>


==> cml-modules/cml_base/templates/lang/java.stg


import "/lang/java/expression.stg"
import "/lang/java/invocation.stg"
import "/lang/java/lambda.stg"
import "/lang/java/type.stg"
import "/lang/java/imports.stg"
import "/lang/common.stg"

// Language

lang ::= [
    "fields": true
]

// Keywords

void() ::= "void"

self() ::= "this"

// Modifiers
```

```
modifier(m) ::= <<
<m><\ >
>>

abstract_keyword() ::= <<
abstract<\ >
>>

new_keyword() ::= <<
new<\ >
>>

// Blocks

block(statements) ::= <<
<\n>{<\n><indent()><statements; anchor><\n>}
>>

block_empty() ::= <<
<\ >{}
>>

// Statement

statement_terminator() ::= ";"

// Call

super_constructor_name() ::= "super"

this_constructor_name() ::= "this"

// Strings

to_string_name() ::= "toString"

to_string_expr(concept) ::= <<
new StringBuilder(<type_name(concept)>.class.getSimpleName())
<indent()><to_string_appends(concept.printableProperties); anchor>
>>

to_string_appends(properties) ::= <<
.append('(')
<to_string_fields(properties, ".append(\", \")\n")><\\>
<newLineIf(properties)><\\>
.append(')')
.toString()
>>

to_string_field(property) ::= <<
.append("<field_name(property)>=")<to_string_field_value(property)>
>>

to_string_field_value_required(property) ::= <<
.append(String.format("\"%s\"", <to_string_getter_call(property)>))
>>

to_string_field_value_optional(property) ::= <<
<if(property.type.boolean)><\\>
<to_string_field_value_required(property)><\\>
<else><\\>
.append(<to_string_getter_call(property)>.isPresent() ? String.format("\"%s\"", <to_string_getter_call(property)>) : "not present")<\\>
<endif>
>>

to_string_field_value_sequence(property) ::= <<
.append(<to_string_getter_call(property)>)
>>

to_string_getter_call(property) ::= <<
<actual_self_field()>.<getter_call(property)>
>>

// Operations

operation_keyword() ::= ""

operation_name(name) ::= <<
<name; format="camel-case">
>>

static_operation_modifier() ::= "static "

operation_header_content(instance, name, params, result_type) ::= <<
<if(result_type)><result_type><\ ><endif><\\>
<name>(<params:var_decl(); separator=", ">)
>>

var_decl(object) ::= <<
<if(object.name || object.type)><\\>
<field_type(object)> <field_name(object)><\\>
<else><\\>
<object><\\>
<endif>
>>

local_var_decl(typed_element) ::= <<
<var_decl(typed_element)>
>>

param_decl(name, type) ::= "<type> <name>"

return(expression) ::= <<
return <expression; anchor>;
>>

// Fields

field_init(named_element) ::= <<
<field_access(named_element)> = <field_name(named_element)>;
>>

field_name(named_element) ::= <<
<named_element.name; format="camel-case">
>>

field_access_prefix() ::= "this."

field_type_optional(type) ::= <<
@Nullable <type_name(type)>
>>

field_type_sequence(type) ::= <<
List\<<type_name(type)>\>
>>
```

```
field_decl(typed_element) ::= <<
private final <var_decl(typed_element)>;
>>

// Getters

getter_name(element) ::= <<
<if(element.type.boolean)><\\>
is<element.name; format="pascal-case"><\\>
<else><\\>
get<element.name; format="pascal-case"><\\>
<endif>
>>

interface_getter_annotations(property) ::= ""

class_getter_annotations(property) ::= ""

getter_type_required(type) ::= <<
<type_name(type)>
>>

getter_type_optional(type) ::= <<
<if(type.boolean)><\\>
<type_name(type)><\\>
<else><\\>
Optional\<<type_name(type)>\><\\>
<endif>
>>

getter_type_sequence(type) ::= <<
List\<<boxed_type_name(type)>\>
>>

getter_field_value_required(property) ::= <<
<field_access(property)>
>>

getter_field_value_optional(property) ::= <<
Optional.ofNullable(<field_access(property)>)
>>

getter_field_value_sequence(property) ::= <<
Collections.unmodifiableList(<field_access(property)>)
>>

getter_call(property) ::= <<
<getter_name(property)>()
>>

// Interfaces

interface_keyword() ::= "interface "

// Classes

class_keyword() ::= "class "

class_type_name(name) ::= <<
<task.classTypeNamePrefix><name; format="pascal-case">
>>

constructor_name(class_name) ::= <<
<class_name; format="pascal-case">
>>

// Ancestors

ancestor_list(abstract, ancestors, interfaces) ::= <<
<if(ancestors)> extends <ancestors:type_name();separator=", "><endif><\\>
<if(interfaces)> implements <interfaces:type_name();separator=", "><endif>
>>

// Type

primitive_type_boolean() ::= "boolean"
primitive_type_integer() ::= "int"
primitive_type_decimal() ::= "BigDecimal"
primitive_type_string() ::= "String"
primitive_type_regex() ::= "Pattern"
primitive_type_byte() ::= "byte"
primitive_type_short() ::= "short"
primitive_type_long() ::= "long"
primitive_type_float() ::= "float"
primitive_type_double() ::= "double"
primitive_type_char() ::= "char"

boxed_primitive_type_boolean() ::= "Boolean"
boxed_primitive_type_integer() ::= "Integer"
boxed_primitive_type_decimal() ::= "BigDecimal"
boxed_primitive_type_string() ::= "String"
boxed_primitive_type_regex() ::= "Pattern"
boxed_primitive_type_byte() ::= "Byte"
boxed_primitive_type_short() ::= "Short"
boxed_primitive_type_long() ::= "Long"
boxed_primitive_type_float() ::= "Float"
boxed_primitive_type_double() ::= "Double"
boxed_primitive_type_char() ::= "Char"

type_hint_name(name) ::=<<
<class_type_name(name)>
>>

// Literal

literal_expr_boolean(text) ::= <<
<text>
>>

literal_expr_string(text) ::= <<
"<text>"
>>

literal_expr_integer(text) ::= <<
<text>
>>

literal_expr_long(text) ::= <<
<text>
>>

literal_expr_short(text) ::= <<
<text>
>>

literal_expr_byte(text) ::= <<
<text>
```

```
>>

literal_expr_decimal(text) ::= <<
new BigDecimal("<text>")
>>

literal_expr_float(text) ::= <<
<text>
>>

literal_expr_double(text) ::= <<
<text>
>>

// Constants

null_value() ::= "null"

integral_zero() ::= "0"

float_zero() ::= "0.0f"

boolean_false() ::= "false"

// Operations

operation_add(left, right) ::= "<nullable_expr(left)> + <nullable_expr(right)>"
operation_sub(left, right) ::= "<nullable_expr(left)> - <nullable_expr(right)>"
operation_mul(left, right) ::= "<nullable_expr(left)> * <nullable_expr(right)>"
operation_div(left, right) ::= "<nullable_expr(left)> / <nullable_expr(right)>"
operation_mod(left, right) ::= "<nullable_expr(left)> % <nullable_expr(right)>"
operation_exp(left, right) ::= "Math.pow(<nullable_expr(left)>, <nullable_expr(right)>)"

operation_string_concat(left, right) ::= <<
<string_concat_operand(left)> + <string_concat_operand(right)>
>>

string_concat_operand(expr) ::= <<
<if(!expr.type.string)>Objects.toString(<endif><nullable_expr(expr)><if(!expr.type.string)>)<endif>
>>

operation_eq(left, right) ::= <<
<if(left.type.string || right.type.string)><\\>
Objects.equals(<operation_eq_operand(left)>, <operation_eq_operand(right)>)<\\>
<else><\\>
<operation_eq_operand(left)> == <operation_eq_operand(right)><\\>
<endif>
>>

operation_eq_operand(operand) ::= <<
<expression(operand)><if(operand.type.optional)>.orElse(<field_default_value(operand.type)>)<endif>
>>

operation_ineq(left, right) ::= "<nullable_expr(left)> != <nullable_expr(right)>"
operation_gt(left, right) ::= "<nullable_expr(left)> > <nullable_expr(right)>"
operation_gte(left, right) ::= "<nullable_expr(left)> >= <nullable_expr(right)>"
operation_lt(left, right) ::= "<expression(left)> \< <nullable_expr(right)>"
operation_lte(left, right) ::= "<nullable_expr(left)> \<= <nullable_expr(right)>"

operation_and(left, right) ::= "<expression(left)> && <expression(right)>"
operation_or(left, right) ::= "<expression(left)> || <expression(right)>"
operation_not(expr) ::= "!<expression(expr)>"

operation_ref_eq_required_required(left, right) ::= "(<expression(left)> == <expression(right)>)"
operation_ref_eq_required_optional(left, right) ::= "(<expression(right)>.isPresent() ? <expression(right)>.get() == <expression(left)> : false)"
operation_ref_eq_optional_optional(left, right) ::= "(<expression(left)>.isPresent() && <expression(right)>.isPresent() ? <expression(left)>.get()

operation_not_ref_eq_required_required(left, right) ::= "(<expression(left)> != <expression(right)>)"
operation_not_ref_eq_required_optional(left, right) ::= "(<expression(right)>.isPresent() ? <expression(right)>.get() != <expression(left)> : true)
operation_not_ref_eq_optional_optional(left, right) ::= "(<expression(left)>.isPresent() && <expression(right)>.isPresent() ? <expression(left)>.ge

// Paths

path_expr(path) ::= <<
<if(path.none)><\\>
Optional.empty()<\\>
<elseif(path.self)><\\>
<actual_self_field()><\\>
<elseif(path.base)><\\>
<path_base_call(path)><\\>
<elseif(path.variable)><\\>
<field_name(path)><\\>
<elseif(!lambda.parameters && lambda.innerExpression)><\\>
<path_root_call_lambda_var_name(path)><\\>
<else><\\>
<path_root_call_self_field(path)><\\>
<endif>
>>

path_base_call(path) ::= <<
<if(path.base.type.sequence || path.base.type.optional)><\\>
<if(path.base.type.optional)><\\>
seq(<path_expr(path.base)>)<\\>
<else><\\>
seq(<path_expr(path.base)>)<\\>
<endif><\\>
.<path_map_call(path)><\\>
<else><\\>
<path_root_call_base(path)><\\>
<endif>
>>

path_root_call_base(path) ::= "<path_expr(path.base)>.<getter_name(path)>()"
path_root_call_lambda_var_name(path) ::= "<lambda_var_name()>.<getter_name(path)>()"
path_root_call_self_field(path) ::= "<actual_self_field()>.<getter_name(path)>()"

path_map_call(path) ::= <<
<if(path.base.type.required && path.type.required)><\\>
map(<field_name(path.base.type)> -> <field_name(path.base.type)>.<getter_name(path)>())<\\>
<else><\\>
flatMap(<field_name(path.base.type)> -> seq(<\\>
<if(path.originalType.required)><\\>
asList(<field_name(path.base.type)>.<getter_name(path)>())<\\>
<else><\\>
<field_name(path.base.type)>.<getter_name(path)>()<\\>
<endif><\\>
))<\\>
<endif><\\>
<if(!path.base.base && path.type.optional)>.findFirst()<if(path.type.boolean)>.orElse(false)<endif><endif>
>>

to_list(expr) ::= <<
<expr>.toList()
>>

expression_conditional(expr) ::= <<
(<expression(expr.condExpr)> ? <expression(expr.thenExpr)> : <expression(expr.elseExpr)>)
```

```
>>


==> cml-modules/cml_base/templates/lang/py/expression.stg


nullable_expr(arg) ::= <<
<expression(arg)>
>>


==> cml-modules/cml_base/templates/lang/py/invocation.stg


invocation_sequence_empty(args) ::= <<
(len(<expression(args.seq)>) == 0)
>>

invocation_optional_empty(args) ::= <<
(<expression(args.seq)> is None)
>>

invocation_required_empty(args) ::= <<
<invocation_optional_empty(args)>
>>

invocation_sequence_present(args) ::= <<
(len(<expression(args.seq)>) > 0)
>>

invocation_optional_present(args) ::= <<
(<expression(args.seq)> is not None)
>>

invocation_required_present(args) ::= <<
<invocation_optional_present(args)>
>>

invocation_sequence_first(args) ::= <<
(list(<expression(args.seq)>) or [None])[0]
>>

invocation_optional_first(args) ::= <<
<expression(args.seq)>
>>

invocation_required_first(args) ::= <<
<expression(args.seq)>
>>

invocation_sequence_last(args) ::= <<
((<expression(args.seq)>) or [None])[-1]
>>

invocation_optional_last(args) ::= <<
<expression(args.seq)>
>>

invocation_required_last(args) ::= <<
<expression(args.seq)>
>>

invocation_sequence_exists(args) ::= <<
any(map(<expression(args.expr)>, <expression(args.seq)>))
>>

invocation_optional_exists(args) ::= <<
any(map(<expression(args.expr)>, <optional_list(args)>))
>>

invocation_required_exists(args) ::= <<
any(map(<expression(args.expr)>, [<expression(args.seq)>]))
>>

invocation_sequence_all(args) ::= <<
all(map(<expression(args.expr)>, <expression(args.seq)>))
>>

invocation_optional_all(args) ::= <<
all(map(<expression(args.expr)>, <optional_list(args)>))
>>

invocation_required_all(args) ::= <<
all(map(<expression(args.expr)>, [<expression(args.seq)>]))
>>

invocation_sequence_none(args) ::= <<
(len(list(filter(<expression(args.expr)>, <expression(args.seq)>))) == 0)
>>

invocation_optional_none(args) ::= <<
(len(list(filter(<expression(args.expr)>, <optional_list(args)>))) == 0)
>>

invocation_required_none(args) ::= <<
(len(list(filter(<expression(args.expr)>, [<expression(args.seq)>]))) == 0)
>>

invocation_sequence_select(args) ::= <<
filter(<expression(args.expr)>, <expression(args.seq)>)
>>

invocation_optional_select(args) ::= <<
filter(<expression(args.expr)>, <optional_list(args)>)
>>

invocation_required_select(args) ::= <<
filter(<expression(args.expr)>, [<expression(args.seq)>])
>>

invocation_sequence_reject(args) ::= <<
itertools.filterfalse(<expression(args.expr)>, <expression(args.seq)>)
>>

invocation_optional_reject(args) ::= <<
itertools.filterfalse(<expression(args.expr)>, <optional_list(args)>)
>>

invocation_required_reject(args) ::= <<
itertools.filterfalse(<expression(args.expr)>, [<expression(args.seq)>])
>>

invocation_sequence_collect(args) ::= <<
map(<expression(args.expr)>, <expression(args.seq)>)
>>
```

```
invocation_optional_collect(args) ::= <<
map(<expression(args.expr)>, <optional_list(args)>)
>>

invocation_required_collect(args) ::= <<
map(<expression(args.expr)>, [<expression(args.seq)>])
>>

optional_list(args) ::= <<
[] if <expression(args.seq)> is None else [<expression(args.seq)>]
>>

invocation_sequence_sort(args) ::= <<
sorted(<expression(args.seq)>, key=functools.cmp_to_key(<expression(args.expr)>))
>>

invocation_compare(args) ::= <<
-1 if (<expression(args.expr1)> \< <expression(args.expr2)>) else +1 if (<expression(args.expr2)> \< <expression(args.expr1)>) else 0
>>

invocation_sequence_reverse(args) ::= <<
reversed(<expression(args.seq)>)
>>

invocation_concat(args) ::= <<
(<expression(args.seq1)> + <expression(args.seq2)>)
>>

invocation_sequence_count(args) ::= <<
len(<expression(args.seq)>)
>>

invocation_sequence_distinct(args) ::= <<
functools.reduce(lambda l, x: l if x in l else list(l)+[x], <expression(args.seq)>, [])
>>
```

==> cml-modules/cml_base/templates/lang/py/lambda.stg

```
expression_lambda(lambda) ::= <<
lambda<\ ><\\>
<if(lambda.parameters)><lambda.parameters; separator=", ", format="underscore-case"><else>item<endif><\\>
:<\ ><expression(lambda.innerExpression)>
>>
```

==> cml-modules/cml_base/templates/lang/py/type.stg

```
type_check_is(expr, type) ::= <<
isinstance(<expression(expr)>, <type_name(type)>)
>>

type_check_isnt(expr, type) ::= <<
(not isinstance(<expression(expr)>, <type_name(type)>))
>>

type_cast_asb_from_required_to_required(expr, type) ::= <<
cast('<type_name(type)>', <expression(expr)>)
>>

type_cast_asb_from_required_to_optional(expr, type) ::= <<
cast('<field_type_optional(type)>', <expression(expr)>)
>>

type_cast_asb_from_required_to_sequence(expr, type) ::= <<
[<type_cast_asb_from_required_to_required(expr, type)>]
>>

type_cast_asb_from_optional_to_required(expr, type) ::= <<
<type_cast_asb_from_required_to_required(expr, type)>
>>

type_cast_asb_from_optional_to_optional(expr, type) ::= <<
<type_cast_asb_from_required_to_optional(expr, type)>
>>

type_cast_asb_from_optional_to_sequence(expr, type) ::= <<
[] if <expression(expr)> is None else <type_cast_asb_from_required_to_sequence(expr, type)>
>>

type_cast_asb_from_sequence_to_sequence(expr, type) ::= <<
map(lambda item: cast('<type_name(type)>', item), <expression(expr)>)
>>

type_cast_asq_from_required_to_optional(expr, type) ::= <<
cast('<type_name(type)>', <expression(expr)>) if isinstance(<expression(expr)>, <type_name(type)>) else None
>>

type_cast_asq_from_required_to_sequence(expr, type) ::= <<
map(lambda item: cast('<type_name(type)>', item), filter(lambda item: isinstance(item, <type_name(type)>), [<expression(expr)>]))
>>

type_cast_asq_from_optional_to_optional(expr, type) ::= <<
<type_cast_asq_from_required_to_optional(expr, type)>
>>

type_cast_asq_from_optional_to_sequence(expr, type) ::= <<
<type_cast_asq_from_required_to_sequence(expr, type)>
>>

type_cast_asq_from_sequence_to_sequence(expr, type) ::= <<
map(lambda item: cast('<type_name(type)>', item), filter(lambda item: isinstance(item, <type_name(type)>), <expression(expr)>))
>>

type_cast_optional_list(expr) ::= <<
[] if <expression(expr)> is None else [<expression(expr)>]
>>
```

==> cml-modules/cml_base/templates/lang/py.stg

```
import "/lang/py/expression.stg"
import "/lang/py/invocation.stg"
import "/lang/py/lambda.stg"
import "/lang/py/type.stg"
import "/lang/common.stg"

// Language

lang ::= [
    "fields": false
]

// Keywords
```

```
void() ::= "None"

self() ::= "self"

// Blocks

block(statements) ::= <<
<\n><indent()><statements; anchor>
>>

block_empty() ::= <<
<\n><indent()>pass
>>

block_header_terminator() ::= ":"

// Call

super_constructor_name() ::= "super().__init__"

// Strings

to_string_name() ::= "__str__"

to_string_expr(concept) ::= <<
"%s(<\\>
<if(concept.printableProperties)><\\>
<concept.printableProperties:{p|<getter_call(p)>=%s}; separator=", ")" % (
<indent()><to_string_expr_params(concept); anchor>
)<\\>
<else><\\>
)" % type(self).__name__<\\>
<endif>
>>

to_string_expr_params(concept) ::= <<
type(self).__name__,
<to_string_fields(concept.printableProperties, ",\n")>
>>

to_string_field(property) ::= <<
<actual_self_field()>.<getter_call(property)>
>>

// Operations

operation_keyword() ::= "def "

operation_name(name) ::= <<
<name; format="underscore-case">
>>

static_operation_modifier() ::= "@staticmethod<\n>"

operation_header_content(instance, name, params, result_type) ::= <<
<name>(<\\>
<if(instance)>self<endif><\\>
<commaIf2(instance, params)><\\>
<params:var_decl(); separator=", ">)<\\>
<\ >-><\ >'<if(result_type)><result_type><else>None<endif>'
>>

var_decl(object) ::= <<
<if(object.name || object.type)><\\>
<param_decl(field_name(object), field_type(object))><\\>
<if(object.value)><\\>
<\ >=<\ ><literal_expr(object.value)><\\>
<endif><\\>
<else><\\>
<object><\\>
<endif>
>>

local_var_decl(typed_element) ::= <<
<field_name(typed_element)>
>>

param_decl(name, type) ::= "<name>: '<type>'"

return(expression) ::= <<
return <expression>
>>

// Fields

field_init(named_element) ::= <<
<field_access(named_element)> = <field_name(named_element)>
>>

field_name(named_element) ::= <<
<named_element.name; format="underscore-case">
>>

field_access_prefix() ::= "self.__"

field_type_optional(type) ::= <<
Optional[<type_name(type)>]
>>

field_type_sequence(type) ::= <<
List[<type_name(type)>]
>>

field_decl(typed_element) ::= ""

// Getters

getter_name(property) ::= <<
<property.name; format="underscore-case">
>>

interface_getter_annotations(property) ::= <<
@abstractproperty<\n>
>>

class_getter_annotations(property) ::= <<
@property<\n>
>>

getter_type_required(type) ::= <<
<type_name(type)>
>>

getter_type_optional(type) ::= <<
<if(type.boolean)><\\>
<type_name(type)><\\>
<else><\\>
```

```
Optional[<type_name(type)>]<\\>
<endif>
>>

getter_type_sequence(type) ::= <<
List[<type_name(type)>]
>>

getter_field_value_required(property) ::= <<
<field_access(property)>
>>

getter_field_value_optional(property) ::= <<
<field_access(property)>
>>

getter_field_value_sequence(property) ::= <<
<field_access(property)>
>>

getter_call(property) ::= <<
<property.name; format="underscore-case">
>>

// Structure

structure_preamble() ::= "<\n>"

// Interfaces

interface_keyword() ::= "class "

interface_ancestor_header(ancestors, ancestor_list) ::= <<
<\ >(ABC<if(ancestors)>, <ancestor_list><endif>)
>>

interface_operation_body() ::= <<
<block_empty()>
>>

// Classes

class_keyword() ::= "class "

class_type_name(name) ::= <<
<task.classTypeNamePrefix><name; format="pascal-case">
>>

constructor_name(class_name) ::= "__init__"

class_static_initializer(concept) ::= ""

// Ancestors

ancestor_list(abstract, ancestors, interfaces) ::= <<
<if(abstract || ancestors || interfaces)>(<endif><\\>
<if(ancestors)><\\>
<ancestors:type_name();separator=", "><\\>
<if(interfaces)>, <endif><\\>
<endif><\\>
<if(interfaces)><interfaces:type_name();separator=", "><endif><\\>
<if(abstract)><if(ancestors || interfaces)>, <endif>ABC<endif><\\>
<if(abstract || ancestors || interfaces)>)<endif>
>>

// Type

primitive_type_boolean() ::= "bool"
primitive_type_integer() ::= "int"
primitive_type_decimal() ::= "Decimal"
primitive_type_string() ::= "str"
primitive_type_regex() ::= "Pattern"
primitive_type_byte() ::= "int"
primitive_type_short() ::= "int"
primitive_type_long() ::= "int"
primitive_type_float() ::= "float"
primitive_type_double() ::= "float"
primitive_type_char() ::= "int"

type_hint_name(name) ::=<<
'<class_type_name(name)>'
>>

// Literal

literal_expr_boolean(text) ::= <<
<text; format="pascal-case">
>>

literal_expr_string(text) ::= <<
"<text>"
>>

literal_expr_integer(text) ::= <<
<text>
>>

literal_expr_long(text) ::= <<
<text>
>>

literal_expr_short(text) ::= <<
<text>
>>

literal_expr_byte(text) ::= <<
<text>
>>

literal_expr_decimal(text) ::= <<
Decimal("<text>")
>>

literal_expr_float(text) ::= <<
<text>
>>

literal_expr_double(text) ::= <<
<text>
>>

// Constants

null_value() ::= "None"

integral_zero() ::= "0"
```

```
float_zero() ::= "0"

boolean_false() ::= "False"

// Operations

operation_add(left, right) ::= "<expression(left)> + <expression(right)>"
operation_sub(left, right) ::= "<expression(left)> - <expression(right)>"
operation_mul(left, right) ::= "<expression(left)> * <expression(right)>"
operation_div(left, right) ::= "<expression(left)> / <expression(right)>"
operation_mod(left, right) ::= "<expression(left)> % <expression(right)>"
operation_exp(left, right) ::= "<expression(left)> ** <expression(right)>"

operation_string_concat(left, right) ::= <<
<string_concat_operand(left)> + <string_concat_operand(right)>
>>

string_concat_operand(expr) ::= <<
<if(!expr.type.string)>str(<endif><nullable_expr(expr)><if(!expr.type.string)>)<endif>
>>

operation_eq(left, right) ::= "<expression(left)> == <expression(right)>"
operation_ineq(left, right) ::= "<expression(left)> != <expression(right)>"
operation_gt(left, right) ::= "<expression(left)> > <expression(right)>"
operation_gte(left, right) ::= "<expression(left)> >= <expression(right)>"
operation_lt(left, right) ::= "<expression(left)> \< <expression(right)>"
operation_lte(left, right) ::= "<expression(left)> \<= <expression(right)>"

operation_and(left, right) ::= "<expression(left)> and <expression(right)>"
operation_or(left, right) ::= "<expression(left)> or <expression(right)>"
operation_not(expr) ::= "not <expression(expr)>"

operation_ref_eq_required_required(left, right) ::= "<expression(left)> == <expression(right)>"
operation_not_ref_eq_required_required(left, right) ::= "<expression(left)> != <expression(right)>"

// Paths

path_expr(path) ::= <<
<if(path.none)><\\>
None<\\>
<elseif(path.self)><\\>
<actual_self_field()><\\>
<elseif(path.base)><\\>
<path_base_call(path)><\\>
<elseif(lambda.parameters)><\\>
<field_name(path)><\\>
<elseif(!lambda.parameters && lambda.innerExpression)><\\>
item.<path_property(path)><\\>
<else><\\>
<path_getter_call(path)><\\>
<endif>
>>

path_getter_call(path) ::= <<
<actual_self_field()>.<path_property(path)>
>>

path_base_call(path) ::= <<
<if(path.base.type.sequence)><\\>
<path_map_call(path); anchor><\\>
<elseif(path.base.type.optional && path.type.boolean)><\\>
False if <path_expr(path.base)> is None else <path_expr(path.base)>.<path_property(path)><\\>
<else><\\>
<path_expr(path.base)>.<path_property(path)><\\>
<endif>
>>

path_map_call(path) ::= <<
<if(path.originalType.sequence)>itertools.chain.from_iterable(<endif><\\>
map(
<indent()>lambda <field_name(path.base.type)>: <field_name(path.base.type)>.<path_property(path)>,
<indent()><path_expr(path.base)>
)<\\>
<if(path.originalType.sequence)>)<endif>
>>

path_property(path) ::= <<
<path.name; format="underscore-case">
>>

to_list(expr) ::= <<
list(<\n><indent()><expr><\n>)
>>

expression_conditional(expr) ::= <<
(<expression(expr.thenExpr)> if <expression(expr.condExpr)> else <expression(expr.elseExpr)>)
>>
```

```
==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_cmlc_java/expected-client-output.txt

Bidirectional Associations (cmlc_java)

Corporation(stock="true", profit="true", name="Walt Disney")
- Employees: [Employee(name="Donald Duck"), Employee(name="Mickey Mouse")]
- Fleet: [Vehicle(plate="DUCK"), Vehicle(plate="MOUSE")]

Employee(name="Donald Duck")
- Employer: Corporation(stock="true", profit="true", name="Walt Disney")
- Vehicle: Optional[Vehicle(plate="DUCK")]

Employee(name="Mickey Mouse")
- Employer: Corporation(stock="true", profit="true", name="Walt Disney")
- Vehicle: Optional[Vehicle(plate="MOUSE")]

Vehicle(plate="DUCK")
- Owner: Corporation(stock="true", profit="true", name="Walt Disney")
- Driver: Optional[Employee(name="Donald Duck")]

Vehicle(plate="MOUSE")
- Owner: Corporation(stock="true", profit="true", name="Walt Disney")
- Driver: Optional[Employee(name="Mickey Mouse")]
```

```
==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_cmlc_java/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-associations-bidirectional</groupId>
    <artifactId>vehicles-cmlc-client</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
    <packaging>jar</packaging>

    <properties>
        <jar.name>vehicles-cmlc-client</jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml-associations-bidirectional</groupId>
            <artifactId>vehicles-cmlc</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${jar.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>vehicles.client.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>expressions-console-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>


==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_cmlc_java/src/main/java/vehicles/client/Launcher.java

package vehicles.client;

import vehicles.cmlc.Corporation;
import vehicles.cmlc.Employee;
import vehicles.cmlc.Vehicle;

import static java.util.Collections.emptyList;

public class Launcher
{
    public static void main(final String[] args)
    {
        System.out.println("Bidirectional Associations (cmlc_java)\n");

        final Corporation corporation = Corporation.createCorporation("Walt Disney", emptyList(), emptyList());
        final Vehicle duck = Vehicle.createVehicle("DUCK", null, corporation);
        final Employee donald = Employee.createEmployee("Donald Duck", corporation, duck);
        final Employee mickey = Employee.createEmployee("Mickey Mouse", corporation, null);
        final Vehicle mouse = Vehicle.createVehicle("MOUSE", mickey, corporation);

        System.out.println(corporation);
        System.out.println("- Employees: " + corporation.getEmployees());
        System.out.println("- Fleet: " + corporation.getFleet());
        System.out.println();

        System.out.println(donald);
        System.out.println("- Employer: " + donald.getEmployer());
        System.out.println("- Vehicle: " + donald.getVehicle());
        System.out.println();

        System.out.println(mickey);
        System.out.println("- Employer: " + mickey.getEmployer());
        System.out.println("- Vehicle: " + mickey.getVehicle());
        System.out.println();

        System.out.println(duck);
        System.out.println("- Owner: " + duck.getOwner());
        System.out.println("- Driver: " + duck.getDriver());
        System.out.println();

        System.out.println(mouse);
        System.out.println("- Owner: " + mouse.getOwner());
        System.out.println("- Driver: " + mouse.getDriver());
    }
}


==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_cmlc_py/client.py


from cml_associations_bidirectional_vehicles_cmlc import *

print("Bidirectional Associations (cmlc_py)\n")

donald = Employee.create_employee("Donald Duck", None, None)
corporation = Corporation.create_corporation("Walt Disney", [donald], [])
mouse = Vehicle.create_vehicle("MOUSE", None, corporation)
mickey = Employee.create_employee("Mickey Mouse", corporation, mouse)
duck = Vehicle.create_vehicle("DUCK", donald, corporation)

print(corporation)
print("- Employees: %s" % ', '.join(map(str, corporation.employees)))
print("- Fleet: %s" % ', '.join(map(str, corporation.fleet)))
print()

print(donald)
print("- Employer: %s" % donald.employer)
print("- Vehicle: %s" % donald.vehicle)
print()

print(mickey)
```

```
print("- Employer: %s" % mickey.employer)
print("- Vehicle: %s" % mickey.vehicle)
print()

print(duck)
print("- Owner: %s" % duck.owner)
print("- Driver: %s" % duck.driver)
print()

print(mouse)
print("- Owner: %s" % mouse.owner)
print("- Driver: %s" % mouse.driver)



==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_cmlc_py/expected-client-output.txt

Bidirectional Associations (cmlc_py)

CorporationImpl(stock=True, profit=True, name=Walt Disney)
- Employees: EmployeeImpl(name=Donald Duck), EmployeeImpl(name=Mickey Mouse)
- Fleet: VehicleImpl(plate=MOUSE), VehicleImpl(plate=DUCK)

EmployeeImpl(name=Donald Duck)
- Employer: CorporationImpl(stock=True, profit=True, name=Walt Disney)
- Vehicle: VehicleImpl(plate=DUCK)

EmployeeImpl(name=Mickey Mouse)
- Employer: CorporationImpl(stock=True, profit=True, name=Walt Disney)
- Vehicle: VehicleImpl(plate=MOUSE)

VehicleImpl(plate=DUCK)
- Owner: CorporationImpl(stock=True, profit=True, name=Walt Disney)
- Driver: EmployeeImpl(name=Donald Duck)

VehicleImpl(plate=MOUSE)
- Owner: CorporationImpl(stock=True, profit=True, name=Walt Disney)
- Driver: EmployeeImpl(name=Mickey Mouse)


==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_poj/expected-client-output.txt

Bidirectional Associations (poj)

Corporation(stock="true", profit="true", name="Walt Disney")
- Employees: [Employee(name="Donald Duck"), Employee(name="Mickey Mouse")]
- Fleet: [Vehicle(plate="MOUSE"), Vehicle(plate="DUCK")]

Employee(name="Donald Duck")
- Employer: Corporation(stock="true", profit="true", name="Walt Disney")
- Vehicle: Optional[Vehicle(plate="DUCK")]

Employee(name="Mickey Mouse")
- Employer: Corporation(stock="true", profit="true", name="Walt Disney")
- Vehicle: Optional[Vehicle(plate="MOUSE")]

Vehicle(plate="DUCK")
- Owner: Corporation(stock="true", profit="true", name="Walt Disney")
- Driver: Optional[Employee(name="Donald Duck")]

Vehicle(plate="MOUSE")
- Owner: Corporation(stock="true", profit="true", name="Walt Disney")
- Driver: Optional[Employee(name="Mickey Mouse")]


==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_poj/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-associations-bidirectional</groupId>
    <artifactId>vehicles-poj-client</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <jar.name>vehicles-poj-client</jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml-associations-bidirectional</groupId>
            <artifactId>vehicles-poj</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${jar.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>vehicles.client.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>expressions-console-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_poj/src/main/java/vehicles/client/Launcher.java

```
package vehicles.client;

import vehicles.poj.Corporation;
import vehicles.poj.Employee;
import vehicles.poj.Vehicle;

import static java.util.Collections.emptyList;

public class Launcher
{
    public static void main(final String[] args)
    {
        System.out.println("Bidirectional Associations (poj)\n");

        final Corporation corporation = new Corporation("Walt Disney", emptyList(), emptyList());
        final Vehicle mouse = new Vehicle("MOUSE", null, corporation);
        final Employee donald = new Employee("Donald Duck", corporation, null);
        final Employee mickey = new Employee("Mickey Mouse", corporation, mouse);
        final Vehicle duck = new Vehicle("DUCK", donald, corporation);

        System.out.println(corporation);
        System.out.println("- Employees: " + corporation.getEmployees());
        System.out.println("- Fleet: " + corporation.getFleet());
        System.out.println();

        System.out.println(donald);
        System.out.println("- Employer: " + donald.getEmployer());
        System.out.println("- Vehicle: " + donald.getVehicle());
        System.out.println();

        System.out.println(mickey);
        System.out.println("- Employer: " + mickey.getEmployer());
        System.out.println("- Vehicle: " + mickey.getVehicle());
        System.out.println();

        System.out.println(duck);
        System.out.println("- Owner: " + duck.getOwner());
        System.out.println("- Driver: " + duck.getDriver());
        System.out.println();

        System.out.println(mouse);
        System.out.println("- Owner: " + mouse.getOwner());
        System.out.println("- Driver: " + mouse.getDriver());
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_pop/client.py

```
from cml_associations_bidirectional_vehicles_pop import *

print("Bidirectional Associations (pop)\n")

donald = Employee("Donald Duck", None, None)
corporation = Corporation("Walt Disney", [donald], [])
mouse = Vehicle("MOUSE", None, corporation)
mickey = Employee("Mickey Mouse", corporation, mouse)
duck = Vehicle("DUCK", donald, corporation)

print(corporation)
print("- Employees: %s" % ', '.join(map(str, corporation.employees)))
print("- Fleet: %s" % ', '.join(map(str, corporation.fleet)))
print()

print(donald)
print("- Employer: %s" % donald.employer)
print("- Vehicle: %s" % donald.vehicle)
print()

print(mickey)
print("- Employer: %s" % mickey.employer)
print("- Vehicle: %s" % mickey.vehicle)
print()

print(duck)
print("- Owner: %s" % duck.owner)
print("- Driver: %s" % duck.driver)
print()

print(mouse)
print("- Owner: %s" % mouse.owner)
print("- Driver: %s" % mouse.driver)
```

==> cml-modules/cml_base/tests/associations/bidirectional/clients/vehicles_pop/expected-client-output.txt

```
Bidirectional Associations (pop)

Corporation(stock=True, profit=True, name=Walt Disney)
- Employees: Employee(name=Donald Duck), Employee(name=Mickey Mouse)
- Fleet: Vehicle(plate=MOUSE), Vehicle(plate=DUCK)

Employee(name=Donald Duck)
- Employer: Corporation(stock=True, profit=True, name=Walt Disney)
- Vehicle: Vehicle(plate=DUCK)

Employee(name=Mickey Mouse)
- Employer: Corporation(stock=True, profit=True, name=Walt Disney)
- Vehicle: Vehicle(plate=MOUSE)

Vehicle(plate=DUCK)
- Owner: Corporation(stock=True, profit=True, name=Walt Disney)
- Driver: Employee(name=Donald Duck)

Vehicle(plate=MOUSE)
- Owner: Corporation(stock=True, profit=True, name=Walt Disney)
- Driver: Employee(name=Mickey Mouse)
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_java/cml-compiler-output.txt

```
model files:
- pom.xml

Vehicle files:
- src/main/java/vehicles/cmlc/Vehicle.java

Employee files:
- src/main/java/vehicles/cmlc/Employee.java
```

```
Organization files:
- src/main/java/vehicles/cmlc/Organization.java

Corporation files:
- src/main/java/vehicles/cmlc/Corporation.java

Employment files:
- src/main/java/vehicles/cmlc/Employment.java

VehicleOwnership files:
- src/main/java/vehicles/cmlc/VehicleOwnership.java

VehicleAssignment files:
- src/main/java/vehicles/cmlc/VehicleAssignment.java


==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_java/pom.xml

<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-associations-bidirectional</groupId>
    <artifactId>vehicles-cmlc</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>15.0</version>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-source-plugin</artifactId>
              <version>2.4</version>
              <executions>
                <execution>
                  <id>attach-sources</id>
                  <goals>
                    <goal>jar</goal>
                  </goals>
                </execution>
              </executions>
            </plugin>
        </plugins>
    </build>

</project>

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_java/src/main/java/vehicles/cmlc/Corporation.java

package vehicles.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Corporation extends Organization
{
    boolean isStock();

    boolean isProfit();

    Corporation getMyself();

    Vehicle getNewVehicle();

    static Corporation createCorporation(String name, List<Employee> employees, List<Vehicle> fleet)
    {
        return createCorporation(name, employees, fleet, true, true);
    }

    static Corporation createCorporation(String name, List<Employee> employees, List<Vehicle> fleet, boolean stock, boolean profit)
    {
        return new CorporationImpl(null, name, employees, fleet, stock, profit);
    }

    static Corporation extendCorporation(@Nullable Corporation actual_self, Organization organization, boolean stock, boolean profit)
    {
        return new CorporationImpl(actual_self, organization, stock, profit);
    }
}

class CorporationImpl implements Corporation
{
    private final @Nullable Corporation actual_self;

    private final Organization organization;

    private final boolean stock;
    private final boolean profit;
```

```
    CorporationImpl(@Nullable Corporation actual_self, String name, List<Employee> employees, List<Vehicle> fleet, boolean stock, boolean profit)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.organization = Organization.extendOrganization(this.actual_self, name, employees, fleet);
        this.stock = stock;
        this.profit = profit;
    }

    CorporationImpl(@Nullable Corporation actual_self, Organization organization, boolean stock, boolean profit)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.organization = organization;
        this.stock = stock;
        this.profit = profit;
    }

    public boolean isStock()
    {
        return this.stock;
    }

    public boolean isProfit()
    {
        return this.profit;
    }

    public Corporation getMyself()
    {
        return this.actual_self;
    }

    public Vehicle getNewVehicle()
    {
        return Vehicle.createVehicle("NEW", seq(this.actual_self.getEmployees()).findFirst().orElse(null), this.actual_self);
    }

    public String getName()
    {
        return this.organization.getName();
    }

    public List<Employee> getEmployees()
    {
        return this.organization.getEmployees();
    }

    public List<Vehicle> getFleet()
    {
        return this.organization.getFleet();
    }

    public String toString()
    {
        return new StringBuilder(Corporation.class.getSimpleName())
                    .append('(')
                    .append("stock=").append(String.format("\"%s\"", this.actual_self.isStock())).append(", ")
                    .append("profit=").append(String.format("\"%s\"", this.actual_self.isProfit())).append(", ")
                    .append("name=").append(String.format("\"%s\"", this.actual_self.getName()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_java/src/main/java/vehicles/cmlc/Employee.java

```
package vehicles.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Employee
{
    String getName();

    Organization getEmployer();

    Optional<Vehicle> getVehicle();

    static Employee createEmployee(String name, Organization employer, @Nullable Vehicle vehicle)
    {
        return new EmployeeImpl(null, name, employer, vehicle);
    }

    static Employee extendEmployee(@Nullable Employee actual_self, String name, Organization employer, @Nullable Vehicle vehicle)
    {
        return new EmployeeImpl(actual_self, name, employer, vehicle);
    }
}

class EmployeeImpl implements Employee
{
    private static Employment employment;
    private static VehicleAssignment vehicleAssignment;

    private final @Nullable Employee actual_self;

    private final String name;

    EmployeeImpl(@Nullable Employee actual_self, String name, Organization employer, @Nullable Vehicle vehicle)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.name = name;

        employment.link(employer, this.actual_self);
        vehicleAssignment.link(vehicle, this.actual_self);
    }

    public String getName()
    {
        return this.name;
    }

    public Organization getEmployer()
    {
        return employment.employerOf(actual_self).get();
    }
```

```
    public Optional<Vehicle> getVehicle()
    {
        return vehicleAssignment.vehicleOf(actual_self);
    }

    public String toString()
    {
        return new StringBuilder(Employee.class.getSimpleName())
                    .append('(')
                    .append("name=").append(String.format("\"%s\"", this.actual_self.getName()))
                    .append(')')
                    .toString();
    }

    static void setEmployment(Employment association)
    {
        employment = association;
    }

    static void setVehicleAssignment(VehicleAssignment association)
    {
        vehicleAssignment = association;
    }

    static
    {
        Employment.init(Employee.class);
        VehicleAssignment.init(Employee.class);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_java/src/main/java/vehicles/cmlc/Employment.java

```
package vehicles.cmlc;

import java.util.*;
import java.math.*;
import org.jetbrains.annotations.*;

public class Employment
{
    private static final Employment singleton = new Employment();

    static void init(Class<?> cls)
    {
        if (Employee.class.isAssignableFrom(cls))
        {
            EmployeeImpl.setEmployment(singleton);
        }
        if (Organization.class.isAssignableFrom(cls))
        {
            OrganizationImpl.setEmployment(singleton);
        }
    }

    private final Map<Employee, Organization> employer = new HashMap<>();
    private final Map<Organization, Set<Employee>> employees = new HashMap<>();

    synchronized void linkMany(Organization employer, List<Employee> employees)
    {
        for (Employee employee: employees) link(employer, employee);
    }

    synchronized void link(Organization organization, Employee employee)
    {
        this.employer.put(employee, organization);

        final Set<Employee> employeeList = this.employees.computeIfAbsent(organization, key -> new LinkedHashSet<>());
        if (!employeeList.contains(employee))
        {
            employeeList.add(employee);
        }
    }

    synchronized Optional<Organization> employerOf(Employee employee)
    {
        return Optional.ofNullable(this.employer.get(employee));
    }

    synchronized List<Employee> employeesOf(Organization organization)
    {
        final Set<Employee> employeeList = this.employees.get(organization);

        return (employeeList == null) ? Collections.emptyList() : new ArrayList<>(employeeList);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_java/src/main/java/vehicles/cmlc/Organization.java

```
package vehicles.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Organization
{
    String getName();

    List<Employee> getEmployees();

    List<Vehicle> getFleet();

    static Organization extendOrganization(@Nullable Organization actual_self, String name, List<Employee> employees, List<Vehicle> fleet)
    {
        return new OrganizationImpl(actual_self, name, employees, fleet);
    }
}

class OrganizationImpl implements Organization
{
    private static Employment employment;
    private static VehicleOwnership vehicleOwnership;

    private final @Nullable Organization actual_self;

    private final String name;

    OrganizationImpl(@Nullable Organization actual_self, String name, List<Employee> employees, List<Vehicle> fleet)
```

```
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.name = name;

        employment.linkMany(this.actual_self, employees);
        vehicleOwnership.linkMany(this.actual_self, fleet);
    }

    public String getName()
    {
        return this.name;
    }

    public List<Employee> getEmployees()
    {
        return employment.employeesOf(actual_self);
    }

    public List<Vehicle> getFleet()
    {
        return vehicleOwnership.fleetOf(actual_self);
    }

    public String toString()
    {
        return new StringBuilder(Organization.class.getSimpleName())
                    .append('(')
                    .append("name=").append(String.format("\"%s\"", this.actual_self.getName()))
                    .append(')')
                    .toString();
    }

    static void setEmployment(Employment association)
    {
        employment = association;
    }

    static void setVehicleOwnership(VehicleOwnership association)
    {
        vehicleOwnership = association;
    }

    static
    {
        Employment.init(Organization.class);
        VehicleOwnership.init(Organization.class);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_java/src/main/java/vehicles/cmlc/Vehicle.java

```
package vehicles.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Vehicle
{
    String getPlate();

    Optional<Employee> getDriver();

    Organization getOwner();

    static Vehicle createVehicle(String plate, @Nullable Employee driver, Organization owner)
    {
        return new VehicleImpl(null, plate, driver, owner);
    }

    static Vehicle extendVehicle(@Nullable Vehicle actual_self, String plate, @Nullable Employee driver, Organization owner)
    {
        return new VehicleImpl(actual_self, plate, driver, owner);
    }
}

class VehicleImpl implements Vehicle
{
    private static VehicleOwnership vehicleOwnership;
    private static VehicleAssignment vehicleAssignment;

    private final @Nullable Vehicle actual_self;

    private final String plate;

    VehicleImpl(@Nullable Vehicle actual_self, String plate, @Nullable Employee driver, Organization owner)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.plate = plate;

        vehicleAssignment.link(driver, this.actual_self);
        vehicleOwnership.link(owner, this.actual_self);
    }

    public String getPlate()
    {
        return this.plate;
    }

    public Optional<Employee> getDriver()
    {
        return vehicleAssignment.driverOf(actual_self);
    }

    public Organization getOwner()
    {
        return vehicleOwnership.ownerOf(actual_self).get();
    }

    public String toString()
    {
        return new StringBuilder(Vehicle.class.getSimpleName())
                    .append('(')
                    .append("plate=").append(String.format("\"%s\"", this.actual_self.getPlate()))
                    .append(')')
                    .toString();
    }
```

```
    static void setVehicleOwnership(VehicleOwnership association)
    {
        vehicleOwnership = association;
    }

    static void setVehicleAssignment(VehicleAssignment association)
    {
        vehicleAssignment = association;
    }

    static
    {
        VehicleOwnership.init(Vehicle.class);
        VehicleAssignment.init(Vehicle.class);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_java/src/main/java/vehicles/cmlc/VehicleAssignment.java

```
package vehicles.cmlc;

import java.util.*;
import java.math.*;
import org.jetbrains.annotations.*;

public class VehicleAssignment
{
    private static final VehicleAssignment singleton = new VehicleAssignment();

    static void init(Class<?> cls)
    {
        if (Vehicle.class.isAssignableFrom(cls))
        {
            VehicleImpl.setVehicleAssignment(singleton);
        }
        if (Employee.class.isAssignableFrom(cls))
        {
            EmployeeImpl.setVehicleAssignment(singleton);
        }
    }

    private final Map<Vehicle, @Nullable Employee> driver = new HashMap<>();
    private final Map<Employee, @Nullable Vehicle> vehicle = new HashMap<>();

    synchronized void link(Employee employee, Vehicle vehicle)
    {
        this.driver.put(vehicle, employee);

        this.vehicle.put(employee, vehicle);
    }

    synchronized void link(Vehicle vehicle, Employee employee)
    {
        this.driver.put(vehicle, employee);

        this.vehicle.put(employee, vehicle);
    }

    synchronized Optional<Employee> driverOf(Vehicle vehicle)
    {
        return Optional.ofNullable(this.driver.get(vehicle));
    }

    synchronized Optional<Vehicle> vehicleOf(Employee employee)
    {
        return Optional.ofNullable(this.vehicle.get(employee));
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_java/src/main/java/vehicles/cmlc/VehicleOwnership.java

```
package vehicles.cmlc;

import java.util.*;
import java.math.*;
import org.jetbrains.annotations.*;

public class VehicleOwnership
{
    private static final VehicleOwnership singleton = new VehicleOwnership();

    static void init(Class<?> cls)
    {
        if (Vehicle.class.isAssignableFrom(cls))
        {
            VehicleImpl.setVehicleOwnership(singleton);
        }
        if (Organization.class.isAssignableFrom(cls))
        {
            OrganizationImpl.setVehicleOwnership(singleton);
        }
    }

    private final Map<Vehicle, Organization> owner = new HashMap<>();
    private final Map<Organization, Set<Vehicle>> fleet = new HashMap<>();

    synchronized void linkMany(Organization owner, List<Vehicle> fleet)
    {
        for (Vehicle vehicle: fleet) link(owner, vehicle);
    }

    synchronized void link(Organization organization, Vehicle vehicle)
    {
        this.owner.put(vehicle, organization);

        final Set<Vehicle> vehicleList = this.fleet.computeIfAbsent(organization, key -> new LinkedHashSet<>());
        if (!vehicleList.contains(vehicle))
        {
            vehicleList.add(vehicle);
        }
    }

    synchronized Optional<Organization> ownerOf(Vehicle vehicle)
    {
        return Optional.ofNullable(this.owner.get(vehicle));
    }

    synchronized List<Vehicle> fleetOf(Organization organization)
    {
        final Set<Vehicle> vehicleList = this.fleet.get(organization);

        return (vehicleList == null) ? Collections.emptyList() : new ArrayList<>(vehicleList);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_py/cml-compiler-output.txt

```
model files:
- setup.py
- cml_associations_bidirectional_vehicles_cmlc/__init__.py


==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_py/cml_associations_bidirectional_vehicles_cmlc/__init__.py

from typing import *
from abc import *
from decimal import *

import functools, itertools

class _Employment:

    _singleton = None

    def __new__(cls) -> '_Employment':
        if cls._singleton is None:
            cls._singleton = super(_Employment, cls).__new__(cls)
        return cls._singleton

    def __init__(self) -> 'None':
        self.__employer = {}  # type: Dict[Employee, Organization]
        self.__employees = {}  # type: Dict[Organization, List[Employee]]

    def link_many(self, employer: 'Organization', employees: 'List[Employee]') -> 'None':
        for employee in employees: self.link(organization=employer, employee=employee)

    def link(self, organization: 'Organization', employee: 'Employee') -> 'None':
        self.__employer[employee] = organization

        if organization in self.__employees:
            employee_list = self.__employees[organization]
        else:
            employee_list = [employee]
        if not (employee in employee_list):
            employee_list.append(employee)
        self.__employees[organization] = employee_list

    def employer_of(self, employee: 'Employee') -> 'Optional[Organization]':
        if employee in self.__employer:
            return self.__employer[employee]
        else:
            return None

    def employees_of(self, organization: 'Organization') -> 'List[Employee]':
        if organization in self.__employees:
            employee_list = self.__employees[organization]
        else:
            employee_list = []
        return list(employee_list)


class _VehicleOwnership:

    _singleton = None

    def __new__(cls) -> '_VehicleOwnership':
        if cls._singleton is None:
            cls._singleton = super(_VehicleOwnership, cls).__new__(cls)
        return cls._singleton

    def __init__(self) -> 'None':
        self.__owner = {}  # type: Dict[Vehicle, Organization]
        self.__fleet = {}  # type: Dict[Organization, List[Vehicle]]

    def link_many(self, owner: 'Organization', fleet: 'List[Vehicle]') -> 'None':
        for vehicle in fleet: self.link(organization=owner, vehicle=vehicle)

    def link(self, organization: 'Organization', vehicle: 'Vehicle') -> 'None':
        self.__owner[vehicle] = organization

        if organization in self.__fleet:
            vehicle_list = self.__fleet[organization]
        else:
            vehicle_list = [vehicle]
        if not (vehicle in vehicle_list):
            vehicle_list.append(vehicle)
        self.__fleet[organization] = vehicle_list

    def owner_of(self, vehicle: 'Vehicle') -> 'Optional[Organization]':
        if vehicle in self.__owner:
            return self.__owner[vehicle]
        else:
            return None

    def fleet_of(self, organization: 'Organization') -> 'List[Vehicle]':
        if organization in self.__fleet:
            vehicle_list = self.__fleet[organization]
        else:
            vehicle_list = []
        return list(vehicle_list)


class _VehicleAssignment:

    _singleton = None

    def __new__(cls) -> '_VehicleAssignment':
        if cls._singleton is None:
            cls._singleton = super(_VehicleAssignment, cls).__new__(cls)
        return cls._singleton

    def __init__(self) -> 'None':
        self.__driver = {}  # type: Dict[Vehicle, Optional[Employee]]
        self.__vehicle = {}  # type: Dict[Employee, Optional[Vehicle]]

    def link(self, employee: 'Employee', vehicle: 'Vehicle') -> 'None':
        self.__driver[vehicle] = employee

        self.__vehicle[employee] = vehicle

    def driver_of(self, vehicle: 'Vehicle') -> 'Optional[Employee]':
        if vehicle in self.__driver:
            return self.__driver[vehicle]
        else:
            return None

    def vehicle_of(self, employee: 'Employee') -> 'Optional[Vehicle]':
        if employee in self.__vehicle:
            return self.__vehicle[employee]
        else:
            return None


class Vehicle(ABC):
```

```
    @abstractproperty
    def plate(self) -> 'str':
        pass

    @abstractproperty
    def driver(self) -> 'Optional[Employee]':
        pass

    @abstractproperty
    def owner(self) -> 'Organization':
        pass

    @staticmethod
    def create_vehicle(plate: 'str', driver: 'Optional[Employee]', owner: 'Organization') -> 'Vehicle':
        return VehicleImpl(None, plate, driver, owner)

    @staticmethod
    def extend_vehicle(actual_self: 'Optional[Vehicle]', plate: 'str', driver: 'Optional[Employee]', owner: 'Organization') -> 'Vehicle':
        return VehicleImpl(actual_self, plate, driver, owner)


class VehicleImpl(Vehicle):

    _vehicle_ownership = _VehicleOwnership()
    _vehicle_assignment = _VehicleAssignment()

    def __init__(self, actual_self: 'Optional[Vehicle]', plate: 'str', driver: 'Optional[Employee]', owner: 'Organization') -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Vehicle]
        else:
            self.__actual_self = actual_self

        self.__plate = plate

        self._vehicle_assignment.link(employee=driver, vehicle=self.__actual_self)
        self._vehicle_ownership.link(organization=owner, vehicle=self.__actual_self)

    @property
    def plate(self) -> 'str':
        return self.__plate

    @property
    def driver(self) -> 'Optional[Employee]':
        return self._vehicle_assignment.driver_of(self.__actual_self)

    @property
    def owner(self) -> 'Organization':
        return self._vehicle_ownership.owner_of(self.__actual_self)

    def __str__(self) -> 'str':
        return "%s(plate=%s)" % (
            type(self).__name__,
            self.__actual_self.plate
        )


class Employee(ABC):

    @abstractproperty
    def name(self) -> 'str':
        pass

    @abstractproperty
    def employer(self) -> 'Organization':
        pass

    @abstractproperty
    def vehicle(self) -> 'Optional[Vehicle]':
        pass

    @staticmethod
    def create_employee(name: 'str', employer: 'Organization', vehicle: 'Optional[Vehicle]') -> 'Employee':
        return EmployeeImpl(None, name, employer, vehicle)

    @staticmethod
    def extend_employee(actual_self: 'Optional[Employee]', name: 'str', employer: 'Organization', vehicle: 'Optional[Vehicle]') -> 'Employee':
        return EmployeeImpl(actual_self, name, employer, vehicle)


class EmployeeImpl(Employee):

    _employment = _Employment()
    _vehicle_assignment = _VehicleAssignment()

    def __init__(self, actual_self: 'Optional[Employee]', name: 'str', employer: 'Organization', vehicle: 'Optional[Vehicle]') -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Employee]
        else:
            self.__actual_self = actual_self

        self.__name = name

        self._employment.link(organization=employer, employee=self.__actual_self)
        self._vehicle_assignment.link(vehicle=vehicle, employee=self.__actual_self)

    @property
    def name(self) -> 'str':
        return self.__name

    @property
    def employer(self) -> 'Organization':
        return self._employment.employer_of(self.__actual_self)

    @property
    def vehicle(self) -> 'Optional[Vehicle]':
        return self._vehicle_assignment.vehicle_of(self.__actual_self)

    def __str__(self) -> 'str':
        return "%s(name=%s)" % (
            type(self).__name__,
            self.__actual_self.name
        )


class Organization(ABC):

    @abstractproperty
    def name(self) -> 'str':
        pass

    @abstractproperty
    def employees(self) -> 'List[Employee]':
        pass

    @abstractproperty
    def fleet(self) -> 'List[Vehicle]':
```

```
        pass

    @staticmethod
    def extend_organization(actual_self: 'Optional[Organization]', name: 'str', employees: 'List[Employee]', fleet: 'List[Vehicle]') -> 'Organizati
        return OrganizationImpl(actual_self, name, employees, fleet)


class OrganizationImpl(Organization):

    _employment = _Employment()
    _vehicle_ownership = _VehicleOwnership()

    def __init__(self, actual_self: 'Optional[Organization]', name: 'str', employees: 'List[Employee]', fleet: 'List[Vehicle]') -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Organization]
        else:
            self.__actual_self = actual_self

        self.__name = name

        self._employment.link_many(self.__actual_self, employees)
        self._vehicle_ownership.link_many(self.__actual_self, fleet)

    @property
    def name(self) -> 'str':
        return self.__name

    @property
    def employees(self) -> 'List[Employee]':
        return self._employment.employees_of(self.__actual_self)

    @property
    def fleet(self) -> 'List[Vehicle]':
        return self._vehicle_ownership.fleet_of(self.__actual_self)

    def __str__(self) -> 'str':
        return "%s(name=%s)" % (
            type(self).__name__,
            self.__actual_self.name
        )


class Corporation(Organization, ABC):

    @abstractproperty
    def stock(self) -> 'bool':
        pass

    @abstractproperty
    def profit(self) -> 'bool':
        pass

    @abstractproperty
    def myself(self) -> 'Corporation':
        pass

    @abstractproperty
    def new_vehicle(self) -> 'Vehicle':
        pass

    @staticmethod
    def create_corporation(name: 'str', employees: 'List[Employee]', fleet: 'List[Vehicle]', stock: 'bool' = True, profit: 'bool' = True) -> 'Corpo
        return CorporationImpl(None, None, stock, profit, name=name, employees=employees, fleet=fleet)

    @staticmethod
    def extend_corporation(actual_self: 'Optional[Corporation]', organization: 'Organization', stock: 'bool' = True, profit: 'bool' = True) -> 'Cor
        return CorporationImpl(actual_self, organization, stock, profit)


class CorporationImpl(Corporation):

    def __init__(self, actual_self: 'Optional[Corporation]', organization: 'Optional[Organization]', stock: 'bool' = True, profit: 'bool' = True, *
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Corporation]
        else:
            self.__actual_self = actual_self

        if organization is None:
            self.__organization = Organization.extend_organization(self.__actual_self, kwargs['name'], kwargs['employees'], kwargs['fleet'])
        else:
            self.__organization = organization
        self.__stock = stock
        self.__profit = profit

    @property
    def stock(self) -> 'bool':
        return self.__stock

    @property
    def profit(self) -> 'bool':
        return self.__profit

    @property
    def myself(self) -> 'Corporation':
        return self.__actual_self

    @property
    def new_vehicle(self) -> 'Vehicle':
        return Vehicle.create_vehicle("NEW", (list(self.__actual_self.employees) or [None])[0], self.__actual_self)

    @property
    def name(self) -> 'str':
        return self.__organization.name

    @property
    def employees(self) -> 'List[Employee]':
        return self.__organization.employees

    @property
    def fleet(self) -> 'List[Vehicle]':
        return self.__organization.fleet

    def __str__(self) -> 'str':
        return "%s(stock=%s, profit=%s, name=%s)" % (
            type(self).__name__,
            self.__actual_self.stock,
            self.__actual_self.profit,
            self.__actual_self.name
        )
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_cmlc_py/setup.py

```
"""A setuptools based setup module.

See:
https://packaging.python.org/en/latest/distributing.html
https://github.com/pypa/sampleproject
```

```
"""

# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
    name='cml_associations_bidirectional_vehicles_cmlc',

    # Versions should comply with PEP440.  For a discussion on single-sourcing
    # the version across setup.py and the project code, see
    # https://packaging.python.org/en/latest/single_source_version.html
    version='1.0',

    description='A sample Python project',

    # The project's main homepage.
    url='https://github.com/pypa/sampleproject',

    # Author details
    author='The Python Packaging Authority',
    author_email='pypa-dev@googlegroups.com',

    # Choose your license
    license='MIT',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',

        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: MIT License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3.6.1',
    ],

    # What does your project relate to?
    keywords='sample setuptools development',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests']),

    # Alternatively, if you want to distribute just a my_module.py, uncomment
    # this:
    #   py_modules=["my_module"],

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    # install_requires=['peppercorn'],

    # List additional groups of dependencies here (e.g. development
    # dependencies). You can install these using the following syntax,
    # for example:
    # $ pip install -e .[dev,test]
    extras_require={
        'dev': ['check-manifest'],
        'test': ['coverage'],
    },

    # If there are data files included in your packages that need to be
    # installed, specify them here.  If using Python 2.6 or less, then these
    # have to be included in MANIFEST.in as well.
    # package_data={
    #     'sample': ['package_data.dat'],
    #},

    # Although 'package_data' is the preferred approach, in some case you may
    # need to place data files outside of your packages. See:
    # http://docs.python.org/3.4/distutils/setupscript.html#installing-additional-files # noqa
    # In this case, 'data_file' will be installed into '<sys.prefix>/my_data'
    #data_files=[('my_data', ['data/data_file'])],

    # To provide executable scripts, use entry points in preference to the
    # "scripts" keyword. Entry points provide cross-platform support and allow
    # pip to create the appropriate form of executable for the target platform.
    #entry_points={
    #    'console_scripts': [
    #        'sample=sample:main',
    #    ],
    #},
)

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_poj/cml-compiler-output.txt

model files:
- pom.xml

Vehicle files:
- src/main/java/vehicles/poj/Vehicle.java

Employee files:
- src/main/java/vehicles/poj/Employee.java

Organization files:
- src/main/java/vehicles/poj/Organization.java

Corporation files:
- src/main/java/vehicles/poj/Corporation.java

Employment files:
- src/main/java/vehicles/poj/Employment.java

VehicleOwnership files:
- src/main/java/vehicles/poj/VehicleOwnership.java

VehicleAssignment files:
- src/main/java/vehicles/poj/VehicleAssignment.java


==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_poj/pom.xml
```

```
<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-associations-bidirectional</groupId>
    <artifactId>vehicles-poj</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>15.0</version>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-source-plugin</artifactId>
              <version>2.4</version>
              <executions>
                <execution>
                  <id>attach-sources</id>
                  <goals>
                    <goal>jar</goal>
                  </goals>
                </execution>
              </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_poj/src/main/java/vehicles/poj/Corporation.java

```
package vehicles.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Corporation extends Organization
{
    private final boolean stock;
    private final boolean profit;

    public Corporation(String name, List<Employee> employees, List<Vehicle> fleet)
    {
        this(name, employees, fleet, true, true);
    }

    public Corporation(String name, List<Employee> employees, List<Vehicle> fleet, boolean stock, boolean profit)
    {
        super(name, employees, fleet);
        this.stock = stock;
        this.profit = profit;
    }

    public boolean isStock()
    {
        return this.stock;
    }

    public boolean isProfit()
    {
        return this.profit;
    }

    public Corporation getMyself()
    {
        return this;
    }

    public Vehicle getNewVehicle()
    {
        return new Vehicle("NEW", seq(this.getEmployees()).findFirst().orElse(null), this);
    }

    public String toString()
    {
        return new StringBuilder(Corporation.class.getSimpleName())
                    .append('(')
                    .append("stock=").append(String.format("\"%s\"", this.isStock())).append(", ")
                    .append("profit=").append(String.format("\"%s\"", this.isProfit())).append(", ")
                    .append("name=").append(String.format("\"%s\"", this.getName()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_poj/src/main/java/vehicles/poj/Employee.java

```
package vehicles.poj;
```

```
import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Employee
{
    private static Employment employment;
    private static VehicleAssignment vehicleAssignment;

    private final String name;

    public Employee(String name, Organization employer, @Nullable Vehicle vehicle)
    {
        this.name = name;

        employment.link(employer, this);
        vehicleAssignment.link(vehicle, this);
    }

    public String getName()
    {
        return this.name;
    }

    public Organization getEmployer()
    {
        return employment.employerOf(this).get();
    }

    public Optional<Vehicle> getVehicle()
    {
        return vehicleAssignment.vehicleOf(this);
    }

    public String toString()
    {
        return new StringBuilder(Employee.class.getSimpleName())
                    .append('(')
                    .append("name=").append(String.format("\"%s\"", this.getName()))
                    .append(')')
                    .toString();
    }

    static void setEmployment(Employment association)
    {
        employment = association;
    }

    static void setVehicleAssignment(VehicleAssignment association)
    {
        vehicleAssignment = association;
    }

    static
    {
        Employment.init(Employee.class);
        VehicleAssignment.init(Employee.class);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_poj/src/main/java/vehicles/poj/Employment.java

```
package vehicles.poj;

import java.util.*;
import java.math.*;
import org.jetbrains.annotations.*;

public class Employment
{
    private static final Employment singleton = new Employment();

    static void init(Class<?> cls)
    {
        if (Employee.class.isAssignableFrom(cls))
        {
            Employee.setEmployment(singleton);
        }
        if (Organization.class.isAssignableFrom(cls))
        {
            Organization.setEmployment(singleton);
        }
    }

    private final Map<Employee, Organization> employer = new HashMap<>();
    private final Map<Organization, Set<Employee>> employees = new HashMap<>();

    synchronized void linkMany(Organization employer, List<Employee> employees)
    {
        for (Employee employee: employees) link(employer, employee);
    }

    synchronized void link(Organization organization, Employee employee)
    {
        this.employer.put(employee, organization);

        final Set<Employee> employeeList = this.employees.computeIfAbsent(organization, key -> new LinkedHashSet<>());
        if (!employeeList.contains(employee))
        {
            employeeList.add(employee);
        }
    }

    synchronized Optional<Organization> employerOf(Employee employee)
    {
        return Optional.ofNullable(this.employer.get(employee));
    }

    synchronized List<Employee> employeesOf(Organization organization)
    {
        final Set<Employee> employeeList = this.employees.get(organization);

        return (employeeList == null) ? Collections.emptyList() : new ArrayList<>(employeeList);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_poj/src/main/java/vehicles/poj/Organization.java

```
package vehicles.poj;
```

```
import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public abstract class Organization
{
    private static Employment employment;
    private static VehicleOwnership vehicleOwnership;

    private final String name;

    public Organization(String name, List<Employee> employees, List<Vehicle> fleet)
    {
        this.name = name;

        employment.linkMany(this, employees);
        vehicleOwnership.linkMany(this, fleet);
    }

    public String getName()
    {
        return this.name;
    }

    public List<Employee> getEmployees()
    {
        return employment.employeesOf(this);
    }

    public List<Vehicle> getFleet()
    {
        return vehicleOwnership.fleetOf(this);
    }

    public String toString()
    {
        return new StringBuilder(Organization.class.getSimpleName())
                    .append('(')
                    .append("name=").append(String.format("\"%s\"", this.getName()))
                    .append(')')
                    .toString();
    }

    static void setEmployment(Employment association)
    {
        employment = association;
    }

    static void setVehicleOwnership(VehicleOwnership association)
    {
        vehicleOwnership = association;
    }

    static
    {
        Employment.init(Organization.class);
        VehicleOwnership.init(Organization.class);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_poj/src/main/java/vehicles/poj/Vehicle.java

```
package vehicles.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Vehicle
{
    private static VehicleOwnership vehicleOwnership;
    private static VehicleAssignment vehicleAssignment;

    private final String plate;

    public Vehicle(String plate, @Nullable Employee driver, Organization owner)
    {
        this.plate = plate;

        vehicleAssignment.link(driver, this);
        vehicleOwnership.link(owner, this);
    }

    public String getPlate()
    {
        return this.plate;
    }

    public Optional<Employee> getDriver()
    {
        return vehicleAssignment.driverOf(this);
    }

    public Organization getOwner()
    {
        return vehicleOwnership.ownerOf(this).get();
    }

    public String toString()
    {
        return new StringBuilder(Vehicle.class.getSimpleName())
                    .append('(')
                    .append("plate=").append(String.format("\"%s\"", this.getPlate()))
                    .append(')')
                    .toString();
    }

    static void setVehicleOwnership(VehicleOwnership association)
    {
        vehicleOwnership = association;
    }
```

```
    static void setVehicleAssignment(VehicleAssignment association)
    {
        vehicleAssignment = association;
    }

    static
    {
        VehicleOwnership.init(Vehicle.class);
        VehicleAssignment.init(Vehicle.class);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_poj/src/main/java/vehicles/poj/VehicleAssignment.java

```
package vehicles.poj;

import java.util.*;
import java.math.*;
import org.jetbrains.annotations.*;

public class VehicleAssignment
{
    private static final VehicleAssignment singleton = new VehicleAssignment();

    static void init(Class<?> cls)
    {
        if (Vehicle.class.isAssignableFrom(cls))
        {
            Vehicle.setVehicleAssignment(singleton);
        }
        if (Employee.class.isAssignableFrom(cls))
        {
            Employee.setVehicleAssignment(singleton);
        }
    }

    private final Map<Vehicle, @Nullable Employee> driver = new HashMap<>();
    private final Map<Employee, @Nullable Vehicle> vehicle = new HashMap<>();

    synchronized void link(Employee employee, Vehicle vehicle)
    {
        this.driver.put(vehicle, employee);

        this.vehicle.put(employee, vehicle);
    }

    synchronized void link(Vehicle vehicle, Employee employee)
    {
        this.driver.put(vehicle, employee);

        this.vehicle.put(employee, vehicle);
    }

    synchronized Optional<Employee> driverOf(Vehicle vehicle)
    {
        return Optional.ofNullable(this.driver.get(vehicle));
    }

    synchronized Optional<Vehicle> vehicleOf(Employee employee)
    {
        return Optional.ofNullable(this.vehicle.get(employee));
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_poj/src/main/java/vehicles/poj/VehicleOwnership.java

```
package vehicles.poj;

import java.util.*;
import java.math.*;
import org.jetbrains.annotations.*;

public class VehicleOwnership
{
    private static final VehicleOwnership singleton = new VehicleOwnership();

    static void init(Class<?> cls)
    {
        if (Vehicle.class.isAssignableFrom(cls))
        {
            Vehicle.setVehicleOwnership(singleton);
        }
        if (Organization.class.isAssignableFrom(cls))
        {
            Organization.setVehicleOwnership(singleton);
        }
    }

    private final Map<Vehicle, Organization> owner = new HashMap<>();
    private final Map<Organization, Set<Vehicle>> fleet = new HashMap<>();

    synchronized void linkMany(Organization owner, List<Vehicle> fleet)
    {
        for (Vehicle vehicle: fleet) link(owner, vehicle);
    }

    synchronized void link(Organization organization, Vehicle vehicle)
    {
        this.owner.put(vehicle, organization);

        final Set<Vehicle> vehicleList = this.fleet.computeIfAbsent(organization, key -> new LinkedHashSet<>());
        if (!vehicleList.contains(vehicle))
        {
            vehicleList.add(vehicle);
        }
    }

    synchronized Optional<Organization> ownerOf(Vehicle vehicle)
    {
        return Optional.ofNullable(this.owner.get(vehicle));
    }

    synchronized List<Vehicle> fleetOf(Organization organization)
    {
        final Set<Vehicle> vehicleList = this.fleet.get(organization);

        return (vehicleList == null) ? Collections.emptyList() : new ArrayList<>(vehicleList);
    }
}
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_pop/cml-compiler-output.txt

```
model files:
- setup.py
- cml_associations_bidirectional_vehicles_pop/__init__.py
```

```
==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_pop/cml_associations_bidirectional_vehicles_pop/__init__.py

from typing import *
from abc import *
from decimal import *

import functools, itertools

class _Employment:

    _singleton = None

    def __new__(cls) -> '_Employment':
        if cls._singleton is None:
            cls._singleton = super(_Employment, cls).__new__(cls)
        return cls._singleton

    def __init__(self) -> 'None':
        self.__employer = {}  # type: Dict[Employee, Organization]
        self.__employees = {}  # type: Dict[Organization, List[Employee]]

    def link_many(self, employer: 'Organization', employees: 'List[Employee]') -> 'None':
        for employee in employees: self.link(organization=employer, employee=employee)

    def link(self, organization: 'Organization', employee: 'Employee') -> 'None':
        self.__employer[employee] = organization

        if organization in self.__employees:
            employee_list = self.__employees[organization]
        else:
            employee_list = [employee]
        if not (employee in employee_list):
            employee_list.append(employee)
        self.__employees[organization] = employee_list

    def employer_of(self, employee: 'Employee') -> 'Optional[Organization]':
        if employee in self.__employer:
            return self.__employer[employee]
        else:
            return None

    def employees_of(self, organization: 'Organization') -> 'List[Employee]':
        if organization in self.__employees:
            employee_list = self.__employees[organization]
        else:
            employee_list = []
        return list(employee_list)


class _VehicleOwnership:

    _singleton = None

    def __new__(cls) -> '_VehicleOwnership':
        if cls._singleton is None:
            cls._singleton = super(_VehicleOwnership, cls).__new__(cls)
        return cls._singleton

    def __init__(self) -> 'None':
        self.__owner = {}  # type: Dict[Vehicle, Organization]
        self.__fleet = {}  # type: Dict[Organization, List[Vehicle]]

    def link_many(self, owner: 'Organization', fleet: 'List[Vehicle]') -> 'None':
        for vehicle in fleet: self.link(organization=owner, vehicle=vehicle)

    def link(self, organization: 'Organization', vehicle: 'Vehicle') -> 'None':
        self.__owner[vehicle] = organization

        if organization in self.__fleet:
            vehicle_list = self.__fleet[organization]
        else:
            vehicle_list = [vehicle]
        if not (vehicle in vehicle_list):
            vehicle_list.append(vehicle)
        self.__fleet[organization] = vehicle_list

    def owner_of(self, vehicle: 'Vehicle') -> 'Optional[Organization]':
        if vehicle in self.__owner:
            return self.__owner[vehicle]
        else:
            return None

    def fleet_of(self, organization: 'Organization') -> 'List[Vehicle]':
        if organization in self.__fleet:
            vehicle_list = self.__fleet[organization]
        else:
            vehicle_list = []
        return list(vehicle_list)


class _VehicleAssignment:

    _singleton = None

    def __new__(cls) -> '_VehicleAssignment':
        if cls._singleton is None:
            cls._singleton = super(_VehicleAssignment, cls).__new__(cls)
        return cls._singleton

    def __init__(self) -> 'None':
        self.__driver = {}  # type: Dict[Vehicle, Optional[Employee]]
        self.__vehicle = {}  # type: Dict[Employee, Optional[Vehicle]]

    def link(self, employee: 'Employee', vehicle: 'Vehicle') -> 'None':
        self.__driver[vehicle] = employee

        self.__vehicle[employee] = vehicle

    def driver_of(self, vehicle: 'Vehicle') -> 'Optional[Employee]':
        if vehicle in self.__driver:
            return self.__driver[vehicle]
        else:
            return None

    def vehicle_of(self, employee: 'Employee') -> 'Optional[Vehicle]':
        if employee in self.__vehicle:
            return self.__vehicle[employee]
        else:
            return None


class Vehicle:

    _vehicle_ownership = _VehicleOwnership()
    _vehicle_assignment = _VehicleAssignment()

    def __init__(self, plate: 'str', driver: 'Optional[Employee]', owner: 'Organization') -> 'None':
```

```
        self.__plate = plate

        self._vehicle_assignment.link(employee=driver, vehicle=self)
        self._vehicle_ownership.link(organization=owner, vehicle=self)

    @property
    def plate(self) -> 'str':
        return self.__plate

    @property
    def driver(self) -> 'Optional[Employee]':
        return self._vehicle_assignment.driver_of(self)

    @property
    def owner(self) -> 'Organization':
        return self._vehicle_ownership.owner_of(self)

    def __str__(self) -> 'str':
        return "%s(plate=%s)" % (
            type(self).__name__,
            self.plate
        )


class Employee:

    _employment = _Employment()
    _vehicle_assignment = _VehicleAssignment()

    def __init__(self, name: 'str', employer: 'Organization', vehicle: 'Optional[Vehicle]') -> 'None':
        self.__name = name

        self._employment.link(organization=employer, employee=self)
        self._vehicle_assignment.link(vehicle=vehicle, employee=self)

    @property
    def name(self) -> 'str':
        return self.__name

    @property
    def employer(self) -> 'Organization':
        return self._employment.employer_of(self)

    @property
    def vehicle(self) -> 'Optional[Vehicle]':
        return self._vehicle_assignment.vehicle_of(self)

    def __str__(self) -> 'str':
        return "%s(name=%s)" % (
            type(self).__name__,
            self.name
        )


class Organization(ABC):

    _employment = _Employment()
    _vehicle_ownership = _VehicleOwnership()

    def __init__(self, name: 'str', employees: 'List[Employee]', fleet: 'List[Vehicle]') -> 'None':
        self.__name = name

        self._employment.link_many(self, employees)
        self._vehicle_ownership.link_many(self, fleet)

    @property
    def name(self) -> 'str':
        return self.__name

    @property
    def employees(self) -> 'List[Employee]':
        return self._employment.employees_of(self)

    @property
    def fleet(self) -> 'List[Vehicle]':
        return self._vehicle_ownership.fleet_of(self)

    def __str__(self) -> 'str':
        return "%s(name=%s)" % (
            type(self).__name__,
            self.name
        )


class Corporation(Organization):

    def __init__(self, name: 'str', employees: 'List[Employee]', fleet: 'List[Vehicle]', stock: 'bool' = True, profit: 'bool' = True) -> 'None':
        super().__init__(name, employees, fleet)
        self.__stock = stock
        self.__profit = profit

    @property
    def stock(self) -> 'bool':
        return self.__stock

    @property
    def profit(self) -> 'bool':
        return self.__profit

    @property
    def myself(self) -> 'Corporation':
        return self

    @property
    def new_vehicle(self) -> 'Vehicle':
        return Vehicle("NEW", (list(self.employees) or [None])[0], self)

    def __str__(self) -> 'str':
        return "%s(stock=%s, profit=%s, name=%s)" % (
            type(self).__name__,
            self.stock,
            self.profit,
            self.name
        )
```

==> cml-modules/cml_base/tests/associations/bidirectional/expected/vehicles_pop/setup.py

```
"""A setuptools based setup module.

See:
https://packaging.python.org/en/latest/distributing.html
https://github.com/pypa/sampleproject
"""

# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
```

```
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
    name='cml_associations_bidirectional_vehicles_pop',

    # Versions should comply with PEP440.  For a discussion on single-sourcing
    # the version across setup.py and the project code, see
    # https://packaging.python.org/en/latest/single_source_version.html
    version='1.0',

    description='A sample Python project',

    # The project's main homepage.
    url='https://github.com/pypa/sampleproject',

    # Author details
    author='The Python Packaging Authority',
    author_email='pypa-dev@googlegroups.com',

    # Choose your license
    license='MIT',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',

        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: MIT License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3.6.1',
    ],

    # What does your project relate to?
    keywords='sample setuptools development',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests']),

    # Alternatively, if you want to distribute just a my_module.py, uncomment
    # this:
    #   py_modules=["my_module"],

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    # install_requires=['peppercorn'],

    # List additional groups of dependencies here (e.g. development
    # dependencies). You can install these using the following syntax,
    # for example:
    # $ pip install -e .[dev,test]
    extras_require={
        'dev': ['check-manifest'],
        'test': ['coverage'],
    },

    # If there are data files included in your packages that need to be
    # installed, specify them here.  If using Python 2.6 or less, then these
    # have to be included in MANIFEST.in as well.
    # package_data={
    #     'sample': ['package_data.dat'],
    #},

    # Although 'package_data' is the preferred approach, in some case you may
    # need to place data files outside of your packages. See:
    # http://docs.python.org/3.4/distutils/setupscript.html#installing-additional-files # noqa
    # In this case, 'data_file' will be installed into '<sys.prefix>/my_data'
    #data_files=[('my_data', ['data/data_file'])],

    # To provide executable scripts, use entry points in preference to the
    # "scripts" keyword. Entry points provide cross-platform support and allow
    # pip to create the appropriate form of executable for the target platform.
    #entry_points={
    #    'console_scripts': [
    #        'sample=sample:main',
    #    ],
    #},
)
```

==> cml-modules/cml_base/tests/associations/bidirectional/source/main.cml

```
@concept Vehicle
{
    plate: String;
    driver: Employee?;
    owner: Organization;
}

@concept Employee
{
    name: String;
    employer: Organization;
    vehicle: Vehicle?;
}

@abstraction Organization
{
    name: String;
    employees: Employee*;
    fleet: Vehicle*;
}

@concept Corporation: Organization
{
    stock: Boolean = true;
    profit: Boolean = true;

    /myself = self;

    /new_vehicle = Vehicle("NEW", first(employees), self);
}
```

```
@association Employment
{
    Employee.employer;
    Organization.employees;
}

@association VehicleOwnership
{
    Vehicle.owner: Organization;
    Organization.fleet: Vehicle*;
}

@association VehicleAssignment
{
    Vehicle.driver: Employee?;
    Employee.vehicle: Vehicle?;
}

@task vehicles_poj: poj
{
    groupId = "cml-associations-bidirectional";
    artifactId = "vehicles-poj";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "vehicles.poj";
    packagePath = "vehicles/poj";
}

@task vehicles_pop: pop
{
    moduleName = "cml_associations_bidirectional_vehicles_pop";
    moduleVersion = "1.0";
}

@task vehicles_cmlc_java: cmlc_java
{
    groupId = "cml-associations-bidirectional";
    artifactId = "vehicles-cmlc";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "vehicles.cmlc";
    packagePath = "vehicles/cmlc";
}

@task vehicles_cmlc_py: cmlc_py
{
    moduleName = "cml_associations_bidirectional_vehicles_cmlc";
    moduleVersion = "1.0";
}



==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_cmlc_java/expected-client-output.txt

Unidirectional Associations (cmlc_java)

Employee: Employee(name="John")
Organization: Organization(name="Acme")
Organization's Employees: [Employee(name="John")]
Vehicle: Vehicle(plate="ABC12345", driver="Optional[Employee(name="John")]", owner="Organization(name="Acme")")


==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_cmlc_java/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-associations-unidirectional</groupId>
    <artifactId>vehicles-cmlc-client</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <jar.name>vehicles-cmlc-client</jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml-associations-unidirectional</groupId>
            <artifactId>vehicles-cmlc</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${jar.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>vehicles.client.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>expressions-console-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_cmlc_java/src/main/java/vehicles/client/Launcher.java

```
package vehicles.client;

import vehicles.cmlc.Employee;
import vehicles.cmlc.Organization;
import vehicles.cmlc.Vehicle;

import static java.util.Collections.singletonList;

public class Launcher
{
    public static void main(final String[] args)
    {
        System.out.println("Unidirectional Associations (cmlc_java)\n");

        final Employee employee = Employee.createEmployee("John");
        final Organization organization = Organization.createOrganization("Acme", singletonList(employee));
        final Vehicle vehicle = Vehicle.createVehicle("ABC12345", employee, organization);

        System.out.println("Employee: " + employee);
        System.out.println("Organization: " + organization);
        System.out.println("Organization's Employees: " + organization.getEmployees());
        System.out.println("Vehicle: " + vehicle);
    }
}
```

==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_cmlc_py/client.py

```
from cml_associations_unidirectional_vehicles_cmlc import *

print("Unidirectional Associations (cmlc_py)\n")

employee = Employee.create_employee("John")
organization = Organization.create_organization("Acme", [employee])
vehicle = Vehicle.create_vehicle("ABC12345", employee, organization)

print("Employee: %s" % employee)
print("Organization: %s" % organization)
print("Organization's Employees: %s" % ','.join(map(str, organization.employees)))
print("Vehicle: %s" % vehicle)
```

==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_cmlc_py/expected-client-output.txt

```
Unidirectional Associations (cmlc_py)

Employee: EmployeeImpl(name=John)
Organization: OrganizationImpl(name=Acme)
Organization's Employees: EmployeeImpl(name=John)
Vehicle: VehicleImpl(plate=ABC12345, driver=EmployeeImpl(name=John), owner=OrganizationImpl(name=Acme))
```

==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_poj/expected-client-output.txt

```
Unidirectional Associations (poj)

Employee: Employee(name="John")
Organization: Organization(name="Acme")
Organization's Employees: [Employee(name="John")]
Vehicle: Vehicle(plate="ABC12345", driver="Optional[Employee(name="John")]", owner="Organization(name="Acme")")
```

==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_poj/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-associations-unidirectional</groupId>
    <artifactId>vehicles-poj-client</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <jar.name>vehicles-poj-client</jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml-associations-unidirectional</groupId>
            <artifactId>vehicles-poj</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${jar.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>vehicles.client.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>expressions-console-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_poj/src/main/java/vehicles/client/Launcher.java

```
package vehicles.client;

import vehicles.poj.Employee;
import vehicles.poj.Organization;
import vehicles.poj.Vehicle;

import static java.util.Collections.singletonList;

public class Launcher
{
    public static void main(final String[] args)
    {
        System.out.println("Unidirectional Associations (poj)\n");

        final Employee employee = new Employee("John");
        final Organization organization = new Organization("Acme", singletonList(employee));
        final Vehicle vehicle = new Vehicle("ABC12345", employee, organization);

        System.out.println("Employee: " + employee);
        System.out.println("Organization: " + organization);
        System.out.println("Organization's Employees: " + organization.getEmployees());
        System.out.println("Vehicle: " + vehicle);
    }
}
```

==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_pop/client.py

```
from cml_associations_unidirectional_vehicles_pop import *

print("Unidirectional Associations (pop)\n")

employee = Employee("John")
organization = Organization("Acme", [employee])
vehicle = Vehicle("ABC12345", employee, organization)

print("Employee: %s" % employee)
print("Organization: %s" % organization)
print("Organization's Employees: %s" % ','.join(map(str, organization.employees)))
print("Vehicle: %s" % vehicle)
```

==> cml-modules/cml_base/tests/associations/unidirectional/clients/vehicles_pop/expected-client-output.txt

```
Unidirectional Associations (pop)

Employee: Employee(name=John)
Organization: Organization(name=Acme)
Organization's Employees: Employee(name=John)
Vehicle: Vehicle(plate=ABC12345, driver=Employee(name=John), owner=Organization(name=Acme))
```

==> cml-modules/cml_base/tests/associations/unidirectional/expected/vehicles_cmlc_java/cml-compiler-output.txt

```
model files:
- pom.xml

VEHICLE files:
- src/main/java/vehicles/cmlc/Vehicle.java

ORGANIZATION files:
- src/main/java/vehicles/cmlc/Organization.java

EMPLOYEE files:
- src/main/java/vehicles/cmlc/Employee.java
```

==> cml-modules/cml_base/tests/associations/unidirectional/expected/vehicles_cmlc_java/ignored-list.txt

```
__ALL__
```

==> cml-modules/cml_base/tests/associations/unidirectional/expected/vehicles_cmlc_py/cml-compiler-output.txt

```
model files:
- setup.py
- cml_associations_unidirectional_vehicles_cmlc/__init__.py
```

==> cml-modules/cml_base/tests/associations/unidirectional/expected/vehicles_cmlc_py/ignored-list.txt

```
setup.py
cml_associations_unidirectional_vehicles_cmlc/__init__.py
```

==> cml-modules/cml_base/tests/associations/unidirectional/expected/vehicles_poj/cml-compiler-output.txt

```
model files:
- pom.xml

VEHICLE files:
- src/main/java/vehicles/poj/Vehicle.java

ORGANIZATION files:
- src/main/java/vehicles/poj/Organization.java

EMPLOYEE files:
- src/main/java/vehicles/poj/Employee.java
```

==> cml-modules/cml_base/tests/associations/unidirectional/expected/vehicles_poj/ignored-list.txt

```
__ALL__
```

==> cml-modules/cml_base/tests/associations/unidirectional/expected/vehicles_pop/cml-compiler-output.txt

```
model files:
- setup.py
- cml_associations_unidirectional_vehicles_pop/__init__.py
```

==> cml-modules/cml_base/tests/associations/unidirectional/expected/vehicles_pop/ignored-list.txt

```
setup.py
cml_associations_unidirectional_vehicles_pop/__init__.py
```

==> cml-modules/cml_base/tests/associations/unidirectional/source/main.cml

```
@concept VEHICLE
{
    plate: STRING;
```

```
    driver: EMPLOYEE?;
    owner: ORGANIZATION;
}

@concept ORGANIZATION
{
    name: STRING;
    employees: EMPLOYEE*;
}

@concept EMPLOYEE
{
    name: STRING;
}

@task vehicles_poj: poj
{
    groupId = "cml-associations-unidirectional";
    artifactId = "vehicles-poj";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "vehicles.poj";
    packagePath = "vehicles/poj";
}

@task vehicles_pop: pop
{
    moduleName = "cml_associations_unidirectional_vehicles_pop";
    moduleVersion = "1.0";
}

@task vehicles_cmlc_java: cmlc_java
{
    groupId = "cml-associations-unidirectional";
    artifactId = "vehicles-cmlc";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "vehicles.cmlc";
    packagePath = "vehicles/cmlc";
}

@task vehicles_cmlc_py: cmlc_py
{
    moduleName = "cml_associations_unidirectional_vehicles_cmlc";
    moduleVersion = "1.0";
}



==> cml-modules/cml_base/tests/expressions/functions/expected/poj/cml-compiler-output.txt

model files:
- pom.xml

Functions files:
- src/main/java/functions/poj/Functions.java

Item files:
- src/main/java/functions/poj/Item.java


==> cml-modules/cml_base/tests/expressions/functions/expected/poj/ignored-list.txt

pom.xml

==> cml-modules/cml_base/tests/expressions/functions/expected/poj/src/main/java/functions/poj/Functions.java

package functions.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Functions
{
    private final Item requiredItem;
    private final @Nullable Item singleItem;
    private final List<Item> items;
    private final List<Item> items2;

    public Functions(Item requiredItem, @Nullable Item singleItem, List<Item> items, List<Item> items2)
    {
        this.requiredItem = requiredItem;
        this.singleItem = singleItem;
        this.items = items;
        this.items2 = items2;
    }

    public Item getRequiredItem()
    {
        return this.requiredItem;
    }

    public Optional<Item> getSingleItem()
    {
        return Optional.ofNullable(this.singleItem);
    }

    public List<Item> getItems()
    {
        return Collections.unmodifiableList(this.items);
    }

    public List<Item> getItems2()
    {
        return Collections.unmodifiableList(this.items2);
    }

    public boolean isEmptyItems()
    {
        return this.getItems().isEmpty();
    }

    public boolean isPresentItems()
    {
        return !this.getItems().isEmpty();
    }

    public boolean isEmptySingleItem()
    {
        return !this.getSingleItem().isPresent();
    }
```

```
public boolean isPresentSingleItem()
{
    return this.getSingleItem().isPresent();
}

public boolean isRequiredEmptySingleItem()
{
    return this.getRequiredItem() == null;
}

public boolean isRequiredPresentSingleItem()
{
    return this.getRequiredItem() != null;
}

public Optional<Item> getItemsFirst()
{
    return seq(this.getItems()).findFirst();
}

public Optional<Item> getItemsLast()
{
    return seq(this.getItems()).findLast();
}

public Optional<Item> getSingleItemFirst()
{
    return seq(this.getSingleItem()).findFirst();
}

public Optional<Item> getSingleItemLast()
{
    return seq(this.getSingleItem()).findLast();
}

public Optional<Item> getRequiredItemFirst()
{
    return Seq.of(this.getRequiredItem()).findFirst();
}

public Optional<Item> getRequiredItemLast()
{
    return Seq.of(this.getRequiredItem()).findLast();
}

public boolean isAtLeastOneLargeItem()
{
    return seq(this.getItems()).anyMatch(item1 -> (item1.getSize() > 100));
}

public boolean isAllLargeItems()
{
    return seq(this.getItems()).allMatch(item2 -> (item2.getSize() > 100));
}

public boolean isNoneLargeItems()
{
    return seq(this.getItems()).noneMatch(item3 -> (item3.getSize() > 100));
}

public boolean isLargeItemExists()
{
    return seq(this.getSingleItem()).anyMatch(item4 -> (item4.getSize() > 100));
}

public boolean isLargeItemAll()
{
    return seq(this.getSingleItem()).allMatch(item5 -> (item5.getSize() > 100));
}

public boolean isLargeItemNone()
{
    return seq(this.getSingleItem()).noneMatch(item6 -> (item6.getSize() > 100));
}

public boolean isRequiredItemExists()
{
    return Seq.of(this.getRequiredItem()).anyMatch(item7 -> (item7.getSize() > 100));
}

public boolean isRequiredItemAll()
{
    return Seq.of(this.getRequiredItem()).allMatch(item8 -> (item8.getSize() > 100));
}

public boolean isRequiredItemNone()
{
    return Seq.of(this.getRequiredItem()).noneMatch(item9 -> (item9.getSize() > 100));
}

public List<Item> getItemsSelect()
{
    return seq(this.getItems()).filter(item10 -> (item10.getSize() > 100)).toList();
}

public List<Item> getItemsReject()
{
    return seq(this.getItems()).removeAll(seq(this.getItems()).filter(item11 -> (item11.getSize() > 100))).toList();
}

public List<Item> getSingleItemSelect()
{
    return seq(this.getSingleItem()).filter(item12 -> (item12.getSize() > 100)).toList();
}

public List<Item> getSingleItemReject()
{
    return seq(this.getSingleItem()).removeAll(seq(this.getSingleItem()).filter(item13 -> (item13.getSize() > 100))).toList();
}

public List<Item> getRequiredItemSelect()
{
    return Seq.of(this.getRequiredItem()).filter(item14 -> (item14.getSize() > 100)).toList();
}

public List<Item> getRequiredItemReject()
{
    return Seq.of(this.getRequiredItem()).removeAll(Seq.of(this.getRequiredItem()).filter(item15 -> (item15.getSize() > 100))).toList();
}

public List<Item> getItemsCollect()
{
    return seq(this.getItems()).map(item16 -> seq(item16.getSubItem()).cast(Item.class).findFirst().get()).toList();
}

public List<Item> getSingleItemCollect()
```

```
    {
        return seq(this.getSingleItem()).map(item17 -> seq(item17.getSubItem()).cast(Item.class).findFirst().get()).toList();
    }

    public List<Item> getRequiredItemCollect()
    {
        return Seq.of(this.getRequiredItem()).map(item18 -> seq(item18.getSubItem()).cast(Item.class).findFirst().get()).toList();
    }

    public List<Item> getSortedItems()
    {
        return seq(this.getItems()).sorted((i1, i2) -> (i1.getSize() < i2.getSize()) ? -1 : ((i2.getSize() < i1.getSize()) ? +1 : 0)).toList();
    }

    public List<Item> getReversedItems()
    {
        return seq(this.getItems()).reverse().toList();
    }

    public Item getNewItem()
    {
        return new Item(12, null);
    }

    public List<Item> getConcatItems()
    {
        return concat(seq(this.getItems()).toList(), seq(this.getItems2()).toList()).toList();
    }

    public long getCountItems()
    {
        return seq(this.getItems()).count();
    }

    public List<Item> getDistinctItems()
    {
        return seq(this.getItems()).distinct().toList();
    }

    public List<Integer> getItemsSizeCollect()
    {
        return seq(this.getItems()).map(item19 -> item19.getSize()).toList();
    }

    public List<Integer> getSingleItemSizeCollect()
    {
        return seq(this.getSingleItem()).map(item20 -> item20.getSize()).toList();
    }

    public List<Integer> getRequiredItemSizeCollect()
    {
        return Seq.of(this.getRequiredItem()).map(item21 -> item21.getSize()).toList();
    }

    public String toString()
    {
        return new StringBuilder(Functions.class.getSimpleName())
                    .append('(')
                    .append("requiredItem=").append(String.format("\"%s\"", this.getRequiredItem())).append(", ")
                    .append("singleItem=").append(this.getSingleItem().isPresent() ? String.format("\"%s\"", this.getSingleItem()) : "not present").
                    .append("emptyItems=").append(String.format("\"%s\"", this.isEmptyItems())).append(", ")
                    .append("presentItems=").append(String.format("\"%s\"", this.isPresentItems())).append(", ")
                    .append("emptySingleItem=").append(String.format("\"%s\"", this.isEmptySingleItem())).append(", ")
                    .append("presentSingleItem=").append(String.format("\"%s\"", this.isPresentSingleItem())).append(", ")
                    .append("requiredEmptySingleItem=").append(String.format("\"%s\"", this.isRequiredEmptySingleItem())).append(", ")
                    .append("requiredPresentSingleItem=").append(String.format("\"%s\"", this.isRequiredPresentSingleItem())).append(", ")
                    .append("atLeastOneLargeItem=").append(String.format("\"%s\"", this.isAtLeastOneLargeItem())).append(", ")
                    .append("allLargeItems=").append(String.format("\"%s\"", this.isAllLargeItems())).append(", ")
                    .append("noneLargeItems=").append(String.format("\"%s\"", this.isNoneLargeItems())).append(", ")
                    .append("largeItemExists=").append(String.format("\"%s\"", this.isLargeItemExists())).append(", ")
                    .append("largeItemAll=").append(String.format("\"%s\"", this.isLargeItemAll())).append(", ")
                    .append("largeItemNone=").append(String.format("\"%s\"", this.isLargeItemNone())).append(", ")
                    .append("requiredItemExists=").append(String.format("\"%s\"", this.isRequiredItemExists())).append(", ")
                    .append("requiredItemAll=").append(String.format("\"%s\"", this.isRequiredItemAll())).append(", ")
                    .append("requiredItemNone=").append(String.format("\"%s\"", this.isRequiredItemNone())).append(", ")
                    .append("countItems=").append(String.format("\"%s\"", this.getCountItems())).append(", ")
                    .append("itemsSizeCollect=").append(this.getItemsSizeCollect()).append(", ")
                    .append("singleItemSizeCollect=").append(this.getSingleItemSizeCollect()).append(", ")
                    .append("requiredItemSizeCollect=").append(this.getRequiredItemSizeCollect())
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/functions/expected/poj/src/main/java/functions/poj/Item.java

```
package functions.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Item
{
    private final int size;
    private final @Nullable Item subItem;

    public Item(int size, @Nullable Item subItem)
    {
        this.size = size;
        this.subItem = subItem;
    }

    public int getSize()
    {
        return this.size;
    }

    public Optional<Item> getSubItem()
    {
        return Optional.ofNullable(this.subItem);
    }

    public String toString()
    {
        return new StringBuilder(Item.class.getSimpleName())
                    .append('(')
                    .append("size=").append(String.format("\"%s\"", this.getSize())).append(", ")
                    .append("subItem=").append(this.getSubItem().isPresent() ? String.format("\"%s\"", this.getSubItem()) : "not present")
                    .append(')')
```

```
                .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/functions/expected/pop/cml-compiler-output.txt

```
model files:
- setup.py
- cml_expressions_functions_pop/__init__.py
```

==> cml-modules/cml_base/tests/expressions/functions/expected/pop/cml_expressions_functions_pop/__init__.py

```
from typing import *
from abc import *
from decimal import *

import functools, itertools

class Item:

    def __init__(self, size: 'int', sub_item: 'Optional[Item]') -> 'None':
        self.__size = size
        self.__sub_item = sub_item

    @property
    def size(self) -> 'int':
        return self.__size

    @property
    def sub_item(self) -> 'Optional[Item]':
        return self.__sub_item

    def __str__(self) -> 'str':
        return "%s(size=%s, sub_item=%s)" % (
            type(self).__name__,
            self.size,
            self.sub_item
        )


class Functions:

    def __init__(self, required_item: 'Item', single_item: 'Optional[Item]', items: 'List[Item]', items_2: 'List[Item]') -> 'None':
        self.__required_item = required_item
        self.__single_item = single_item
        self.__items = items
        self.__items_2 = items_2

    @property
    def required_item(self) -> 'Item':
        return self.__required_item

    @property
    def single_item(self) -> 'Optional[Item]':
        return self.__single_item

    @property
    def items(self) -> 'List[Item]':
        return self.__items

    @property
    def items_2(self) -> 'List[Item]':
        return self.__items_2

    @property
    def empty_items(self) -> 'bool':
        return (len(self.items) == 0)

    @property
    def present_items(self) -> 'bool':
        return (len(self.items) > 0)

    @property
    def empty_single_item(self) -> 'bool':
        return (self.single_item is None)

    @property
    def present_single_item(self) -> 'bool':
        return (self.single_item is not None)

    @property
    def required_empty_single_item(self) -> 'bool':
        return (self.required_item is None)

    @property
    def required_present_single_item(self) -> 'bool':
        return (self.required_item is not None)

    @property
    def items_first(self) -> 'Optional[Item]':
        return (list(self.items) or [None])[0]

    @property
    def items_last(self) -> 'Optional[Item]':
        return ((self.items) or [None])[-1]

    @property
    def single_item_first(self) -> 'Optional[Item]':
        return self.single_item

    @property
    def single_item_last(self) -> 'Optional[Item]':
        return self.single_item

    @property
    def required_item_first(self) -> 'Optional[Item]':
        return self.required_item

    @property
    def required_item_last(self) -> 'Optional[Item]':
        return self.required_item

    @property
    def at_least_one_large_item(self) -> 'bool':
        return any(map(lambda item: (item.size > 100), self.items))

    @property
    def all_large_items(self) -> 'bool':
        return all(map(lambda item: (item.size > 100), self.items))

    @property
    def none_large_items(self) -> 'bool':
        return (len(list(filter(lambda item: (item.size > 100), self.items))) == 0)

    @property
    def large_item_exists(self) -> 'bool':
```

```
    return any(map(lambda item: (item.size > 100), [] if self.single_item is None else [self.single_item]))

@property
def large_item_all(self) -> 'bool':
    return all(map(lambda item: (item.size > 100), [] if self.single_item is None else [self.single_item]))

@property
def large_item_none(self) -> 'bool':
    return (len(list(filter(lambda item: (item.size > 100), [] if self.single_item is None else [self.single_item]))) == 0)

@property
def required_item_exists(self) -> 'bool':
    return any(map(lambda item: (item.size > 100), [self.required_item]))

@property
def required_item_all(self) -> 'bool':
    return all(map(lambda item: (item.size > 100), [self.required_item]))

@property
def required_item_none(self) -> 'bool':
    return (len(list(filter(lambda item: (item.size > 100), [self.required_item]))) == 0)

@property
def items_select(self) -> 'List[Item]':
    return list(
        filter(lambda item: (item.size > 100), self.items)
    )

@property
def items_reject(self) -> 'List[Item]':
    return list(
        itertools.filterfalse(lambda item: (item.size > 100), self.items)
    )

@property
def single_item_select(self) -> 'List[Item]':
    return list(
        filter(lambda item: (item.size > 100), [] if self.single_item is None else [self.single_item])
    )

@property
def single_item_reject(self) -> 'List[Item]':
    return list(
        itertools.filterfalse(lambda item: (item.size > 100), [] if self.single_item is None else [self.single_item])
    )

@property
def required_item_select(self) -> 'List[Item]':
    return list(
        filter(lambda item: (item.size > 100), [self.required_item])
    )

@property
def required_item_reject(self) -> 'List[Item]':
    return list(
        itertools.filterfalse(lambda item: (item.size > 100), [self.required_item])
    )

@property
def items_collect(self) -> 'List[Item]':
    return list(
        map(lambda item: cast('Item', item.sub_item), self.items)
    )

@property
def single_item_collect(self) -> 'List[Item]':
    return list(
        map(lambda item: cast('Item', item.sub_item), [] if self.single_item is None else [self.single_item])
    )

@property
def required_item_collect(self) -> 'List[Item]':
    return list(
        map(lambda item: cast('Item', item.sub_item), [self.required_item])
    )

@property
def sorted_items(self) -> 'List[Item]':
    return list(
        sorted(self.items, key=functools.cmp_to_key(lambda i_1, i_2: -1 if (i_1.size < i_2.size) else +1 if (i_2.size < i_1.size) else 0))
    )

@property
def reversed_items(self) -> 'List[Item]':
    return list(
        reversed(self.items)
    )

@property
def new_item(self) -> 'Item':
    return Item(12, None)

@property
def concat_items(self) -> 'List[Item]':
    return list(
        (self.items + self.items_2)
    )

@property
def count_items(self) -> 'int':
    return len(self.items)

@property
def distinct_items(self) -> 'List[Item]':
    return list(
        functools.reduce(lambda l, x: l if x in l else list(l)+[x], self.items, [])
    )

@property
def items_size_collect(self) -> 'List[int]':
    return list(
        map(lambda item: item.size, self.items)
    )

@property
def single_item_size_collect(self) -> 'List[int]':
    return list(
        map(lambda item: item.size, [] if self.single_item is None else [self.single_item])
    )

@property
def required_item_size_collect(self) -> 'List[int]':
    return list(
        map(lambda item: item.size, [self.required_item])
    )

def __str__(self) -> 'str':
```

```
        return "%s(required_item=%s, single_item=%s, empty_items=%s, present_items=%s, empty_single_item=%s, present_single_item=%s, required_empty
            type(self).__name__,
            self.required_item,
            self.single_item,
            self.empty_items,
            self.present_items,
            self.empty_single_item,
            self.present_single_item,
            self.required_empty_single_item,
            self.required_present_single_item,
            self.at_least_one_large_item,
            self.all_large_items,
            self.none_large_items,
            self.large_item_exists,
            self.large_item_all,
            self.large_item_none,
            self.required_item_exists,
            self.required_item_all,
            self.required_item_none,
            self.count_items,
            self.items_size_collect,
            self.single_item_size_collect,
            self.required_item_size_collect
        )
```

==> cml-modules/cml_base/tests/expressions/functions/expected/pop/setup.py

```
"""A setuptools based setup module.

See:
https://packaging.python.org/en/latest/distributing.html
https://github.com/pypa/sampleproject
"""

# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
    name='cml_expressions_functions_pop',

    # Versions should comply with PEP440.  For a discussion on single-sourcing
    # the version across setup.py and the project code, see
    # https://packaging.python.org/en/latest/single_source_version.html
    version='1.0',

    description='A sample Python project',

    # The project's main homepage.
    url='https://github.com/pypa/sampleproject',

    # Author details
    author='The Python Packaging Authority',
    author_email='pypa-dev@googlegroups.com',

    # Choose your license
    license='MIT',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',

        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: MIT License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3.6.1',
    ],

    # What does your project relate to?
    keywords='sample setuptools development',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests']),

    # Alternatively, if you want to distribute just a my_module.py, uncomment
    # this:
    #   py_modules=["my_module"],

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    # install_requires=['peppercorn'],

    # List additional groups of dependencies here (e.g. development
    # dependencies). You can install these using the following syntax,
    # for example:
    # $ pip install -e .[dev,test]
    extras_require={
        'dev': ['check-manifest'],
        'test': ['coverage'],
    },

    # If there are data files included in your packages that need to be
    # installed, specify them here.  If using Python 2.6 or less, then these
    # have to be included in MANIFEST.in as well.
    # package_data={
    #     'sample': ['package_data.dat'],
    #},

    # Although 'package_data' is the preferred approach, in some case you may
    # need to place data files outside of your packages. See:
    # http://docs.python.org/3.4/distutils/setupscript.html#installing-additional-files # noqa
    # In this case, 'data_file' will be installed into '<sys.prefix>/my_data'
    #data_files=[('my_data', ['data/data_file'])],

    # To provide executable scripts, use entry points in preference to the
    # "scripts" keyword. Entry points provide cross-platform support and allow
    # pip to create the appropriate form of executable for the target platform.
    #entry_points={
    #    'console_scripts': [
    #        'sample=sample:main',
```

```
    #    ],
    #},
)

==> cml-modules/cml_base/tests/expressions/functions/source/main.cml


@concept Functions
{
    required_item: Item;
    single_item: Item?;
    items: Item*; items2: Item*;

    /empty_items = empty(items);
    /present_items = present(items);

    /empty_single_item = empty(single_item);
    /present_single_item = present(single_item);

    /required_empty_single_item = empty(required_item);
    /required_present_single_item = present(required_item);

    /items_first = first(items);
    /items_last = last(items);

    /single_item_first = first(single_item);
    /single_item_last = last(single_item);

    /required_item_first = first(required_item);
    /required_item_last = last(required_item);

    /at_least_one_large_item = items | exists: size > 100;
    /all_large_items = items | all: size > 100;
    /none_large_items = items | none: size > 100;

    /large_item_exists = single_item | exists: size > 100;
    /large_item_all = single_item | all: size > 100;
    /large_item_none = single_item | none: size > 100;

    /required_item_exists = required_item | exists: size > 100;
    /required_item_all = required_item | all: size > 100;
    /required_item_none = required_item | none: size > 100;

    /items_select = items | select: size > 100;
    /items_reject = items | reject: size > 100;

    /single_item_select = single_item | select: size > 100;
    /single_item_reject = single_item | reject: size > 100;

    /required_item_select = required_item | select: size > 100;
    /required_item_reject = required_item | reject: size > 100;

    /items_collect = items | collect: subItem as! Item;
    /single_item_collect = single_item | collect: subItem as! Item;
    /required_item_collect = required_item | collect: subItem as! Item;

    /sorted_items = items | sort: i1, i2 -> compare(i1.size, i2.size);
    /reversed_items = reverse(items);

    /new_item = Item(12, none);

    /concat_items = concat(items, items2);

    /count_items = count(items);
    /distinct_items = distinct(items);

    /items_size_collect = items | collect: size;
    /single_item_size_collect = single_item | collect: size;
    /required_item_size_collect = required_item | collect: size;
}

@concept Item
{
    size: integer;
    subItem: Item?;
}

@task poj: poj
{
    groupId = "cml-expressions";
    artifactId = "cml-expressions-functions";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "functions.poj";
    packagePath = "functions/poj";
}

@task pop: pop
{
    moduleName = "cml_expressions_functions_pop";
    moduleVersion = "1.0";
}


==> cml-modules/cml_base/tests/expressions/let/expected/poj/cml-compiler-output.txt

model files:
- pom.xml

LetExpressions files:
- src/main/java/let/poj/LetExpressions.java


==> cml-modules/cml_base/tests/expressions/let/expected/poj/ignored-list.txt

pom.xml

==> cml-modules/cml_base/tests/expressions/let/expected/poj/src/main/java/let/poj/LetExpressions.java

package let.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class LetExpressions
{
    public int getSquare()
    {
        return new Supplier<Integer>() {
                    public Integer get()
```

```
                {
                    final int a = 2;
                    return (a * a);
                }
            }.get();
    }

    public String toString()
    {
        return new StringBuilder(LetExpressions.class.getSimpleName())
                .append('(')
                .append("square=").append(String.format("\"%s\"", this.getSquare()))
                .append(')')
                .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/let/source/main.cml

```
@concept LetExpressions
{
  /square = let a = 2 in a * a;
}

@task poj: poj
{
    groupId = "cml-expressions";
    artifactId = "cml-expressions-let";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "let.poj";
    packagePath = "let/poj";
}
```

==> cml-modules/cml_base/tests/expressions/literals/clients/literals_poj/expected-client-output.txt

```
Literal Expressions

LiteralTrueBoolean = true
LiteralFalseBoolean = false
LiteralStringInit =     Some "String"

LiteralIntegerInit = 123
LiteralDecimalInit = 123.456
LiteralDecimalInit2 = 0.456
```

==> cml-modules/cml_base/tests/expressions/literals/clients/literals_poj/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-expressions</groupId>
    <artifactId>cml-expressions-literals-client</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <jar.name>${project.artifactId}</jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml-expressions</groupId>
            <artifactId>cml-expressions-literals</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${jar.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>literals.client.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>literals-client-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

==> cml-modules/cml_base/tests/expressions/literals/clients/literals_poj/src/main/java/literals/client/Launcher.java

```
package literals.client;

import literals.poj.LiteralExpressions;

public class Launcher
{
    public static void main(final String[] args)
    {
        System.out.println("Literal Expressions");
        System.out.println();
```

```
        final LiteralExpressions expressions = new LiteralExpressions();

        System.out.println("LiteralTrueBoolean = " + expressions.isLiteralTrueBoolean());
        System.out.println("LiteralFalseBoolean = " + expressions.isLiteralFalseBoolean());
        System.out.println("LiteralStringInit =  " + expressions.getLiteralStringInit());
        System.out.println("LiteralIntegerInit = " + expressions.getLiteralIntegerInit());
        System.out.println("LiteralDecimalInit = " + expressions.getLiteralDecimalInit());
        System.out.println("LiteralDecimalInit2 = " + expressions.getLiteralDecimalInit2());
    }
}
```

==> cml-modules/cml_base/tests/expressions/literals/clients/literals_pop/client.py

```
from cml_expressions_literals_pop import *

expressions = LiteralExpressions()

print("Expression Literals\n")

print("literal_true_boolean = %s" % expressions.literal_true_boolean)
print("literal_false_boolean = %s" % expressions.literal_false_boolean)
print("literal_string_init = %s" % expressions.literal_string_init)
print("literal_integer_init = %d" % expressions.literal_integer_init)
print("literal_decimal_init = %.3f" % expressions.literal_decimal_init)
print("literal_decimal_init_2 = %.3f" % expressions.literal_decimal_init_2)
```

==> cml-modules/cml_base/tests/expressions/literals/clients/literals_pop/expected-client-output.txt

```
Expression Literals

literal_true_boolean = True
literal_false_boolean = False
literal_string_init =   Some "String"

literal_integer_init = 123
literal_decimal_init = 123.456
literal_decimal_init_2 = 0.456
```

==> cml-modules/cml_base/tests/expressions/literals/expected/literals_poj/cml-compiler-output.txt

```
model files:
- pom.xml

LITERAL_EXPRESSIONS files:
- src/main/java/literals/poj/LiteralExpressions.java
```

==> cml-modules/cml_base/tests/expressions/literals/expected/literals_poj/src/main/java/literals/poj/LiteralExpressions.java

```
package literals.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class LiteralExpressions
{
    private final boolean literalTrueBoolean;
    private final boolean literalFalseBoolean;
    private final String literalStringInit;
    private final int literalIntegerInit;
    private final BigDecimal literalDecimalInit;
    private final BigDecimal literalDecimalInit2;

    public LiteralExpressions()
    {
        this(true, false, "\tSome \"String\"\n", 123, new BigDecimal("123.456"), new BigDecimal(".456"));
    }

    public LiteralExpressions(boolean literalTrueBoolean, boolean literalFalseBoolean, String literalStringInit, int literalIntegerInit, BigDecimal
    {
        this.literalTrueBoolean = literalTrueBoolean;
        this.literalFalseBoolean = literalFalseBoolean;
        this.literalStringInit = literalStringInit;
        this.literalIntegerInit = literalIntegerInit;
        this.literalDecimalInit = literalDecimalInit;
        this.literalDecimalInit2 = literalDecimalInit2;
    }

    public boolean isLiteralTrueBoolean()
    {
        return this.literalTrueBoolean;
    }

    public boolean isLiteralFalseBoolean()
    {
        return this.literalFalseBoolean;
    }

    public String getLiteralStringInit()
    {
        return this.literalStringInit;
    }

    public int getLiteralIntegerInit()
    {
        return this.literalIntegerInit;
    }

    public BigDecimal getLiteralDecimalInit()
    {
        return this.literalDecimalInit;
    }

    public BigDecimal getLiteralDecimalInit2()
    {
        return this.literalDecimalInit2;
    }

    public boolean isComparedStrings()
    {
        return (Objects.equals(this.getLiteralStringInit(), "another string"));
    }

    public String getStrConcat()
    {
```

```
        return (this.getLiteralStringInit() + "another string");
    }

    public String getStrConcat2()
    {
        return ((this.getLiteralStringInit() + Objects.toString(321)) + Objects.toString(this.getLiteralDecimalInit()));
    }

    public String getStrConcat3()
    {
        return (Objects.toString(321) + Objects.toString(this.getLiteralDecimalInit()));
    }

    public String getStrConcat4()
    {
        return (this.getLiteralStringInit() + "another string");
    }

    public String toString()
    {
        return new StringBuilder(LiteralExpressions.class.getSimpleName())
                    .append('(')
                    .append("literalTrueBoolean=").append(String.format("\"%s\"", this.isLiteralTrueBoolean())).append(", ")
                    .append("literalFalseBoolean=").append(String.format("\"%s\"", this.isLiteralFalseBoolean())).append(", ")
                    .append("literalStringInit=").append(String.format("\"%s\"", this.getLiteralStringInit())).append(", ")
                    .append("literalIntegerInit=").append(String.format("\"%s\"", this.getLiteralIntegerInit())).append(", ")
                    .append("literalDecimalInit=").append(String.format("\"%s\"", this.getLiteralDecimalInit())).append(", ")
                    .append("literalDecimalInit2=").append(String.format("\"%s\"", this.getLiteralDecimalInit2())).append(", ")
                    .append("comparedStrings=").append(String.format("\"%s\"", this.isComparedStrings())).append(", ")
                    .append("strConcat=").append(String.format("\"%s\"", this.getStrConcat())).append(", ")
                    .append("strConcat2=").append(String.format("\"%s\"", this.getStrConcat2())).append(", ")
                    .append("strConcat3=").append(String.format("\"%s\"", this.getStrConcat3())).append(", ")
                    .append("strConcat4=").append(String.format("\"%s\"", this.getStrConcat4()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/literals/expected/literals_pop/cml-compiler-output.txt

```
model files:
- setup.py
- cml_expressions_literals_pop/__init__.py
```

==> cml-modules/cml_base/tests/expressions/literals/expected/literals_pop/cml_expressions_literals_pop/__init__.py

```
from typing import *
from abc import *
from decimal import *

import functools, itertools

class LiteralExpressions:

    def __init__(self, literal_true_boolean: 'bool' = True, literal_false_boolean: 'bool' = False, literal_string_init: 'str' = "\tSome \"String\"\
        self.__literal_true_boolean = literal_true_boolean
        self.__literal_false_boolean = literal_false_boolean
        self.__literal_string_init = literal_string_init
        self.__literal_integer_init = literal_integer_init
        self.__literal_decimal_init = literal_decimal_init
        self.__literal_decimal_init_2 = literal_decimal_init_2

    @property
    def literal_true_boolean(self) -> 'bool':
        return self.__literal_true_boolean

    @property
    def literal_false_boolean(self) -> 'bool':
        return self.__literal_false_boolean

    @property
    def literal_string_init(self) -> 'str':
        return self.__literal_string_init

    @property
    def literal_integer_init(self) -> 'int':
        return self.__literal_integer_init

    @property
    def literal_decimal_init(self) -> 'Decimal':
        return self.__literal_decimal_init

    @property
    def literal_decimal_init_2(self) -> 'Decimal':
        return self.__literal_decimal_init_2

    @property
    def compared_strings(self) -> 'bool':
        return (self.literal_string_init == "another string")

    @property
    def str_concat(self) -> 'str':
        return (self.literal_string_init + "another string")

    @property
    def str_concat_2(self) -> 'str':
        return ((self.literal_string_init + str(321)) + str(self.literal_decimal_init))

    @property
    def str_concat_3(self) -> 'str':
        return (str(321) + str(self.literal_decimal_init))

    @property
    def str_concat_4(self) -> 'str':
        return (self.literal_string_init + "another string")

    def __str__(self) -> 'str':
        return "%s(literal_true_boolean=%s, literal_false_boolean=%s, literal_string_init=%s, literal_integer_init=%s, literal_decimal_init=%s, lit
            type(self).__name__,
            self.literal_true_boolean,
            self.literal_false_boolean,
            self.literal_string_init,
            self.literal_integer_init,
            self.literal_decimal_init,
            self.literal_decimal_init_2,
            self.compared_strings,
            self.str_concat,
            self.str_concat_2,
            self.str_concat_3,
            self.str_concat_4
        )
```

==> cml-modules/cml_base/tests/expressions/literals/source/main.cml

```
@concept LITERAL_EXPRESSIONS
```

```
{
    LiteralTrueBoolean = true;
    LiteralFalseBoolean = false;
    LiteralStringInit = "\tSome \"String\"\n";
    LiteralIntegerInit = 123;
    LiteralDecimalInit = 123.456;
    LiteralDecimalInit2 = .456;

    /compared_strings = LiteralStringInit == "another string";
    /str_concat = LiteralStringInit & "another string";
    /str_concat2 = LiteralStringInit & 321 & LiteralDecimalInit;
    /str_concat3 = 321 & LiteralDecimalInit;
    /str_concat4 = LiteralStringInit & "another string";
}

@task literals_poj: poj
{
    groupId = "cml-expressions";
    artifactId = "cml-expressions-literals";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "literals.poj";
    packagePath = "literals/poj";
}

@task literals_pop: pop
{
    moduleName = "cml_expressions_literals_pop";
    moduleVersion = "1.0";
}


==> cml-modules/cml_base/tests/expressions/paths/clients/paths_poj/expected-client-output.txt

Paths Client (poj)

self_var = ExpressionCases(foo="foo", somePath="SomeConcept(value="-1", bar="Bar()", oneMorePath="AnotherConcept(etc="Etc()", flag="true")")", sing
single_var = foo

path_var = Bar()
path_var2 = Etc()

path_var3 = [Etc(), Etc()]
path_bars = [Bar(), Bar()]
path_foos = [AnotherConcept(etc="Etc()", flag="true"), AnotherConcept(etc="Etc()", flag="true"), AnotherConcept(etc="Etc()", flag="true"), AnotherC

somePathList = [SomeConcept(value="2", bar="Bar()", oneMorePath="AnotherConcept(etc="Etc()", flag="true")"), SomeConcept(value="-1", bar="Bar()", o
sorted_list = [SomeConcept(value="-1", bar="Bar()", oneMorePath="AnotherConcept(etc="Etc()", flag="true")"), SomeConcept(value="2", bar="Bar()", on


==> cml-modules/cml_base/tests/expressions/paths/clients/paths_poj/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-expressions</groupId>
    <artifactId>cml-expressions-paths-client</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <jar.name>${project.artifactId}</jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml-expressions</groupId>
            <artifactId>cml-expressions-paths</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${jar.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>paths.client.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>paths-client-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>


==> cml-modules/cml_base/tests/expressions/paths/clients/paths_poj/src/main/java/paths/client/Launcher.java

package paths.client;

import paths.poj.*;

import static java.util.Arrays.asList;

public class Launcher
{
    public static void main(final String[] args)
    {
```

```
        System.out.println("Paths Client (poj)");
        System.out.println();

        final AnotherConcept anotherConcept = new AnotherConcept(new Etc());
        final AnotherConcept anotherConcept2 = new AnotherConcept(new Etc());
        final SomeConcept someConcept = new SomeConcept(-1, new Bar(), asList(anotherConcept, anotherConcept2), anotherConcept);
        final SomeConcept someConcept2 = new SomeConcept(2, new Bar(), asList(anotherConcept2, anotherConcept), anotherConcept2);
        final ExpressionCases cases = new ExpressionCases("foo", someConcept, asList(someConcept2, someConcept), null);

        System.out.println("self_var = " + cases.getSelfVar());
        System.out.println("single_var = " + cases.getSingleVar());
        System.out.println();

        System.out.println("path_var = " + cases.getPathVar());
        System.out.println("path_var2 = " + cases.getPathVar2());
        System.out.println();

        System.out.println("path_var3 = " + cases.getPathVar3());
        System.out.println("path_bars = " + cases.getPathBars());
        System.out.println("path_foos = " + cases.getPathFoos());
        System.out.println();

        System.out.println("somePathList = " + cases.getSomePathList());
        System.out.println("sorted_list = " + cases.getSortedList());
    }
}
```

==> cml-modules/cml_base/tests/expressions/paths/clients/paths_pop/client.py

```
from cml_expressions_paths_pop import *

anotherConcept = AnotherConcept(Etc())
anotherConcept2 = AnotherConcept(Etc())
someConcept = SomeConcept(-1, Bar(), [anotherConcept, anotherConcept2], anotherConcept)
someConcept2 = SomeConcept(2, Bar(), [anotherConcept2, anotherConcept], anotherConcept2)
cases = ExpressionCases("foo", someConcept, [someConcept2, someConcept], None)

print("Paths Client (pop)\n")

print("self_var = " + str(cases.self_var))
print("single_var = " + cases.single_var)
print()

print("path_var = " + str(cases.path_var))
print("path_var_2 = " + str(cases.path_var_2))
print()

print("path_var_3 = " + str(list(map(lambda item: str(item), cases.path_var_3))))
print("path_bars = " + str(list(map(lambda item: str(item), cases.path_bars))))
print("path_foos = " + str(list(map(lambda item: str(item), cases.path_foos))))

print("some_path_list = " + str(list(map(lambda item: str(item), cases.some_path_list))))
print("sorted_list = " + str(list(map(lambda item: str(item), cases.sorted_list))))
```

==> cml-modules/cml_base/tests/expressions/paths/clients/paths_pop/expected-client-output.txt

```
Paths Client (pop)

self_var = ExpressionCases(foo=foo, some_path=SomeConcept(value=-1, bar=Bar(), one_more_path=AnotherConcept(etc=Etc(), flag=True)), single_var=foo,
single_var = foo

path_var = Bar()
path_var_2 = Etc()

path_var_3 = ['Etc()', 'Etc()']
path_bars = ['Bar()', 'Bar()']
path_foos = ['AnotherConcept(etc=Etc(), flag=True)', 'AnotherConcept(etc=Etc(), flag=True)', 'AnotherConcept(etc=Etc(), flag=True)', 'AnotherConcep
some_path_list = ['SomeConcept(value=2, bar=Bar(), one_more_path=AnotherConcept(etc=Etc(), flag=True))', 'SomeConcept(value=-1, bar=Bar(), one_more
sorted_list = ['SomeConcept(value=-1, bar=Bar(), one_more_path=AnotherConcept(etc=Etc(), flag=True))', 'SomeConcept(value=2, bar=Bar(), one_more_pa
```

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_java/cml-compiler-output.txt

```
model files:
- pom.xml

ExpressionCases files:
- src/main/java/paths/cmlc/ExpressionCases.java

SomeConcept files:
- src/main/java/paths/cmlc/SomeConcept.java

AnotherConcept files:
- src/main/java/paths/cmlc/AnotherConcept.java

Bar files:
- src/main/java/paths/cmlc/Bar.java

Etc files:
- src/main/java/paths/cmlc/Etc.java
```

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_java/pom.xml

```
<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-expressions</groupId>
    <artifactId>cml-expressions-paths-cmlc</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>15.0</version>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-source-plugin</artifactId>
              <version>2.4</version>
              <executions>
                <execution>
                  <id>attach-sources</id>
                  <goals>
                    <goal>jar</goal>
                  </goals>
                </execution>
              </executions>
            </plugin>
        </plugins>
    </build>

</project>

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_java/src/main/java/paths/cmlc/AnotherConcept.java

package paths.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface AnotherConcept
{
    Etc getEtc();

    boolean isFlag();

    static AnotherConcept createAnotherConcept(Etc etc)
    {
        return createAnotherConcept(etc, true);
    }

    static AnotherConcept createAnotherConcept(Etc etc, boolean flag)
    {
        return new AnotherConceptImpl(null, etc, flag);
    }

    static AnotherConcept extendAnotherConcept(@Nullable AnotherConcept actual_self, Etc etc, boolean flag)
    {
        return new AnotherConceptImpl(actual_self, etc, flag);
    }
}

class AnotherConceptImpl implements AnotherConcept
{
    private final @Nullable AnotherConcept actual_self;

    private final Etc etc;
    private final boolean flag;

    AnotherConceptImpl(@Nullable AnotherConcept actual_self, Etc etc, boolean flag)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.etc = etc;
        this.flag = flag;
    }

    public Etc getEtc()
    {
        return this.etc;
    }

    public boolean isFlag()
    {
        return this.flag;
    }

    public String toString()
    {
        return new StringBuilder(AnotherConcept.class.getSimpleName())
                    .append('(')
                    .append("etc=").append(String.format("\"%s\"", this.actual_self.getEtc())).append(", ")
                    .append("flag=").append(String.format("\"%s\"", this.actual_self.isFlag()))
                    .append(')')
                    .toString();
    }
}

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_java/src/main/java/paths/cmlc/Bar.java

package paths.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Bar {}

class BarImpl implements Bar
{
    private final @Nullable Bar actual_self;

    BarImpl(@Nullable Bar actual_self)
    {
        this.actual_self = actual_self == null ? this : actual_self;
```

```
    }

    public String toString()
    {
        return new StringBuilder(Bar.class.getSimpleName())
                    .append('(')
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_java/src/main/java/paths/cmlc/Etc.java

```
package paths.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Etc {}

class EtcImpl implements Etc
{
    private final @Nullable Etc actual_self;

    EtcImpl(@Nullable Etc actual_self)
    {
        this.actual_self = actual_self == null ? this : actual_self;


    }

    public String toString()
    {
        return new StringBuilder(Etc.class.getSimpleName())
                    .append('(')
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_java/src/main/java/paths/cmlc/ExpressionCases.java

```
package paths.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface ExpressionCases
{
    String getFoo();

    SomeConcept getSomePath();

    List<SomeConcept> getSomePathList();

    ExpressionCases getSelfVar();

    String getSingleVar();

    SomeConcept getDerivedSomePath();

    Bar getPathVar();

    Etc getPathVar2();

    List<Etc> getPathVar3();

    List<Bar> getPathBars();

    List<AnotherConcept> getPathFoos();

    List<SomeConcept> getSortedList();

    Optional<AnotherConcept> getOptProp();

    boolean isOptFlag();

    Optional<SomeConcept> getNoneProp();

    static ExpressionCases createExpressionCases(String foo, SomeConcept somePath, List<SomeConcept> somePathList, @Nullable AnotherConcept optProp
    {
        return new ExpressionCasesImpl(null, foo, somePath, somePathList, optProp);
    }

    static ExpressionCases extendExpressionCases(@Nullable ExpressionCases actual_self, String foo, SomeConcept somePath, List<SomeConcept> somePat
    {
        return new ExpressionCasesImpl(actual_self, foo, somePath, somePathList, optProp);
    }
}

class ExpressionCasesImpl implements ExpressionCases
{
    private final @Nullable ExpressionCases actual_self;

    private final String foo;
    private final SomeConcept somePath;
    private final List<SomeConcept> somePathList;
    private final @Nullable AnotherConcept optProp;

    ExpressionCasesImpl(@Nullable ExpressionCases actual_self, String foo, SomeConcept somePath, List<SomeConcept> somePathList, @Nullable AnotherC
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.foo = foo;
        this.somePath = somePath;
        this.somePathList = somePathList;
        this.optProp = optProp;
    }

    public String getFoo()
```

```
    {
        return this.foo;
    }

    public SomeConcept getSomePath()
    {
        return this.somePath;
    }

    public List<SomeConcept> getSomePathList()
    {
        return Collections.unmodifiableList(this.somePathList);
    }

    public Optional<AnotherConcept> getOptProp()
    {
        return Optional.ofNullable(this.optProp);
    }

    public ExpressionCases getSelfVar()
    {
        return this.actual_self;
    }

    public String getSingleVar()
    {
        return this.actual_self.getFoo();
    }

    public SomeConcept getDerivedSomePath()
    {
        return this.actual_self.getSomePath();
    }

    public Bar getPathVar()
    {
        return this.actual_self.getSomePath().getBar();
    }

    public Etc getPathVar2()
    {
        return this.actual_self.getSomePath().getOneMorePath().getEtc();
    }

    public List<Etc> getPathVar3()
    {
        return seq(seq(this.actual_self.getSomePathList()).flatMap(someConcept -> seq(asList(someConcept.getOneMorePath())))).flatMap(anotherConcep
    }

    public List<Bar> getPathBars()
    {
        return seq(this.actual_self.getSomePathList()).flatMap(someConcept -> seq(asList(someConcept.getBar()))).toList();
    }

    public List<AnotherConcept> getPathFoos()
    {
        return seq(this.actual_self.getSomePathList()).flatMap(someConcept -> seq(someConcept.getFoos())).toList();
    }

    public List<SomeConcept> getSortedList()
    {
        return seq(this.actual_self.getSomePathList()).sorted((item1, item2) -> (item1.getValue() < item2.getValue()) ? -1 : ((item2.getValue() < i
    }

    public boolean isOptFlag()
    {
        return seq(this.actual_self.getOptProp()).anyMatch(item1 -> item1.isFlag());
    }

    public Optional<SomeConcept> getNoneProp()
    {
        return Optional.empty();
    }

    public String toString()
    {
        return new StringBuilder(ExpressionCases.class.getSimpleName())
                    .append('(')
                    .append("foo=").append(String.format("\"%s\"", this.actual_self.getFoo())).append(", ")
                    .append("somePath=").append(String.format("\"%s\"", this.actual_self.getSomePath())).append(", ")
                    .append("singleVar=").append(String.format("\"%s\"", this.actual_self.getSingleVar())).append(", ")
                    .append("optProp=").append(this.actual_self.getOptProp().isPresent() ? String.format("\"%s\"", this.actual_self.getOptProp()) :
                    .append("optFlag=").append(String.format("\"%s\"", this.actual_self.isOptFlag()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_java/src/main/java/paths/cmlc/SomeConcept.java

```
package paths.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface SomeConcept
{
    int getValue();

    Bar getBar();

    List<AnotherConcept> getFoos();

    AnotherConcept getOneMorePath();

    static SomeConcept createSomeConcept(int value, Bar bar, List<AnotherConcept> foos, AnotherConcept oneMorePath)
    {
        return new SomeConceptImpl(null, value, bar, foos, oneMorePath);
    }

    static SomeConcept extendSomeConcept(@Nullable SomeConcept actual_self, int value, Bar bar, List<AnotherConcept> foos, AnotherConcept oneMorePa
    {
        return new SomeConceptImpl(actual_self, value, bar, foos, oneMorePath);
    }
}

class SomeConceptImpl implements SomeConcept
{
    private final @Nullable SomeConcept actual_self;
```

```
    private final int value;
    private final Bar bar;
    private final List<AnotherConcept> foos;
    private final AnotherConcept oneMorePath;

    SomeConceptImpl(@Nullable SomeConcept actual_self, int value, Bar bar, List<AnotherConcept> foos, AnotherConcept oneMorePath)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.value = value;
        this.bar = bar;
        this.foos = foos;
        this.oneMorePath = oneMorePath;
    }

    public int getValue()
    {
        return this.value;
    }

    public Bar getBar()
    {
        return this.bar;
    }

    public List<AnotherConcept> getFoos()
    {
        return Collections.unmodifiableList(this.foos);
    }

    public AnotherConcept getOneMorePath()
    {
        return this.oneMorePath;
    }

    public String toString()
    {
        return new StringBuilder(SomeConcept.class.getSimpleName())
                    .append('(')
                    .append("value=").append(String.format("\"%s\"", this.actual_self.getValue())).append(", ")
                    .append("bar=").append(String.format("\"%s\"", this.actual_self.getBar())).append(", ")
                    .append("oneMorePath=").append(String.format("\"%s\"", this.actual_self.getOneMorePath()))
                    .append(')')
                    .toString();
    }
}
```

```
==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_py/cml-compiler-output.txt

model files:
- setup.py
- cml_expressions_paths_cmlc/__init__.py


==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_py/cml_expressions_paths_cmlc/__init__.py

from typing import *
from abc import *
from decimal import *

import functools, itertools

class AnotherConcept(ABC):

    @abstractproperty
    def etc(self) -> 'Etc':
        pass

    @abstractproperty
    def flag(self) -> 'bool':
        pass

    @staticmethod
    def create_another_concept(etc: 'Etc', flag: 'bool' = True) -> 'AnotherConcept':
        return AnotherConceptImpl(None, etc, flag)

    @staticmethod
    def extend_another_concept(actual_self: 'Optional[AnotherConcept]', etc: 'Etc', flag: 'bool' = True) -> 'AnotherConcept':
        return AnotherConceptImpl(actual_self, etc, flag)


class AnotherConceptImpl(AnotherConcept):

    def __init__(self, actual_self: 'Optional[AnotherConcept]', etc: 'Etc', flag: 'bool' = True) -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[AnotherConcept]
        else:
            self.__actual_self = actual_self

        self.__etc = etc
        self.__flag = flag

    @property
    def etc(self) -> 'Etc':
        return self.__etc

    @property
    def flag(self) -> 'bool':
        return self.__flag

    def __str__(self) -> 'str':
        return "%s(etc=%s, flag=%s)" % (
            type(self).__name__,
            self.__actual_self.etc,
            self.__actual_self.flag
        )


class Bar(ABC):
    pass


class BarImpl(Bar):

    def __init__(self, actual_self: 'Optional[Bar]') -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Bar]
        else:
            self.__actual_self = actual_self



    def __str__(self) -> 'str':
        return "%s()" % type(self).__name__
```

```
class Etc(ABC):
    pass


class EtcImpl(Etc):

    def __init__(self, actual_self: 'Optional[Etc]') -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Etc]
        else:
            self.__actual_self = actual_self



    def __str__(self) -> 'str':
        return "%s()" % type(self).__name__


class SomeConcept(ABC):

    @abstractproperty
    def value(self) -> 'int':
        pass

    @abstractproperty
    def bar(self) -> 'Bar':
        pass

    @abstractproperty
    def foos(self) -> 'List[AnotherConcept]':
        pass

    @abstractproperty
    def one_more_path(self) -> 'AnotherConcept':
        pass

    @staticmethod
    def create_some_concept(value: 'int', bar: 'Bar', foos: 'List[AnotherConcept]', one_more_path: 'AnotherConcept') -> 'SomeConcept':
        return SomeConceptImpl(None, value, bar, foos, one_more_path)

    @staticmethod
    def extend_some_concept(actual_self: 'Optional[SomeConcept]', value: 'int', bar: 'Bar', foos: 'List[AnotherConcept]', one_more_path: 'AnotherCo
        return SomeConceptImpl(actual_self, value, bar, foos, one_more_path)


class SomeConceptImpl(SomeConcept):

    def __init__(self, actual_self: 'Optional[SomeConcept]', value: 'int', bar: 'Bar', foos: 'List[AnotherConcept]', one_more_path: 'AnotherConcept
        if actual_self is None:
            self.__actual_self = self  # type: Optional[SomeConcept]
        else:
            self.__actual_self = actual_self

        self.__value = value
        self.__bar = bar
        self.__foos = foos
        self.__one_more_path = one_more_path

    @property
    def value(self) -> 'int':
        return self.__value

    @property
    def bar(self) -> 'Bar':
        return self.__bar

    @property
    def foos(self) -> 'List[AnotherConcept]':
        return self.__foos

    @property
    def one_more_path(self) -> 'AnotherConcept':
        return self.__one_more_path

    def __str__(self) -> 'str':
        return "%s(value=%s, bar=%s, one_more_path=%s)" % (
            type(self).__name__,
            self.__actual_self.value,
            self.__actual_self.bar,
            self.__actual_self.one_more_path
        )


class ExpressionCases(ABC):

    @abstractproperty
    def foo(self) -> 'str':
        pass

    @abstractproperty
    def some_path(self) -> 'SomeConcept':
        pass

    @abstractproperty
    def some_path_list(self) -> 'List[SomeConcept]':
        pass

    @abstractproperty
    def self_var(self) -> 'ExpressionCases':
        pass

    @abstractproperty
    def single_var(self) -> 'str':
        pass

    @abstractproperty
    def derived_some_path(self) -> 'SomeConcept':
        pass

    @abstractproperty
    def path_var(self) -> 'Bar':
        pass

    @abstractproperty
    def path_var_2(self) -> 'Etc':
        pass

    @abstractproperty
    def path_var_3(self) -> 'List[Etc]':
        pass

    @abstractproperty
    def path_bars(self) -> 'List[Bar]':
        pass

    @abstractproperty
    def path_foos(self) -> 'List[AnotherConcept]':
```

```
        pass

    @abstractproperty
    def sorted_list(self) -> 'List[SomeConcept]':
        pass

    @abstractproperty
    def opt_prop(self) -> 'Optional[AnotherConcept]':
        pass

    @abstractproperty
    def opt_flag(self) -> 'bool':
        pass

    @abstractproperty
    def none_prop(self) -> 'Optional[SomeConcept]':
        pass

    @staticmethod
    def create_expression_cases(foo: 'str', some_path: 'SomeConcept', some_path_list: 'List[SomeConcept]', opt_prop: 'Optional[AnotherConcept]') ->
        return ExpressionCasesImpl(None, foo, some_path, some_path_list, opt_prop)

    @staticmethod
    def extend_expression_cases(actual_self: 'Optional[ExpressionCases]', foo: 'str', some_path: 'SomeConcept', some_path_list: 'List[SomeConcept]'
        return ExpressionCasesImpl(actual_self, foo, some_path, some_path_list, opt_prop)


class ExpressionCasesImpl(ExpressionCases):

    def __init__(self, actual_self: 'Optional[ExpressionCases]', foo: 'str', some_path: 'SomeConcept', some_path_list: 'List[SomeConcept]', opt_pro
        if actual_self is None:
            self.__actual_self = self  # type: Optional[ExpressionCases]
        else:
            self.__actual_self = actual_self

        self.__foo = foo
        self.__some_path = some_path
        self.__some_path_list = some_path_list
        self.__opt_prop = opt_prop

    @property
    def foo(self) -> 'str':
        return self.__foo

    @property
    def some_path(self) -> 'SomeConcept':
        return self.__some_path

    @property
    def some_path_list(self) -> 'List[SomeConcept]':
        return self.__some_path_list

    @property
    def opt_prop(self) -> 'Optional[AnotherConcept]':
        return self.__opt_prop

    @property
    def self_var(self) -> 'ExpressionCases':
        return self.__actual_self

    @property
    def single_var(self) -> 'str':
        return self.__actual_self.foo

    @property
    def derived_some_path(self) -> 'SomeConcept':
        return self.__actual_self.some_path

    @property
    def path_var(self) -> 'Bar':
        return self.__actual_self.some_path.bar

    @property
    def path_var_2(self) -> 'Etc':
        return self.__actual_self.some_path.one_more_path.etc

    @property
    def path_var_3(self) -> 'List[Etc]':
        return list(
            map(
                lambda another_concept: another_concept.etc,
                map(
                    lambda some_concept: some_concept.one_more_path,
                    self.__actual_self.some_path_list
                )
            )
        )

    @property
    def path_bars(self) -> 'List[Bar]':
        return list(
            map(
                lambda some_concept: some_concept.bar,
                self.__actual_self.some_path_list
            )
        )

    @property
    def path_foos(self) -> 'List[AnotherConcept]':
        return list(
            itertools.chain.from_iterable(map(
                lambda some_concept: some_concept.foos,
                self.__actual_self.some_path_list
            ))
        )

    @property
    def sorted_list(self) -> 'List[SomeConcept]':
        return list(
            sorted(self.__actual_self.some_path_list, key=functools.cmp_to_key(lambda item_1, item_2: -1 if (item_1.value < item_2.value) else +1 i
        )

    @property
    def opt_flag(self) -> 'bool':
        return any(map(lambda item: item.flag, [] if self.__actual_self.opt_prop is None else [self.__actual_self.opt_prop]))

    @property
    def none_prop(self) -> 'Optional[SomeConcept]':
        return None

    def __str__(self) -> 'str':
        return "%s(foo=%s, some_path=%s, single_var=%s, opt_prop=%s, opt_flag=%s)" % (
            type(self).__name__,
            self.__actual_self.foo,
            self.__actual_self.some_path,
            self.__actual_self.single_var,
            self.__actual_self.opt_prop,
```

```
            self.__actual_self.opt_flag
        )

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_cmlc_py/setup.py

"""A setuptools based setup module.

See:
https://packaging.python.org/en/latest/distributing.html
https://github.com/pypa/sampleproject
"""

# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
    name='cml_expressions_paths_cmlc',

    # Versions should comply with PEP440.  For a discussion on single-sourcing
    # the version across setup.py and the project code, see
    # https://packaging.python.org/en/latest/single_source_version.html
    version='1.0',

    description='A sample Python project',

    # The project's main homepage.
    url='https://github.com/pypa/sampleproject',

    # Author details
    author='The Python Packaging Authority',
    author_email='pypa-dev@googlegroups.com',

    # Choose your license
    license='MIT',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',

        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: MIT License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3.6.1',
    ],

    # What does your project relate to?
    keywords='sample setuptools development',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests']),

    # Alternatively, if you want to distribute just a my_module.py, uncomment
    # this:
    #   py_modules=["my_module"],

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    # install_requires=['peppercorn'],

    # List additional groups of dependencies here (e.g. development
    # dependencies). You can install these using the following syntax,
    # for example:
    # $ pip install -e .[dev,test]
    extras_require={
        'dev': ['check-manifest'],
        'test': ['coverage'],
    },

    # If there are data files included in your packages that need to be
    # installed, specify them here.  If using Python 2.6 or less, then these
    # have to be included in MANIFEST.in as well.
    # package_data={
    #     'sample': ['package_data.dat'],
    #},

    # Although 'package_data' is the preferred approach, in some case you may
    # need to place data files outside of your packages. See:
    # http://docs.python.org/3.4/distutils/setupscript.html#installing-additional-files # noqa
    # In this case, 'data_file' will be installed into '<sys.prefix>/my_data'
    #data_files=[('my_data', ['data/data_file'])],

    # To provide executable scripts, use entry points in preference to the
    # "scripts" keyword. Entry points provide cross-platform support and allow
    # pip to create the appropriate form of executable for the target platform.
    #entry_points={
    #    'console_scripts': [
    #        'sample=sample:main',
    #    ],
    #},
)

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_poj/cml-compiler-output.txt

model files:
- pom.xml

ExpressionCases files:
- src/main/java/paths/poj/ExpressionCases.java

SomeConcept files:
- src/main/java/paths/poj/SomeConcept.java

AnotherConcept files:
- src/main/java/paths/poj/AnotherConcept.java

Bar files:
- src/main/java/paths/poj/Bar.java

Etc files:
```

```
- src/main/java/paths/poj/Etc.java


==> cml-modules/cml_base/tests/expressions/paths/expected/paths_poj/pom.xml

<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-expressions</groupId>
    <artifactId>cml-expressions-paths</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>15.0</version>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-source-plugin</artifactId>
              <version>2.4</version>
              <executions>
                <execution>
                  <id>attach-sources</id>
                  <goals>
                    <goal>jar</goal>
                  </goals>
                </execution>
              </executions>
            </plugin>
        </plugins>
    </build>

</project>

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_poj/src/main/java/paths/poj/AnotherConcept.java

package paths.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class AnotherConcept
{
    private final Etc etc;
    private final boolean flag;

    public AnotherConcept(Etc etc)
    {
        this(etc, true);
    }

    public AnotherConcept(Etc etc, boolean flag)
    {
        this.etc = etc;
        this.flag = flag;
    }

    public Etc getEtc()
    {
        return this.etc;
    }

    public boolean isFlag()
    {
        return this.flag;
    }

    public String toString()
    {
        return new StringBuilder(AnotherConcept.class.getSimpleName())
                    .append('(')
                    .append("etc=").append(String.format("\"%s\"", this.getEtc())).append(", ")
                    .append("flag=").append(String.format("\"%s\"", this.isFlag()))
                    .append(')')
                    .toString();
    }
}

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_poj/src/main/java/paths/poj/Bar.java

package paths.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
```

```
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Bar
{
    public String toString()
    {
        return new StringBuilder(Bar.class.getSimpleName())
                    .append('(')
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_poj/src/main/java/paths/poj/Etc.java

```
package paths.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Etc
{
    public String toString()
    {
        return new StringBuilder(Etc.class.getSimpleName())
                    .append('(')
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_poj/src/main/java/paths/poj/ExpressionCases.java

```
package paths.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class ExpressionCases
{
    private final String foo;
    private final SomeConcept somePath;
    private final List<SomeConcept> somePathList;
    private final @Nullable AnotherConcept optProp;

    public ExpressionCases(String foo, SomeConcept somePath, List<SomeConcept> somePathList, @Nullable AnotherConcept optProp)
    {
        this.foo = foo;
        this.somePath = somePath;
        this.somePathList = somePathList;
        this.optProp = optProp;
    }

    public String getFoo()
    {
        return this.foo;
    }

    public SomeConcept getSomePath()
    {
        return this.somePath;
    }

    public List<SomeConcept> getSomePathList()
    {
        return Collections.unmodifiableList(this.somePathList);
    }

    public Optional<AnotherConcept> getOptProp()
    {
        return Optional.ofNullable(this.optProp);
    }

    public ExpressionCases getSelfVar()
    {
        return this;
    }

    public String getSingleVar()
    {
        return this.getFoo();
    }

    public SomeConcept getDerivedSomePath()
    {
        return this.getSomePath();
    }

    public Bar getPathVar()
    {
        return this.getSomePath().getBar();
    }

    public Etc getPathVar2()
    {
        return this.getSomePath().getOneMorePath().getEtc();
    }

    public List<Etc> getPathVar3()
    {
        return seq(seq(this.getSomePathList()).flatMap(someConcept -> seq(asList(someConcept.getOneMorePath())))).flatMap(anotherConcept -> seq(asL
    }

    public List<Bar> getPathBars()
    {
        return seq(this.getSomePathList()).flatMap(someConcept -> seq(asList(someConcept.getBar()))).toList();
    }

    public List<AnotherConcept> getPathFoos()
```

```
    {
        return seq(this.getSomePathList()).flatMap(someConcept -> seq(someConcept.getFoos())).toList();
    }

    public List<SomeConcept> getSortedList()
    {
        return seq(this.getSomePathList()).sorted((item1, item2) -> (item1.getValue() < item2.getValue()) ? -1 : ((item2.getValue() < item1.getValu
    }

    public boolean isOptFlag()
    {
        return seq(this.getOptProp()).anyMatch(item1 -> item1.isFlag());
    }

    public Optional<SomeConcept> getNoneProp()
    {
        return Optional.empty();
    }

    public String toString()
    {
        return new StringBuilder(ExpressionCases.class.getSimpleName())
                    .append('(')
                    .append("foo=").append(String.format("\"%s\"", this.getFoo())).append(", ")
                    .append("somePath=").append(String.format("\"%s\"", this.getSomePath())).append(", ")
                    .append("singleVar=").append(String.format("\"%s\"", this.getSingleVar())).append(", ")
                    .append("optProp=").append(this.getOptProp().isPresent() ? String.format("\"%s\"", this.getOptProp()) : "not present").append(",
                    .append("optFlag=").append(String.format("\"%s\"", this.isOptFlag()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_poj/src/main/java/paths/poj/SomeConcept.java

```
package paths.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class SomeConcept
{
    private final int value;
    private final Bar bar;
    private final List<AnotherConcept> foos;
    private final AnotherConcept oneMorePath;

    public SomeConcept(int value, Bar bar, List<AnotherConcept> foos, AnotherConcept oneMorePath)
    {
        this.value = value;
        this.bar = bar;
        this.foos = foos;
        this.oneMorePath = oneMorePath;
    }

    public int getValue()
    {
        return this.value;
    }

    public Bar getBar()
    {
        return this.bar;
    }

    public List<AnotherConcept> getFoos()
    {
        return Collections.unmodifiableList(this.foos);
    }

    public AnotherConcept getOneMorePath()
    {
        return this.oneMorePath;
    }

    public String toString()
    {
        return new StringBuilder(SomeConcept.class.getSimpleName())
                    .append('(')
                    .append("value=").append(String.format("\"%s\"", this.getValue())).append(", ")
                    .append("bar=").append(String.format("\"%s\"", this.getBar())).append(", ")
                    .append("oneMorePath=").append(String.format("\"%s\"", this.getOneMorePath()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_pop/cml-compiler-output.txt

```
model files:
- setup.py
- cml_expressions_paths_pop/__init__.py


==> cml-modules/cml_base/tests/expressions/paths/expected/paths_pop/cml_expressions_paths_pop/__init__.py

from typing import *
from abc import *
from decimal import *

import functools, itertools

class AnotherConcept:

    def __init__(self, etc: 'Etc', flag: 'bool' = True) -> 'None':
        self.__etc = etc
        self.__flag = flag

    @property
    def etc(self) -> 'Etc':
        return self.__etc

    @property
    def flag(self) -> 'bool':
        return self.__flag

    def __str__(self) -> 'str':
        return "%s(etc=%s, flag=%s)" % (
            type(self).__name__,
```

```
            self.etc,
            self.flag
        )


class Bar:

    def __str__(self) -> 'str':
        return "%s()" % type(self).__name__


class Etc:

    def __str__(self) -> 'str':
        return "%s()" % type(self).__name__


class SomeConcept:

    def __init__(self, value: 'int', bar: 'Bar', foos: 'List[AnotherConcept]', one_more_path: 'AnotherConcept') -> 'None':
        self.__value = value
        self.__bar = bar
        self.__foos = foos
        self.__one_more_path = one_more_path

    @property
    def value(self) -> 'int':
        return self.__value

    @property
    def bar(self) -> 'Bar':
        return self.__bar

    @property
    def foos(self) -> 'List[AnotherConcept]':
        return self.__foos

    @property
    def one_more_path(self) -> 'AnotherConcept':
        return self.__one_more_path

    def __str__(self) -> 'str':
        return "%s(value=%s, bar=%s, one_more_path=%s)" % (
            type(self).__name__,
            self.value,
            self.bar,
            self.one_more_path
        )


class ExpressionCases:

    def __init__(self, foo: 'str', some_path: 'SomeConcept', some_path_list: 'List[SomeConcept]', opt_prop: 'Optional[AnotherConcept]') -> 'None':
        self.__foo = foo
        self.__some_path = some_path
        self.__some_path_list = some_path_list
        self.__opt_prop = opt_prop

    @property
    def foo(self) -> 'str':
        return self.__foo

    @property
    def some_path(self) -> 'SomeConcept':
        return self.__some_path

    @property
    def some_path_list(self) -> 'List[SomeConcept]':
        return self.__some_path_list

    @property
    def opt_prop(self) -> 'Optional[AnotherConcept]':
        return self.__opt_prop

    @property
    def self_var(self) -> 'ExpressionCases':
        return self

    @property
    def single_var(self) -> 'str':
        return self.foo

    @property
    def derived_some_path(self) -> 'SomeConcept':
        return self.some_path

    @property
    def path_var(self) -> 'Bar':
        return self.some_path.bar

    @property
    def path_var_2(self) -> 'Etc':
        return self.some_path.one_more_path.etc

    @property
    def path_var_3(self) -> 'List[Etc]':
        return list(
            map(
                lambda another_concept: another_concept.etc,
                map(
                    lambda some_concept: some_concept.one_more_path,
                    self.some_path_list
                )
            )
        )

    @property
    def path_bars(self) -> 'List[Bar]':
        return list(
            map(
                lambda some_concept: some_concept.bar,
                self.some_path_list
            )
        )

    @property
    def path_foos(self) -> 'List[AnotherConcept]':
        return list(
            itertools.chain.from_iterable(map(
                lambda some_concept: some_concept.foos,
                self.some_path_list
            ))
        )

    @property
    def sorted_list(self) -> 'List[SomeConcept]':
        return list(
```

```
            sorted(self.some_path_list, key=functools.cmp_to_key(lambda item_1, item_2: -1 if (item_1.value < item_2.value) else +1 if (item_2.valu
        )

    @property
    def opt_flag(self) -> 'bool':
        return any(map(lambda item: item.flag, [] if self.opt_prop is None else [self.opt_prop]))

    @property
    def none_prop(self) -> 'Optional[SomeConcept]':
        return None

    def __str__(self) -> 'str':
        return "%s(foo=%s, some_path=%s, single_var=%s, opt_prop=%s, opt_flag=%s)" % (
            type(self).__name__,
            self.foo,
            self.some_path,
            self.single_var,
            self.opt_prop,
            self.opt_flag
        )

==> cml-modules/cml_base/tests/expressions/paths/expected/paths_pop/setup.py

"""A setuptools based setup module.

See:
https://packaging.python.org/en/latest/distributing.html
https://github.com/pypa/sampleproject
"""

# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
    name='cml_expressions_paths_pop',

    # Versions should comply with PEP440.  For a discussion on single-sourcing
    # the version across setup.py and the project code, see
    # https://packaging.python.org/en/latest/single_source_version.html
    version='1.0',

    description='A sample Python project',

    # The project's main homepage.
    url='https://github.com/pypa/sampleproject',

    # Author details
    author='The Python Packaging Authority',
    author_email='pypa-dev@googlegroups.com',

    # Choose your license
    license='MIT',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',

        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: MIT License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3.6.1',
    ],

    # What does your project relate to?
    keywords='sample setuptools development',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests']),

    # Alternatively, if you want to distribute just a my_module.py, uncomment
    # this:
    #   py_modules=["my_module"],

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    # install_requires=['peppercorn'],

    # List additional groups of dependencies here (e.g. development
    # dependencies). You can install these using the following syntax,
    # for example:
    # $ pip install -e .[dev,test]
    extras_require={
        'dev': ['check-manifest'],
        'test': ['coverage'],
    },

    # If there are data files included in your packages that need to be
    # installed, specify them here.  If using Python 2.6 or less, then these
    # have to be included in MANIFEST.in as well.
    # package_data={
    #     'sample': ['package_data.dat'],
    #},

    # Although 'package_data' is the preferred approach, in some case you may
    # need to place data files outside of your packages. See:
    # http://docs.python.org/3.4/distutils/setupscript.html#installing-additional-files # noqa
    # In this case, 'data_file' will be installed into '<sys.prefix>/my_data'
    #data_files=[('my_data', ['data/data_file'])],

    # To provide executable scripts, use entry points in preference to the
    # "scripts" keyword. Entry points provide cross-platform support and allow
    # pip to create the appropriate form of executable for the target platform.
    #entry_points={
    #    'console_scripts': [
    #        'sample=sample:main',
    #    ],
    #},
)
```

```
==> cml-modules/cml_base/tests/expressions/paths/source/main.cml


@concept ExpressionCases
{
    // Used by paths below:
    foo: String;
    somePath: SomeConcept;
    somePathList: SomeConcept*;

    // Path-derived properties:
    /self_var = self;
    /single_var = foo;
    /derived_some_path = self.somePath;
    /path_var = somePath.bar;
    /path_var2 = somePath.oneMorePath.etc;
    /path_var3 = somePathList.oneMorePath.etc;
    /path_bars = somePathList.bar;
    /path_foos = somePathList.foos;

    /sorted_list = somePathList | sort: item1, item2 -> compare(item1.value, item2.value);

    opt_prop: AnotherConcept?;
    /opt_flag = opt_prop | exists: flag;

    /none_prop: SomeConcept? = none;
}

@concept SomeConcept
{
    value: integer;
    bar: Bar;
    foos: AnotherConcept*;
    oneMorePath: AnotherConcept;
}

@concept AnotherConcept
{
    etc: Etc;
    flag: boolean = true;
}

@concept Bar;
@concept Etc;

@task paths_poj: poj
{
    groupId = "cml-expressions";
    artifactId = "cml-expressions-paths";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "paths.poj";
    packagePath = "paths/poj";
}

@task paths_cmlc_java: cmlc_java
{
    groupId = "cml-expressions";
    artifactId = "cml-expressions-paths-cmlc";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "paths.cmlc";
    packagePath = "paths/cmlc";
}

@task paths_pop: pop
{
    moduleName = "cml_expressions_paths_pop";
    moduleVersion = "1.0";
}

@task paths_cmlc_py: cmlc_py
{
    moduleName = "cml_expressions_paths_cmlc";
    moduleVersion = "1.0";
}


==> cml-modules/cml_base/tests/expressions/type_validations/expected/type_validations/cml-compiler-errors.txt


Failed validation: required property_type_specified_or_inferred: in property Types.req_to_req_type_cast_asq: UNDEFINED (11:5)
Cannot use 'as?' to cast to required-cardinality type:
- left operand is 'req: Ancestor'
- right operand is 'Descendant'

Failed validation: required property_type_specified_or_inferred: in property Types.opt_to_req_type_cast_asq: UNDEFINED (12:5)
Cannot use 'as?' to cast to required-cardinality type:
- left operand is 'opt: Ancestor?'
- right operand is 'Descendant'

Failed validation: required property_type_specified_or_inferred: in property Types.seq_to_req_type_cast_asq: UNDEFINED (13:5)
Cannot use 'as?' to cast to required-cardinality type:
- left operand is 'seq: Ancestor*'
- right operand is 'Descendant'

Failed validation: required property_type_specified_or_inferred: in property Types.seq_to_req_type_cast_asb: UNDEFINED (17:5)
Cannot cast from sequence type to required-cardinality type:
- left operand is 'seq: Ancestor*'
- right operand is 'Descendant'

Failed validation: required property_type_specified_or_inferred: in property Types.seq_to_opt_type_cast_asb: UNDEFINED (25:5)
Cannot cast from sequence type to optional-cardinality type:
- left operand is 'seq: Ancestor*'
- right operand is 'Descendant?'

Failed validation: required property_type_specified_or_inferred: in property Types.seq_to_opt_type_cast_asq: UNDEFINED (26:5)
Cannot cast from sequence type to optional-cardinality type:
- left operand is 'seq: Ancestor*'
- right operand is 'Descendant?'


==> cml-modules/cml_base/tests/expressions/type_validations/source/main.cml


@concept Types
{
    req: Ancestor;
    opt: Ancestor?;
    seq: Ancestor*;

    /type_check_is = req is Descendant;
    /type_check_is_not = req isnt Descendant;

    /req_to_req_type_cast_asq = req as? Descendant; // <- error: cannot use 'as?' to cast to required type
    /opt_to_req_type_cast_asq = opt as? Descendant; // <- error: cannot use 'as?' to cast to required type
    /seq_to_req_type_cast_asq = seq as? Descendant; // <- error: cannot use 'as?' to cast to required type

    /req_to_req_type_cast_asb = req as! Descendant;
```

```
    /opt_to_req_type_cast_asb = opt as! Descendant;
    /seq_to_req_type_cast_asb = seq as! Descendant; // <- error: cannot cast from sequence to required type

    /req_to_opt_type_cast_asb = req as! Descendant?;
    /req_to_opt_type_cast_asq = req as? Descendant?;

    /opt_to_opt_type_cast_asb = opt as! Descendant?;
    /opt_to_opt_type_cast_asq = opt as? Descendant?;

    /seq_to_opt_type_cast_asb = seq as! Descendant?; // <- error: cannot cast from sequence to optional
    /seq_to_opt_type_cast_asq = seq as? Descendant?; // <- error: cannot cast from sequence to optional

    /req_to_seq_type_cast_asb = req as! Descendant*;
    /opt_to_seq_type_cast_asb = opt as! Descendant*;
    /seq_to_seq_type_cast_asb = seq as! Descendant*;

    /req_to_seq_type_cast_asq = req as? Descendant*;
    /opt_to_seq_type_cast_asq = opt as? Descendant*;
    /seq_to_seq_type_cast_asq = seq as? Descendant*;

    /descendants = for a in seq | select: a is Descendant;

    flag: Boolean?;
    ints: Integer*;
}

@concept Ancestor;

@concept Descendant: Ancestor;

@task type_validations: poj
{
    groupId = "cml-expressions";
    artifactId = "cml-expressions-type-validations";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "types.poj";
    packagePath = "types/poj";
}


==> cml-modules/cml_base/tests/expressions/types/expected/poj/cml-compiler-output.txt

model files:
- pom.xml

Types files:
- src/main/java/types/poj/Types.java

Ancestor files:
- src/main/java/types/poj/Ancestor.java

Descendant files:
- src/main/java/types/poj/Descendant.java


==> cml-modules/cml_base/tests/expressions/types/expected/poj/ignored-list.txt

pom.xml

==> cml-modules/cml_base/tests/expressions/types/expected/poj/src/main/java/types/poj/Ancestor.java

package types.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Ancestor
{
    public String toString()
    {
        return new StringBuilder(Ancestor.class.getSimpleName())
                    .append('(')
                    .append(')')
                    .toString();
    }
}

==> cml-modules/cml_base/tests/expressions/types/expected/poj/src/main/java/types/poj/Descendant.java

package types.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Descendant extends Ancestor
{
    public String toString()
    {
        return new StringBuilder(Descendant.class.getSimpleName())
                    .append('(')
                    .append(')')
                    .toString();
    }
}

==> cml-modules/cml_base/tests/expressions/types/expected/poj/src/main/java/types/poj/Types.java

package types.poj;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Types
```

```
{
    private final Ancestor req;
    private final @Nullable Ancestor opt;
    private final List<Ancestor> seq;

    public Types(Ancestor req, @Nullable Ancestor opt, List<Ancestor> seq)
    {
        this.req = req;
        this.opt = opt;
        this.seq = seq;
    }

    public Ancestor getReq()
    {
        return this.req;
    }

    public Optional<Ancestor> getOpt()
    {
        return Optional.ofNullable(this.opt);
    }

    public List<Ancestor> getSeq()
    {
        return Collections.unmodifiableList(this.seq);
    }

    public boolean isTypeCheckIs()
    {
        return this.getReq() instanceof Descendant;
    }

    public boolean isTypeCheckIsNot()
    {
        return !(this.getReq() instanceof Descendant);
    }

    public Descendant getReqToReqTypeCastAsb()
    {
        return (Descendant)this.getReq();
    }

    public Descendant getOptToReqTypeCastAsb()
    {
        return seq(this.getOpt()).cast(Descendant.class).findFirst().get();
    }

    public Optional<Descendant> getReqToOptTypeCastAsb()
    {
        return Seq.of(this.getReq()).cast(Descendant.class).findFirst();
    }

    public Optional<Descendant> getReqToOptTypeCastAsq()
    {
        return Seq.of(this.getReq()).ofType(Descendant.class).findFirst();
    }

    public Optional<Descendant> getOptToOptTypeCastAsb()
    {
        return seq(this.getOpt()).cast(Descendant.class).findFirst();
    }

    public Optional<Descendant> getOptToOptTypeCastAsq()
    {
        return seq(this.getOpt()).ofType(Descendant.class).findFirst();
    }

    public List<Descendant> getReqToSeqTypeCastAsb()
    {
        return Seq.of(this.getReq()).cast(Descendant.class).toList();
    }

    public List<Descendant> getOptToSeqTypeCastAsb()
    {
        return seq(this.getOpt()).cast(Descendant.class).toList();
    }

    public List<Descendant> getSeqToSeqTypeCastAsb()
    {
        return seq(this.getSeq()).cast(Descendant.class).toList();
    }

    public List<Descendant> getReqToSeqTypeCastAsq()
    {
        return Seq.of(this.getReq()).ofType(Descendant.class).toList();
    }

    public List<Descendant> getOptToSeqTypeCastAsq()
    {
        return seq(this.getOpt()).ofType(Descendant.class).toList();
    }

    public List<Descendant> getSeqToSeqTypeCastAsq()
    {
        return seq(this.getSeq()).ofType(Descendant.class).toList();
    }

    public List<Ancestor> getDescendants()
    {
        return seq(this.getSeq()).filter((a) -> a instanceof Descendant).toList();
    }

    public String toString()
    {
        return new StringBuilder(Types.class.getSimpleName())
                    .append('(')
                    .append("req=").append(String.format("\"%s\"", this.getReq())).append(", ")
                    .append("opt=").append(this.getOpt().isPresent() ? String.format("\"%s\"", this.getOpt()) : "not present").append(", ")
                    .append("typeCheckIs=").append(String.format("\"%s\"", this.isTypeCheckIs())).append(", ")
                    .append("typeCheckIsNot=").append(String.format("\"%s\"", this.isTypeCheckIsNot()))
                    .append(')')
                    .toString();
    }
}

==> cml-modules/cml_base/tests/expressions/types/expected/pop/cml-compiler-output.txt

model files:
- setup.py
- cml_expressions_types_pop/__init__.py


==> cml-modules/cml_base/tests/expressions/types/expected/pop/cml_expressions_types_pop/__init__.py

from typing import *
from abc import *
from decimal import *
```

```
import functools, itertools

class Ancestor:

    def __str__(self) -> 'str':
        return "%s()" % type(self).__name__


class Descendant(Ancestor):

    def __str__(self) -> 'str':
        return "%s()" % type(self).__name__


class Types:

    def __init__(self, req: 'Ancestor', opt: 'Optional[Ancestor]', seq: 'List[Ancestor]') -> 'None':
        self.__req = req
        self.__opt = opt
        self.__seq = seq

    @property
    def req(self) -> 'Ancestor':
        return self.__req

    @property
    def opt(self) -> 'Optional[Ancestor]':
        return self.__opt

    @property
    def seq(self) -> 'List[Ancestor]':
        return self.__seq

    @property
    def type_check_is(self) -> 'bool':
        return isinstance(self.req, Descendant)

    @property
    def type_check_is_not(self) -> 'bool':
        return (not isinstance(self.req, Descendant))

    @property
    def req_to_req_type_cast_asb(self) -> 'Descendant':
        return cast('Descendant', self.req)

    @property
    def opt_to_req_type_cast_asb(self) -> 'Descendant':
        return cast('Descendant', self.opt)

    @property
    def req_to_opt_type_cast_asb(self) -> 'Optional[Descendant]':
        return cast('Optional[Descendant]', self.req)

    @property
    def req_to_opt_type_cast_asq(self) -> 'Optional[Descendant]':
        return cast('Descendant', self.req) if isinstance(self.req, Descendant) else None

    @property
    def opt_to_opt_type_cast_asb(self) -> 'Optional[Descendant]':
        return cast('Optional[Descendant]', self.opt)

    @property
    def opt_to_opt_type_cast_asq(self) -> 'Optional[Descendant]':
        return cast('Descendant', self.opt) if isinstance(self.opt, Descendant) else None

    @property
    def req_to_seq_type_cast_asb(self) -> 'List[Descendant]':
        return list(
            [cast('Descendant', self.req)]
        )

    @property
    def opt_to_seq_type_cast_asb(self) -> 'List[Descendant]':
        return list(
            [] if self.opt is None else [cast('Descendant', self.opt)]
        )

    @property
    def seq_to_seq_type_cast_asb(self) -> 'List[Descendant]':
        return list(
            map(lambda item: cast('Descendant', item), self.seq)
        )

    @property
    def req_to_seq_type_cast_asq(self) -> 'List[Descendant]':
        return list(
            map(lambda item: cast('Descendant', item), filter(lambda item: isinstance(item, Descendant), [self.req]))
        )

    @property
    def opt_to_seq_type_cast_asq(self) -> 'List[Descendant]':
        return list(
            map(lambda item: cast('Descendant', item), filter(lambda item: isinstance(item, Descendant), [self.opt]))
        )

    @property
    def seq_to_seq_type_cast_asq(self) -> 'List[Descendant]':
        return list(
            map(lambda item: cast('Descendant', item), filter(lambda item: isinstance(item, Descendant), self.seq))
        )

    @property
    def descendants(self) -> 'List[Ancestor]':
        return list(
            filter(lambda a: isinstance(a, Descendant), self.seq)
        )

    def __str__(self) -> 'str':
        return "%s(req=%s, opt=%s, type_check_is=%s, type_check_is_not=%s)" % (
            type(self).__name__,
            self.req,
            self.opt,
            self.type_check_is,
            self.type_check_is_not
        )
```

==> cml-modules/cml_base/tests/expressions/types/expected/pop/ignored-list.txt

```
setup.py
```

==> cml-modules/cml_base/tests/expressions/types/source/main.cml

```
@concept Types
{
    req: Ancestor;
```

```
    opt: Ancestor?;
    seq: Ancestor*;

    /type_check_is = req is Descendant;
    /type_check_is_not = req isnt Descendant;

//  /req_to_req_type_cast_asq = req as? Descendant; // <- error: cannot use 'as?' to cast to required type
//  /opt_to_req_type_cast_asq = opt as? Descendant; // <- error: cannot use 'as?' to cast to required type
//  /seq_to_req_type_cast_asq = seq as? Descendant; // <- error: cannot use 'as?' to cast to required type

    /req_to_req_type_cast_asb = req as! Descendant;
    /opt_to_req_type_cast_asb = opt as! Descendant;
//  /seq_to_req_type_cast_asb = seq as! Descendant; // <- error: cannot cast from sequence to required type

    /req_to_opt_type_cast_asb = req as! Descendant?;
    /req_to_opt_type_cast_asq = req as? Descendant?;

    /opt_to_opt_type_cast_asb = opt as! Descendant?;
    /opt_to_opt_type_cast_asq = opt as? Descendant?;

//  /seq_to_opt_type_cast_asb = seq as! Descendant?; // <- error: cannot cast from sequence to optional
//  /seq_to_opt_type_cast_asq = seq as? Descendant?; // <- error: cannot cast from sequence to optional

    /req_to_seq_type_cast_asb = req as! Descendant*;
    /opt_to_seq_type_cast_asb = opt as! Descendant*;
    /seq_to_seq_type_cast_asb = seq as! Descendant*;

    /req_to_seq_type_cast_asq = req as? Descendant*;
    /opt_to_seq_type_cast_asq = opt as? Descendant*;
    /seq_to_seq_type_cast_asq = seq as? Descendant*;

    /descendants = for a in seq | select: a is Descendant;
}

@concept Ancestor;

@concept Descendant: Ancestor;

@task poj: poj
{
    groupId = "cml-expressions";
    artifactId = "cml-expressions-types";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "types.poj";
    packagePath = "types/poj";
}

@task pop: pop
{
    moduleName = "cml_expressions_types_pop";
    moduleVersion = "1.0";
}


==> cml-modules/cml_base/tests/expressions/validations/expected/expressions_validations/cml-compiler-errors.txt


Failed validation: required property_type_specified_or_inferred: in property InfixValidations.derived_flag: UNDEFINED (7:5)
Unable to find type of member: some.unknown

Failed validation: required property_type_specified_or_inferred: in property InfixValidations.invalid_flag: UNDEFINED (11:5)
Incompatible operand(s) for operator 'or':
- left operand is 'flag: boolean'
- right operand is 'number: integer'

Failed validation: required property_type_specified_or_inferred: in property InfixValidations.rel_equality: UNDEFINED (13:5)
Incompatible operand(s) for operator '==':
- left operand is 'some: Something'
- right operand is 'number: integer'

Failed validation: required property_type_specified_or_inferred: in property InfixValidations.rel_inequality: UNDEFINED (14:5)
Incompatible operand(s) for operator '!=':
- left operand is 'some: Something'
- right operand is 'number: integer'

Failed validation: required property_type_specified_or_inferred: in property InfixValidations.ref_equality: UNDEFINED (16:5)
Incompatible operand(s) for operator '===':
- left operand is 'some: Something'
- right operand is 'number: integer'

Failed validation: required property_type_specified_or_inferred: in property InfixValidations.ref_inequality: UNDEFINED (17:5)
Incompatible operand(s) for operator '!==':
- left operand is 'some: Something'
- right operand is 'number: integer'

Failed validation: required property_type_specified_or_inferred: in property InfixValidations.bool_rel_equality: UNDEFINED (19:5)
Incompatible operand(s) for operator '==':
- left operand is 'flag: boolean'
- right operand is 'flag: boolean'

Failed validation: required property_type_specified_or_inferred: in property InfixValidations.bool_rel_inequality: UNDEFINED (20:5)
Incompatible operand(s) for operator '!=':
- left operand is 'flag: boolean'
- right operand is 'flag: boolean'

Failed validation: required property_type_specified_or_inferred: in property InfixValidations.bool_ref_equality: UNDEFINED (22:5)
Incompatible operand(s) for operator '===':
- left operand is 'flag: boolean'
- right operand is 'flag: boolean'

Failed validation: required property_type_specified_or_inferred: in property InfixValidations.bool_ref_inequality: UNDEFINED (23:5)
Incompatible operand(s) for operator '!==':
- left operand is 'flag: boolean'
- right operand is 'flag: boolean'

Failed validation: required property_type_specified_or_inferred: in property UnaryValidations.derived_flag: UNDEFINED (30:5)
Unable to find type of member: some.unknown

Failed validation: required property_type_specified_or_inferred: in property UnaryValidations.invalid_flag: UNDEFINED (34:5)
Incompatible operand 'number' of type 'integer' for operator 'not'.

Failed validation: required property_type_specified_or_inferred: in property UnaryValidations.invalid_number: UNDEFINED (38:5)
Incompatible operand 'flag' of type 'boolean' for operator '-'.

Failed validation: required property_type_specified_or_inferred: in property InvocationValidations.derived: UNDEFINED (46:5)
Unable to find function of invocation: unknown_function

Failed validation: required matching_function_for_invocation: in unknown_function(value) -> UNDEFINED (46:16)
Unable to find function of invocation: unknown_function


==> cml-modules/cml_base/tests/expressions/validations/source/main.cml


@concept InfixValidations
{
    flag: boolean;
```

```
    some: Something;

    /derived_flag = flag or some.unknown; // the type of 'derived_flag' should be: UNDEFINED

    number: integer;

    /invalid_flag = flag or number;

    /rel_equality = some == number;
    /rel_inequality = some != number;

    /ref_equality = some === number;
    /ref_inequality = some !== number;

    /bool_rel_equality = flag == flag;
    /bool_rel_inequality = flag != flag;

    /bool_ref_equality = flag === flag;
    /bool_ref_inequality = flag !== flag;
}

@concept UnaryValidations
{
    some: Something;

    /derived_flag = not some.unknown; // the type of 'derived_flag' should be: UNDEFINED

    number: integer;

    /invalid_flag = not number;

    flag: boolean;

    /invalid_number = - flag;
}

@concept Something {}

@concept InvocationValidations
{
    value: string;
    /derived = unknown_function(value);
}

@task expressions_validations: poj {}


==> cml-modules/cml_base/tests/inheritance/multiple/clients/mcml_java/expected-client-output.txt

Mini-CML Compiler

Model(parent=not present)
Concept(abstracted="true", name="SomeConcept", parent=not present)


==> cml-modules/cml_base/tests/inheritance/multiple/clients/mcml_java/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-inheritance-multiple</groupId>
    <artifactId>mcml-client</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <jar.name>${project.artifactId}</jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml-inheritance-multiple</groupId>
            <artifactId>mcml-cmlc</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${jar.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>mcml.client.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>mcml-client-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>


==> cml-modules/cml_base/tests/inheritance/multiple/clients/mcml_java/src/main/java/mcml/client/Launcher.java

package mcml.client;

import mcml.cmlc.Concept;
import mcml.cmlc.Model;
```

```
import static java.util.Collections.emptyList;

public class Launcher
{
    public static void main(final String[] args)
    {
        final Model model = Model.createModel(null, emptyList());
        final Concept concept = Concept.createConcept( "SomeConcept", null, emptyList(), true);

        System.out.println("Mini-CML Compiler");
        System.out.println();
        System.out.println(model);
        System.out.println(concept);
    }
}
```

==> cml-modules/cml_base/tests/inheritance/multiple/clients/mcml_py/client.py

```
from cml_inheritance_multiple_mcml_cmlc import Model, Concept

model = Model.create_model(None, [])
concept = Concept.create_concept("SomeConcept", None, [], True)
print("Mini-CML Compiler")
print()
print(model)
print(concept)
```

==> cml-modules/cml_base/tests/inheritance/multiple/clients/mcml_py/expected-client-output.txt

```
Mini-CML Compiler

ModelImpl(parent=None)
ConceptImpl(abstracted=True, name=SomeConcept, parent=None)
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_java/cml-compiler-output.txt

```
model files:
- pom.xml

ModelElement files:
- src/main/java/mcml/cmlc/ModelElement.java

NamedElement files:
- src/main/java/mcml/cmlc/NamedElement.java

PropertyList files:
- src/main/java/mcml/cmlc/PropertyList.java

Concept files:
- src/main/java/mcml/cmlc/Concept.java

Model files:
- src/main/java/mcml/cmlc/Model.java
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_java/pom.xml

```
<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-inheritance-multiple</groupId>
    <artifactId>mcml-cmlc</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>15.0</version>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-source-plugin</artifactId>
              <version>2.4</version>
              <executions>
                <execution>
                  <id>attach-sources</id>
                  <goals>
                    <goal>jar</goal>
                  </goals>
                </execution>
              </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_java/src/main/java/mcml/cmlc/Concept.java

```
package mcml.cmlc;

import java.util.*;
```

```
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Concept extends NamedElement, PropertyList
{
    boolean isAbstracted();

    static Concept createConcept(String name, @Nullable ModelElement parent, List<ModelElement> elements, boolean abstracted)
    {
        return new ConceptImpl(null, name, parent, elements, abstracted);
    }

    static Concept extendConcept(@Nullable Concept actual_self, ModelElement modelElement, NamedElement namedElement, PropertyList propertyList, bo
    {
        return new ConceptImpl(actual_self, modelElement, namedElement, propertyList, abstracted);
    }
}

class ConceptImpl implements Concept
{
    private final @Nullable Concept actual_self;

    private final ModelElement modelElement;
    private final NamedElement namedElement;
    private final PropertyList propertyList;

    private final boolean abstracted;

    ConceptImpl(@Nullable Concept actual_self, String name, @Nullable ModelElement parent, List<ModelElement> elements, boolean abstracted)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.modelElement = ModelElement.extendModelElement(this.actual_self, parent, elements);
        this.namedElement = NamedElement.extendNamedElement(this.actual_self, this.modelElement, name);
        this.propertyList = PropertyList.extendPropertyList(this.actual_self, this.modelElement);
        this.abstracted = abstracted;
    }

    ConceptImpl(@Nullable Concept actual_self, ModelElement modelElement, NamedElement namedElement, PropertyList propertyList, boolean abstracted)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.modelElement = modelElement;
        this.namedElement = namedElement;
        this.propertyList = propertyList;
        this.abstracted = abstracted;
    }

    public boolean isAbstracted()
    {
        return this.abstracted;
    }

    public String getName()
    {
        return this.namedElement.getName();
    }

    public Optional<ModelElement> getParent()
    {
        return this.modelElement.getParent();
    }

    public List<ModelElement> getElements()
    {
        return this.modelElement.getElements();
    }

    public String toString()
    {
        return new StringBuilder(Concept.class.getSimpleName())
                    .append('(')
                    .append("abstracted=").append(String.format("\"%s\"", this.actual_self.isAbstracted())).append(", ")
                    .append("name=").append(String.format("\"%s\"", this.actual_self.getName())).append(", ")
                    .append("parent=").append(this.actual_self.getParent().isPresent() ? String.format("\"%s\"", this.actual_self.getParent()) : "no
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_java/src/main/java/mcml/cmlc/Model.java

```
package mcml.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Model extends ModelElement
{
    static Model createModel(@Nullable ModelElement parent, List<ModelElement> elements)
    {
        return new ModelImpl(null, parent, elements);
    }

    static Model extendModel(@Nullable Model actual_self, ModelElement modelElement)
    {
        return new ModelImpl(actual_self, modelElement);
    }
}

class ModelImpl implements Model
{
    private final @Nullable Model actual_self;

    private final ModelElement modelElement;

    ModelImpl(@Nullable Model actual_self, @Nullable ModelElement parent, List<ModelElement> elements)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.modelElement = ModelElement.extendModelElement(this.actual_self, parent, elements);
```

```
    }

    ModelImpl(@Nullable Model actual_self, ModelElement modelElement)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.modelElement = modelElement;
    }

    public Optional<ModelElement> getParent()
    {
        return this.modelElement.getParent();
    }

    public List<ModelElement> getElements()
    {
        return this.modelElement.getElements();
    }

    public String toString()
    {
        return new StringBuilder(Model.class.getSimpleName())
                    .append('(')
                    .append("parent=").append(this.actual_self.getParent().isPresent() ? String.format("\"%s\"", this.actual_self.getParent()) : "no
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_java/src/main/java/mcml/cmlc/ModelElement.java

```
package mcml.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface ModelElement
{
    Optional<ModelElement> getParent();

    List<ModelElement> getElements();

    static ModelElement extendModelElement(@Nullable ModelElement actual_self, @Nullable ModelElement parent, List<ModelElement> elements)
    {
        return new ModelElementImpl(actual_self, parent, elements);
    }
}

class ModelElementImpl implements ModelElement
{
    private final @Nullable ModelElement actual_self;

    private final @Nullable ModelElement parent;
    private final List<ModelElement> elements;

    ModelElementImpl(@Nullable ModelElement actual_self, @Nullable ModelElement parent, List<ModelElement> elements)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.parent = parent;
        this.elements = elements;
    }

    public Optional<ModelElement> getParent()
    {
        return Optional.ofNullable(this.parent);
    }

    public List<ModelElement> getElements()
    {
        return Collections.unmodifiableList(this.elements);
    }

    public String toString()
    {
        return new StringBuilder(ModelElement.class.getSimpleName())
                    .append('(')
                    .append("parent=").append(this.actual_self.getParent().isPresent() ? String.format("\"%s\"", this.actual_self.getParent()) : "no
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_java/src/main/java/mcml/cmlc/NamedElement.java

```
package mcml.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface NamedElement extends ModelElement
{
    String getName();

    static NamedElement extendNamedElement(@Nullable NamedElement actual_self, ModelElement modelElement, String name)
    {
        return new NamedElementImpl(actual_self, modelElement, name);
    }
}

class NamedElementImpl implements NamedElement
{
    private final @Nullable NamedElement actual_self;

    private final ModelElement modelElement;

    private final String name;

    NamedElementImpl(@Nullable NamedElement actual_self, @Nullable ModelElement parent, List<ModelElement> elements, String name)
    {
        this.actual_self = actual_self == null ? this : actual_self;
```

```
        this.modelElement = ModelElement.extendModelElement(this.actual_self, parent, elements);
        this.name = name;
    }

    NamedElementImpl(@Nullable NamedElement actual_self, ModelElement modelElement, String name)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.modelElement = modelElement;
        this.name = name;
    }

    public String getName()
    {
        return this.name;
    }

    public Optional<ModelElement> getParent()
    {
        return this.modelElement.getParent();
    }

    public List<ModelElement> getElements()
    {
        return this.modelElement.getElements();
    }

    public String toString()
    {
        return new StringBuilder(NamedElement.class.getSimpleName())
                    .append('(')
                    .append("name=").append(String.format("\"%s\"", this.actual_self.getName())).append(", ")
                    .append("parent=").append(this.actual_self.getParent().isPresent() ? String.format("\"%s\"", this.actual_self.getParent()) : "no
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_java/src/main/java/mcml/cmlc/PropertyList.java

```
package mcml.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface PropertyList extends ModelElement
{
    static PropertyList extendPropertyList(@Nullable PropertyList actual_self, ModelElement modelElement)
    {
        return new PropertyListImpl(actual_self, modelElement);
    }
}

class PropertyListImpl implements PropertyList
{
    private final @Nullable PropertyList actual_self;

    private final ModelElement modelElement;

    PropertyListImpl(@Nullable PropertyList actual_self, @Nullable ModelElement parent, List<ModelElement> elements)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.modelElement = ModelElement.extendModelElement(this.actual_self, parent, elements);

    }

    PropertyListImpl(@Nullable PropertyList actual_self, ModelElement modelElement)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.modelElement = modelElement;
    }

    public Optional<ModelElement> getParent()
    {
        return this.modelElement.getParent();
    }

    public List<ModelElement> getElements()
    {
        return this.modelElement.getElements();
    }

    public String toString()
    {
        return new StringBuilder(PropertyList.class.getSimpleName())
                    .append('(')
                    .append("parent=").append(this.actual_self.getParent().isPresent() ? String.format("\"%s\"", this.actual_self.getParent()) : "no
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_py/cml-compiler-output.txt

```
model files:
- setup.py
- cml_inheritance_multiple_mcml_cmlc/__init__.py
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_py/cml_inheritance_multiple_mcml_cmlc/__init__.py

```
from typing import *
from abc import *
from decimal import *

import functools, itertools

class ModelElement(ABC):

    @abstractproperty
    def parent(self) -> 'Optional[ModelElement]':
        pass

    @abstractproperty
    def elements(self) -> 'List[ModelElement]':
        pass
```

```
    @staticmethod
    def extend_model_element(actual_self: 'Optional[ModelElement]', parent: 'Optional[ModelElement]', elements: 'List[ModelElement]') -> 'ModelElem
        return ModelElementImpl(actual_self, parent, elements)


class ModelElementImpl(ModelElement):

    def __init__(self, actual_self: 'Optional[ModelElement]', parent: 'Optional[ModelElement]', elements: 'List[ModelElement]') -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[ModelElement]
        else:
            self.__actual_self = actual_self

        self.__parent = parent
        self.__elements = elements

    @property
    def parent(self) -> 'Optional[ModelElement]':
        return self.__parent

    @property
    def elements(self) -> 'List[ModelElement]':
        return self.__elements

    def __str__(self) -> 'str':
        return "%s(parent=%s)" % (
            type(self).__name__,
            self.__actual_self.parent
        )


class NamedElement(ModelElement, ABC):

    @abstractproperty
    def name(self) -> 'str':
        pass

    @staticmethod
    def extend_named_element(actual_self: 'Optional[NamedElement]', model_element: 'ModelElement', name: 'str') -> 'NamedElement':
        return NamedElementImpl(actual_self, model_element, name)


class NamedElementImpl(NamedElement):

    def __init__(self, actual_self: 'Optional[NamedElement]', model_element: 'Optional[ModelElement]', name: 'str') -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[NamedElement]
        else:
            self.__actual_self = actual_self

        self.__model_element = model_element
        self.__name = name

    @property
    def name(self) -> 'str':
        return self.__name

    @property
    def parent(self) -> 'Optional[ModelElement]':
        return self.__model_element.parent

    @property
    def elements(self) -> 'List[ModelElement]':
        return self.__model_element.elements

    def __str__(self) -> 'str':
        return "%s(name=%s, parent=%s)" % (
            type(self).__name__,
            self.__actual_self.name,
            self.__actual_self.parent
        )


class PropertyList(ModelElement, ABC):

    @staticmethod
    def extend_property_list(actual_self: 'Optional[PropertyList]', model_element: 'ModelElement') -> 'PropertyList':
        return PropertyListImpl(actual_self, model_element)


class PropertyListImpl(PropertyList):

    def __init__(self, actual_self: 'Optional[PropertyList]', model_element: 'Optional[ModelElement]') -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[PropertyList]
        else:
            self.__actual_self = actual_self

        self.__model_element = model_element


    @property
    def parent(self) -> 'Optional[ModelElement]':
        return self.__model_element.parent

    @property
    def elements(self) -> 'List[ModelElement]':
        return self.__model_element.elements

    def __str__(self) -> 'str':
        return "%s(parent=%s)" % (
            type(self).__name__,
            self.__actual_self.parent
        )


class Concept(NamedElement, PropertyList, ABC):

    @abstractproperty
    def abstracted(self) -> 'bool':
        pass

    @staticmethod
    def create_concept(name: 'str', parent: 'Optional[ModelElement]', elements: 'List[ModelElement]', abstracted: 'bool') -> 'Concept':
        return ConceptImpl(None, None, None, None, abstracted, name=name, parent=parent, elements=elements)

    @staticmethod
    def extend_concept(actual_self: 'Optional[Concept]', model_element: 'ModelElement', named_element: 'NamedElement', property_list: 'PropertyList
        return ConceptImpl(actual_self, model_element, named_element, property_list, abstracted)


class ConceptImpl(Concept):

    def __init__(self, actual_self: 'Optional[Concept]', model_element: 'Optional[ModelElement]', named_element: 'Optional[NamedElement]', property
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Concept]
```

```
        else:
            self.__actual_self = actual_self

        if model_element is None:
            self.__model_element = ModelElement.extend_model_element(self.__actual_self, kwargs['parent'], kwargs['elements'])
        else:
            self.__model_element = model_element
        if named_element is None:
            self.__named_element = NamedElement.extend_named_element(self.__actual_self, self.__model_element, kwargs['name'])
        else:
            self.__named_element = named_element
        if property_list is None:
            self.__property_list = PropertyList.extend_property_list(self.__actual_self, self.__model_element)
        else:
            self.__property_list = property_list
        self.__abstracted = abstracted

    @property
    def abstracted(self) -> 'bool':
        return self.__abstracted

    @property
    def name(self) -> 'str':
        return self.__named_element.name

    @property
    def parent(self) -> 'Optional[ModelElement]':
        return self.__model_element.parent

    @property
    def elements(self) -> 'List[ModelElement]':
        return self.__model_element.elements

    def __str__(self) -> 'str':
        return "%s(abstracted=%s, name=%s, parent=%s)" % (
            type(self).__name__,
            self.__actual_self.abstracted,
            self.__actual_self.name,
            self.__actual_self.parent
        )


class Model(ModelElement, ABC):

    @staticmethod
    def create_model(parent: 'Optional[ModelElement]', elements: 'List[ModelElement]') -> 'Model':
        return ModelImpl(None, None, parent=parent, elements=elements)

    @staticmethod
    def extend_model(actual_self: 'Optional[Model]', model_element: 'ModelElement') -> 'Model':
        return ModelImpl(actual_self, model_element)


class ModelImpl(Model):

    def __init__(self, actual_self: 'Optional[Model]', model_element: 'Optional[ModelElement]', **kwargs) -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Model]
        else:
            self.__actual_self = actual_self

        if model_element is None:
            self.__model_element = ModelElement.extend_model_element(self.__actual_self, kwargs['parent'], kwargs['elements'])
        else:
            self.__model_element = model_element


    @property
    def parent(self) -> 'Optional[ModelElement]':
        return self.__model_element.parent

    @property
    def elements(self) -> 'List[ModelElement]':
        return self.__model_element.elements

    def __str__(self) -> 'str':
        return "%s(parent=%s)" % (
            type(self).__name__,
            self.__actual_self.parent
        )
```

==> cml-modules/cml_base/tests/inheritance/multiple/expected/mcml_py/setup.py

```
"""A setuptools based setup module.

See:
https://packaging.python.org/en/latest/distributing.html
https://github.com/pypa/sampleproject
"""

# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
    name='cml_inheritance_multiple_mcml_cmlc',

    # Versions should comply with PEP440.  For a discussion on single-sourcing
    # the version across setup.py and the project code, see
    # https://packaging.python.org/en/latest/single_source_version.html
    version='1.0',

    description='A sample Python project',

    # The project's main homepage.
    url='https://github.com/pypa/sampleproject',

    # Author details
    author='The Python Packaging Authority',
    author_email='pypa-dev@googlegroups.com',

    # Choose your license
    license='MIT',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',

        # Indicate who your project is intended for
```

```
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: MIT License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3.6.1',
    ],

    # What does your project relate to?
    keywords='sample setuptools development',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests']),

    # Alternatively, if you want to distribute just a my_module.py, uncomment
    # this:
    #   py_modules=["my_module"],

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    # install_requires=['peppercorn'],

    # List additional groups of dependencies here (e.g. development
    # dependencies). You can install these using the following syntax,
    # for example:
    # $ pip install -e .[dev,test]
    extras_require={
        'dev': ['check-manifest'],
        'test': ['coverage'],
    },

    # If there are data files included in your packages that need to be
    # installed, specify them here.  If using Python 2.6 or less, then these
    # have to be included in MANIFEST.in as well.
    # package_data={
    #     'sample': ['package_data.dat'],
    #},

    # Although 'package_data' is the preferred approach, in some case you may
    # need to place data files outside of your packages. See:
    # http://docs.python.org/3.4/distutils/setupscript.html#installing-additional-files # noqa
    # In this case, 'data_file' will be installed into '<sys.prefix>/my_data'
    #data_files=[('my_data', ['data/data_file'])],

    # To provide executable scripts, use entry points in preference to the
    # "scripts" keyword. Entry points provide cross-platform support and allow
    # pip to create the appropriate form of executable for the target platform.
    #entry_points={
    #    'console_scripts': [
    #        'sample=sample:main',
    #    ],
    #},
)
```

```
==> cml-modules/cml_base/tests/inheritance/multiple/source/main.cml


@abstraction ModelElement
{
    parent: ModelElement?;
    elements: ModelElement*;
}

@abstraction NamedElement: ModelElement
{
    name: String;
}

@abstraction PropertyList: ModelElement
{
}

@concept Concept: NamedElement, PropertyList
{
    abstracted: Boolean;
}

@concept Model: ModelElement
{
}

@task mcml_java: cmlc_java
{
    groupId = "cml-inheritance-multiple";
    artifactId = "mcml-cmlc";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "mcml.cmlc";
    packagePath = "mcml/cmlc";
}

@task mcml_py: cmlc_py
{
    moduleName = "cml_inheritance_multiple_mcml_cmlc";
    moduleVersion = "1.0";
}



==> cml-modules/cml_base/tests/modules/livir_books/clients/livir_poj/expected-client-output.txt

Livir Client (poj)

Classes:
- livir.books.BookStore
- livir.books.Book

Book(isbn="1234", title="Programming Adventures", name="Programming Adventures", price="10.00")


==> cml-modules/cml_base/tests/modules/livir_books/clients/livir_poj/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>livir</groupId>
    <artifactId>livir-client</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
    <packaging>jar</packaging>

    <properties>
        <jar.name>${project.artifactId}</jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>livir</groupId>
            <artifactId>livir-books</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${jar.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>livir.client.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>livir-client-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>


==> cml-modules/cml_base/tests/modules/livir_books/clients/livir_poj/src/main/java/livir/client/Launcher.java

package livir.client;

import livir.books.Book;
import livir.books.BookStore;

import java.math.BigDecimal;

public class Launcher
{
    public static void main(final String[] args)
    {
        System.out.println("Livir Client (poj)");
        System.out.println();
        System.out.println("Classes:");
        System.out.println("- " + BookStore.class.getName());
        System.out.println("- " + Book.class.getName());
        System.out.println();
        System.out.println(new Book(new BigDecimal("10.00"), "1234", "Programming Adventures", "Programming Adventures"));
    }
}


==> cml-modules/cml_base/tests/modules/livir_books/clients/livir_pop/client.py


from livir_books import Book
from decimal import *

print("Livir Client (pop)\n")

book = Book(Decimal("13.00"), "ISBN-1234", "Python's Book", "Python's Book")

print("Book: %s" % book)


==> cml-modules/cml_base/tests/modules/livir_books/clients/livir_pop/expected-client-output.txt

Livir Client (pop)

Book: Book(isbn=ISBN-1234, title=Python's Book, name=Python's Book, price=13.00)


==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_poj/cml-compiler-output.txt

model files:
- pom.xml

Book files:
- src/main/java/livir/books/Book.java

Order files:
- src/main/java/livir/books/Order.java

Item files:
- src/main/java/livir/books/Item.java

BookStore files:
- src/main/java/livir/books/BookStore.java

Product files:
- src/main/java/livir/books/Product.java

Customer files:
- src/main/java/livir/books/Customer.java


==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_poj/pom.xml

<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>livir</groupId>
    <artifactId>livir-books</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>15.0</version>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-source-plugin</artifactId>
              <version>2.4</version>
              <executions>
                <execution>
                  <id>attach-sources</id>
                  <goals>
                    <goal>jar</goal>
                  </goals>
                </execution>
              </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_poj/src/main/java/livir/books/Book.java

```
package livir.books;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Book extends Product
{
    private final String isbn;
    private final String title;
    private final String name;

    public Book()
    {
        this(new BigDecimal("0.00"), "Not Provided", "No Title", "");
    }

    public Book(BigDecimal price, String isbn, String title, String name)
    {
        super(name, price);
        this.isbn = isbn;
        this.title = title;
        this.name = name;
    }

    public String getIsbn()
    {
        return this.isbn;
    }

    public String getTitle()
    {
        return this.title;
    }

    public String getName()
    {
        return this.name;
    }

    public String toString()
    {
        return new StringBuilder(Book.class.getSimpleName())
                    .append('(')
                    .append("isbn=").append(String.format("\"%s\"", this.getIsbn())).append(", ")
                    .append("title=").append(String.format("\"%s\"", this.getTitle())).append(", ")
                    .append("name=").append(String.format("\"%s\"", this.getName())).append(", ")
                    .append("price=").append(String.format("\"%s\"", this.getPrice()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_poj/src/main/java/livir/books/BookStore.java

```
package livir.books;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;
```

```
import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class BookStore
{
    private final List<Book> books;
    private final List<Customer> customers;

    public BookStore(List<Book> books, List<Customer> customers)
    {
        this.books = books;
        this.customers = customers;
    }

    public List<Book> getBooks()
    {
        return Collections.unmodifiableList(this.books);
    }

    public List<Customer> getCustomers()
    {
        return Collections.unmodifiableList(this.customers);
    }

    public String toString()
    {
        return new StringBuilder(BookStore.class.getSimpleName())
                .append('(')
                .append(')')
                .toString();
    }
}
```

==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_poj/src/main/java/livir/books/Customer.java

```
package livir.books;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Customer
{
    private final String name;

    public Customer(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return this.name;
    }

    public String toString()
    {
        return new StringBuilder(Customer.class.getSimpleName())
                .append('(')
                .append("name=").append(String.format("\"%s\"", this.getName()))
                .append(')')
                .toString();
    }
}
```

==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_poj/src/main/java/livir/books/Item.java

```
package livir.books;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Item
{
    private final Book book;
    private final int quantity;

    public Item(Book book, int quantity)
    {
        this.book = book;
        this.quantity = quantity;
    }

    public Book getBook()
    {
        return this.book;
    }

    public int getQuantity()
    {
        return this.quantity;
    }

    public String toString()
    {
        return new StringBuilder(Item.class.getSimpleName())
                .append('(')
                .append("book=").append(String.format("\"%s\"", this.getBook())).append(", ")
                .append("quantity=").append(String.format("\"%s\"", this.getQuantity()))
                .append(')')
                .toString();
    }
}
```

==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_poj/src/main/java/livir/books/Order.java

```
package livir.books;

import java.util.*;
```

```
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Order
{
    private final Customer customer;
    private final List<Item> items;

    public Order(Customer customer, List<Item> items)
    {
        this.customer = customer;
        this.items = items;
    }

    public Customer getCustomer()
    {
        return this.customer;
    }

    public List<Item> getItems()
    {
        return Collections.unmodifiableList(this.items);
    }

    public String toString()
    {
        return new StringBuilder(Order.class.getSimpleName())
                    .append('(')
                    .append("customer=").append(String.format("\"%s\"", this.getCustomer()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_poj/src/main/java/livir/books/Product.java

```
package livir.books;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public class Product
{
    private final String name;
    private final BigDecimal price;

    public Product(String name)
    {
        this(name, new BigDecimal("0.00"));
    }

    public Product(String name, BigDecimal price)
    {
        this.name = name;
        this.price = price;
    }

    public String getName()
    {
        return this.name;
    }

    public BigDecimal getPrice()
    {
        return this.price;
    }

    public String toString()
    {
        return new StringBuilder(Product.class.getSimpleName())
                    .append('(')
                    .append("name=").append(String.format("\"%s\"", this.getName())).append(", ")
                    .append("price=").append(String.format("\"%s\"", this.getPrice()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_pop/cml-compiler-output.txt

```
model files:
- setup.py
- livir_books/__init__.py
```

==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_pop/livir_books/__init__.py

```
from typing import *
from abc import *
from decimal import *

import functools, itertools

class Product:

    def __init__(self, name: 'str', price: 'Decimal' = Decimal("0.00")) -> 'None':
        self.__name = name
        self.__price = price

    @property
    def name(self) -> 'str':
        return self.__name

    @property
    def price(self) -> 'Decimal':
        return self.__price

    def __str__(self) -> 'str':
        return "%s(name=%s, price=%s)" % (
            type(self).__name__,
            self.name,
            self.price
```

```
        )


class Book(Product):

    def __init__(self, price: 'Decimal' = Decimal("0.00"), isbn: 'str' = "Not Provided", title: 'str' = "No Title", name: 'str' = "") -> 'None':
        super().__init__(name, price)
        self.__isbn = isbn
        self.__title = title
        self.__name = name

    @property
    def isbn(self) -> 'str':
        return self.__isbn

    @property
    def title(self) -> 'str':
        return self.__title

    @property
    def name(self) -> 'str':
        return self.__name

    def __str__(self) -> 'str':
        return "%s(isbn=%s, title=%s, name=%s, price=%s)" % (
            type(self).__name__,
            self.isbn,
            self.title,
            self.name,
            self.price
        )


class Item:

    def __init__(self, book: 'Book', quantity: 'int') -> 'None':
        self.__book = book
        self.__quantity = quantity

    @property
    def book(self) -> 'Book':
        return self.__book

    @property
    def quantity(self) -> 'int':
        return self.__quantity

    def __str__(self) -> 'str':
        return "%s(book=%s, quantity=%s)" % (
            type(self).__name__,
            self.book,
            self.quantity
        )


class Customer:

    def __init__(self, name: 'str') -> 'None':
        self.__name = name

    @property
    def name(self) -> 'str':
        return self.__name

    def __str__(self) -> 'str':
        return "%s(name=%s)" % (
            type(self).__name__,
            self.name
        )


class Order:

    def __init__(self, customer: 'Customer', items: 'List[Item]') -> 'None':
        self.__customer = customer
        self.__items = items

    @property
    def customer(self) -> 'Customer':
        return self.__customer

    @property
    def items(self) -> 'List[Item]':
        return self.__items

    def __str__(self) -> 'str':
        return "%s(customer=%s)" % (
            type(self).__name__,
            self.customer
        )


class BookStore:

    def __init__(self, books: 'List[Book]', customers: 'List[Customer]') -> 'None':
        self.__books = books
        self.__customers = customers

    @property
    def books(self) -> 'List[Book]':
        return self.__books

    @property
    def customers(self) -> 'List[Customer]':
        return self.__customers

    def __str__(self) -> 'str':
        return "%s()" % type(self).__name__
```

==> cml-modules/cml_base/tests/modules/livir_books/expected/livir_pop/setup.py

```
"""A setuptools based setup module.

See:
https://packaging.python.org/en/latest/distributing.html
https://github.com/pypa/sampleproject
"""

# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
```

```
    name='livir_books',

    # Versions should comply with PEP440.  For a discussion on single-sourcing
    # the version across setup.py and the project code, see
    # https://packaging.python.org/en/latest/single_source_version.html
    version='1.0',

    description='A sample Python project',

    # The project's main homepage.
    url='https://github.com/pypa/sampleproject',

    # Author details
    author='The Python Packaging Authority',
    author_email='pypa-dev@googlegroups.com',

    # Choose your license
    license='MIT',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',

        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: MIT License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3.6.1',
    ],

    # What does your project relate to?
    keywords='sample setuptools development',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests']),

    # Alternatively, if you want to distribute just a my_module.py, uncomment
    # this:
    #   py_modules=["my_module"],

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    # install_requires=['peppercorn'],

    # List additional groups of dependencies here (e.g. development
    # dependencies). You can install these using the following syntax,
    # for example:
    # $ pip install -e .[dev,test]
    extras_require={
        'dev': ['check-manifest'],
        'test': ['coverage'],
    },

    # If there are data files included in your packages that need to be
    # installed, specify them here.  If using Python 2.6 or less, then these
    # have to be included in MANIFEST.in as well.
    # package_data={
    #     'sample': ['package_data.dat'],
    #},

    # Although 'package_data' is the preferred approach, in some case you may
    # need to place data files outside of your packages. See:
    # http://docs.python.org/3.4/distutils/setupscript.html#installing-additional-files # noqa
    # In this case, 'data_file' will be installed into '<sys.prefix>/my_data'
    #data_files=[('my_data', ['data/data_file'])],

    # To provide executable scripts, use entry points in preference to the
    # "scripts" keyword. Entry points provide cross-platform support and allow
    # pip to create the appropriate form of executable for the target platform.
    #entry_points={
    #    'console_scripts': [
    #        'sample=sample:main',
    #    ],
    #},
)
```

==> cml-modules/cml_base/tests/modules/livir_books/source/main.cml

```
@module livir_books
{
    @import livir_products;
}

@concept Book: Product
{
    isbn: String = "Not Provided";
    title: String = "No Title";
    name = title;
}

@concept Order
{
    customer: Customer;
    items: Item*;

//    /total = (for item in items | from sum = 0 reduce sum + item.total);
}

@concept Item
{
    book: Book;
    quantity: Integer;

//    /total = book.price * quantity;
}

@concept BookStore
{
    books: Book*;
    customers: Customer*;
}

@task livir_poj: poj
{
```

```
    groupId = "livir";
    artifactId = "livir-books";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "livir.books";
    packagePath = "livir/books";
}

@task livir_pop: pop
{
    moduleName = "livir_books";
    moduleVersion = "1.0";
}
```

==> cml-modules/cml_base/tests/modules/livir_products/source/main.cml

```
@concept Product
{
    name: String;
    price: Decimal = 0.00;
}

@concept Customer
{
    name: String;
}
```

==> cml-modules/cml_base/tests/specializations/abstract_property_redefinition/expected/abstract_property_redefinition/cml-compiler-errors.txt

```
Failed validation: required abstract_property_redefinition: in concept Circle (8:1)
- property Shape.area: double (4:5)

Failed validation: required abstract_property_redefinition: in concept UnitCircleA (11:1)
- property Shape.area: double (4:5)
```

==> cml-modules/cml_base/tests/specializations/abstract_property_redefinition/source/main.cml

```
@abstraction Shape
{
    -- A derived property without an expression is considered abstract:
    /area: Double;
}

-- In order to be considered concrete, Circle should have redefined "area".
@concept Circle: Shape;

-- Also missing redefinition in specialization of Circle
@concept UnitCircleA: Circle;

-- Properly redefined "area" as a derived property:
@concept UnitCircleB: Circle
{
    /area = 1.0d;
}

-- Properly redefined "area" as a concrete property:
@concept UnitCircleC: Circle
{
    area = 1.0d;
}

-- OK to miss redefinition in this specialization because it is an abstraction:
@abstraction UnitCircleD: Circle;

@task abstract_property_redefinition;
```

==> cml-modules/cml_base/tests/specializations/compatible_generalizations/expected/compatible_generalizations/cml-compiler-errors.txt

```
Failed validation: required compatible_generalizations: in concept C (13:1)
- property A.number: integer (4:5)
- property B.number: decimal (9:5)

Failed validation: required compatible_generalizations: in concept F (18:1)
- property A.number: integer (4:5)
- property B.number: decimal (9:5)
```

==> cml-modules/cml_base/tests/specializations/compatible_generalizations/source/main.cml

```
@concept A
{
    number: INTEGER;
}

@concept B
{
    number: DECIMAL;
}

// The "number" property has incompatible types in A and B:
@concept C: A, B;

// Indirectly inheriting conflicting definitions of "number":
@concept D: A;
@concept E: B;
@concept F: D, E;

@task compatible_generalizations;
```

==> cml-modules/cml_base/tests/specializations/conflict_redefinition/expected/conflict_redefinition/cml-compiler-errors.txt

```
Failed validation: required conflict_redefinition: in concept C (24:1)
- property A.p1: integer (8:5)
- property B.p1: integer (18:5)

Failed validation: required conflict_redefinition: in concept E (38:1)
- property A.p1: integer (8:5)
- property D.p1: integer (29:5)
- property A.r1: decimal (10:5)
- property D.r1: decimal (31:5)
- property A.s2: decimal (12:5)
- property D.s2: decimal (33:5)

Failed validation: required conflict_redefinition: in concept F (44:1)
- property A.t1: string (13:5)
- property D.t1: string (34:5)
```

```
==> cml-modules/cml_base/tests/specializations/conflict_redefinition/source/main.cml

@concept Base
{
    a: STRING;
}

@concept A: Base
{
    p1: INTEGER = 0;
    q1: STRING;
    r1: DECIMAL = 1.0;
    s1: DECIMAL;
    s2: DECIMAL;
    t1: STRING = "A";
}

@concept B: Base
{
    p1: INTEGER = 1;
    q2: STRING;
}

// Property "p1" should have been redefined
// in order to resolve definition conflict between A and B:
@concept C: A, B;

// Redefinition as expected:
@concept D: A, B
{
    p1: INTEGER = 2;
    q3: STRING;
    r1: DECIMAL = 2.0;
    s1: DECIMAL;
    /s2 = 3.0;
    t1: STRING = "D";
}

// Missing redefinition of "p1" and "r1":
@concept E: A, D
{
    t1: STRING = "E";
}

// Redefinitions as expected (except for "t1"):
@concept F: A, D, Base
{
    p1: INTEGER = 3;
    q4: STRING;
    r1: DECIMAL;

    // No need to redefine "s1" because no expression has been defined for it in A and D.

    // However, we do need to redefine "s2" because it is a derived property in D, but not in A:
    s2 = 4.0;
}

@task conflict_redefinition;

==> cml-modules/cml_base/tests/specializations/generalization_compatible_redefinition/expected/generalization_compatible_redefinition/cml-compiler-


Failed validation: required generalization_compatible_redefinition: in property C.p1: string (27:5)
- property A.p1: integer (8:5)
- property B.p1: integer (19:5)

Failed validation: required generalization_compatible_redefinition: in property D.p2: integer (38:5)
- property A.p2: string (9:5)

Failed validation: required compatible_generalizations: in concept E (47:1)
- property A.p2: string (9:5)
- property D.p2: integer (38:5)

Failed validation: required generalization_compatible_redefinition: in property E.p1: string (49:5)
- property A.p1: integer (8:5)
- property D.p1: integer (37:5)

Failed validation: required generalization_compatible_redefinition: in property E.r1: string (51:5)
- property A.r1: decimal (11:5)
- property D.r1: decimal (40:5)

Failed validation: required compatible_generalizations: in concept F (57:1)
- property A.p2: string (9:5)
- property D.p2: integer (38:5)

Failed validation: required conflict_redefinition: in concept F (57:1)
- property A.t2: string (14:5)
- property D.t2: string (43:5)

Failed validation: required generalization_compatible_redefinition: in property F.t1: boolean (62:5)
- property A.t1: string (13:5)
- property D.t1: string (42:5)


==> cml-modules/cml_base/tests/specializations/generalization_compatible_redefinition/source/main.cml

@concept Base
{
    a: STRING;
}

@concept A: Base
{
    p1: INTEGER = 0;
    p2: STRING;
    q1: STRING;
    r1: DECIMAL = 1.0;
    s1: DECIMAL;
    t1: STRING = "A";
    t2: STRING = "A";
}

@concept B: Base
{
    p1: INTEGER = 1;
    q2: STRING;
}

// Property "p1" should have been redefined as INTEGER
// in order to resolve definition conflict between A and B:
@concept C: A, B
{
    p1: STRING;
    q3: STRING;
    r1: DECIMAL = 2.0;
    s1: DECIMAL;
```

```
    t1: STRING = "D";
}

// Redefinition as expected:
@concept D: A, B
{
    p1: INTEGER = 2;
    p2: INTEGER; // Incompatible with A
    q3: STRING;
    r1: DECIMAL = 2.0;
    s1: DECIMAL;
    t1: STRING = "D";
    t2: STRING = "D";
}

// Incorrect type in redefinition of "p1" and "r1":
@concept E: A, D
{
    p1: STRING;
    q4: STRING;
    r1: STRING;
    t1: STRING = "E";
    t2: STRING = "E";
}

// Redefinitions as expected (except for "t1", which should be "STRING"):
@concept F: A, D, Base
{
    p1: INTEGER = 3;
    q4: STRING;
    r1: DECIMAL;
    t1: BOOLEAN;

    // Also missing "t2" redefinition.
}

@task generalization_compatible_redefinition;
```

==> cml-modules/cml_base/tests/specializations/not_own_generalization/expected/not_own_generalization/cml-compiler-errors.txt

```
Failed validation: required not_own_generalization: in concept A (3:1)

Failed validation: required not_own_generalization: in concept A1 (6:1)

Failed validation: required not_own_generalization: in concept A2 (7:1)

Failed validation: required not_own_generalization: in concept B1 (10:1)

Failed validation: required not_own_generalization: in concept B2 (11:1)

Failed validation: required not_own_generalization: in concept B3 (12:1)
```

==> cml-modules/cml_base/tests/specializations/not_own_generalization/source/main.cml

```
// Direct cycle:
@concept A: A;

// Indirect cycle:
@concept A1: A2;
@concept A2: A1;

// Third-level cycle:
@concept B1: B2, C;
@concept B2: B3, C;
@concept B3: B1, C;
@concept C: D;
@concept D;

@task not_own_generalization;
```

==> cml-modules/cml_base/tests/specializations/redefinitions/clients/shapes_cmlc_java/expected-client-output.txt

```
Shapes Client (cmlc_java)

Rectangle(width="10.0", height="20.0", area="200.0", color="red", totalArea="200.0")
Rhombus(p="14.1421356237", q="14.1421356237", area="99.9999999956229", color="green", totalArea="99.9999999956229")
Square(sideLength="10.0", width="10.0", height="10.0", p="14.1421356237", q="14.1421356237", area="100.0", color="blue", totalArea="100.0")
Circle(radius="10.0", area="314.159", color="red", totalArea="314.159")
UnitCircle(area="3.14159", radius="1.0", color="Blue", totalArea="3.14159")
RedUnitCircle(color="Red", area="3.14159", radius="1.0", totalArea="3.14159")
```

==> cml-modules/cml_base/tests/specializations/redefinitions/clients/shapes_cmlc_java/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-specializations-redefinitions</groupId>
    <artifactId>shapes-client</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <jar.name>${project.artifactId}</jar.name>
    </properties>

    <dependencies>
        <dependency>
            <groupId>cml-specializations-redefinitions</groupId>
            <artifactId>shapes-cmlc</artifactId>
            <version>${project.version}</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>${jar.name}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
```

```
                        <manifest>
                            <mainClass>shapes.client.Launcher</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>shapes-client-jar-with-dependencies</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

==> cml-modules/cml_base/tests/specializations/redefinitions/clients/shapes_cmlc_java/src/main/java/shapes/client/Launcher.java

```
package shapes.client;

import static shapes.cmlc.Circle.createCircle;
import static shapes.cmlc.Rectangle.createRectangle;
import static shapes.cmlc.RedUnitCircle.createRedUnitCircle;
import static shapes.cmlc.Rhombus.createRhombus;
import static shapes.cmlc.Square.createSquare;
import static shapes.cmlc.UnitCircle.createUnitCircle;

public class Launcher
{
    public static void main(final String[] args)
    {
        System.out.println("Shapes Client (cmlc_java)\n");
        System.out.println(createRectangle("red", 10, 20));
        System.out.println(createRhombus("green", 14.1421356237, 14.1421356237));
        System.out.println(createSquare("blue", 10));
        System.out.println(createCircle(10, "red"));
        System.out.println(createUnitCircle());
        System.out.println(createRedUnitCircle());
    }
}
```

==> cml-modules/cml_base/tests/specializations/redefinitions/clients/shapes_cmlc_py/client.py

```
from cml_specializations_redefinitions_shapes_cmlc import Rectangle, Rhombus, Square, Circle, UnitCircle, RedUnitCircle

print("Shapes Client (cmlc_py)\n")
print(Rectangle.create_rectangle("red", 10, 20))
print(Rhombus.create_rhombus("green", 14.1421356237, 14.1421356237))
print(Square.create_square("blue", 10))
print(Circle.create_circle(10, "red"))
print(UnitCircle.create_unit_circle())
print(RedUnitCircle.create_red_unit_circle())
```

==> cml-modules/cml_base/tests/specializations/redefinitions/clients/shapes_cmlc_py/expected-client-output.txt

```
Shapes Client (cmlc_py)

RectangleImpl(width=10, height=20, area=200, color=red, total_area=200)
RhombusImpl(p=14.1421356237, q=14.1421356237, area=99.99999999956229, color=green, total_area=99.99999999956229)
SquareImpl(side_length=10, width=10, height=10, p=14.1421356237, q=14.1421356237, area=100.0, color=blue, total_area=100.0)
CircleImpl(radius=10, area=314.159, color=red, total_area=314.159)
UnitCircleImpl(area=3.14159, radius=1.0, color=Blue, total_area=3.14159)
RedUnitCircleImpl(color=Red, area=3.14159, radius=1.0, total_area=3.14159)
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_java/cml-compiler-output.txt

```
model files:
- pom.xml

Shape files:
- src/main/java/shapes/cmlc/Shape.java

Rectangle files:
- src/main/java/shapes/cmlc/Rectangle.java

Rhombus files:
- src/main/java/shapes/cmlc/Rhombus.java

Square files:
- src/main/java/shapes/cmlc/Square.java

Circle files:
- src/main/java/shapes/cmlc/Circle.java

UnitCircle files:
- src/main/java/shapes/cmlc/UnitCircle.java

RedUnitCircle files:
- src/main/java/shapes/cmlc/RedUnitCircle.java
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_java/pom.xml

```
<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml-specializations-redefinitions</groupId>
    <artifactId>shapes-cmlc</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.jetbrains</groupId>
```

```
            <artifactId>annotations</artifactId>
            <version>15.0</version>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-source-plugin</artifactId>
              <version>2.4</version>
              <executions>
                <execution>
                  <id>attach-sources</id>
                  <goals>
                    <goal>jar</goal>
                  </goals>
                </execution>
              </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_java/src/main/java/shapes/cmlc/Circle.java

```
package shapes.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Circle extends Shape
{
    double getRadius();

    double getArea();

    String getColor();

    static Circle createCircle(double radius)
    {
        return createCircle(radius, "Blue");
    }

    static Circle createCircle(double radius, String color)
    {
        return new CircleImpl(null, radius, color);
    }

    static Circle extendCircle(@Nullable Circle actual_self, Shape shape, double radius, String color)
    {
        return new CircleImpl(actual_self, shape, radius, color);
    }
}

class CircleImpl implements Circle
{
    private final @Nullable Circle actual_self;

    private final Shape shape;

    private final double radius;
    private final String color;

    CircleImpl(@Nullable Circle actual_self, double radius, String color)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = Shape.extendShape(this.actual_self, color);
        this.radius = radius;
        this.color = color;
    }

    CircleImpl(@Nullable Circle actual_self, Shape shape, double radius, String color)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = shape;
        this.radius = radius;
        this.color = color;
    }

    public double getRadius()
    {
        return this.radius;
    }

    public String getColor()
    {
        return this.color;
    }

    public double getArea()
    {
        return (3.14159 * (Math.pow(this.actual_self.getRadius(), 2.0)));
    }

    public double getTotalArea()
    {
        return this.shape.getTotalArea();
    }

    public String toString()
```

```
    {
        return new StringBuilder(Circle.class.getSimpleName())
                    .append('(')
                    .append("radius=").append(String.format("\"%s\"", this.actual_self.getRadius())).append(", ")
                    .append("area=").append(String.format("\"%s\"", this.actual_self.getArea())).append(", ")
                    .append("color=").append(String.format("\"%s\"", this.actual_self.getColor())).append(", ")
                    .append("totalArea=").append(String.format("\"%s\"", this.actual_self.getTotalArea()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_java/src/main/java/shapes/cmlc/Rectangle.java

```
package shapes.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Rectangle extends Shape
{
    double getWidth();

    double getHeight();

    double getArea();

    static Rectangle createRectangle(String color, double width, double height)
    {
        return new RectangleImpl(null, color, width, height);
    }

    static Rectangle extendRectangle(@Nullable Rectangle actual_self, Shape shape, double width, double height)
    {
        return new RectangleImpl(actual_self, shape, width, height);
    }
}

class RectangleImpl implements Rectangle
{
    private final @Nullable Rectangle actual_self;

    private final Shape shape;

    private final double width;
    private final double height;

    RectangleImpl(@Nullable Rectangle actual_self, String color, double width, double height)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = Shape.extendShape(this.actual_self, color);
        this.width = width;
        this.height = height;
    }

    RectangleImpl(@Nullable Rectangle actual_self, Shape shape, double width, double height)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = shape;
        this.width = width;
        this.height = height;
    }

    public double getWidth()
    {
        return this.width;
    }

    public double getHeight()
    {
        return this.height;
    }

    public double getArea()
    {
        return (this.actual_self.getWidth() * this.actual_self.getHeight());
    }

    public String getColor()
    {
        return this.shape.getColor();
    }

    public double getTotalArea()
    {
        return this.shape.getTotalArea();
    }

    public String toString()
    {
        return new StringBuilder(Rectangle.class.getSimpleName())
                    .append('(')
                    .append("width=").append(String.format("\"%s\"", this.actual_self.getWidth())).append(", ")
                    .append("height=").append(String.format("\"%s\"", this.actual_self.getHeight())).append(", ")
                    .append("area=").append(String.format("\"%s\"", this.actual_self.getArea())).append(", ")
                    .append("color=").append(String.format("\"%s\"", this.actual_self.getColor())).append(", ")
                    .append("totalArea=").append(String.format("\"%s\"", this.actual_self.getTotalArea()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_java/src/main/java/shapes/cmlc/RedUnitCircle.java

```
package shapes.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;
```

```
public interface RedUnitCircle extends UnitCircle
{
    String getColor();

    static RedUnitCircle createRedUnitCircle()
    {
        return createRedUnitCircle(3.14159, "Red");
    }

    static RedUnitCircle createRedUnitCircle(double area, String color)
    {
        return new RedUnitCircleImpl(null, area, color);
    }

    static RedUnitCircle extendRedUnitCircle(@Nullable RedUnitCircle actual_self, Shape shape, Circle circle, UnitCircle unitCircle, String color)
    {
        return new RedUnitCircleImpl(actual_self, shape, circle, unitCircle, color);
    }
}

class RedUnitCircleImpl implements RedUnitCircle
{
    private final @Nullable RedUnitCircle actual_self;

    private final Shape shape;
    private final Circle circle;
    private final UnitCircle unitCircle;

    private final String color;

    RedUnitCircleImpl(@Nullable RedUnitCircle actual_self, double area, String color)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = Shape.extendShape(this.actual_self, color);
        this.circle = Circle.extendCircle(this.actual_self, this.shape, 0.0f, color);
        this.unitCircle = UnitCircle.extendUnitCircle(this.actual_self, this.shape, this.circle, area);
        this.color = color;
    }

    RedUnitCircleImpl(@Nullable RedUnitCircle actual_self, Shape shape, Circle circle, UnitCircle unitCircle, String color)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = shape;
        this.circle = circle;
        this.unitCircle = unitCircle;
        this.color = color;
    }

    public String getColor()
    {
        return this.color;
    }

    public double getArea()
    {
        return this.unitCircle.getArea();
    }

    public double getRadius()
    {
        return this.unitCircle.getRadius();
    }

    public double getTotalArea()
    {
        return this.shape.getTotalArea();
    }

    public String toString()
    {
        return new StringBuilder(RedUnitCircle.class.getSimpleName())
                    .append('(')
                    .append("color=").append(String.format("\"%s\"", this.actual_self.getColor())).append(", ")
                    .append("area=").append(String.format("\"%s\"", this.actual_self.getArea())).append(", ")
                    .append("radius=").append(String.format("\"%s\"", this.actual_self.getRadius())).append(", ")
                    .append("totalArea=").append(String.format("\"%s\"", this.actual_self.getTotalArea()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_java/src/main/java/shapes/cmlc/Rhombus.java

```
package shapes.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Rhombus extends Shape
{
    double getP();

    double getQ();

    double getArea();

    static Rhombus createRhombus(String color, double p, double q)
    {
        return new RhombusImpl(null, color, p, q);
    }

    static Rhombus extendRhombus(@Nullable Rhombus actual_self, Shape shape, double p, double q)
    {
        return new RhombusImpl(actual_self, shape, p, q);
    }
}

class RhombusImpl implements Rhombus
{
    private final @Nullable Rhombus actual_self;

    private final Shape shape;

    private final double p;
    private final double q;

    RhombusImpl(@Nullable Rhombus actual_self, String color, double p, double q)
```

```
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = Shape.extendShape(this.actual_self, color);
        this.p = p;
        this.q = q;
    }

    RhombusImpl(@Nullable Rhombus actual_self, Shape shape, double p, double q)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = shape;
        this.p = p;
        this.q = q;
    }

    public double getP()
    {
        return this.p;
    }

    public double getQ()
    {
        return this.q;
    }

    public double getArea()
    {
        return ((this.actual_self.getP() * this.actual_self.getQ()) / 2.0);
    }

    public String getColor()
    {
        return this.shape.getColor();
    }

    public double getTotalArea()
    {
        return this.shape.getTotalArea();
    }

    public String toString()
    {
        return new StringBuilder(Rhombus.class.getSimpleName())
                    .append('(')
                    .append("p=").append(String.format("\"%s\"", this.actual_self.getP())).append(", ")
                    .append("q=").append(String.format("\"%s\"", this.actual_self.getQ())).append(", ")
                    .append("area=").append(String.format("\"%s\"", this.actual_self.getArea())).append(", ")
                    .append("color=").append(String.format("\"%s\"", this.actual_self.getColor())).append(", ")
                    .append("totalArea=").append(String.format("\"%s\"", this.actual_self.getTotalArea()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_java/src/main/java/shapes/cmlc/Shape.java

```
package shapes.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Shape
{
    String getColor();

    double getArea();

    double getTotalArea();

    static Shape extendShape(@Nullable Shape actual_self, String color)
    {
        return new ShapeImpl(actual_self, color);
    }
}

class ShapeImpl implements Shape
{
    private final @Nullable Shape actual_self;

    private final String color;

    ShapeImpl(@Nullable Shape actual_self, String color)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.color = color;
    }

    public String getColor()
    {
        return this.color;
    }

    public double getArea()
    {
        return 0.0f;
    }

    public double getTotalArea()
    {
        return this.actual_self.getArea();
    }

    public String toString()
    {
        return new StringBuilder(Shape.class.getSimpleName())
                    .append('(')
                    .append("color=").append(String.format("\"%s\"", this.actual_self.getColor())).append(", ")
                    .append("area=").append(String.format("\"%s\"", this.actual_self.getArea())).append(", ")
                    .append("totalArea=").append(String.format("\"%s\"", this.actual_self.getTotalArea()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_java/src/main/java/shapes/cmlc/Square.java

```
package shapes.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface Square extends Rectangle, Rhombus
{
    double getSideLength();

    double getWidth();

    double getHeight();

    double getP();

    double getQ();

    double getArea();

    static Square createSquare(String color, double sideLength)
    {
        return new SquareImpl(null, color, sideLength);
    }

    static Square extendSquare(@Nullable Square actual_self, Shape shape, Rectangle rectangle, Rhombus rhombus, double sideLength)
    {
        return new SquareImpl(actual_self, shape, rectangle, rhombus, sideLength);
    }
}

class SquareImpl implements Square
{
    private final @Nullable Square actual_self;

    private final Shape shape;
    private final Rectangle rectangle;
    private final Rhombus rhombus;

    private final double sideLength;

    SquareImpl(@Nullable Square actual_self, String color, double sideLength)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = Shape.extendShape(this.actual_self, color);
        this.rectangle = Rectangle.extendRectangle(this.actual_self, this.shape, 0.0f, 0.0f);
        this.rhombus = Rhombus.extendRhombus(this.actual_self, this.shape, 0.0f, 0.0f);
        this.sideLength = sideLength;
    }

    SquareImpl(@Nullable Square actual_self, Shape shape, Rectangle rectangle, Rhombus rhombus, double sideLength)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = shape;
        this.rectangle = rectangle;
        this.rhombus = rhombus;
        this.sideLength = sideLength;
    }

    public double getSideLength()
    {
        return this.sideLength;
    }

    public double getWidth()
    {
        return this.actual_self.getSideLength();
    }

    public double getHeight()
    {
        return this.actual_self.getSideLength();
    }

    public double getP()
    {
        return (this.actual_self.getSideLength() * 1.41421356237);
    }

    public double getQ()
    {
        return this.actual_self.getP();
    }

    public double getArea()
    {
        return (Math.pow(this.actual_self.getSideLength(), 2.0));
    }

    public String getColor()
    {
        return this.shape.getColor();
    }

    public double getTotalArea()
    {
        return this.shape.getTotalArea();
    }

    public String toString()
    {
        return new StringBuilder(Square.class.getSimpleName())
                    .append('(')
                    .append("sideLength=").append(String.format("\"%s\"", this.actual_self.getSideLength())).append(", ")
                    .append("width=").append(String.format("\"%s\"", this.actual_self.getWidth())).append(", ")
                    .append("height=").append(String.format("\"%s\"", this.actual_self.getHeight())).append(", ")
                    .append("p=").append(String.format("\"%s\"", this.actual_self.getP())).append(", ")
                    .append("q=").append(String.format("\"%s\"", this.actual_self.getQ())).append(", ")
                    .append("area=").append(String.format("\"%s\"", this.actual_self.getArea())).append(", ")
                    .append("color=").append(String.format("\"%s\"", this.actual_self.getColor())).append(", ")
                    .append("totalArea=").append(String.format("\"%s\"", this.actual_self.getTotalArea()))
                    .append(')')
                    .toString();
    }
}
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_java/src/main/java/shapes/cmlc/UnitCircle.java

```
package shapes.cmlc;

import java.util.*;
import java.util.function.*;
import java.math.*;
import org.jetbrains.annotations.*;
import org.jooq.lambda.*;

import static java.util.Arrays.*;
import static java.util.Collections.*;
import static java.util.stream.Collectors.*;
import static org.jooq.lambda.Seq.*;

public interface UnitCircle extends Circle
{
    double getArea();

    double getRadius();

    static UnitCircle createUnitCircle()
    {
        return createUnitCircle("Blue", 3.14159);
    }

    static UnitCircle createUnitCircle(String color, double area)
    {
        return new UnitCircleImpl(null, color, area);
    }

    static UnitCircle extendUnitCircle(@Nullable UnitCircle actual_self, Shape shape, Circle circle, double area)
    {
        return new UnitCircleImpl(actual_self, shape, circle, area);
    }
}

class UnitCircleImpl implements UnitCircle
{
    private final @Nullable UnitCircle actual_self;

    private final Shape shape;
    private final Circle circle;

    private final double area;

    UnitCircleImpl(@Nullable UnitCircle actual_self, String color, double area)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = Shape.extendShape(this.actual_self, color);
        this.circle = Circle.extendCircle(this.actual_self, this.shape, 0.0f, color);
        this.area = area;
    }

    UnitCircleImpl(@Nullable UnitCircle actual_self, Shape shape, Circle circle, double area)
    {
        this.actual_self = actual_self == null ? this : actual_self;

        this.shape = shape;
        this.circle = circle;
        this.area = area;
    }

    public double getArea()
    {
        return this.area;
    }

    public double getRadius()
    {
        return 1.0;
    }

    public String getColor()
    {
        return this.circle.getColor();
    }

    public double getTotalArea()
    {
        return this.shape.getTotalArea();
    }

    public String toString()
    {
        return new StringBuilder(UnitCircle.class.getSimpleName())
                    .append('(')
                    .append("area=").append(String.format("\"%s\"", this.actual_self.getArea())).append(", ")
                    .append("radius=").append(String.format("\"%s\"", this.actual_self.getRadius())).append(", ")
                    .append("color=").append(String.format("\"%s\"", this.actual_self.getColor())).append(", ")
                    .append("totalArea=").append(String.format("\"%s\"", this.actual_self.getTotalArea()))
                    .append(')')
                    .toString();
    }
}
```

```
==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_py/cml-compiler-output.txt

model files:
- setup.py
- cml_specializations_redefinitions_shapes_cmlc/__init__.py


==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_py/cml_specializations_redefinitions_shapes_cmlc/__init__.py

from typing import *
from abc import *
from decimal import *

import functools, itertools

class Shape(ABC):

    @abstractproperty
    def color(self) -> 'str':
        pass

    @abstractproperty
    def area(self) -> 'float':
        pass

    @abstractproperty
    def total_area(self) -> 'float':
        pass

    @staticmethod
    def extend_shape(actual_self: 'Optional[Shape]', color: 'str') -> 'Shape':
        return ShapeImpl(actual_self, color)
```

```
class ShapeImpl(Shape):

    def __init__(self, actual_self: 'Optional[Shape]', color: 'str') -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Shape]
        else:
            self.__actual_self = actual_self

        self.__color = color

    @property
    def color(self) -> 'str':
        return self.__color

    @property
    def area(self) -> 'float':
        return 0

    @property
    def total_area(self) -> 'float':
        return self.__actual_self.area

    def __str__(self) -> 'str':
        return "%s(color=%s, area=%s, total_area=%s)" % (
            type(self).__name__,
            self.__actual_self.color,
            self.__actual_self.area,
            self.__actual_self.total_area
        )


class Rectangle(Shape, ABC):

    @abstractproperty
    def width(self) -> 'float':
        pass

    @abstractproperty
    def height(self) -> 'float':
        pass

    @abstractproperty
    def area(self) -> 'float':
        pass

    @staticmethod
    def create_rectangle(color: 'str', width: 'float', height: 'float') -> 'Rectangle':
        return RectangleImpl(None, None, width, height, color=color)

    @staticmethod
    def extend_rectangle(actual_self: 'Optional[Rectangle]', shape: 'Shape', width: 'float', height: 'float') -> 'Rectangle':
        return RectangleImpl(actual_self, shape, width, height)


class RectangleImpl(Rectangle):

    def __init__(self, actual_self: 'Optional[Rectangle]', shape: 'Optional[Shape]', width: 'float', height: 'float', **kwargs) -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Rectangle]
        else:
            self.__actual_self = actual_self

        if shape is None:
            self.__shape = Shape.extend_shape(self.__actual_self, kwargs['color'])
        else:
            self.__shape = shape
        self.__width = width
        self.__height = height

    @property
    def width(self) -> 'float':
        return self.__width

    @property
    def height(self) -> 'float':
        return self.__height

    @property
    def area(self) -> 'float':
        return (self.__actual_self.width * self.__actual_self.height)

    @property
    def color(self) -> 'str':
        return self.__shape.color

    @property
    def total_area(self) -> 'float':
        return self.__shape.total_area

    def __str__(self) -> 'str':
        return "%s(width=%s, height=%s, area=%s, color=%s, total_area=%s)" % (
            type(self).__name__,
            self.__actual_self.width,
            self.__actual_self.height,
            self.__actual_self.area,
            self.__actual_self.color,
            self.__actual_self.total_area
        )


class Rhombus(Shape, ABC):

    @abstractproperty
    def p(self) -> 'float':
        pass

    @abstractproperty
    def q(self) -> 'float':
        pass

    @abstractproperty
    def area(self) -> 'float':
        pass

    @staticmethod
    def create_rhombus(color: 'str', p: 'float', q: 'float') -> 'Rhombus':
        return RhombusImpl(None, None, p, q, color=color)

    @staticmethod
    def extend_rhombus(actual_self: 'Optional[Rhombus]', shape: 'Shape', p: 'float', q: 'float') -> 'Rhombus':
        return RhombusImpl(actual_self, shape, p, q)


class RhombusImpl(Rhombus):
```

```
    def __init__(self, actual_self: 'Optional[Rhombus]', shape: 'Optional[Shape]', p: 'float', q: 'float', **kwargs) -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Rhombus]
        else:
            self.__actual_self = actual_self

        if shape is None:
            self.__shape = Shape.extend_shape(self.__actual_self, kwargs['color'])
        else:
            self.__shape = shape
        self.__p = p
        self.__q = q

    @property
    def p(self) -> 'float':
        return self.__p

    @property
    def q(self) -> 'float':
        return self.__q

    @property
    def area(self) -> 'float':
        return ((self.__actual_self.p * self.__actual_self.q) / 2.0)

    @property
    def color(self) -> 'str':
        return self.__shape.color

    @property
    def total_area(self) -> 'float':
        return self.__shape.total_area

    def __str__(self) -> 'str':
        return "%s(p=%s, q=%s, area=%s, color=%s, total_area=%s)" % (
            type(self).__name__,
            self.__actual_self.p,
            self.__actual_self.q,
            self.__actual_self.area,
            self.__actual_self.color,
            self.__actual_self.total_area
        )


class Square(Rectangle, Rhombus, ABC):

    @abstractproperty
    def side_length(self) -> 'float':
        pass

    @abstractproperty
    def width(self) -> 'float':
        pass

    @abstractproperty
    def height(self) -> 'float':
        pass

    @abstractproperty
    def p(self) -> 'float':
        pass

    @abstractproperty
    def q(self) -> 'float':
        pass

    @abstractproperty
    def area(self) -> 'float':
        pass

    @staticmethod
    def create_square(color: 'str', side_length: 'float') -> 'Square':
        return SquareImpl(None, None, None, None, side_length, color=color)

    @staticmethod
    def extend_square(actual_self: 'Optional[Square]', shape: 'Shape', rectangle: 'Rectangle', rhombus: 'Rhombus', side_length: 'float') -> 'Square
        return SquareImpl(actual_self, shape, rectangle, rhombus, side_length)


class SquareImpl(Square):

    def __init__(self, actual_self: 'Optional[Square]', shape: 'Optional[Shape]', rectangle: 'Optional[Rectangle]', rhombus: 'Optional[Rhombus]', s
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Square]
        else:
            self.__actual_self = actual_self

        if shape is None:
            self.__shape = Shape.extend_shape(self.__actual_self, kwargs['color'])
        else:
            self.__shape = shape
        if rectangle is None:
            self.__rectangle = Rectangle.extend_rectangle(self.__actual_self, self.__shape, 0, 0)
        else:
            self.__rectangle = rectangle
        if rhombus is None:
            self.__rhombus = Rhombus.extend_rhombus(self.__actual_self, self.__shape, 0, 0)
        else:
            self.__rhombus = rhombus
        self.__side_length = side_length

    @property
    def side_length(self) -> 'float':
        return self.__side_length

    @property
    def width(self) -> 'float':
        return self.__actual_self.side_length

    @property
    def height(self) -> 'float':
        return self.__actual_self.side_length

    @property
    def p(self) -> 'float':
        return (self.__actual_self.side_length * 1.41421356237)

    @property
    def q(self) -> 'float':
        return self.__actual_self.p

    @property
    def area(self) -> 'float':
        return (self.__actual_self.side_length ** 2.0)

    @property
    def color(self) -> 'str':
```

```
        return self.__shape.color

    @property
    def total_area(self) -> 'float':
        return self.__shape.total_area

    def __str__(self) -> 'str':
        return "%s(side_length=%s, width=%s, height=%s, p=%s, q=%s, area=%s, color=%s, total_area=%s)" % (
            type(self).__name__,
            self.__actual_self.side_length,
            self.__actual_self.width,
            self.__actual_self.height,
            self.__actual_self.p,
            self.__actual_self.q,
            self.__actual_self.area,
            self.__actual_self.color,
            self.__actual_self.total_area
        )


class Circle(Shape, ABC):

    @abstractproperty
    def radius(self) -> 'float':
        pass

    @abstractproperty
    def area(self) -> 'float':
        pass

    @abstractproperty
    def color(self) -> 'str':
        pass

    @staticmethod
    def create_circle(radius: 'float', color: 'str' = "Blue") -> 'Circle':
        return CircleImpl(None, None, radius, color)

    @staticmethod
    def extend_circle(actual_self: 'Optional[Circle]', shape: 'Shape', radius: 'float', color: 'str' = "Blue") -> 'Circle':
        return CircleImpl(actual_self, shape, radius, color)


class CircleImpl(Circle):

    def __init__(self, actual_self: 'Optional[Circle]', shape: 'Optional[Shape]', radius: 'float', color: 'str' = "Blue") -> 'None':
        if actual_self is None:
            self.__actual_self = self  # type: Optional[Circle]
        else:
            self.__actual_self = actual_self

        if shape is None:
            self.__shape = Shape.extend_shape(self.__actual_self, color)
        else:
            self.__shape = shape
        self.__radius = radius
        self.__color = color

    @property
    def radius(self) -> 'float':
        return self.__radius

    @property
    def color(self) -> 'str':
        return self.__color

    @property
    def area(self) -> 'float':
        return (3.14159 * (self.__actual_self.radius ** 2.0))

    @property
    def total_area(self) -> 'float':
        return self.__shape.total_area

    def __str__(self) -> 'str':
        return "%s(radius=%s, area=%s, color=%s, total_area=%s)" % (
            type(self).__name__,
            self.__actual_self.radius,
            self.__actual_self.area,
            self.__actual_self.color,
            self.__actual_self.total_area
        )


class UnitCircle(Circle, ABC):

    @abstractproperty
    def area(self) -> 'float':
        pass

    @abstractproperty
    def radius(self) -> 'float':
        pass

    @staticmethod
    def create_unit_circle(color: 'str' = "Blue", area: 'float' = 3.14159) -> 'UnitCircle':
        return UnitCircleImpl(None, None, None, area, color=color)

    @staticmethod
    def extend_unit_circle(actual_self: 'Optional[UnitCircle]', shape: 'Shape', circle: 'Circle', area: 'float' = 3.14159) -> 'UnitCircle':
        return UnitCircleImpl(actual_self, shape, circle, area)


class UnitCircleImpl(UnitCircle):

    def __init__(self, actual_self: 'Optional[UnitCircle]', shape: 'Optional[Shape]', circle: 'Optional[Circle]', area: 'float' = 3.14159, **kwargs
        if actual_self is None:
            self.__actual_self = self  # type: Optional[UnitCircle]
        else:
            self.__actual_self = actual_self

        if shape is None:
            self.__shape = Shape.extend_shape(self.__actual_self, kwargs['color'])
        else:
            self.__shape = shape
        if circle is None:
            self.__circle = Circle.extend_circle(self.__actual_self, self.__shape, 0, kwargs['color'])
        else:
            self.__circle = circle
        self.__area = area

    @property
    def area(self) -> 'float':
        return self.__area

    @property
    def radius(self) -> 'float':
```

```
        return 1.0

    @property
    def color(self) -> 'str':
        return self.__circle.color

    @property
    def total_area(self) -> 'float':
        return self.__shape.total_area

    def __str__(self) -> 'str':
        return "%s(area=%s, radius=%s, color=%s, total_area=%s)" % (
            type(self).__name__,
            self.__actual_self.area,
            self.__actual_self.radius,
            self.__actual_self.color,
            self.__actual_self.total_area
        )


class RedUnitCircle(UnitCircle, ABC):

    @abstractproperty
    def color(self) -> 'str':
        pass

    @staticmethod
    def create_red_unit_circle(area: 'float' = 3.14159, color: 'str' = "Red") -> 'RedUnitCircle':
        return RedUnitCircleImpl(None, None, None, None, color, area=area)

    @staticmethod
    def extend_red_unit_circle(actual_self: 'Optional[RedUnitCircle]', shape: 'Shape', circle: 'Circle', unit_circle: 'UnitCircle', color: 'str' =
        return RedUnitCircleImpl(actual_self, shape, circle, unit_circle, color)


class RedUnitCircleImpl(RedUnitCircle):

    def __init__(self, actual_self: 'Optional[RedUnitCircle]', shape: 'Optional[Shape]', circle: 'Optional[Circle]', unit_circle: 'Optional[UnitCir
        if actual_self is None:
            self.__actual_self = self  # type: Optional[RedUnitCircle]
        else:
            self.__actual_self = actual_self

        if shape is None:
            self.__shape = Shape.extend_shape(self.__actual_self, color)
        else:
            self.__shape = shape
        if circle is None:
            self.__circle = Circle.extend_circle(self.__actual_self, self.__shape, 0, color)
        else:
            self.__circle = circle
        if unit_circle is None:
            self.__unit_circle = UnitCircle.extend_unit_circle(self.__actual_self, self.__shape, self.__circle, kwargs['area'])
        else:
            self.__unit_circle = unit_circle
        self.__color = color

    @property
    def color(self) -> 'str':
        return self.__color

    @property
    def area(self) -> 'float':
        return self.__unit_circle.area

    @property
    def radius(self) -> 'float':
        return self.__unit_circle.radius

    @property
    def total_area(self) -> 'float':
        return self.__shape.total_area

    def __str__(self) -> 'str':
        return "%s(color=%s, area=%s, radius=%s, total_area=%s)" % (
            type(self).__name__,
            self.__actual_self.color,
            self.__actual_self.area,
            self.__actual_self.radius,
            self.__actual_self.total_area
        )
```

==> cml-modules/cml_base/tests/specializations/redefinitions/expected/shapes_cmlc_py/setup.py

```
"""A setuptools based setup module.

See:
https://packaging.python.org/en/latest/distributing.html
https://github.com/pypa/sampleproject
"""

# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

setup(
    name='cml_specializations_redefinitions_shapes_cmlc',

    # Versions should comply with PEP440.  For a discussion on single-sourcing
    # the version across setup.py and the project code, see
    # https://packaging.python.org/en/latest/single_source_version.html
    version='1.0',

    description='A sample Python project',

    # The project's main homepage.
    url='https://github.com/pypa/sampleproject',

    # Author details
    author='The Python Packaging Authority',
    author_email='pypa-dev@googlegroups.com',

    # Choose your license
    license='MIT',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',
```

```
        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: MIT License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3.6.1',
    ],

    # What does your project relate to?
    keywords='sample setuptools development',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests']),

    # Alternatively, if you want to distribute just a my_module.py, uncomment
    # this:
    #   py_modules=["my_module"],

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    # install_requires=['peppercorn'],

    # List additional groups of dependencies here (e.g. development
    # dependencies). You can install these using the following syntax,
    # for example:
    # $ pip install -e .[dev,test]
    extras_require={
        'dev': ['check-manifest'],
        'test': ['coverage'],
    },

    # If there are data files included in your packages that need to be
    # installed, specify them here.  If using Python 2.6 or less, then these
    # have to be included in MANIFEST.in as well.
    # package_data={
    #    'sample': ['package_data.dat'],
    #},

    # Although 'package_data' is the preferred approach, in some case you may
    # need to place data files outside of your packages. See:
    # http://docs.python.org/3.4/distutils/setupscript.html#installing-additional-files # noqa
    # In this case, 'data_file' will be installed into '<sys.prefix>/my_data'
    #data_files=[('my_data', ['data/data_file'])],

    # To provide executable scripts, use entry points in preference to the
    # "scripts" keyword. Entry points provide cross-platform support and allow
    # pip to create the appropriate form of executable for the target platform.
    #entry_points={
    #    'console_scripts': [
    #        'sample=sample:main',
    #    ],
    #},
)
```

```
==> cml-modules/cml_base/tests/specializations/redefinitions/source/main.cml
```

```
-- Generalization of Circle and Rectangle.
-- As an abstraction, no direct instances of Shape are ever created.
@abstraction Shape
{
    -- Abstractions may also have concrete properties.
    -- Specializations below share the color attribute as-is.
    color: String;

    -- Specializations below redefine the area attribute:
    -- A derived property without an expression is considered abstract.
    -- Only abstractions may have abstract properties.
    /area: Double;

    -- If specializations redefine "area", "total_area" should match its value.
    /total_area = area;
}

// Specialization of Shape:
@concept Rectangle: Shape
{
    // New attributes that characterize a rectangle:
    width: Double;
    height: Double;

    // Redefinition of the area attribute:
    /area = width * height;
}

// Another specialization of Shape:
@concept Rhombus: Shape
{
    // Diagonal attributes that characterize a rhombus:
    p: Double;
    q: Double;

    // Another redefinition of the area attribute:
    /area = (p * q) / 2.0d;
}

// Specialization of both Rectangle and Rhombus:
@concept Square: Rectangle, Rhombus
{
    // Only attribute needed to characterize a square:
    side_length: Double;

    // Redefinitions of Rectangle's attributes:
    /width = side_length;
    /height = side_length;

    // Redefinitions of Rhombus' attributes:
    /p = side_length * 1.41421356237f; // square root of 2
    /q = p;

    // Required to redefine area in order to resolve conflict
    // between Rectangle's area and Rhombus' area:
    /area = side_length ^ 2.0f;
}

-- All instances of Circle are in turn instances of Shape.
@concept Circle: Shape
{
    radius: Double;
```

```
    -- In order to be considered a concrete concept,
    -- Circle must redefine the abstract properties
    -- inherited from Shape.
    /area = 3.14159d * radius ^ 2.0f;

    -- Circle may also redefine concrete properties of Shape.
    -- However, the redefinition is not required in this case.
    color = "Blue";
}

@concept UnitCircle: Circle
{
    -- Observe below that the redefinition of
    -- an abstract property may be concrete;
    -- that is, it does not have to be derived
    -- as it was done in Circle.
    area = 3.14159d;

    -- In the case above, however,
    -- it is desirable to redefine "area" as a derived property,
    -- in order to guarantee area's value cannot be modified
    -- after the instantiation of UnitCircle.
    -- This is done with the redefinition of "radius" below.
    -- Notice that, in Circle, radius was concrete,
    -- but its redefinition below makes it derived.
    -- That's allowed in CML just as the other way around,
    -- as it was done with "area" above.
    /radius = 1.0d;
}

@concept RedUnitCircle: UnitCircle
{
    color = "Red";
}

@task shapes_cmlc_java: cmlc_java
{
    groupId = "cml-specializations-redefinitions";
    artifactId = "shapes-cmlc";
    artifactVersion = "1.0-SNAPSHOT";
    packageName = "shapes.cmlc";
    packagePath = "shapes/cmlc";
}

@task shapes_cmlc_py: cmlc_py
{
    moduleName = "cml_specializations_redefinitions_shapes_cmlc";
    moduleVersion = "1.0";
}



==> cml-acceptance/cases/errors/constructor_undefined/compiler-output.txt

Error: no constructor defined for task: some_task


==> cml-acceptance/cases/errors/constructor_undefined/source/main.cml

@concept Book;

// Task's constructor not defined on purpose:
@task some_task;


==> cml-acceptance/cases/errors/constructor_unknown/compiler-output.txt

Error: unable to find templates for constructor: unknown


==> cml-acceptance/cases/errors/constructor_unknown/source/main.cml

@task unknown_constructor_task: unknown;


==> cml-acceptance/cases/errors/invalid_module_name/compiler-output.txt

Error: Module declaration name (some_module_name) should match the module's directory name: invalid_module_name


==> cml-acceptance/cases/errors/invalid_module_name/source/main.cml


// Module name should be: invalid_module_name
@module some_module_name
{
}

@concept SomeConcept;

@task some_task: poj;


==> cml-acceptance/cases/errors/missing-source/compiler-output.txt

Error: no source files in module: missing-source


==> cml-acceptance/cases/errors/missing-source/source/READ.txt

This directory does not have the main.cml source file purposely. It is used by one of the acceptance tests.

==> cml-acceptance/cases/errors/missing_ancestor/compiler-output.txt

Error: Unable to find ancestors: MissingAncestor, AnotherMissingAncestor


==> cml-acceptance/cases/errors/missing_ancestor/source/main.cml


@concept SomeConcept: MissingAncestor, AnotherMissingAncestor;

@task some_task;


==> cml-acceptance/cases/errors/parsing_failed/compiler-output.txt

Syntax Error: mismatched input '<EOF>' expecting NAME (7:1)
Error: No name provided for concept.


==> cml-acceptance/cases/errors/parsing_failed/READ.txt
```

```
The main.cml source file is purposely empty; in order to fail the parsing.

==> cml-acceptance/cases/errors/parsing_failed/source/main.cml


@task some_task;

// Syntax error - abstract keyword after the concept keyword:
@concept


==> cml-acceptance/cases/errors/task_undeclared/compiler-output.txt

Error: no source file has declared task named: some_task_name


==> cml-acceptance/cases/errors/task_undeclared/README.txt

No source dir, thus no task declared.

==> cml-acceptance/cases/target_dirs/target_dir_cleaned/compiler-output.txt

model files:
- pom.xml

BookStore files:
- src/main/java/books/BookStore.java


==> cml-acceptance/cases/target_dirs/target_dir_cleaned/source/main.cml

@concept BookStore;

@task some_task: poj
{
    groupId = "books";
    artifactId = "books";
    artifactVersion = "1.0.1-SNAPSHOT";
    packageName = "books";
    packagePath = "books";
}



==> cml-acceptance/cases/target_dirs/target_dir_created/compiler-output.txt

model files:
- pom.xml

Book files:
- src/main/java/books/Book.java


==> cml-acceptance/cases/target_dirs/target_dir_created/source/main.cml

@concept Book;

@task some_task: poj
{
    groupId = "books";
    artifactId = "books";
    artifactVersion = "1.0.1-SNAPSHOT";
    packageName = "books";
    packagePath = "books";
}



==> cml-acceptance/cases/validations/abstract_property_in_abstract_concept/compiler-output.txt


Failed validation: required abstract_property_in_abstract_concept: in property Shape.area: double (5:5)

Failed validation: required abstract_property_redefinition: in concept CircleA (9:1)
- property Shape.area: double (5:5)


==> cml-acceptance/cases/validations/abstract_property_in_abstract_concept/source/main.cml

-- Shape should be tagged abstract since it has an abstract property:
@concept Shape
{
    -- A derived property without an expression is considered abstract:
    /area: Double;
}

-- In order to be considered concrete, Circle should have redefined "area".
@concept CircleA: Shape;

-- Properly redefined "area" as a concrete property:
@concept CircleB: Shape
{
    area = 4.0d;
}

-- OK to miss redefinition in this specialization because it is an abstraction:
@abstraction CircleC: Shape;

-- Properly redefined "area" as a derived property:
@concept UnitCircle: CircleA
{
    /area = 1.0d;
}



==> cml-acceptance/cases/validations/association_end_property_found_in_model/compiler-output.txt


Failed validation: required association_end_property_found_in_model: in association end Employee.employer (15:5)
- concept Employee (2:1)

Failed validation: required association_end_property_found_in_model: in association end Organiza.employees (16:5)


==> cml-acceptance/cases/validations/association_end_property_found_in_model/source/main.cml


@concept Employee
{
    name: String;
}

@concept Organization
{
```

```
    name: String;
    employees: Employee*;
}

@association Employment
{
    Employee.employer;
    Organiza.employees;
}
```

```
==> cml-acceptance/cases/validations/association_end_type_matches_property_type/compiler-output.txt
```

```
Failed validation: required association_end_type_matches_property_type: in association end Vehicle.owner: Organization* (15:5)
- property Vehicle.owner: Organization (4:5)

Failed validation: required association_end_type_matches_property_type: in association end Organization.fleet: Vehicle (16:5)
- property Organization.fleet: Vehicle* (10:5)
```

```
==> cml-acceptance/cases/validations/association_end_type_matches_property_type/source/main.cml
```

```
@concept Vehicle
{
    plate: String;
    owner: Organization;
}

@concept Organization
{
    name: String;
    fleet: Vehicle*;
}

@association VehicleOwnership
{
    Vehicle.owner: Organization*;
    Organization.fleet: Vehicle;
}
```

```
==> cml-acceptance/cases/validations/association_end_types_must_match/compiler-output.txt
```

```
Failed validation: required association_end_types_must_match: in association Employment (18:1)
- association end Employee.employer (20:5)
- association end Employee.employer (21:5)

Failed validation: required association_end_types_must_match: in association VehicleOwnership (24:1)
- association end Vehicle.owner: Organization (26:5)
- association end Organization.employees: Employee* (27:5)
```

```
==> cml-acceptance/cases/validations/association_end_types_must_match/source/main.cml
```

```
@concept Vehicle
{
    driver: Employee?;
    owner: Organization;
}

@concept Employee
{
    employer: Organization;
}

@concept Organization
{
    employees: Employee*;
    fleet: Vehicle*;
}

@association Employment
{
    Employee.employer;
    Employee.employer;
}

@association VehicleOwnership
{
    Vehicle.owner: Organization;
    Organization.employees: Employee*;
}
```

```
==> cml-acceptance/cases/validations/association_must_have_two_association_ends/compiler-output.txt
```

```
Failed validation: required association_must_have_two_association_ends: in association Employment (18:1)
- association end Employee.employer (20:5)
- association end Organization.employees (21:5)
- association end Vehicle.owner: Organization (22:5)

Failed validation: required association_must_have_two_association_ends: in association VehicleOwnership (25:1)
- association end Organization.fleet: Vehicle* (27:5)

Failed validation: required association_must_have_two_association_ends: in association Empty (30:1)
```

```
==> cml-acceptance/cases/validations/association_must_have_two_association_ends/source/main.cml
```

```
@concept Vehicle
{
    driver: Employee?;
    owner: Organization;
}

@concept Employee
{
    employer: Organization;
}

@concept Organization
{
    employees: Employee*;
    fleet: Vehicle*;
}

@association Employment
{
    Employee.employer;
    Organization.employees;
    Vehicle.owner: Organization;
}
```

```
@association VehicleOwnership
{
    Organization.fleet: Vehicle*;
}

@association Empty {}
```

==> cml-acceptance/cases/validations/property_type_assignable_from_expression_type/compiler-output.txt

```
Failed validation: required property_type_assignable_from_expression_type: in property Attributes.bad_prop1: integer (12:5)
Declared type is integer but type inferred from expression is string.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.bad_prop2: string (13:5)
Declared type is string but type inferred from expression is integer.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.bad_prop3: double (14:5)
Declared type is double but type inferred from expression is decimal.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.decimal_to_double: double (25:5)
Declared type is double but type inferred from expression is decimal.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.double_to_decimal: decimal (26:5)
Declared type is decimal but type inferred from expression is double.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.decimal_to_byte: byte (27:5)
Declared type is byte but type inferred from expression is decimal.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.a_to_b: B (28:5)
Declared type is B but type inferred from expression is A.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.a6: A (51:5)
Declared type is A but type inferred from expression is A*.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.a7: A? (52:5)
Declared type is A? but type inferred from expression is A*.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.a8: A (53:5)
Declared type is A but type inferred from expression is A?.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.ab7: A (54:5)
Declared type is A but type inferred from expression is A*.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.ab8: A? (55:5)
Declared type is A? but type inferred from expression is A*.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.ab9: A (56:5)
Declared type is A but type inferred from expression is A?.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.d4: decimal (57:5)
Declared type is decimal but type inferred from expression is decimal*.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.d5: decimal? (58:5)
Declared type is decimal? but type inferred from expression is decimal*.

Failed validation: required property_type_assignable_from_expression_type: in property Attributes.d6: decimal (59:5)
Declared type is decimal but type inferred from expression is decimal?.
```

==> cml-acceptance/cases/validations/property_type_assignable_from_expression_type/source/main.cml

```
@concept A;
@concept B: A;

@abstraction Attributes
{
    a: A;
    b: B;
    b2: B?;
    b3: B*;

    // Type and expression mismatch:
    bad_prop1: INTEGER = "STRING"; // type is INTEGER but expression is STRING
    /bad_prop2: STRING = 0; // type is STRING but expression is INTEGER
    /bad_prop3: DOUBLE = 2.0; // type is DOUBLE but expression is DECIMAL

    // Assignable Types:
    byte_to_short: SHORT = 1b;
    short_to_int: INTEGER = 1s;
    int_to_long: LONG = 1;
    long_to_decimal: DECIMAL = 1l;
    float_to_double: DOUBLE = 1.0f;
    b_to_a: A = b;

    // Non-Assignable Types:
    decimal_to_double: DOUBLE = 1.0;
    double_to_decimal: DECIMAL = 1.0d;
    decimal_to_byte: BYTE = 1.0;
    a_to_b: B = a;

    // Valid cases where either the type is specified or an expression is provided:
    prop1: STRING; // type specified
    prop2 = 0; // init expression specified - type inferred
    /prop3 = 1.0; // derived expression specified - type inferred
    /prop4: BOOLEAN; // type specified for abstract property

    // Cardinality - Assignable:
    a2: A = a;
    a3: A? = a;
    a4: A* = a2;
    a5: A* = a3;
    ab2: A = b;
    ab3: A? = b;
    ab4: A* = b;
    ab5: A* = b2;
    ab6: A* = b3;
    d1: DECIMAL? = long_to_decimal;
    d2: DECIMAL* = long_to_decimal;
    d3: DECIMAL* = d1;

    // Cardinality - Non-Assignable:
    a6: A = a4;
    a7: A? = a5;
    a8: A = a3;
    ab7: A = ab4;
    ab8: A? = ab5;
    ab9: A = ab3;
    d4: DECIMAL = d2;
    d5: DECIMAL? = d3;
    d6: DECIMAL = d1;
}
```

==> cml-acceptance/cases/validations/property_type_specified_or_inferred/compiler-output.txt

```
Failed validation: required property_type_specified_or_inferred: in property Attributes.bad_prop1: UNDEFINED (4:5)
```

```
No type or expression defined for property: bad_prop1

Failed validation: required property_type_specified_or_inferred: in property Attributes.bad_prop2: UNDEFINED (5:5)
No type or expression defined for property: bad_prop2


==> cml-acceptance/cases/validations/property_type_specified_or_inferred/source/main.cml

@abstraction Attributes
{
    // Missing type and expression:
    bad_prop1;
    /bad_prop2;

    // All valid cases:
    prop1: String; // type specified
    prop2 = 0; // init expression specified
    /prop3 = 1.0; // derived expression specified
    /prop4: Boolean; // type specified for abstract property
}

==> cml-acceptance/cases/validations/unique_property_name/compiler-output.txt


Failed validation: required unique_property_name: in property Attributes.prop_name: string (3:5)
- property Attributes.prop_name: integer (4:5)
- property Attributes.prop_name: decimal (5:5)
- property Attributes.prop_name: boolean (6:5)

Failed validation: required unique_property_name: in property Attributes.prop_name: integer (4:5)
- property Attributes.prop_name: string (3:5)
- property Attributes.prop_name: decimal (5:5)
- property Attributes.prop_name: boolean (6:5)

Failed validation: required unique_property_name: in property Attributes.prop_name: decimal (5:5)
- property Attributes.prop_name: string (3:5)
- property Attributes.prop_name: integer (4:5)
- property Attributes.prop_name: boolean (6:5)

Failed validation: required unique_property_name: in property Attributes.prop_name: boolean (6:5)
- property Attributes.prop_name: string (3:5)
- property Attributes.prop_name: integer (4:5)
- property Attributes.prop_name: decimal (5:5)


==> cml-acceptance/cases/validations/unique_property_name/source/main.cml

@abstraction Attributes
{
    prop_name: STRING;
    prop_name = 0;
    /prop_name = 1.0;
    /prop_name: BOOLEAN;
}

==> cml-acceptance/pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>cml</groupId>
    <artifactId>cml-acceptance</artifactId>
    <version>1.1.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>15.0</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.maven.shared</groupId>
            <artifactId>maven-invoker</artifactId>
            <version>2.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.codehaus.plexus</groupId>
            <artifactId>plexus-utils</artifactId>
            <version>3.0.22</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>21.0</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.jooq</groupId>
            <artifactId>jool</artifactId>
            <version>0.9.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>


==> cml-acceptance/src/test/java/cml/acceptance/AcceptanceTest.java

package cml.acceptance;
```

```
import com.google.common.io.Files;
import org.apache.maven.shared.invoker.*;
import org.codehaus.plexus.util.cli.CommandLineException;
import org.codehaus.plexus.util.cli.Commandline;
import org.codehaus.plexus.util.cli.WriterStreamConsumer;
import org.jooq.lambda.Seq;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.experimental.theories.DataPoints;
import org.junit.experimental.theories.FromDataPoints;
import org.junit.experimental.theories.Theories;
import org.junit.experimental.theories.Theory;
import org.junit.runner.RunWith;

import java.io.*;
import java.nio.charset.Charset;
import java.util.List;

import static java.util.Arrays.asList;
import static java.util.Collections.singletonList;
import static junit.framework.TestCase.assertEquals;
import static org.codehaus.plexus.util.FileUtils.*;
import static org.codehaus.plexus.util.cli.CommandLineUtils.executeCommandLine;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.core.Is.is;
import static org.jooq.lambda.Seq.seq;

@RunWith(Theories.class)
public class AcceptanceTest
{
    private static final Charset OUTPUT_FILE_ENCODING = Charset.forName("UTF-8");
    private static final int PROCESS_TIMEOUT_IN_SECONDS = 360;

    private static final String BASE_DIR = "..";

    private static final String COMPILER_DIR = BASE_DIR + "/cml-compiler";
    private static final String FRONTEND_DIR = COMPILER_DIR + "/cml-frontend";
    private static final String FRONTEND_TARGET_DIR = FRONTEND_DIR + "/target";
    private static final String COMPILER_JAR = FRONTEND_TARGET_DIR + "/cml-compiler-jar-with-dependencies.jar";

    private static final String CML_MODULES_BASE_DIR = BASE_DIR + "/" + "cml-modules";

    private static final String CML_BOOTSTRAPPING_BASE_DIR = COMPILER_DIR + "/" + "cml-bootstrapping";

    private static final String POJ = "poj"; // plain old Java

    private static final String JAVA = "java";
    private static final String PYTHON = "py";

    private static final int SUCCESS = 0;
    private static final int FAILURE_SOURCE_FILE_NOT_FOUND = 2;
    private static final int FAILURE_FAILED_LOADING_MODEL = 3;
    private static final int FAILURE_MODEL_VALIDATION = 4;
    private static final int FAILURE_CONSTRUCTOR_UNKNOWN = 101;
    private static final int FAILURE_CONSTRUCTOR_UNDEFINED = 102;
    private static final int FAILURE_TASK_UNDECLARED = 103;
    private static final String SOME_TASK = "some_task";

    @DataPoints("validation-modules")
    public static String[] validationModules = {
        "abstract_property_in_abstract_concept",
        "association_end_property_found_in_model",
        "association_end_type_matches_property_type",
        "association_end_types_must_match",
        "association_must_have_two_association_ends",
        "property_type_specified_or_inferred",
        "property_type_assignable_from_expression_type",
        "unique_property_name"
    };

    @DataPoints("failing-modules")
    public static String[] failingModules = {
        "invalid_module_name",
        "missing_ancestor",
        "parsing_failed"
    };

    @BeforeClass
    public static void buildCompiler() throws MavenInvocationException
    {
        buildMavenModule(COMPILER_DIR);

        final File targetDir = new File(FRONTEND_TARGET_DIR);
        assertThat("Target dir must exist: " + targetDir, targetDir.exists(), is(true));
    }

    @Test
    public void cml_modules()
    {
        moduleDirs(CML_MODULES_BASE_DIR).forEach(this::testModule);
    }

    @Test
    public void cml_bootstrapping_modules()
    {
        moduleDirs(CML_BOOTSTRAPPING_BASE_DIR).forEach(this::testModule);
    }

    private Seq<File> moduleDirs(String baseDir)
    {
        final File[] subDirs = new File(baseDir).listFiles(File::isDirectory);

        return seq(asList(subDirs == null ? new File[0] : subDirs)).filter(AcceptanceTest::isModuleDir);
    }

    private static boolean isModuleDir(File subDir)
    {
        return new File(subDir, "source").isDirectory() || new File(subDir, "templates").isDirectory();
    }

    private void testModule(File moduleDir)
    {
        try
        {
            System.out.println("--------");
            System.out.println("Testing module: " + moduleDir.getName());

            executeJar(moduleDir.getCanonicalPath(), COMPILER_JAR, singletonList("test"), SUCCESS, System.out);
        }
        catch (IOException exception)
        {
            throw new RuntimeException("IOException: " + exception.getMessage(), exception);
        }
        catch (CommandLineException exception)
        {
            throw new RuntimeException("CommandLineException: " + exception.getMessage(), exception);
```

```
    }
}

@Theory
public void model_validation(@FromDataPoints("validation-modules") final String moduleName) throws Exception
{
    final String modulePath = getValidationModulePath(moduleName);

    cleanTargetDir(modulePath, SOME_TASK);
    compileWithTaskAndVerifyOutput(modulePath, SOME_TASK, FAILURE_MODEL_VALIDATION);
}

@Theory
public void error_loading_model(@FromDataPoints("failing-modules") final String moduleName) throws Exception
{
    final String modulePath = getErrorModulePath(moduleName);

    cleanTargetDir(modulePath, SOME_TASK);
    compileWithTaskAndVerifyOutput(modulePath, SOME_TASK, FAILURE_FAILED_LOADING_MODEL);
}

@Test
public void missing_source_file() throws Exception
{
    final String modulePath = getErrorModulePath("missing-source");

    cleanTargetDir(modulePath, POJ);
    compileAndVerifyOutput(modulePath, POJ, FAILURE_SOURCE_FILE_NOT_FOUND);
}

@Test
public void constructor_unknown() throws Exception
{
    final String modulePath = getErrorModulePath("constructor_unknown");

    cleanTargetDir(modulePath, "unknown_constructor_task");
    compileWithTaskAndVerifyOutput(modulePath, "unknown_constructor_task", FAILURE_CONSTRUCTOR_UNKNOWN);
}

@Test
public void task_undeclared() throws Exception
{
    final String modulePath = getErrorModulePath("task_undeclared");

    cleanTargetDir(modulePath, "some_task_name");
    compileAndVerifyOutput(modulePath, "some_task_name", FAILURE_TASK_UNDECLARED);
}

@Test
public void constructor_undefined() throws Exception
{
    final String modulePath = getErrorModulePath("constructor_undefined");

    cleanTargetDir(modulePath, "some_task");
    compileAndVerifyOutput(modulePath, "some_task", FAILURE_CONSTRUCTOR_UNDEFINED);
}

@Test
public void target_dir_created() throws Exception
{
    final String modulePath = SuccessCase.CASES_DIR + "/target_dirs/target_dir_created";
    final File targetDir = new File(getTargetDirPath(modulePath, SOME_TASK));

    forceDelete(targetDir);
    assertThat("Target dir must NOT exist: " + targetDir, targetDir.exists(), is(false));

    compileAndVerifyOutput(modulePath, SOME_TASK, SUCCESS);
    assertThat("Target dir must exist: " + targetDir, targetDir.exists(), is(true));
}

@Test
public void target_dir_cleaned() throws Exception
{
    final String modulePath = SuccessCase.CASES_DIR + "/target_dirs/target_dir_cleaned";
    final File targetDir = new File(getTargetDirPath(modulePath, SOME_TASK));

    final File bookFile = new File(targetDir, "src/main/java/books/Book.java");
    final File bookStoreFile = new File(targetDir, "src/main/java/books/BookStore.java");

    // Ensures there is already content in the target dir:
    final String tempModulePath = SuccessCase.CASES_DIR + "/target_dirs/target_dir_created";
    compileAndVerifyOutput(tempModulePath, SOME_TASK, SUCCESS);
    cleanTargetDir(modulePath, SOME_TASK);
    copyDirectoryStructure(
        new File(getTargetDirPath(tempModulePath, SOME_TASK)),
        new File(getTargetDirPath(modulePath, SOME_TASK)));
    assertThat("Book must exist: " + bookFile, bookFile.exists(), is(true));
    assertThat("BookStore must NOT exist: " + bookFile, bookStoreFile.exists(), is(false));

    // Verifies that the previously generated target has been cleaned before generating the new one:
    compileAndVerifyOutput(modulePath, SOME_TASK, SUCCESS);
    assertThat("Book must NOT exist: " + bookFile, bookFile.exists(), is(false));
    assertThat("BookStore must exist: " + bookFile, bookStoreFile.exists(), is(true));
}

private void compileAndVerifyOutput(
    final String modulePath,
    final String taskName,
    final int expectedExitCode) throws CommandLineException, IOException
{
    compileWithTaskAndVerifyOutput(modulePath, taskName, expectedExitCode);
}

private void compileWithTaskAndVerifyOutput(
    final String modulePath,
    final String taskName,
    final int expectedExitCode) throws CommandLineException, IOException
{
    compileAndVerifyOutput(
        modulePath,
        taskName,
        modulePath + File.separator + "compiler-output.txt",
        expectedExitCode);
}

private void compileAndVerifyOutput(
    final String modulePath,
    final String taskName,
    final String expectedOutputPath,
    final int expectedExitCode) throws CommandLineException, IOException
{
    final String actualCompilerOutput = executeJar(
        modulePath,
        COMPILER_JAR,
        singletonList(taskName),
        expectedExitCode);
```

```
    assertThatOutputMatches("compiler's output", expectedOutputPath, actualCompilerOutput);
}

private void cleanTargetDir(String currentDirPath, String taskName) throws IOException
{
    final String targetDirPath = getTargetDirPath(currentDirPath, taskName);

    System.out.println("\n----------------------------");
    System.out.println("Cleaning target dir: " + targetDirPath);

    forceMkdir(new File(targetDirPath));
    cleanDirectory(targetDirPath);
}

private String getTargetDirPath(String currentDirPath, String taskName)
{
    return currentDirPath + "/targets/" + taskName;
}

private void assertThatOutputMatches(
    final String reason,
    final String expectedOutputPath,
    final String actualOutput) throws IOException
{
    final String expectedOutput = Files.toString(new File(expectedOutputPath), OUTPUT_FILE_ENCODING);
    assertEquals(reason, expectedOutput, actualOutput);
}

private static void buildMavenModule(final String baseDir) throws MavenInvocationException
{
    System.out.println("Building: " + baseDir);

    System.setProperty("maven.home", System.getenv("M2_HOME"));

    final InvocationRequest request = new DefaultInvocationRequest();
    request.setBaseDirectory(new File(baseDir));
    request.setGoals(singletonList("install"));
    request.setInteractive(false);

    final Invoker invoker = new DefaultInvoker();
    final InvocationResult result = invoker.execute(request);

    if (result.getExitCode() != 0) throw new MavenInvocationException("Exit code: " + result.getExitCode());
}

private static String executeJar(
    final String currentDirPath,
    final String jarPath,
    final List<String> args,
    final int expectedExitCode) throws CommandLineException, IOException
{
    final ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

    try
    {
        executeJar(currentDirPath, jarPath, args, expectedExitCode, outputStream);
    }
    catch (AssertionError error)
    {
        System.out.println("Failure in output: \n" + new String(outputStream.toByteArray(), OUTPUT_FILE_ENCODING));

        throw error;
    }
    finally
    {
        outputStream.close();
    }

    return new String(outputStream.toByteArray(), OUTPUT_FILE_ENCODING);
}

private static void executeJar(
        final String currentDirPath,
        final String jarPath,
        final List<String> args,
        final int expectedExitCode,
        final OutputStream outputStream) throws CommandLineException, IOException
{
    final int actualExitCode = executeJar(currentDirPath, jarPath, args, outputStream);

    assertThat("exit code", actualExitCode, is(expectedExitCode));
}

private static int executeJar(
    final String currentDirPath,
    final String jarPath,
    final List<String> args,
    final OutputStream outputStream) throws CommandLineException, IOException
{
    final File jarFile = new File(jarPath);
    assertThat("Jar file must exit: " + jarFile, jarFile.exists(), is(true));

    final File javaBinDir = new File(System.getProperty("java.home"), "bin");
    assertThat("Java bin dir must exit: " + javaBinDir, javaBinDir.exists(), is(true));

    final File javaExecFile = new File(javaBinDir, "java");
    assertThat("Java exec file must exit: " + javaExecFile, javaExecFile.exists(), is(true));

    final Commandline commandLine = new Commandline();
    commandLine.addEnvironment("CML_MODULES_PATH", new File(CML_MODULES_BASE_DIR).getCanonicalPath());
    commandLine.setWorkingDirectory(currentDirPath);
    commandLine.setExecutable(javaExecFile.getAbsolutePath());

    commandLine.createArg().setValue("-jar");
    commandLine.createArg().setValue(jarFile.getAbsolutePath());

    for (final String arg : args) commandLine.createArg().setValue(arg);

    final Writer writer = new OutputStreamWriter(outputStream);
    final WriterStreamConsumer systemOut = new WriterStreamConsumer(writer);
    final WriterStreamConsumer systemErr = new WriterStreamConsumer(writer);

    System.out.println("Launching jar: " + commandLine);

    final int exitCode = executeCommandLine(commandLine, systemOut, systemErr, PROCESS_TIMEOUT_IN_SECONDS);

    System.out.println("Jar's exit code: " + exitCode);

    return exitCode;
}

private static String getErrorModulePath(String moduleName)
{
    return SuccessCase.CASES_DIR + File.separator + "errors" + File.separator + moduleName;
}

private static String getValidationModulePath(String moduleName)
```

```
    {
        return SuccessCase.CASES_DIR + File.separator + "validations" + File.separator + moduleName;
    }
}


==> cml-acceptance/src/test/java/cml/acceptance/SuccessCase.java

package cml.acceptance;

import org.jetbrains.annotations.Nullable;

import java.io.File;

import static java.lang.String.format;

class SuccessCase
{
    static final String CASES_DIR = "cases";

    private static final String CLIENT_PATH = "/%s-clients/%s";

    private static final String COMPILER_OUTPUT_FILENAME = "output-%s-compiler-%s.txt";
    private static final String CLIENT_OUTPUT_FILENAME = "output-%s-client-%s.txt";

    private final String moduleName;
    private final String clientName;
    private final String taskName;
    private final String targetLanguageExtension;
    private final @Nullable String pythonModuleName;

    SuccessCase(String moduleName, String clientName, String taskName, String targetLanguageExtension)
    {
        this(moduleName, clientName, taskName, targetLanguageExtension, moduleName.replace("-", "_"));
    }

    SuccessCase(String moduleName, String clientName, String taskName, String targetLanguageExtension, String pythonModuleName)
    {
        this.moduleName = moduleName;
        this.clientName = clientName;
        this.taskName = taskName;
        this.targetLanguageExtension = targetLanguageExtension;
        this.pythonModuleName = pythonModuleName;
    }

    String getModuleName()
    {
        return moduleName;
    }

    String getPythonModuleDir(String baseDir)
    {
        return baseDir + File.separator + pythonModuleName;
    }

    String getTaskName()
    {
        return taskName;
    }

    String getTargetLanguageExtension()
    {
        return targetLanguageExtension;
    }

    String getClientName()
    {
        return clientName;
    }

    String getClientPath()
    {
        return format(CLIENT_PATH, targetLanguageExtension, clientName);
    }

    String getModulePath()
    {
        return CASES_DIR + "/success/" + getModuleName();
    }

    String getTargetDirPath()
    {
        return getModulePath() + "/targets/" + getTaskName();
    }

    String getExpectedCompilerOutputPath()
    {
        return getModulePath() + File.separator + getExpectedCompilerOutputFilename();
    }

    private String getExpectedCompilerOutputFilename()
    {
        return format(COMPILER_OUTPUT_FILENAME, getTargetLanguageExtension(), getTaskName());
    }

    String getExpectedClientOutputPath()
    {
        return getModulePath() + File.separator + getExpectedClientOutputFilename();
    }

    private String getExpectedClientOutputFilename()
    {
        return format(CLIENT_OUTPUT_FILENAME, getTargetLanguageExtension(), getClientName());
    }

}
```