INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




dySMS.config Model Driven Development

EVARISTO MARTINS FIGUEIREDO
Outubro de 2019

POLITÉCNICO
DO PORTO

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA

isep

dySMS.config

MODEL DRIVEN DEVOLPEMENT

Instituto Superior de Engenharia do Porto

dySMS.config

Model driven software engineering

Evaristo Martins Figueiredo

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de software

Orientador: Dr. Paulo Alexandre Fangueiro Oliveira Maio

Coorientador: Dr. Nuno Alexandre Pinto da Silva

Porto, 10 2019

Resumo

A empresa DigitalWind, com intuito de fazer face às necessidades de reconversão do seu sistema legado de software - UEBE.Q - promoveu a criação de um projeto de I&D em parceria com o ISEP. Deste projeto, designado dySMS, resultou, entre outras coisas, a criação de um conjunto de componentes de software que permitem agilizar o referido processo de reconversão.

Associado aos referidos componentes de software existe um grande volume de dados de configuração que permitem a adaptação do funcionamento do software às problemáticas inerentes de um projeto desta natureza.

Uma vez que é necessária a manipulação frequente dos referidos dados de configuração, por utilizadores que não necessitam de possuir grandes conhecimentos técnicos e que a linguagem utilizada para os codificar é de baixo nível, esta tarefa revelou-se muito morosa e propensa a erros.

O presente trabalho, realizado no âmbito da Dissertação de Mestrado em Engenharia Informática - área de especialização de Engenharia de Software, do Instituto Superior de Engenharia do Porto (ISEP), documenta o processo de investigação e desenvolvimento, que suportou a criação do componente de configuração do referido sistema de software.

Para suportar a realização do referido projeto foi adotada uma abordagem orientada a modelos. Foi dada grande enfase à criação de linguagens de domínio específico, que promovem uma elevação do nível de abstração dos conceitos necessários à manipulação dos dados, permitido aos operadores realizar as tarefas de configuração recorrendo a uma linguagem muito mais próxima da linguagem do domínio de negócio.

No núcleo da aplicação encontra-se uma ontologia onde são capturados os vários conceitos de negócio e as suas relações, que permite definir uma estrutura conceptual base, numa linguagem amigável e de alto nível. Com base neste componente central, foi então desenvolvido um processo de mapeamento dos referidos conceitos e relações de alto nível com as estruturas de dados complexas do software legado. Uma vez conseguido este relacionamento bidirecional foram adicionadas várias funcionalidades para automatizar a configuração da interface de utilizador e da sua tradução em vários idiomas e das regras de autentificação e autorização de acesso a dados.

Palavras-Chave: DSL. MPS. MDE. MDD. Configurador. Sistemas Legados, Desenvolvimento orientado a modelos

Abstract

The company Digital Wind, to meet the needs of reconversion of its legacy software system-Uebe.Q-promoted the creation of an I&D project in partnership with ISEP. This project, called dySMS, resulted in, among other things, the creation of a set of software components that allow to expedite the process of reconversion.

Associated with these software components there is a large volume of configuration data that allow the adaptation of the software operation to the inherent problems of a project of this nature.

Since the frequent manipulation of the configuration data is necessary, by users who do not need to have great technical knowledge and that the language used for the code is low-level, this task has proved very lengthy and error-prone.

The present work carried out in the scope of the master's thesis in Informatics Engineering-area of specialization of Software Engineering, of the Instituto Superior de Engenharia do Porto (ISEP), documents the process of research and development, which supported the creation of the configuration component of the software system.

To support the realization of this project, a model-oriented approach was adopted. It was given great emphasis to the creation of specific domain languages, which promote an elevation of the level of abstraction of the concepts necessary to the manipulation of the data, allowing operators to perform the configuration tasks using a language very Closest to the language of the business domain.

At the core of the application is an ontology where the various business concepts and their relationships are captured, which allows defining a basic conceptual structure, in a friendly and high-level language. Based on this central component, a process was developed to map these concepts and high-level relationships with the complex data structures of legacy software. Once this bi-directional relationship has been achieved, several features have been added to automate the configuration of the user interface and its translation into multiple languages and the rules for authentications and data access authorization.

Key words: DSL. MPS. MDE. MDD. Configurator. Legacy systems, model-driven development

Índice

Resumo ii
Abstract iv
Índice vi
Lista de Figuras xi
Lista de Tabelas xiv
Acrónimos e Símbolos xvi
1 Introdução 1
1.1 Contexto 1
1.2 Problema 3
1.3 Objetivos 4
1.4 Abordagem 5
1.5 Estrutura do documento 7
2 Estado da arte 8
2.1 Gestão de ficheiros de configuração 8
2.1.1 Gestão de ficheiros de configuração 8
2.1.2 Gestão de configuração de software 8
2.1.3 Ferramentas de edição de formato especifico 9
2.1.4 Conclusão 10
2.2 Análise conceptual 10
2.2.2 Modelos 12
2.2.3 Software como produto 12
2.2.4 Abordagens orientadas a modelos 17
2.2.5 Domain-driven design 19
2.2.6 Model driven design 20
2.2.7 Domain Driven vs Model Driven 24
2.2.8 DSL 25
2.2.9 Language Workbench 25
2.2.10 Use-Case 2.0 26
2.2.11 Avaliação 27
2.3 Análise tecnológica 29
2.3.1 Intentional platform 29
2.3.2 MetaEdit+ 31
2.3.3 Eclipse Xtext 34
2.3.4 SDF 34

2.3.5 Monticore 35
2.3.6 Modeling SDK for Visual Studio 36
2.3.7 JetBrains MPS 39
3 Análise de valor 40
3.1 Partes interessadas 40
3.2 Desenvolvimento do conceito do produto 41
3.2.1 Identificação de oportunidades 41
3.2.2 Análise de oportunidade 42
3.2.3 Geração e enriquecimento da ideia 43
3.2.4 Seleção de ideias 43
3.2.5 Definição de conceito 44
3.3 Conceito de negócio e análise de valor 44
3.3.1 Análise SOWT 2
3.3.2 CANVAS 5
4 Requisitos 10
4.1 Perfis de utilizadores 10
4.2 Funcionalidades 11
4.2.1 Funcionalidades especificas de domínio 11
4.2.2 Funcionalidades não especificas de domínio 16
4.3 Requisitos não funcionais 16
4.3.1 Usabilidade 16
4.3.2 Interface 16
4.3.3 Implementação 18
5 Design 20
5.1 Visão Geral 20
5.2 DSL 21
5.2.1 Ontology 22
5.2.2 Persistence 24
5.2.3 Mappings 26
5.2.4 UI 28
5.2.5 Multilingual 30
5.2.6 Authorizations 31
5.3 Editor 32
5.3.1 Presentation 33
5.3.2 Application 34
5.3.3 Domain 34

5.3.4 Persistence 35
5.3.5 Repositories 35
6 Implementação 36
6.1 MPS 36
6.1.1 Estrutura de um projeto 37
6.1.2 Integração de bibliotecas externas 38
6.1.3 Compilação do projeto 41
6.1.4 Implantação 43
6.2 Design orientado a modelos 44
6.2.1 Componentes 45
6.2.2 *Layers* 46
6.3 DSL 47
6.3.1 Esquema 47
6.3.2 Restrições 48
6.3.3 Editor 49
6.3.4 Gerador 51
6.4 Editor 52
6.4.1 Criar projeto 52
6.4.2 Definir configuração 54
6.4.3 Importar ficheiros de configuração 55
6.4.4 Biblioteca system 56
7 Testes 61
7.1 Casos de uso 61
7.2 Testes unitários 62
7.2.1 Desenvolvimento 62
7.2.2 Testes 63
7.3 Testes funcionais 64
7.3.1 Execução automatizada 64
7.3.2 Testes 65
8 Avaliação 66
8.1 Estabelecimento de requisitos de avaliação 66
8.2 Especificação da avaliação 67
8.2.1 Selecionar métricas 68
8.2.2 Níveis de pontuação para as métricas 68
8.2.3 Critérios para julgamento 70
8.3 Projetar a avaliação 70

8.4 Execução da Avaliação 70
9 Conclusão 72
9.1 Resultados 72
9.1.1 Prática 72
9.1.2 Académicos 72
9.2 Considerações finais 72
9.2.1 Model driven design 73
9.2.2 Qualidade 73
9.2.3 JetBrains MPS 73
9.3 Trabalho futuro 74
9.3.1 Melhoria 1 – Estrutura dos tipos coleções 75
9.3.2 Melhoria 2 – Acrescentar cardinalidade às propriedades dos conceitos 76
9.3.3 Melhoria 3 – Importação das relações entre tabelas da BD relacional 76
9.3.4 Melhoria 4 – Criar conceitos a partir da informação de uma tabela 76
9.3.5 Domínio aplicacional 1 – Suporte para novos tipos de fontes de dados 76
9.3.6 Melhoria 5 – Acrescentar novos tipos de drivers para BD relacionais 77
9.3.7 Melhoria 6 – Alargar o subconjunto de especificações dos ficheiros hibernate suportadas pela aplicação 77
9.3.8 Melhoria 7 – Melhorar o mapeando de tipos 77
9.3.9 Domínio aplicacional 2 – Acrescentar perfil para os distribuidores 77
9.3.10 Domínio aplicacional 3 – Novos ambientes de desenvolvimento para os UI 78
Referências 79
A. Anexos 82
A.1 Catalogo de elementos arquiteturais 82
A.2 Documento de definição de requisitos 83
A.3 Exemplo de ficheiro hibernate 95

Lista de Figuras

Figura 1 - Diagrama de contexto 2
Figura 2 - Editor de ficheiros hibernate 9
Figura 3 - Editor de ficheiros Drools 9
Figura 4 - Software como produto, plataformas e transformações 13
Figura 5 - Abordagens orientadas a modelos 17
Figura 6 - DDD vs. MDD 20
Figura 7 - Blocos fundamentais do design orientado a modelos 21
Figura 8 - Arquitetura em camadas 22
Figura 9 - Dados na camada de apresentação 22
Figura 10 – Áreas da qualidade de software 28
Figura 11 - Processo de avaliação 29
Figura 12 - Diagrama da plataforma Itentional 30
Figura 13 - Exemplo de uma DSL na plataforma Intentional 30
Figura 14 - Passo 1, definição do meta-modelo 31
Figura 15 - Passo 2, desenho da notação 32
Figura 16 - Passo 3, Criar geradores 32
Figura 17 - Editor de diagramas 33
Figura 18 - Editor de matrizes 33
Figura 19 – Browser 33
Figura 20 - Exemplo de uma DSL no Visual Studio 37
Figura 21 - Exemplo de um modelo de DSL no Visual Studio 38
Figura 22 - Exemplo de instância de uma DSL 38
Figura 23 - AST 39
Figura 24 - Análise SWOT 2
Figura 25 - Canvas 5
Figura 26 - Diagrama de casos de uso 11
Figura 27 - Diagrama de componentes - Visão geral 21
Figura 28 - Diagrama de componentes – DSL 21
Figura 29 - Diagrama de classes dos Tipos 23
Figura 30 - Diagrama de classes dos Conceitos 24
Figura 31 - Diagrama de classes da persistência 25
Figura 32 - Diagrama de classes dos mapeamentos 26
Figura 33 - Diagrama de classes do UI 29
Figura 34 - Diagrama de classes do componente multilingue 30
Figura 35 - Diagrama de classes do componente authorizations 32
Figura 36 - Diagrama de componentes – Editor 33
Figura 37 - Estrutura de um projeto MPS 37
Figura 38 - Integração de classes JAVA, passo 1 39
Figura 39 - Integração de classes JAVA, passo 2 40
Figura 40 - Usar classes JAVA integradas 41
Figura 41 - Projeto de compilação 41
Figura 42 - Cabeçalho do script de build 42
Figura 43 - Estrutura do projeto no script de build 42
Figura 44 - Livrarias JAVA eternas no script de compilação 43
Figura 45 - Diagrama de implantação 44
Figura 46 - Diagrama de componentes 45

Figura 47 - Arquitetura em camadas 47
Figura 48 – Esquema de uma DSL 48
Figura 49 - Restrições de uma DSL 49
Figura 50 – Diagrama de sequência para o editor 50
Figura 51 - Editor de uma DSL 50
Figura 52 - Diagrama de sequência para o gerador 52
Figura 53 – Diagrama de sequência da criação de um projeto 53
Figura 54 - Projeto em tempo de execução 53
Figura 55 - Diagrama de pacotes do projeto em tempo de execução 54
Figura 56 – Diagrama de sequencia para definição da configuração de um projeto 55
Figura 57 – Diagrama de sequência para a importação de ficheiros 56
Figura 58 –Fluxo de execução base do sistema 61
Figura 59 - Processo de desenvolvimento 62
Figura 60 – Processo automatizado de execução 65
Figura 61 - Questionário de avaliação de qualidade 68
Figura 62 - Escala das características 69
Figura 63 - Tipos de dados coleção 75

Lista de Tabelas

Tabela 1 - Partes interessadas Digital Wind 40
Tabela 2 - Partes interessadas ISEP 41
Tabela 3 - Perfis de operadores 10
Tabela 4 - RF1 – Criar a ontologia 12
Tabela 5 - RF2 Criar ficheiro de configuração dos mapeamentos 12
Tabela 6 - RF3 - Criar ficheiro de configuração da UI 13
Tabela 7 - RF4 - Criar ficheiro de configuração multilingue 15
Tabela 8 - RF5 Criar ficheiro de configuração da lógica de negocio 15
Tabela 9 - Estrutura configuração UI - Coleção de instâncias 18
Tabela 10 - Estrutura de configuração UI - Instância única 18
Tabela 11 - Divisão de requisitos em "slices" para o design 20
Tabela 12 - *Slices* adotados na fase de implementação 36
Tabela 13 - Modelo de qualidade adotado 67
Tabela 14 -Ponderação das questões 69
Tabela 15 - Planeamento da avaliação 70
Tabela 16 - Resultados da avaliação 71
Tabela 17 – Estereótipos personalizados 82

Acrónimos e Símbolos

AST Abstract syntax tree

BD Base de dados

CRUD Create, Read, Update, Delete

DDD Domain-Driven Design

MDD Model-Driven Design or Development

DSL Domain-specific language

GPL General-purpose programming language

IDE Integrated development environment

M2M Model to model transformation

M2T Model to text transformation

MDE Model-driven engineering

NCD New Concept Development

O/R Object/Relational

OO Object oriented

OMG Object Management Group

ORM Object-relational mapping

RAD Rapid Application Development

REST Representational State Transfer

SAAS Software as a service

SDK Software development kit

UC Use case

UI User Interface

UML Unified Modeling Language

VSIX Visual Studio Integration Extension

1 Introdução

Nesta seção faz-se uma apresentação preliminar da dissertação. Começa-se com a exposição do ambiente situacional no qual é produzido este documento, com o intuito de permitir o seu correto enquadramento. De seguida, apresenta-se a uma descrição dos problemas basilares, que se encontram na génese do desenvolvimento deste projeto.

Uma vez exposto o contexto e avaliadas as motivações, são então enunciados os objetivos, a partir dos quais é possível orientar a elaboração do projeto e permitir a posterior aferição do grau de sucesso das várias fases do projeto, comparativamente com as aspirações iniciais subjacentes à realização do mesmo.

Posteriormente, explana-se a abordagem adotada no desenvolvimento do projeto, com o propósito de esclarecer as decisões subjacentes à escolha das técnicas adotadas, em detrimento de outras possíveis.

Por fim, apresenta-se uma visão geral da totalidade da dissertação.

1.1 Contexto

O projeto atual, dySMS.config, que se enquadra no âmbito da dissertação para a obtenção do Mestrado em Engenharia Informática (MEI), especialização na área de Engenharia de Software, do Instituto Superior de Engenharia do Porto (ISEP), é parte integrante de um projeto em copromoção no âmbito do programa nacional P2020, designado dySMS. Este projeto de investigação e desenvolvimento foi proposto pela empresa Digital Wind em parceria com o ISEP e tem por objetivo promover a reconversão de um sistema legado de software designado UEBE.Q, que é propriedade da referida empresa com o intuito deste satisfazer mais e melhor os novos requisitos e desafios de negócio com que a Digital Wind se depara.

No âmbito do referido projeto foram desenvolvidos vários componentes de software, que formam a nova versão do sistema UEBE.Q onde o projeto atual dySMS.config está integrado.

A Figura 1 apresenta uma visão geral do sistema através de um diagrama de componentes e, simultaneamente representativo da integração do mesmo com o projeto descrito neste documento.

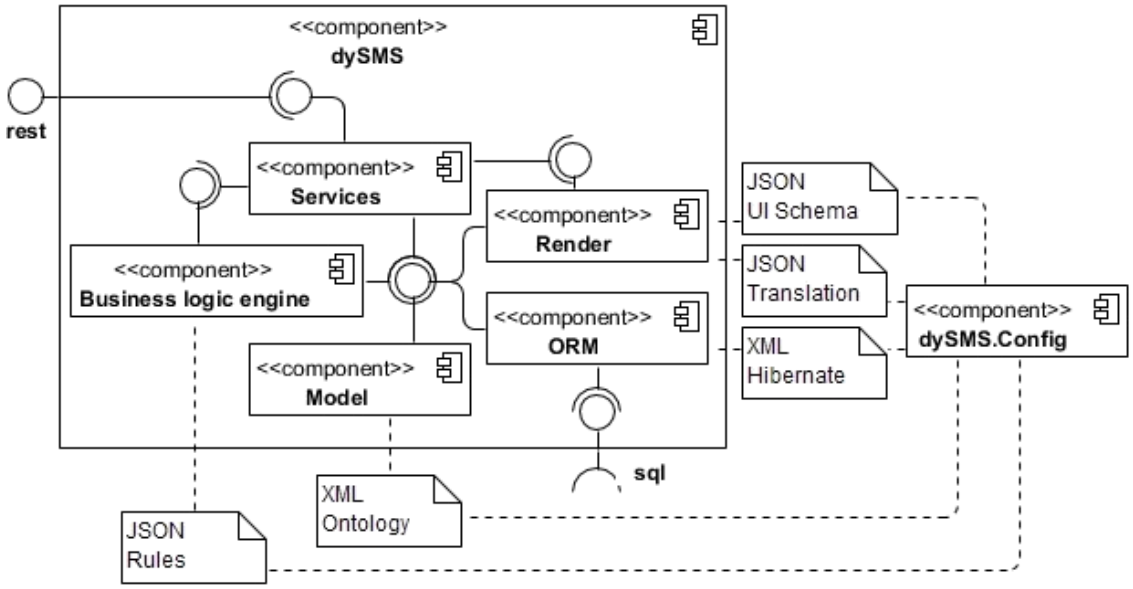

Figura 1 - Diagrama de contexto

A dissertação atual versa exclusivamente sobre projeto dySMS.config. Porém, com o dySMS.config tem origem no contexto do projeto integrador – dySMS, é apresentada uma breve descrição dos componentes deste último.

1) Model: a aplicação UEBE.Q tem como objetivo a gestão de processos de certificação de qualidade. Este domínio requer uma grande capacidade de adaptação da aplicação às especificidades e terminologia subjacentes à área de negócio de cada cliente, pois a gestão da qualidade pode ser aplicada às mais diversas áreas. Este facto obriga a que aplicação, ao contrário da grande parte dos sistemas de gestão, não tenha a terminologia e as regras de negócio coligidas de forma rígida no seu código. A aplicação tem que dispor de inúmeras funcionalidades de configuração para permitir alterar os conceitos e os fluxos dos processos de negócio, de acordo com a realidade concreta de cada cliente. A forma encontrada para ultrapassar este problema foi a criação de uma ontologia onde é possível capturar os conceitos de negócio, os respetivos atributos, as relações existentes entre conceitos e as ações a eles associadas. Desta forma é possível garantir o ajustamento da terminologia utilizada na aplicação à realidade de cada cliente do UEBE.Q:

2) ORM: este componente é responsável pelo acesso aos dados, armazenados de forma persistente numa base de dados relacional. Utiliza a técnica de mapeamento objeto-relacional (ORM) para providenciar, dinamicamente, instâncias dos conceitos da ontologia. Importa salientar que a estrutura da base de dados pode ser significativamente diferente da ontologia em uso.

3) Busines logic engine: este componente permite alargar as potencialidades de personalização da aplicação para que estas abranjam os processos e regras de negócio, nomeadamente no que diz respeito à manipulação de dados associadas a cada um dos perfis de operadores. O funcionamento deste componente é suportado pela ferramenta DROOLS que é um sistema de gestão de regras de negócio, com um mecanismo de regras baseadas em inferência de encadeamento direto e inverso. Este sistema permite a avaliação rápida e confiável de regras de e o processamento de eventos.

4) Services: neste componente encontra-se a implementação da camada de serviços e algumas funcionalidades responsáveis pela interação com a camada de negócio do sistema legado (não representado na figura). Estas últimas facilitam a transição do sistema legado para o novo. Esta implementação é feita com base no estilo de arquitetura Representational State Transfer (REST). Os web services compatíveis com REST permitem que os sistemas clientes acedam e manipulem representações textuais de recursos da Web, fazendo uso um conjunto uniforme e predefinido de operações sem estado (Consortium, 2004).

5) Render: neste componente encontra-se a implementação da camada de apresentação, que no caso concreto do dySMS é uma single-page application (SPA) altamente configurável de forma adaptar-se aos dados recebido de acordo com a ontologia, que contém informação sobre a estrutura das interfaces de utilizador (UI).

Antes de continuar, importa realçar que a Figura 1 apresenta uma visão lógica do sistema e não corresponde exatamente à implementação concreta. Na realidade a informação sobre a ontologia e sobre os mapeamentos hibernate, é obtida pelo sistema dySMS a partir de apenas um único ficheiro de configuração. Isto é relevante na medida em que nas fases seguintes do projeto não existe referências nem ao componente, nem ficheiro – Ontology e este facto poderia ser interpretado como um erro.

Uma vez explicados os componentes do sistema integrador, passa-se agora a abordar o sistema dySMS.config e a sua relação com o sistema onde este se integra.

O componente dySMS.config agrega todas as responsabilidades inerentes à gestão da informação de configuração e interage com o sistema integrador de forma passiva. As alterações dos ficheiros de configuração que resultam da operação do componente dySMS.config, são consumidos pelo sistema integrador durante o seu arranque e determinam a sua forma de funcionamento.

1.2 Problema

O principal problema que motivou o desenvolvimento do projeto dySMS.config, foi a complexidade inerente às linguagens utilizadas nos vários ficheiros de configuração de cada um dos componentes que constituem o sistema dySMS.

As linguagens em que os referidos ficheiros estão escritos, são de caracter muito técnico e direcionadas para processamento automatizado por software. Inerentemente este tipo de linguagens são de baixo nível e de difícil compreensão por humanos. Além disto também existe redundância de informação entre os vários ficheiros de configuração e que facilmente conduz à existência de incoerências/inconsistências entre os mesmos.

Como existe a necessidade de manipulação frequente destes ficheiros por utilizadores sem grandes conhecimentos técnicos, constata-se facilmente que esta tarefa é muito morosa e propensa a erros. Sendo assim tornou-se óbvio que era necessário complementar o projeto dySMS com um novo componente (designado dySMS.config), que consiga oferecer aos referidos utilizadores um ambiente onde possam realizar a manipulação da grande quantidade de dados de configuração de uma forma simples, correta e segura.

1.3 Objetivos

O projeto dySMS.config tem como propósito o desenvolvimento de uma aplicação, que suporte o processo de recolha e manipulação da informação necessária para a correta configuração dos vários componentes que suportam o processo de reconversão do sistema legado de software – UEBE.Q.

A recolha desta informação deve ser feita uma única vez de forma centralizada, com o intuito de que a mesma possa ser utilizada pelos vários componentes minimizando as configurações necessárias e evitando redundâncias.

A manipulação da informação de configuração deve ser suportada por mecanismos que auxiliem o utilizador através do uso de sugestões automáticas e da validação do conteúdo.

Sempre que seja possível, o sistema deve fornecer ao utilizador funções de automatização das ações a serem realizadas.

A implementação do sistema deve assegurar a interoperabilidade com vários sistemas operativos (OS). Como se pretende estender o uso do projeto dySMS.config aos revendedores, não é possível prever os OS em utilização em cada um deles.

No seguimento da postulação prévia dos objetivos gerais almejados para o projeto atual, e com o intuito de promover a sua clarificação e de aumentar a sua objetividade, são de seguida enumerados os objetivos específicos:

6) Permitir a especificação de um ontologia que capture os conceitos de negócio existentes no dySMS, bem como os atributos dos respetivos conceitos e relações existente entre conceitos e o seu ajustamento (e.g. terminologia adotada) à realidade de cada cliente do UEBE.Q;

7) Capturar as ações de negócio que ocorrem sobre cada conceito da ontologia;

8) Permitir o mapeamento dos elementos que compõem a ontologia (modelo OO) para uma estrutura de dados relacional já existente e consequentemente heterogénea. Este mapeamento permitirá instanciar (i.e. popular com instâncias) os conceitos da ontologia.

9) Especificar para cada par conceito ontológico/ação de negócio uma (ou mais) estruturas de visualização adequadas.

10) Deve ser possível definir as regras de autorização e acesso para cada uma das ações associadas aos conceitos. A partir desta informação o sistema deve ser capaz de exportar as regras definidas no formato utilizado pelo sistema de gestão de regras de negócio – DROOLS.

11) Deve ser possível agregar aos conceitos existentes na ontologia informação que permita definir as traduções dos vários termos utilizados na interface de utilizador para os vários idiomas que o sistema deve contemplar.

Dada a forte componente académica associada ao projeto atual, além dos objetivos puramente funcionais, importa ainda salientar que o desenvolvimento deste projeto pretende servir como caso de estudo, que permita ajudar a aferir acerca da adequabilidade das metodologias orientadas a modelos, especificamente o uso de linguagens especificas de domínio (DSL), em cenários que envolvem o processamento de informação de configuração.

1.4 Abordagem

A abordagem adotada para o desenvolvimento do projeto dySMS.config, foi uma abordagem baseada na utilização de linguagens de domínio específico – DSL.

De acordo com (Tolvanen, 2007), as DSL são "linguagens de programação que elevam o nível de abstração além da programação, permitindo a especificação da solução através do uso direto de conceitos e regras de um domínio de problema específico".

Uma linguagem de domínio específico (DSL) pode oferecer várias vantagens importantes sobre uma linguagem de propósito geral (GPL) (Czarnecki, 2004):

12) Abstrações específicas do domínio.

Uma DSL fornece abstrações predefinidas para representar diretamente os conceitos do domínio de negócio. Consequentemente, os peritos do domínio podem eles mesmos compreender, validar, modificar, e frequentemente mesmo desenvolver DSL (Deursen, et al., 2000).

13) Sintaxe concreta específica de domínio.

Uma DSL oferece uma notação natural para um determinado domínio e evita a desordem sintática que muitas vezes resulta ao usar uma GPL.

14) Verificação de erros específica do domínio.

Uma DSL permite a criação de analisadores estáticos que podem encontrar mais erros do que os analisadores similares para uma GPL e que podem relatar os erros numa linguagem familiar ao especialista em domínio.

15) Otimizações específicas de domínio.

Um DSL cria oportunidades para gerar uma base de código otimizada em conhecimento específico do domínio, que normalmente não está disponível para um compilador para uma GPL.

16) Suporte a ferramentas específicas de domínio.

Uma DSL cria oportunidades para melhorar qualquer aspeto operacional de um ambiente de desenvolvimento, incluindo, editores, depuradores, controle de versão, etc.

O conhecimento específico de domínio que é explicitamente capturado por uma DSL pode ser usado para fornecer suporte a ferramentas mais inteligentes para os programadores.

Deursen et al. (Deursen, et al., 2000) acrescentam que as DSL encarnam o conhecimento do domínio, e assim permitem a conservação e reutilização deste conhecimento.

Eric Evans (Evans, 2003) também enfatiza que para abordar a complexidade do desenvolvimento de software é necessária uma linguagem compreensível por programadores e especialistas em domínio.

DSL e o desenvolvimento orientado a modelos - MDD (Kent, 2002) são complementares e intrinsecamente dependentes, o que se torna uma vantagem adicional, pois às potencialidades das DSL é possível adicionar o suporte proporcionado por uma metodologia de desenvolvimento de software, cujo foco se mantém no domínio do negócio e na modelação dos conceitos do mesmo (como as DSL).

A adoção desta metodologia acrescenta recursos que promovem a sistematização na abordagem dos aspetos arquitetónicos e na abordagem do processo de desenvolvimento de software com base em princípios de engenharia.

MDD é necessário para definir o tipo de modelos necessários para descrever um software e para definir um processo de engenharia e manutenção, ou seja, como construir e manter o software usando modelos. DSL são necessárias para definir as linguagens para expressar os modelos. Com o uso das dimensões do modelo MDD (Kent, 2002), o âmbito das DSL pode ser definido de forma mais precisa. Isso é muito importante porque uma das maiores armadilhas

do design de DSL é que o seu âmbito cresce e cresce, devido à mudança de requisitos, tornando a DSL original num tipo de linguagem de propósito geral (GPL).

A MDD é frequentemente associada a modelos gráficos devido ao foco do Object Management Group (OMG) no seu padrão Unified Modeling Language (UML). No entanto os princípios da MDE defendem que a linguagem usada para expressar um modelo deve ser o mais adequada possível. Não só ao nível de semântica, mas também ao nível da sintaxe o que torna o uso de DSL textuais uma parte integrante da MDE.

Tendo em conta que o problema que se encontra na origem do desenvolvimento do projeto atual, é o facto de ser necessário manipular informação que se encontram numa linguagem de baixo nível de uma forma amigável e com termos mais próximos da linguagem do domínio do negócio, é possível concluir que o uso de DSL e uma abordagem pelo menos teoricamente viável e foi explorada neste trabalho.

1.5 Estrutura do documento

A estrutura deste trabalho académico visa garantir uma organização lógica e formal adequada à documentação do trabalho, de pesquisa e desenvolvimento, realizado para obter o grau de mestre, no Instituto Superior de Engenharia do Porto (ISEP). Em conformidade com o framework adotado pelo Departamento de Engenharia Informática (DEI) (Pereira & Baltarejo, 2016), esta dissertação, do ponto de vista lógico, encontra-se dividida, em três grandes partes: Introdução, Corpo da Dissertação e Conclusão, que por sua vez estão formalmente organizadas sobre a forma de capítulos, secções e subsecções (Pereira & Baltarejo, 2016).

A introdução corresponde ao primeiro capítulo, onde se apresenta o projeto e a dissertação. A explicação introdutória do projeto abrange as primeiras quatro secções: Contexto, Problema, Objetivos e Abordagem. A secção atual encerra o capítulo com a descrição da organização adotada na estruturação deste documento.

O Corpo da Tese inclui o levantamento do estado da arte (capítulo 2), análise de valor (capítulo 3) e a descrição, nos sucessivos capítulos (4 a 7), de todos os pontos importantes do trabalho realizado e respetivos resultados. Esta descrição é estruturada com base nas atividades padrão do ciclo de vida do processo de desenvolvimento de software: Requisitos, Design, Implementação e Testes.

A parte das conclusões é composta pelos dois últimos capítulos: Avaliação e Conclusão. Foi adotada a norma ISO/IEC 14598, que define o processo de avaliação de produtos de software, para consubstanciar o balanço final do trabalho que é apresentado no último capítulo, estruturado em três secções: Resultados, Trabalho futuro e Considerações finais.

2 Estado da arte

Este capítulo descreve o estudo realizado sobre as temáticas: Gestão de ficheiros configuração e DSL/MDD, subjacentes ao problema a ser resolvido e á abordagem adotada. É composto por três secções: Gestão de ficheiros de configuração, Análise conceptual e Análise tecnológica.

A informação recolhida faz referência a conhecimento já existente, permitindo enquadrar o resultado da investigação atual e relaciona-la no contexto da sua temática. Além disso, é obviamente importante para o desenvolvimento do novo produto.

Na primeira secção são apresentados os resultados da investigação realizada sobre estudos existentes e ferramentas relevantes, na área da gestão de ficheiros de configuração.

Tendo em conta a adoção de uma abordagem baseada em DSL, enquadradas na metodologia orientada a modelos, as duas secções seguintes apresentam uma análise conceptual a este tipo de abordagem, e uma análise tecnológica das ferramentas disponíveis.

2.1 Gestão de ficheiros de configuração

No que respeita a esta temática, não foi possível encontrar estudos relevantes e apenas foram encontradas ferramentas provenientes de outros domínios, que oferecem algumas funcionalidades nesta área, mas nenhuma com características similares ao produto a ser criado.

Estes resultados permitem atestar a originalidade e a relevância do trabalho realizado e são apresentados nas subsecções seguintes estruturados de acordo com os domínios onde se inserem.

2.1.1 Gestão de ficheiros de configuração

Estas são ferramentas especificas para realizar a gestão de ficheiros de configuração, mas fazem-no numa perspetiva de DevOps ou de gestão de redes.

Neste domínio a lista é bastante reduzida, apenas conseguimos encontrar duas: ConfigApp (ConfigApp, 2019) e MenageEngine (Zoho, 2019).

2.1.2 Gestão de configuração de software

Neste domínio, existem inúmeras soluções disponíveis, das quais destacamos as seguintes: CFEngine (Northern.tech, 2019), puppet (puppet, 2019), chef (Chef, 2019) e ansible (HAT, 2019).

Cada uma possui recursos específicos que a tornam melhor para algumas situações do que para outras. No entanto, as quatro referidas têm várias funcionalidades em comum, que fazem delas a referência na área de DevOps: todas possuem uma licença de código-fonte aberto, usam arquivos de definição de configuração externos, são executados sem supervisão e são programáveis[1].

2.1.3 Ferramentas de edição de formato especifico

Estas ferramentas permitem editar ficheiros com os formatos específicos utilizados no projeto.

2.1.3.1 Hibernate (RedHat, 2019)

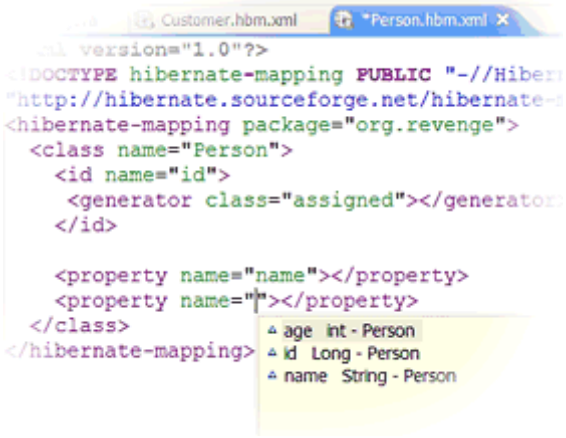

Figura 2 - Editor de ficheiros hibernate

2.1.3.2 Drools (RedHat, 2019)

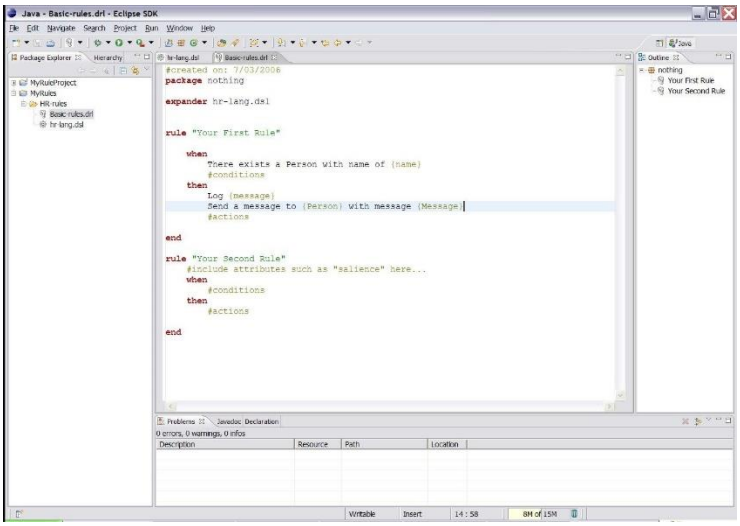

Figura 3 - Editor de ficheiros Drools

[1] Informações dos repositórios de software e sites das ferramentas

2.1.4 Conclusão

No que diz respeito à gestão de ficheiros de configuração, as ferramentas analisadas, permitem gerir aspetos como o versionamento ou a automatização e gestão da implantação, mas nenhuma delas oferece funcionalidades de edição gráfica assistida, transformação, validação e elevação do nível da linguagem, que são fundamentais no projeto atual.

No caso dos editores, as funcionalidades são iguais ás dos editores de linguagens de domínio geral, ficando muito aquém do poder oferecido pelo uso de editores projecionais adaptados a cada DSL.

2.2 Análise conceptual

Esta secção reveste-se de uma importância capital para o desenvolvimento do projeto atual, pois nela é feito um resumo das bases teóricas, que suportam a abordagem adotada na implementação do mesmo.

2.2.1.1 Os vários conceitos e teorias aqui apresentados complementam a introdução feita na seção 1.4, permitem perceber o enquadramento adotado na escolha e análise das várias ferramentas abrangidas na secção 2.2.11.2 Processo de avaliação

Segundo a análise da qualidade de software pode ser dividida em duas grandes áreas de conhecimento: a qualidades do processo de software e a qualidade do produto de software. Na Figura 10 é apresentado um diagrama que representa a divisão da análise de qualidade de software em áreas de conhecimento e respetivas metodologias.

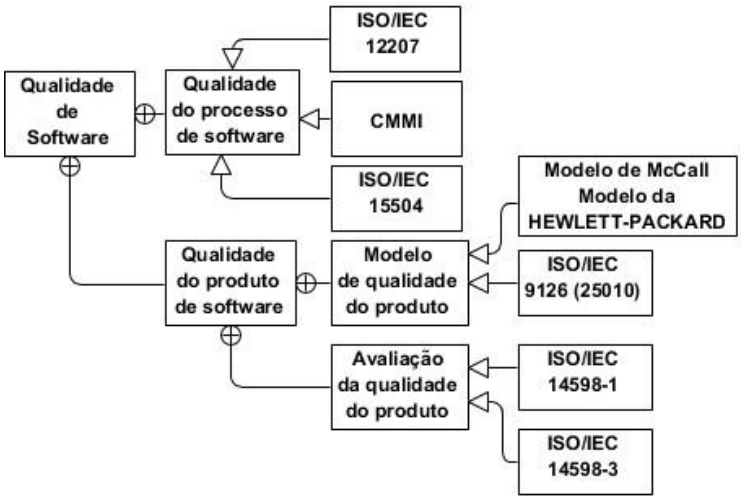

Figura 10 – Áreas da qualidade de software

A primeira área tem a ver com qualidade das atividades/tarefas que compõem os processos de ciclo de vida de software (requisitos, design, construção, testes, manutenção, etc.). Nesta área o objetivo é avaliar maturidade organizacional ou a capacidade das áreas da empresa relacionadas com o processo de desenvolvimento e estabelecer prioridades para melhoria .

A área de qualidade do produto de software, por sua vez, pode ser dividida em duas subáreas: modelo de qualidade do produto e avaliação da qualidade do produto. A primeira subárea tem como objetivo a definição dos atributos de qualidade e dos critérios e métricas para produtos de software, ao passo que a segunda visa a definição de um processo de avaliação de produtos de software.

Os modelos McCall e HEWLETT-PACKARD, são modelos antigos e que foram substituídos pela norma ISO/IEC 9126, que oferece uma abordagem mais abrangente e padronizada a nível mundial. A referência a estes modelos apenas foi incluída por motivos históricos e académicos.

A norma ISO/IEC 9126, por sua vez, foi substituída pela norma ISO/IEC 25010 em 2011, mas para o trabalho atual as alterações introduzidas não se revestem de importância significativa. Devido a este facto são mantidas referências à norma original, por esta ser mais conhecida.

A norma ISO/IEC 14598 está subdividida em 6 partes: (14598-1) visão geral; (14598-2) planeamento e gestão; (14598-3) processo para programadores; (14598-4) processo para compradores; (14598-5) processo para avaliadores; (14598-6) documentação de módulos de avaliação. Na Figura 10, não são referidas todas as partes por razões de simplificação.

Na Figura 11 encontra-se representado o processo de avaliação de produtos de software, preconizado pela norma 14598-1. Este processo é composto por quatro fases: i) Estabelecimento de requisitos de avaliação, ii) Especificação da avaliação, iii) Projeto da avaliação e iv) Execução da avaliação.

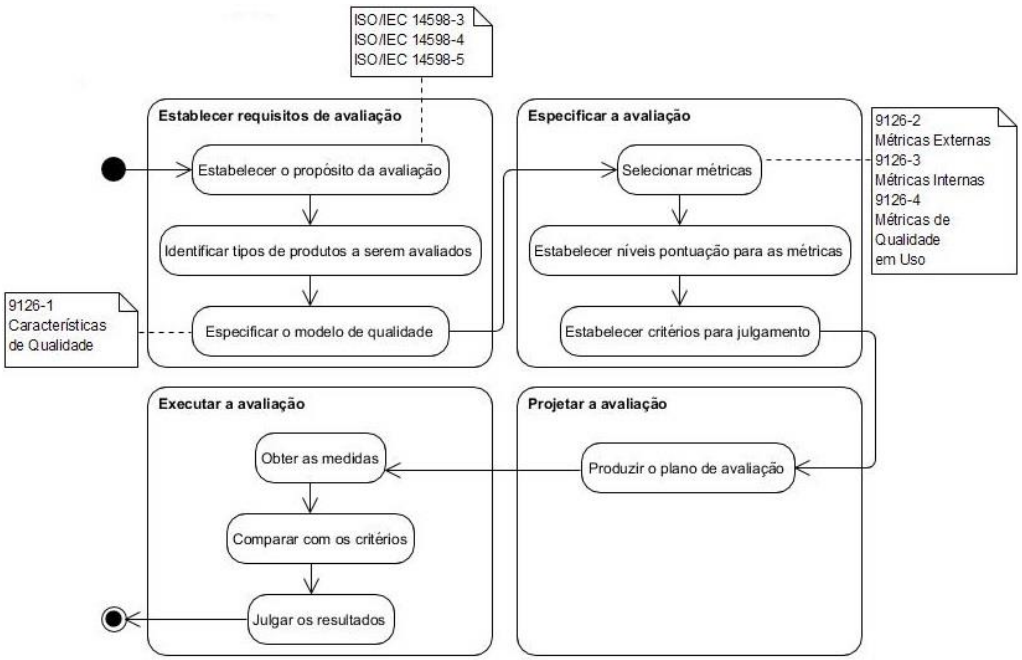

Figura 11 - Processo de avaliação

As caixas de comentário, que aparecem na Figura 11 ao lado de cada uma das fases, identificam os padrões ISSO a serem seguidos em cada uma das fases.

Análise tecnológica, servem de fundamento para explicar várias decisões arquiteturais apresentadas no capítulo 5. Design e funcionam como uma introdução teórica fundamental para uma compreensão mais aprofundada do capítulo 6. Implementação.

2.2.2 Modelos

Um modelo é uma abstração de um sistema frequentemente usada para substituir o sistema em estudo (Ludewig, 2003). Em geral, um modelo representa uma visão parcial e simplificada de um sistema, portanto, a criação de vários modelos é geralmente necessária para melhor representar e compreender a totalidade do sistema em estudo.

A modelação é uma técnica bem conhecida, adotada pelos vários campos de engenharia assim como por outras áreas nomeadamente: física, matemática, biologia, economia, política e filosofia (Favre, 2005).

No entanto, no âmbito deste documento, o termo modelo é usado no contexto da engenharia de software e MDD (2.2.3 Software como produto), o que significa que os modelos no contexto do projeto atual têm uma natureza baseada em linguagem (2.2.8 DSL), e pretendem descrever um sistema em oposição, por exemplo, a modelos em Matemática que são entendidos como a interpretação de uma teoria (Chang & Keisler, 1990).

Os modelos permitem partilhar uma visão e um conhecimento comuns entre os intervenientes técnicos e não técnicos, facilitando e promovendo a comunicação entre eles. Além disso, os modelos tornam o planeamento do projeto mais eficaz e eficiente proporcionando uma visão mais adequada do sistema a ser desenvolvido, permitindo que o controlo do projeto seja alcançado de acordo com critérios objetivos (Booch, 1994).

2.2.3 Software como produto

A abordagem MDD defende que o uso de linguagens de modelação ajuda a especificar modelos num determinado nível de abstração, e também que esses modelos são usados para apoiar o desenvolvimento de aplicações de software (Atkinson & Kuhne, 2003).

Nesta abordagem a aplicação de software (ou produto de software) é encarada como um sistema composto a partir da integração não trivial de plataformas de software, artefactos gerados por meio de transformações de modelo para texto, artefactos diretamente escritos por programadores e, eventualmente, modelos diretamente executáveis no contexto de uma plataforma de software específica. A Figura 4 (Silva, 2015) mostra como, nesta abordagem, os componentes para desenvolvimento de um produto de software se relacionam entre si.

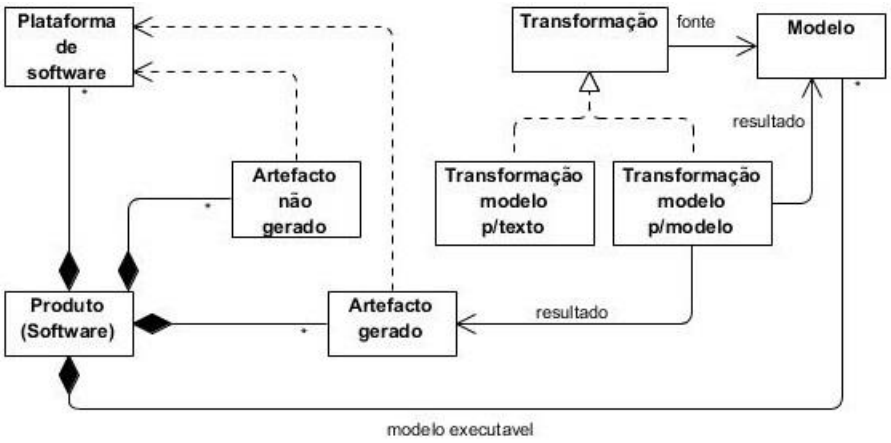

Figura 4 - Software como produto, plataformas e transformações

2.2.3.1 Plataformas de software

As plataformas de software representam um conjunto integrado de elementos computacionais que possibilitam o desenvolvimento e a execução de uma classe de produtos de software (Brambilla, et al., 2017).

2.2.3.2 Importa destacar a importância dada às plataformas de software e o facto de ser proposta a adoção de uma ou mais plataformas de suporte ao desenvolvimento do software, proposta esta que foi adotada no desenvolvimento do projeto atual. Exemplos de plataformas (2.2.9 *Language Workbench*) podem ser encontrados na secção 27 Processo de avaliação

Segundo a análise da qualidade de software pode ser dividida em duas grandes áreas de conhecimento: a qualidades do processo de software e a qualidade do produto de software. Na Figura 10 é apresentado um diagrama que representa a divisão da análise de qualidade de software em áreas de conhecimento e respetivas metodologias.

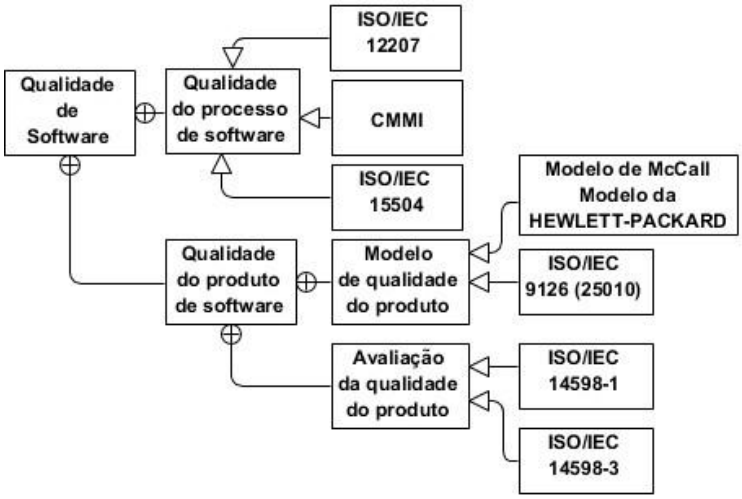

Figura 10 – Áreas da qualidade de software

A primeira área tem a ver com qualidade das atividades/tarefas que compõem os processos de ciclo de vida de software (requisitos, design, construção, testes, manutenção, etc.). Nesta área o objetivo é avaliar maturidade organizacional ou a capacidade das áreas da empresa relacionadas com o processo de desenvolvimento e estabelecer prioridades para melhoria .

A área de qualidade do produto de software, por sua vez, pode ser dividida em duas subáreas: modelo de qualidade do produto e avaliação da qualidade do produto. A primeira subárea tem como objetivo a definição dos atributos de qualidade e dos critérios e métricas para produtos de software, ao passo que a segunda visa a definição de um processo de avaliação de produtos de software.

Os modelos McCall e HEWLETT-PACKARD, são modelos antigos e que foram substituídos pela norma ISO/IEC 9126, que oferece uma abordagem mais abrangente e padronizada a nível mundial. A referência a estes modelos apenas foi incluída por motivos históricos e académicos.

A norma ISO/IEC 9126, por sua vez, foi substituída pela norma ISO/IEC 25010 em 2011, mas para o trabalho atual as alterações introduzidas não se revestem de importância significativa. Devido a este facto são mantidas referências à norma original, por esta ser mais conhecida.

A norma ISO/IEC 14598 está subdividida em 6 partes: (14598-1) visão geral; (14598-2) planeamento e gestão; (14598-3) processo para programadores; (14598-4) processo para compradores; (14598-5) processo para avaliadores; (14598-6) documentação de módulos de avaliação. Na Figura 10, não são referidas todas as partes por razões de simplificação.

Na Figura 11 encontra-se representado o processo de avaliação de produtos de software, preconizado pela norma 14598-1. Este processo é composto por quatro fases: i) Estabelecimento de requisitos de avaliação, ii) Especificação da avaliação, iii) Projeto da avaliação e iv) Execução da avaliação.

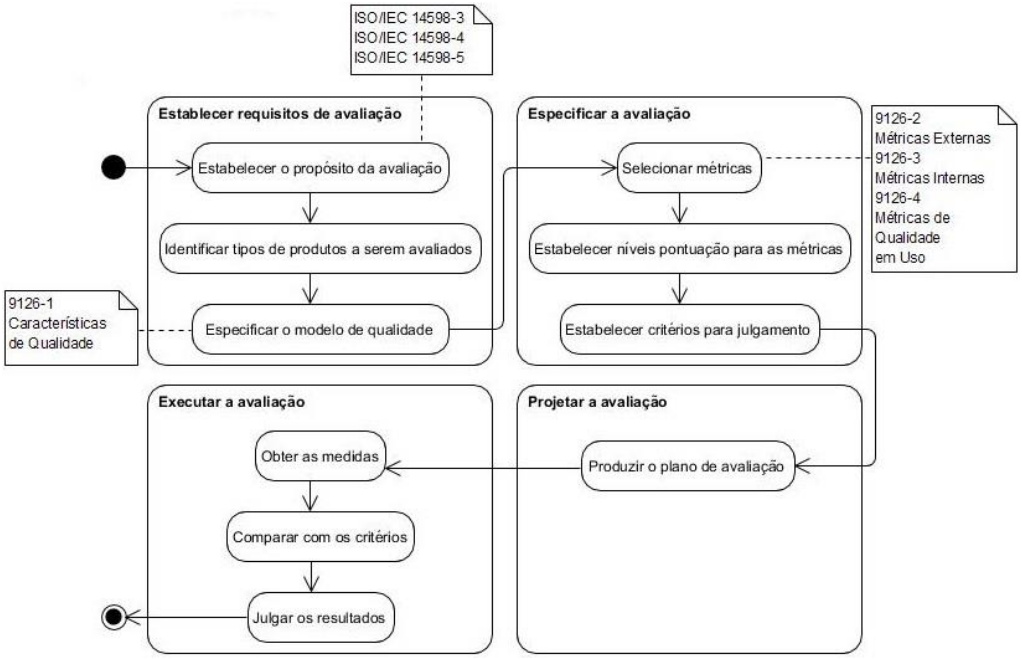

Figura 11 - Processo de avaliação

As caixas de comentário, que aparecem na Figura 11 ao lado de cada uma das fases, identificam os padrões ISSO a serem seguidos em cada uma das fases.

Análise tecnológica.

A teoria, princípios e metodologia subjacentes ao desenvolvimento orientado a modelos permitiu uma grande evolução no processo de desenvolvimento de software. Esta abordagem tornou possível a criação de plataformas de software, capazes de aportar ao referido processo uma abordagem baseada em sólidos princípios de engenharia.

Estas plataformas proporcionam um ambiente de desenvolvimento de software, onde é possível realizar as tarefas inerentes a esta atividade de uma forma sistemática, estruturada e assistida por inúmeros processos automatizados.

Com a abordagem orientada a modelos o processo de desenvolvimento é divido em duas áreas: Engenharia de domínio e Engenharia aplicacional.

Numa primeira fase procede-se à modelação do domínio da aplicação e ao desenvolvimento de artefactos e transformações, baseadas nos modelos criados, que automatizam o desenvolvimento de software (engenharia aplicacional).

Na fase seguinte os artefactos anteriormente criados, são usados na especificação, design, desenvolvimento e teste da aplicação final. A utilização dos referidos artefactos oferece um nível de abstração mais elevado, um enquadramento de apoio e a automatização das atividades da engenharia aplicacional.

2.2.3.2.1 Artefactos

Os artefactos gerados e não gerados são elementos da aplicação de software. Alguns destes artefactos podem ser relevantes somente durante o tempo de desenvolvimento, enquanto outros artefactos podem ser relevantes em tempo de execução. No entanto, ambos os tipos de artefactos são fortemente dependentes das plataformas envolvidas.

Como exemplo de artefactos podem ser considerados: arquivos de código-fonte e binários, scripts de configuração e implantação, scripts de base de dados e até mesmo arquivos de documentação, incluindo os próprios modelos.

2.2.3.2.2 Transformações

Existem dois tipos principais de transformações que tendem a ser considerados na abordagem MDD. Por um lado, as transformações modelo para texto (M2T) que geram ou produzem artefactos de software – tipicamente código fonte, XML e outros arquivos de texto –, a partir de modelos. A técnica mais comum para essa classe de transformações é conhecida como geração de código, e existem várias soluções e técnicas, conforme discutido por Czarnecki e outros (Czarnecki & UW, 2000).

Por outro lado, as transformações modelo para modelo (M2M) que permitem traduzir modelos noutro conjunto de modelos, normalmente mais próximos do domínio da solução ou que satisfazem necessidades específicas para as diferentes partes interessadas.

Estas transformações são especificadas por meio de linguagens distintas, como as linguagens de programação generalistas, mas também por linguagens de transformação de modelo especializadas em finalidades diferentes e com paradigmas de modelação diferente, como QVT, Acceleo, ATL, VIATRA, DSLTrans (Wimmer, et al., 2009).

2.2.3.2.3 Modelos

Os modelos são um conceito central da abordagem MDD.

Por um lado, um modelo pode ser criado diretamente pelos utilizadores (ou seja, designers de modelo) ou pode ser produzido automaticamente a partir de transformações modelo-para-modelo e, em seguida, ainda editado e refinado.

Por outro lado, os modelos podem ser usados para produzir artefactos de software gerados ou não gerados, respetivamente por meio de transformações M2T ou de criação direta pelos utilizadores (ou seja, engenheiros de software).

Além disso, em algumas situações particulares, os modelos podem ser interpretados e executados diretamente por plataformas específicas integradas na aplicação de software (Luz & Silva, 2004).

É importante salientar que no contexto de MDD a utilização de modelos implica uma definição consistente e rigorosa. Em geral, é necessário um certo nível de qualidade para que esses

modelos possam ser usados corretamente em cenários model M2M ou M2T. Para este efeito, existem características que as ferramentas devem fornecer, tais como análise de modelo, validação e simulação, conforme discutido em (Vangheluwe, et al., 2002).

2.2.4 Abordagens orientadas a modelos

Nesta seção são discutidas várias abordagens orientadas a modelos. Esta questão é importante para promover a compreensão clara dos seus significados e respetivas relações.

A Figura 5 apresenta o modelo conceptual proposto por (Silva, 2015), que introduz os termos utilizados no âmbito das abordagens orientadas a modelos e as suas relações.

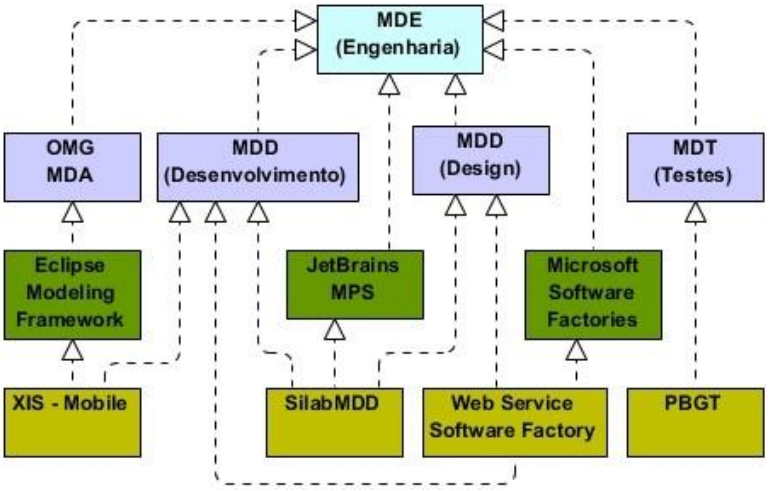

Figura 5 - Abordagens orientadas a modelos

O modelo apresentado na Figura 5, agrupa hierarquicamente, através de cores e relacionamentos de generalização e especialização, as abordagens orientadas a modelos com base no nível de abstração. No topo da hierarquia, encontra-se o termo mais abstrato (MDE), seguido pelas metodologias, algumas plataformas e por fim algumas abordagens concretas. Nas secções seguintes é apresentada uma breve descrição dos conceitos mais relevantes.

2.2.4.1 MDE

MDE é uma abordagem de engenharia de software que considera modelos não apenas artefactos de documentação, mas cidadãos de primeira classe. Os modelos podem ser usados em todas as disciplinas de engenharia e em qualquer domínio de aplicação.

Neste contexto, o MDE é melhor classificado como um paradigma de engenharia de software e assim, não tem nenhuma sustentação concreta por parte das ferramentas.

2.2.4.2 MDD

A abordagem de desenvolvimento orientado a modelos (MDD) concentra-se principalmente nos requisitos, análise e design, e disciplinas de implementação. As abordagens concretas MDD

tendem a definir linguagens de modelação para especificar o sistema em estudo em diferentes níveis de abstração, para fornecer transformações M2M e M2T, a fim de melhorar a produtividade e a qualidade do processo e o sistema de software final.

2.2.4.3 MDT

A abordagem aos testes baseados em modelos concentra-se principalmente na automatização da disciplina de testes. Os modelos de teste são usados para representar o comportamento desejado do sistema teste (SUT), para representar as estratégias de teste e o ambiente de teste.

Um modelo de teste descrevendo o SUT é geralmente uma representação abstrata do comportamento desejado do SUT. Casos de teste derivados de um modelo são testes funcionais no mesmo nível de abstração como o modelo e, em seguida, podem ser mapeados em testes executáveis que podem se comunicar diretamente com o SUT em ferramentas de teste específicas e estruturas.

2.2.4.4 MDA

A arquitetura orientada a modelos (MDA) é a abordagem orientada a modelos proposta pela OMG, disponível desde o início de 2000, e focada principalmente na definição de modelos e suas transformações.

A MDA defende a definição de modelos em diferentes níveis de abstração, ou seja, modelos independentes computacionais (CIM), modelos independentes de plataforma (PIM) e modelos específicos de plataforma (PSM).

As plataformas computacionais correspondem a implementações concretas de servidores de aplicativos, servidores de base de dados, sistemas de gestão de conteúdo, framework e arquiteturas de software; essas plataformas podem ser descritas por meio de modelos de descrição de plataforma (PDM).

O MDA também considera diferentes tipos de transformações modelo a modelo, a saber: CIM-CIM, CIM-PIM, PIM-PIM, PIM-PSM e PSM-PSM. Além disso, considera a transformação de modelos de PSM em código-fonte e outros tipos de artefactos textuais (PSM-Text). Em teoria, um aplicativo desenvolvido na abordagem MDA é independente de plataforma, que permite que ele seja instalado em diferentes plataformas de computação e suporta adequadamente diferentes tecnologias, graças a essas transformações, especialmente para PIM-PSM PSM-PSM e transformações de texto PSM.

Apesar de não ser uma abordagem orientada a modelos concreta, porque não especifica linguagens de modelação concreta e ferramentas associadas. A MDA defende o uso de várias especificações de OMG concretas, a saber: (i) MOF como um componente central de sua arquitetura de meta-modelação; (ii) QVT 6 como um conjunto de linguagens para consulta e transformações do modelo; e (iii) perfis UML como uma forma simples, mas prática de definir DSML gráficas.

2.2.4.5 Plataformas

No nível médio identificamos as abordagens orientadas a modelos, conforme proposto pelas respetivas empresas ou comunidades.

Essas abordagens são baseadas em tecnologias e geralmente suportadas por ferramentas complexas a que o autor se refere como "Meta-Ferramentas" e são comumente conhecidas como "*Language Workbench*". A maioria dessas ferramentas fornece uma coleção de recursos para ajudar os utilizadores a definir DSL, com editores específicos, validação de modelo, transformação de modelo, etc.

2.2.4.6 Abordagens concretas

Finalmente são apresentados alguns exemplos de abordagens orientadas a modelos concretas, ou seja, XIS-Mobile, WebService Software Factory, SilabMDD e PBGT. Estes exemplos mostram a aplicabilidade de modelos de uma forma direta.

2.2.5 Domain-driven design

O design orientado por domínio (DDD) é uma abordagem para o desenvolvimento de software com base em dois princípios principais (Evans, 2003):

1) O foco principal de um projeto de software deve ser o próprio domínio e não os detalhes técnicos.

17) Designs de domínio complexos devem ser baseados num modelo.

Como tal, DDD enfatiza a importância de uma representação adequada e efetiva do domínio do problema do sistema a ser desenvolvido. Para este efeito, o DDD fornece um extenso conjunto de práticas de design e técnicas, destinadas a ajudar os programadores de software e especialistas em domínio a partilharem os seus conhecimentos do domínio, através da representação dos mesmos sobre a forma de modelos.

Claramente, o DDD compartilha muitos aspetos com o MDD. Ambos defendem a necessidade de usar modelos para representar o conhecimento do domínio e que o processo de desenvolvimento, deve começar por concentrar os esforços em aspetos independentes da plataforma.

Neste sentido, MDD pode ser considerado como uma estrutura que fornece as técnicas para colocar o DDD em prática (para modelar o domínio, criar DSL que facilitam a comunicação entre especialistas de domínio e programadores, etc.)

Ao mesmo tempo, o MDD complementa o DDD ajudando os programadores a beneficiarem ainda mais dos modelos de domínio, graças às transformações de modelos e técnicas de geração de código.

O modelo de domínio pode ser usado não só para representar o domínio (estrutura, regras, dinâmica, etc.), mas também para gerar o sistema de software real que será usado para gerir o referido modelo.

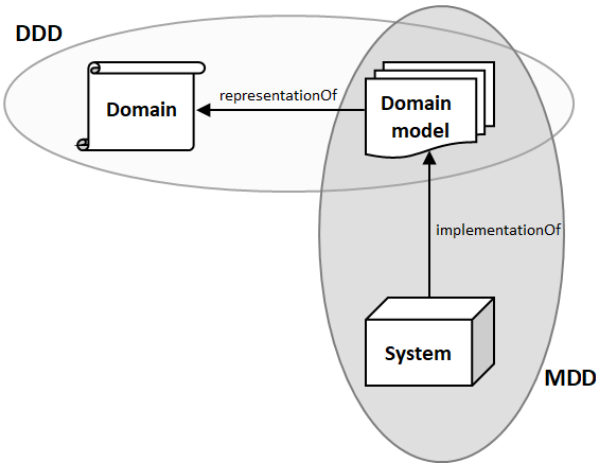

Figura 6 - DDD vs. MDD

A Figura 6 - DDD vs. MDD, visa destacar de uma forma simples o foco diferente de cada técnica e como eles podem ser combinadas entre si.

2.2.6 Model driven design

De acordo com (Evans, 2003), a metodologia de desenvolvimento orientada a modelos não pretende introduzir fundamentos de design radicalmente diferentes, mas sim alterar a ênfase que é dada aos diferentes elementos constituintes das práticas adotadas no design de software. Desta forma os modelos passam a ser os elementos basilares do processo de desenvolvimento de software.

2.2.6.1 Padrões de design

Na Figura 7 é apresentado o esquema, proposto por (Evans, 2003), onde são identificados os padrões de design usados no MDD e as suas relações.

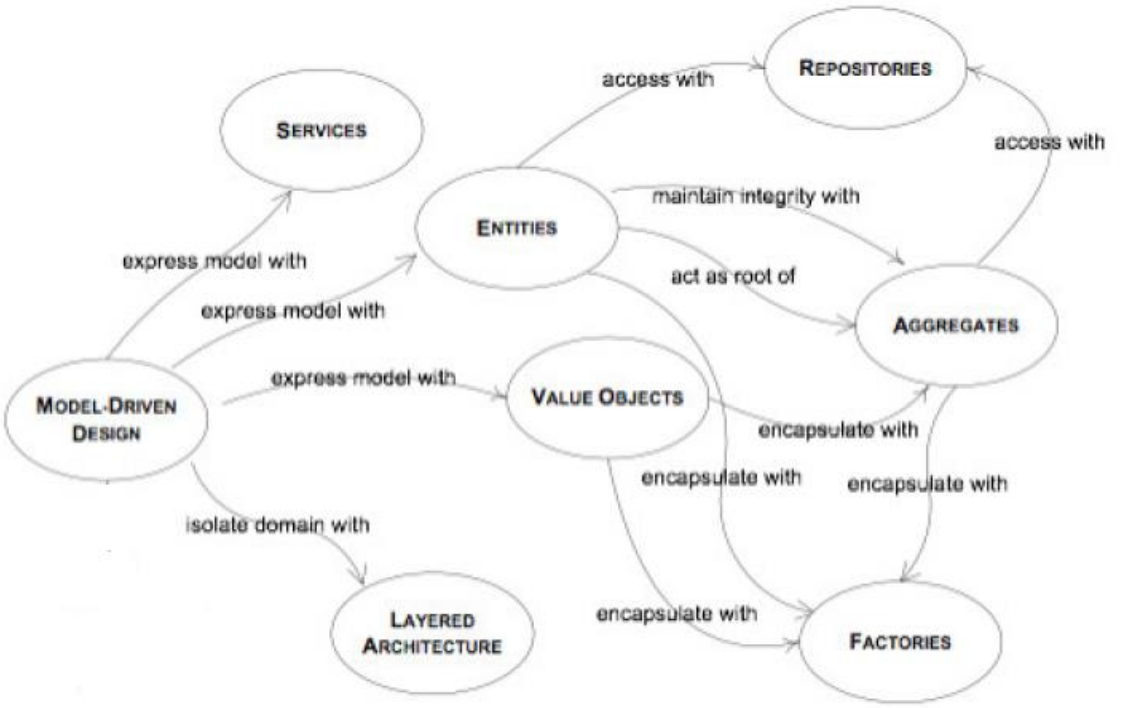

Figura 7 - Blocos fundamentais do design orientado a modelos

Embora o os padrões apresentados na Figura 7não sejam novos, a sua utilização no design orientado a modelos adquire contornos próprios, como se explica a seguir.

1) Serviços: são objetos desprovidos de estado, que executam uma ação.

2) Objetos valor: normalmente podem ser representados como objetos imutáveis. Alterar uma propriedade de um objeto de valor essencialmente destrói o objeto antigo e cria um novo porque foi alterada a definição do objeto de valor.

3) Entidades: são objetos que são acessíveis com uma identidade. Cada entidade deve ter uma maneira operacional de estabelecer sua identidade com outros objetos, e é distinguível até mesmo de outro objeto com os mesmos atributos descritivos.

4) Repositórios: formam uma camada intermedia entre o modelo de domínio e a persistência de estados em fontes de dados. Permitem gravar e recuperar o estado das várias instâncias dos modelos, a cada momento através de abstrações responsáveis por fornecer os métodos necessários de comunicação com a infraestrutura de persistência.

5) Agregados: um agregado é um cluster de objetos associados, que são tratados como uma unidade para fins de alterações de dados.

6) Fabricas: são um padrão que permite encapsular a complexidade da construção de objetos. Estas estruturas são responsáveis pela criação de instâncias de objetos que exigem certa lógica de construção que se pretende oculta.

7) Arquitetura em camadas: este estilo arquitetural defende o particionamento de um sistema complexo em camadas. Dentro de cada camada, deve ser adotado um design coeso e que depende apenas das camadas abaixo. Devem ser adotados padrões arquitetónicos standard (MVC, Boundary-Control-Entity, etc) para fornecer independência relativamente ás camadas superiores.

2.2.6.2 Arquitetura em camadas orientada a modelos

Todo o código relacionado ao modelo de domínio, deve ficar separado numa camada e isolá-lo da interface de utilizador, da camada de aplicação e do código de infraestrutura.

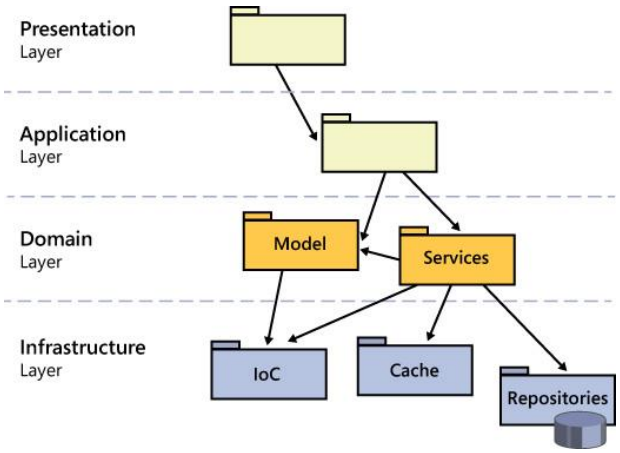

Figura 8 - Arquitetura em camadas

2.2.6.2.1 Camada de apresentação

De acordo com (Esposito & Saltarello, 2014) a camada de apresentação é responsável por fornecer as interfaces de utilizador(UI) para realizar todas as tarefas. Apresentação é uma coleção de ecrãs; cada ecrã é preenchido por um conjunto de dados (modelo de visualização) e qualquer ação que começa a partir do ecrã devolve outro conjunto de dados (modelo de leitura).

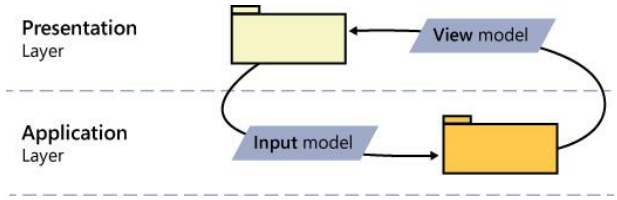

Figura 9 - Dados na camada de apresentação

2.2.6.2.2 Camada de aplicação

A camada de aplicação é uma excelente maneira de separar camadas de apresentação e domínio. Ao fazê-lo, a camada de aplicação contribui grandemente para a clareza de todo o design.

A camada de aplicação é a camada adicional que reporta à apresentação e é onde se realiza a orquestração da interação entre vários casos de uso e entre as ações das entidades de domínio que os compõem.

Cada controlo da interface de utilizador (por exemplo, botões) aciona uma ação no back-end do sistema. Em alguns cenários simples, a ação que resulta do clique leva apenas à execução de um único passo. Mas na maioria das vezes, em vez disso, o clique normalmente dispara algo como um fluxo de trabalho.

A camada de aplicação é o ponto de entrada no back-end do sistema e o ponto de contato entre a apresentação e o back-end.

Por exemplo, numa aplicação MVC, é espectável que as classes da camada de aplicação estejam intimamente relacionadas com os controladores.

Esta camada não tem estado que reflete a situação de negócio, mas pode ter estado que reflete a evolução de uma tarefa. Não contém regras ou informações de negócios, mas apenas coordena tarefas e delega trabalho nos agregados de objetos de domínio na próxima camada em baixo.

Resumindo, a camada de aplicação é responsável pela orquestração da sequência de execução de várias unidades funcionais, que constituem uma operação completa. Tudo o que faz é controlar a sequência de execução, em função da informação que recebe da camada de apresentação, e delegar trabalho nas camadas inferiores da pilha.

Tipicamente pode haver dois tipos de uma camada de aplicação dentro ou fora da lógica de negócios. Nenhum dos tipos é preferível ao outro; tem tudo a ver como de vislumbra o sistema.

2.2.6.2.3 Camada de domínio

A camada de domínio aloja toda a lógica de negócio especifica das entidades do domínio. Por outras palavras, a camada de domínio contém toda a lógica de negócios que permanece depois de ser ter estruturado a camada de aplicativo.

A camada de domínio é composta por um modelo de domínio e, pelos serviços de domínio. A natureza do modelo pode variar. Na maior parte das vezes, é um modelo de entidades e relacionamentos, e as entidades no modelo devem expor os dados e o comportamento.

O objetivo final de um modelo de domínio é implementar a linguagem e expressar as ações que compõe os processos de negócios.

Os serviços de domínio são partes da lógica de domínio que, por algum motivo, não se encaixam em nenhuma das entidades existentes. Tipicamente um serviço de domínio é uma classe e agrupa comportamentos logicamente relacionados que normalmente operam em várias entidades de domínio. Um serviço de domínio geralmente também requer acesso à camada de infraestrutura para operações de leitura/gravação.

2.2.6.2.4 Camada de infraestrutura

A camada de infraestrutura é algo relacionado ao uso de tecnologias concretas, seja a persistência de dados (estruturas de O/RM), API de segurança específica, registro em log, rastreamento, cache e outros.

O componente mais proeminente da camada de infraestrutura é a camada de persistência — que não é nada mais do que a camada de acesso a dados, apenas possivelmente estendida para abranger algumas fontes de dados que não sejam armazenamentos de dados relacionais simples. A camada de persistência sabe como ler e/ou gravar dados.

2.2.7 Domain Driven vs Model Driven

De acordo com (Evans, 2004) como dois nomes são semelhantes, e porque os dois conceitos normalmente são usados em conjunto, eles são muitas vezes confundidos. Mas, na realidade, são duas maneiras distintas de abordar o desenvolvimento de software que também se reforçam mutuamente.

Design orientado ao domínio baseia-se na premissa que o cerne do desenvolvimento de software é o conhecimento da área onde se encontra o problema e a importância em encontrar maneiras úteis de compreender essa área.

A complexidade que devemos abordar é a complexidade do próprio domínio — não a arquitetura técnica, não a interface de utilizador, nem mesmo recursos específicos. Isto significa projetar tudo em torno de nossa compreensão e conceção dos conceitos mais essenciais do negócio e justificar qualquer outro desenvolvimento apenas na perspetiva de como ele suporta esse núcleo.

Para fazer isso, especialistas em domínio e especialistas em software têm de experimentar juntos, para encontrar maneiras de organizar seu conhecimento do domínio que justifica o desenvolvimento do software. O referido conhecimento do domínio, deve condicionar a definição das prioridades por forma a que o software deve ser construído, em primeiro lugar, para representar e raciocinar sobre o domínio. Devem ser afastados até os aspetos superficiais desse mesmo domínio e extrair apenas os princípios nucleares.

Esta destilação do conhecimento num conjunto claro de conceitos é um processo de modelação.

Estes modelos têm que ser incorporados na linguagem da equipa, e cada conversa nesta linguagem pode ser uma sessão de modelação. E, portanto, o design orientado ao domínio conduz inevitavelmente à modelação porque a modelação é a maneira como se lida com a compreensão de domínios complexos.

Mas quando os modelos apenas flutuam no ar, não estão a justificar plenamente a sua adoção. Para cada funcionalidade que o software aborda, ainda é necessário realizar o seu design e nesta fase existe o risco de o modelo se tornar irrelevante ou até mesmo enganoso

Um design orientado a modelos permite a criação de software estruturado em torno de algum conjunto de conceitos, mas é a aplicação de design orientado a modelos do domínio que permite que um projeto de desenvolvimento realmente atinja os seus objetivos.

O MDD permite incorporar os modelos (destilados através da análise do conhecimento do domínio, realizada pelos programadores em parceria com os especialistas de domínio) no próprio tecido do software.

Para o design orientado a modelos, o design não é apenas baseado no modelo de domínio, um modelo também é escolhido para atender às considerações de design e implementação.

É por isso que o design orientado a modelos é um companheiro natural e indispensável para o design orientado ao domínio. Pois permite que uma compreensão profunda do domínio, possa passar das mentes dos especialistas de domínio e dos utilizadores para os programadores, através do uso de uma linguagem comum e omnipresente.

2.2.8 DSL

De uma forma geral é possível definir DSL (linguagem específica de domínio) como sendo uma linguagem de programação de computadores de expressividade limitada focada num domínio específico (Fowler, 2010).

No entanto no contexto do documento atual o acrónimo DSL é utilizado de forma mais abrangente e pretende referenciar o tipo de DSL orientada a modelo (2.2.3 Software como produto).

Este tipo de DSL, embora comunguem das características elencadas na definição genérica do termo, remetem para uma abordagem mais concreta que é descrita na secção 2.2.9 *Language Workbench*.

Tendo em conta estas considerações, o acrónimo DSL é utilizado ao longo do documento em contextos que remetem não só para a definição geral do termo, mas também para a sua natureza orientada a modelo e para as suas características no contexto dos *Language Workbench*.

2.2.9 Language Workbench

O termo *Language Workbench* foi cunhado por Martin Fowler em 2005 (Fowler, 2005) no seu artigo onde ele os define como sendo ferramentas com as seguintes características:

Os utilizadores podem livremente definir novas linguagens (DSL), que são passiveis de serem completamente integradas umas nas outras;

A principal fonte de informação é uma representação abstrata persistente;

A DSL é definida em quatro partes principais: (i) esquema, (ii) restrições, (iii) editor(es) e (iv) gerador(es);

Os utilizadores das linguagens manipulam uma DSL através de um editor projecional, modulação de linguagem e IDE, extensão e composição.

2.2.10 Use-Case 2.0

É uma prática ágil (Jacobson, et al., 2011) que usa casos de uso para capturar os requisitos funcionais do sistema e que fornece uma estrutura para os decompor, de forma incremental, á medida que se realiza o design, implementação e testes. Numa primeira fase são identificados os atores e casos de uso. Posteriormente, em cada uma das fases do desenvolvimento, os casos de uso são organizados em *Slices*, que representam as unidades básicas de trabalho para cada uma das fases.

Os casos de uso capturam objetivos específicos do sistema para um tipo especifico de utilizador, por forma a permitir descrever o sistema caso a caso em vez de o abordar de uma só vez.

Para entender um caso de uso, são contadas histórias. Um caso de uso é descrito numa narrativa composta por vários fluxos. Cada fluxo representa uma história descrita em múltiplos passos. Cada passo pode representar uma tarefa simples ou uma nova história (fluxo). Desta forma tornam-se claras as relações entre as maneiras de usar o sistema.

Os fluxos, dos vários casos de uso, podem ser agrupados ou divididos para formar *Slices*, em função das necessidades de cada uma das fases de desenvolvimento. Permitindo desta forma organizar o trabalho e rastear os requisitos.

A narrativa é apresentada sobre a forma de uma tabela para cada caso de uso. Esta tabela é composta por duas colunas: Fluxo básico e Fluxos alternativos. O fluxo básico descreve a maneira mais simples de atingir o objetivo do caso de uso. Os demais são apresentados como fluxos alternativos. Um fluxo alternativo também pode ser usado para definir o fluxo que descreve a forma de lidar com quaisquer problemas que possam ocorrer.

A narrativa é complementada por informação de suporte, que visa esclarecer e fundamentar as histórias representadas nos fluxos que compõem a narrativa. Na informação de suporte também devem ser apresentadas eventuais explicações que se apliquem a mais do que um caso de uso.

O foco da narrativa do caso de uso deve ser na forma como o sistema é usado e não em longas listas de funções ou recursos que o sistema oferece. Uma vez que, para os utilizadores e outras partes interessadas, o valor de um sistema apenas é gerado se o sistema for realmente usado. Esta abordagem permite identificar primeiro as formas mais valiosas de utilização do sistema, e que as menos valiosas sejam adicionadas mais tarde, sem destruir ou alterar o que foi feito antes.

Ao usar a narração de histórias como uma técnica para comunicar requisitos, é essencial garantir que as histórias sejam capturadas de maneira a torná-las executáveis e testáveis. Um conjunto de casos de teste deve acompanhar cada uma das narrativas, porque eles tornam as histórias reais e seu uso pode demonstrar inequivocamente que o sistema faz o que deve fazer. São os casos de teste que definem o que significa implementar com sucesso o caso de uso.

2.2.11 Avaliação

2.2.11.1 Qualidade

De acordo com (Sanders, 1994) qualidade de software é: "Um produto de software apresenta qualidade dependendo do grau de satisfação das necessidades dos clientes sob todos os aspetos do produto".

Segundo (Pressman, 2001), "Qualidade de software é a conformidade a requisitos funcionais e de desempenho que foram explicitamente declarados, a padrões de desenvolvimento claramente documentados, e a características implícitas que são esperadas de todo software desenvolvido por profissionais".

As definições enfatizam três aspetos importantes:

1) Os requisitos de software são a base a partir da qual a qualidade é medida. A falta de conformidade aos requisitos significa falta de qualidade.

2) Padrões especificados definem um conjunto de critérios de desenvolvimento que orientam a maneira segundo a qual o software passa pelo trabalho de engenharia. Se os critérios não forem seguidos, o resultado quase que seguramente será a falta de qualidade.

3) Existe um conjunto de requisitos implícitos (por exemplo o desejo de uma boa manutenibilidade. Se o software se adequar aos seus requisitos funcionais, mas deixar de cumprir seus requisitos não funcionais, a qualidade do software fica comprometida.

2.2.11.2 Processo de avaliação

Segundo (Bourque & Fairley, 2014) a análise da qualidade de software pode ser dividida em duas grandes áreas de conhecimento: a qualidades do processo de software e a qualidade do produto de software. Na Figura 10 é apresentado um diagrama que representa a divisão da análise de qualidade de software em áreas de conhecimento e respetivas metodologias.

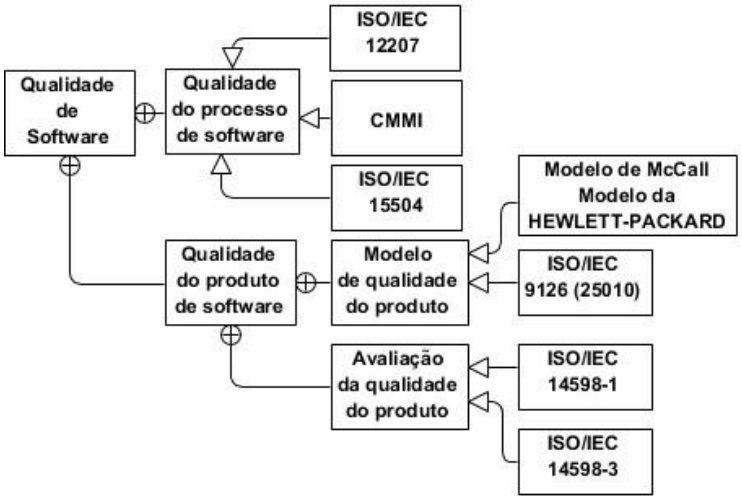

Figura 10 – Áreas da qualidade de software

A primeira área tem a ver com qualidade das atividades/tarefas que compõem os processos de ciclo de vida de software (requisitos, design, construção, testes, manutenção, etc.). Nesta área o objetivo é avaliar maturidade organizacional ou a capacidade das áreas da empresa relacionadas com o processo de desenvolvimento e estabelecer prioridades para melhoria (ISO/IEC12207, 2008) (CMMI, 2018).

A área de qualidade do produto de software, por sua vez, pode ser dividida em duas subáreas: modelo de qualidade do produto e avaliação da qualidade do produto. A primeira subárea (ISO/IEC25010, 2011) tem como objetivo a definição dos atributos de qualidade e dos critérios e métricas para produtos de software, ao passo que a segunda (ISO/IEC14598-1, 1999) visa a definição de um processo de avaliação de produtos de software.

Os modelos McCall e HEWLETT-PACKARD, são modelos antigos e que foram substituídos pela norma ISO/IEC 9126, que oferece uma abordagem mais abrangente e padronizada a nível mundial. A referência a estes modelos apenas foi incluída por motivos históricos e académicos.

A norma ISO/IEC 9126, por sua vez, foi substituída pela norma ISO/IEC 25010 em 2011, mas para o trabalho atual as alterações introduzidas não se revestem de importância significativa. Devido a este facto são mantidas referências à norma original, por esta ser mais conhecida.

A norma ISO/IEC 14598 está subdividida em 6 partes: (14598-1) visão geral; (14598-2) planeamento e gestão; (14598-3) processo para programadores; (14598-4) processo para compradores; (14598-5) processo para avaliadores; (14598-6) documentação de módulos de avaliação. Na Figura 10, não são referidas todas as partes por razões de simplificação.

Na Figura 11 encontra-se representado o processo de avaliação de produtos de software, preconizado pela norma 14598-1. Este processo é composto por quatro fases: i) Estabelecimento de requisitos de avaliação, ii) Especificação da avaliação, iii) Projeto da avaliação e iv) Execução da avaliação.

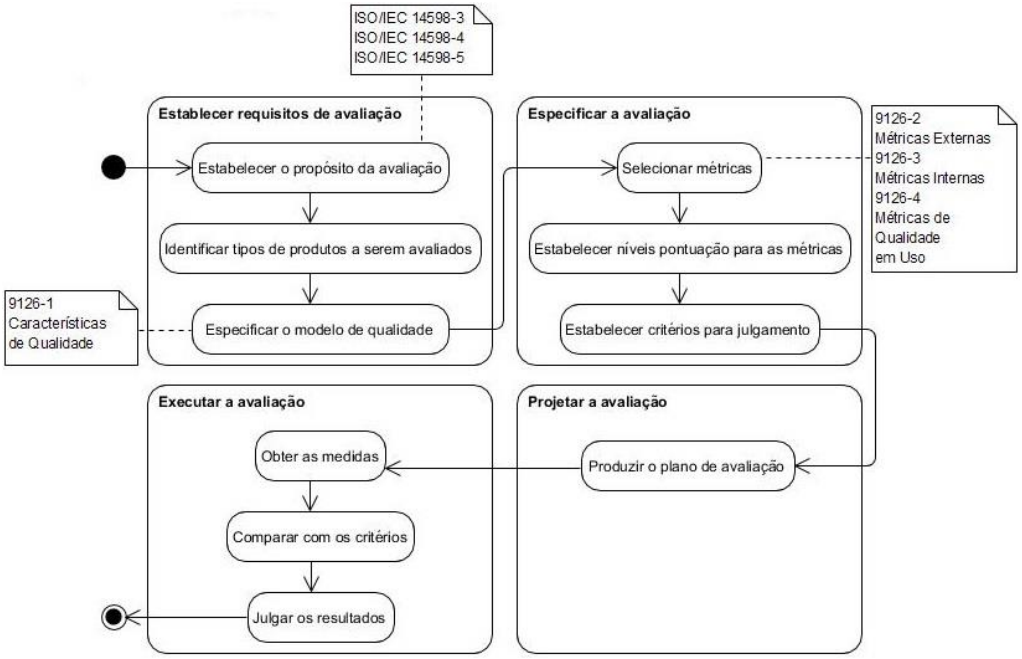

Figura 11 - Processo de avaliação

As caixas de comentário, que aparecem na Figura 11 ao lado de cada uma das fases, identificam os padrões ISSO a serem seguidos em cada uma das fases.

2.3 Análise tecnológica

Nesta Seção encontra-se o resumo do resultado da investigação realizada durante este projeto na área das ferramentas que suportam o desenvolvimento de DSL. A investigação realizada tentou identificar as plataformas e ferramentas com mais relevância e mais conhecidas.

2.3.1 Intentional platform

O sistema de programação intencional (Intentional Software, 2017) permite que os programas sejam escritos e visualizados numa variedade de notações específicas, e também permite a integração harmoniosa de linguagens específicas de domínio e de uso geral.

Esta plataforma é produzida pela empresa Intentional Software, que visa desenvolver e comercializar software para o processamento do conhecimento.

A plataforma intencional é uma plataforma para desenvolver e executar aplicações. Neste tipo de aplicações, o conhecimento torna-se um ativo tangível que os computadores podem processar. Semelhante a outras tecnologias de processamento-como o processamento de texto, processamento de linguagem natural ou processamento de transações processamento do conhecimento opera sobre o conhecimento e transforma-o em output útil. Exemplos incluem automatizar um processo de negócios ou gerar software.

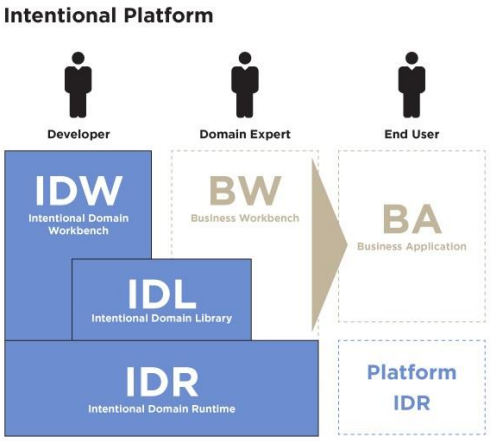

Figura 12 - Diagrama da plataforma Itentional

Linguagens específicas de domínio são usadas na plataforma intencional para gravar conhecimento e são projetadas através de um editor de projeção para fornecer uma interface intuitiva para o especialista em domínio. A descrição do conhecimento tem sintaxe, que é o que está a ser representado e projetado, e tem semântica, que é o que está a ser executado. O conhecimento é representado numa "linguagem" como "código" no sentido de computação tradicional; "linguagem de domínio" e "código de domínio", ou seja, linguagem e código para um domínio específico.

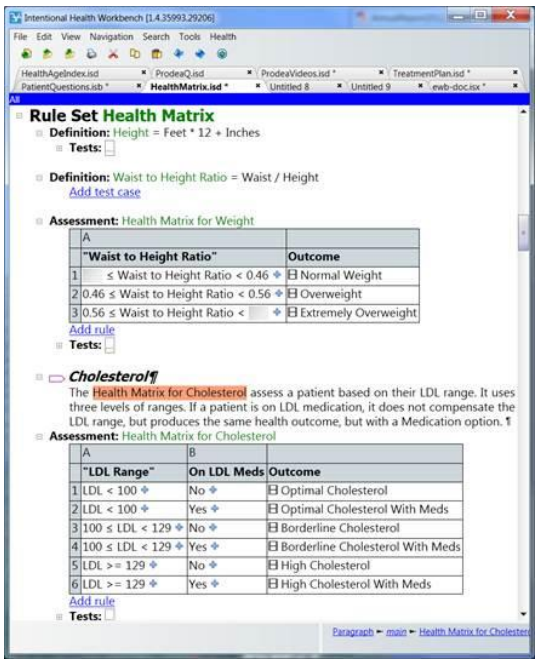

Figura 13 - Exemplo de uma DSL na plataforma Intentional

A plataforma intencional usa DSL na sua base. As DSL tendem a ser muito técnicas e, portanto, não são adequadas para utilizadores de negócios. Portanto, a plataforma intencional projeta a DSL numa interface de utilizador com que os utilizadores de negócios se sintam mais

confortáveis. A plataforma intencional usa um editor de projecional para elevar o nível da linguagem para perto da linguagem de negócio. As projeções das DSL podem ser ajustadas às necessidades exatas do perito do domínio, -a pessoa com o conhecimento do domínio do negócio.

2.3.2 MetaEdit+

MetaEdit+ (Metacase, 2018) é um exemplo de uma Workbench de linguagem puramente gráfica (tabelas e formulários também podem ser usados). As linguagens são definidas através de meta modelos, restrições, geradores de código e símbolos gráficos associados com o meta-modelo. Os conceitos de linguagem podem ser representados por símbolos diferentes em tipos de diagramas diferentes e elementos de várias linguagens podem ser usados num diagrama. Elementos da linguagem A podem referenciar elementos na linguagem B. Isto não é surpreendente uma vez que as Workbench baseadas em linguagens gráficas são (e foram sempre) projecionais.

A criação de linguagens nesta ferramenta pode ser dividida em três passos, que ilustram o funcionamento da mesma.

O primeiro passo requer a definição dos conceitos e regras da linguagem a ser desenvolvida, com recurso a ferramentas de meta-modelação gráficas ou baseadas em formulários.

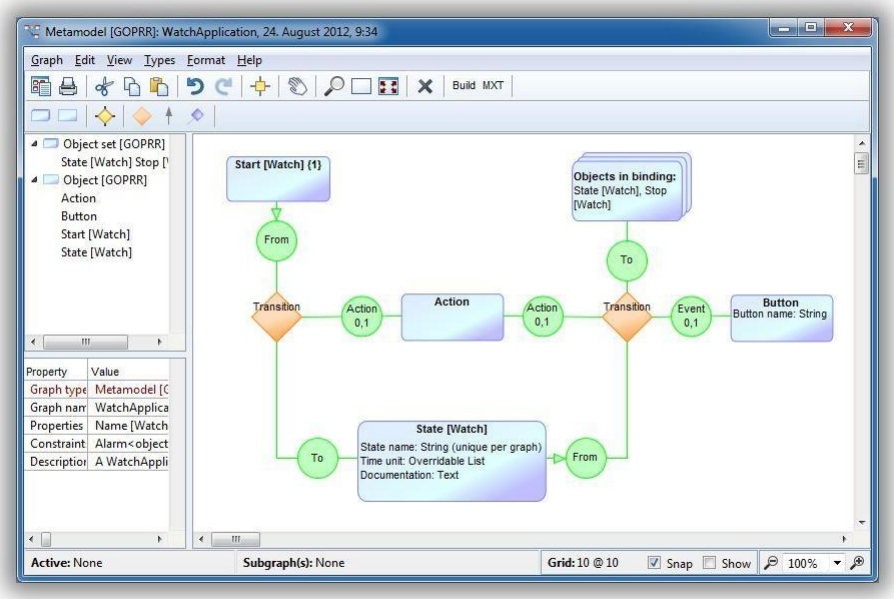

Figura 14 - Passo 1, definição do meta-modelo

O segundo passo é o desenho da notação a ser usada na linguagem e pode realizado com o editor de símbolos, que também permite o reaproveitamento de símbolos pré-existentes através da importação dos mesmos

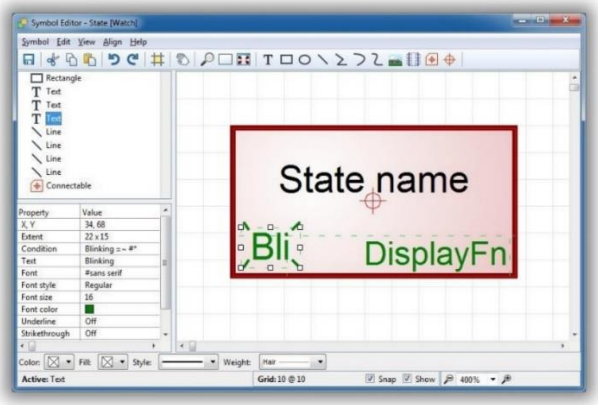

Figura 15 - Passo 2, desenho da notação

Por fim, é necessário construir os geradores para que estes produzam o código, configurações, analise, dados de teste, etc.

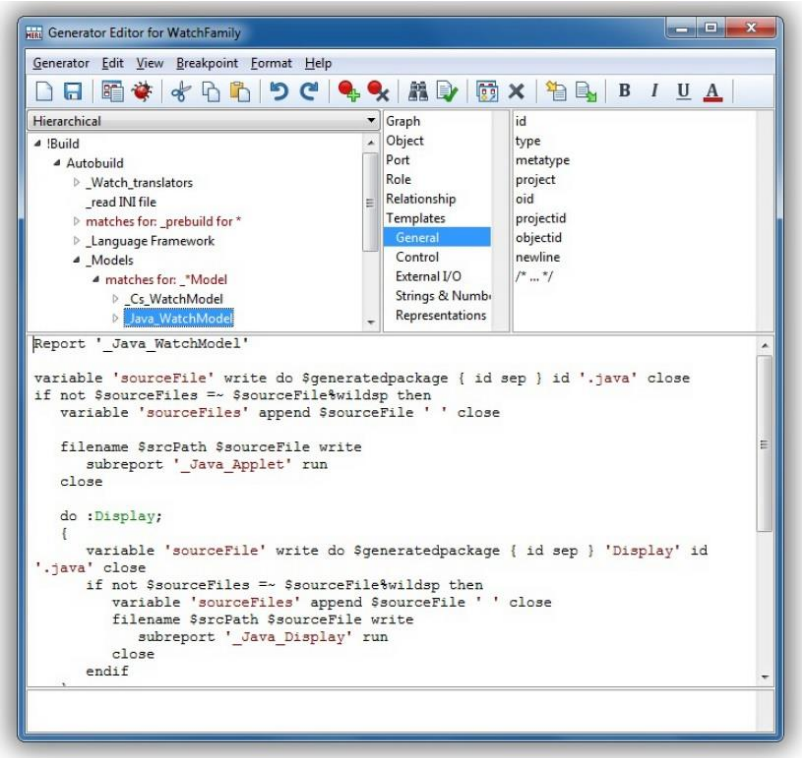

Figura 16 - Passo 3, Criar geradores

Assim que a linguagem se encontra definida esta framework oferece um conjunto de ferramentas de modelação que se adaptam automaticamente a linguagem previamente definida.

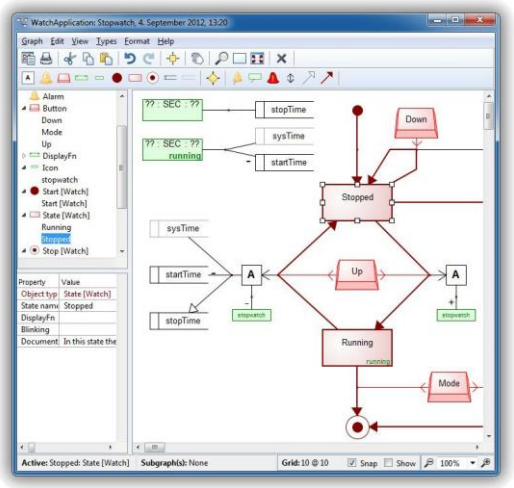

Figura 17 - Editor de diagramas

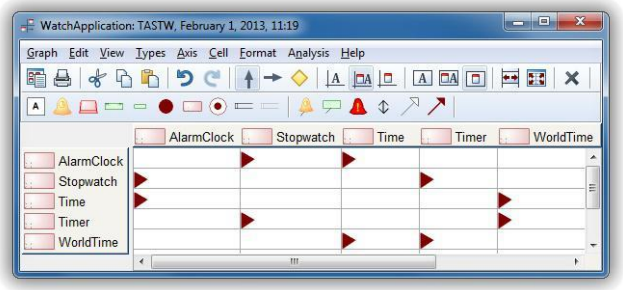

Figura 18 - Editor de matrizes

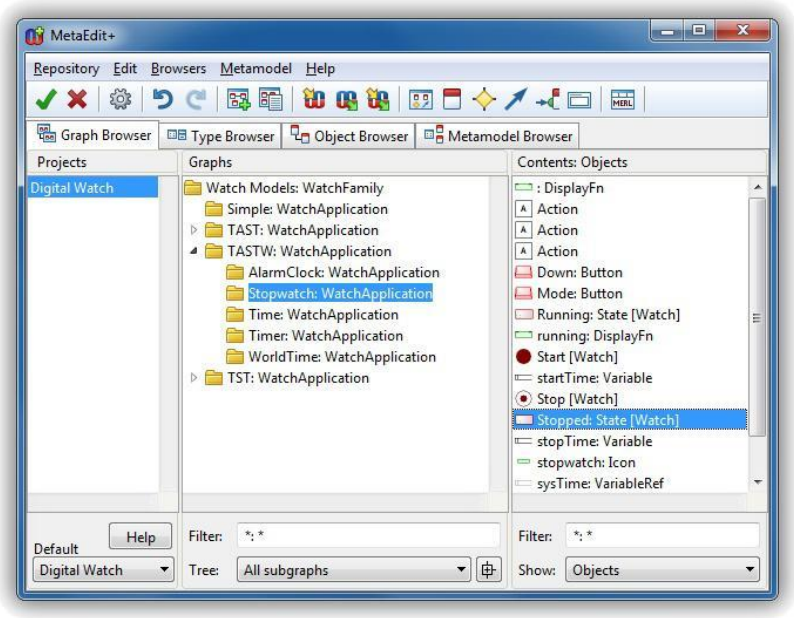

Figura 19 – Browser

Nas figuras Figura 17, Figura 18, Figura 19 podemos ver ilustrações das várias ferramentas de modelação que a plataforma fornece, com o intuito de manipular instâncias das linguagens que foram criadas.

2.3.3 Eclipse Xtext

A tecnologia Eclipse Xtext (Eclipse foundation, 2018) suporta a criação de editores de texto extremamente poderosos (com auto completar de código, verificação de erros e coloração de sintaxe) a partir da definição reforçada de uma gramática do tipo EBNF. Também gera um meta-modelo que representa a sintaxe abstrata da gramática, bem como um analisador que avalia frases da linguagem.

Como esta ferramenta usa o eclipse EMF como base para seus meta-modelos, pode ser usada em conjunto com qualquer ferramenta de transformação e geração de código baseada em EMF (exemplos incluem Xpand, ATL e Acceleo).

A combinação linguística é facilmente conseguida através da conclusão de código com referências de outros modelos, bem como com verificações cruzadas entre modelos e-linguagens.

A reutilização de linguagens, extensão e incorporação são bastante limitados, no entanto, é possível fazer uma linguagem estender uma outra linguagem.

Os conceitos da linguagem base podem ser usados na linguagem subordinada e é possível redefinir as regras gramaticais definidas na linguagem base.

Também é possível criar novos subtipos de elementos da linguagem base.

No entanto, não é possível definir representações diferentes do mesmo elemento e não é possível incorporar linguagens arbitrárias ou módulos de linguagem. Isto é principalmente porque a tecnologia subjacente do analisador é ANTLR que é um analisador clássico de duas fases que tem problemas com a composição gramatical.

Apesar de a herança única ser útil, e muitas DSL interessantes poderem ser construídos com XText, a extensão de linguagem de herança única não é suficiente. Na prática; seria como programação orientada a objetos com apenas herança e sem delegação ou “traits”.

2.3.4 SDF

SDF (StrateGeoxT, 2018) é uma linguagem para definir a sintaxe. Ele prevê um nível sem precedentes de concisão em definições de sintaxe. (i) A sintaxe pode ser definida em módulos. (ii) As sintaxes léxicas e livres de contexto são integradas num único formalismo no qual a sintaxe completa de uma linguagem pode ser definida. (iii) A SDF inclui construções declarativas

de desambiguação. Portanto, não há necessidade de introduzir símbolos não-terminais para lidar com as ambiguidades em uma língua.

Desenvolvido pela Universidade de Delft (Holanda), usa Parsers que não precisam de Scanner o que permite que as linguagens possam facilmente ser incorporados dentro de outras linguagens.

Os geradores de código são implementados através da substituição de termos na sintaxe abstrata, mas renderizados na sintaxe concreta.

Atualmente, a SDF é principalmente um conjunto de ferramentas de linha de comando, mas o suporte para IDE (com editores de Eclipse gerados automaticamente) está em desenvolvimento como parte do Spoofax (http://strategoxt.org/Spoofax).

O formalismo de definição de sintaxe SDF permite uma expressão concisa e natural da sintaxe de uma linguagem livre de contexto. O SDF integra num único formalismo de sintaxe a sintaxe léxica e livre de contexto. A sintaxe completa de uma linguagem é, portanto, definida numa única definição.

A SDF oferece suporte para gramáticas livres de contexto arbitrários. Portanto, a SDF não restringe as gramáticas a uma subclasse das gramáticas livres de contexto, como ll ou LALR.

As definições de sintaxe de SDF podem ser modulares. Os módulos SDF podem ser reutilizados em diferentes definições de sintaxe. A desambiguação de gramáticas não é feita através do hacking da gramática, mas através da utilização de funcionalidades de desambiguação de propósito, tais como prioridades, rejeição de produções, e seguimento de restrições.

SDF é uma linguagem declarativa. Isso significa que uma definição de sintaxe pode ser usada para fins diferentes além da utilização habitual, nomeadamente: a geração de um analisador. Em vez disso, as definições de sintaxe de SDF podem ser usadas para gerar outras ferramentas de definições de sintaxe, por exemplo, “Pretty-Print” e definições de tipo de dados.

A SDF suporta as seguintes funcionalidades: módulos; integração da sintaxe léxica e livre de contexto; classes de caracteres; expressões regulares; desambiguação por prioridades relativas; desambiguação por associatividade; desambiguação por preferência; aliases; renomeação e parametrização de módulos; restrições lookahead; rejeitar produções; palavras-chave insensíveis a maiúsculas e minúsculas.

2.3.5 Monticore

É outro ambiente de engenharia de linguagens (Monticore, 2018) baseado em Parsers que gera analisadores, meta-modelos e editores baseados em gramáticas estendidas. As linguagens podem estender-se mutuamente e podem ser incorporadas umas dentro das outras. Uma ideia importante é a capacidade de não ter de regenerar o parser ou qualquer uma das ferramentas relacionadas depois da definição de uma linguagem combinada.

MontiCore é uma workbench de linguagens de pleno direito para o design e realização de linguagens específicas de domínio textual (DSL). Permite a pesquisa de métodos de desenvolvimento de software baseados em modelos que empregam uma variedade de linguagens de DSL e modelação.

Esta ferramenta e as DSL resultantes são usados com sucesso em projetos de pesquisa acadêmica e industrial em vários domínios, como modelação de software automóvel, arquitetura de nuvem e modelação de segurança, robótica baseada em modelos, gestão inteligente de energia, modelação de redes neurais.

A lógica de design da MontiCore é fornecer uma workbench poderosa e eficiente para a criação ágil de DSL, juntamente com sua infraestrutura de acompanhamento, tais como análises, transformações e geradores de código. Apresenta uma arquitetura funcional e altamente extensível que permite personalizar ainda mais o processo de desenvolvimento de DSL em si.

As características mais notáveis de MontiCore são: especificação combinada de sintaxe concreta e abstrata numa gramática livre de contexto; geração personalizável de analisador e árvore de sintaxe abstrata; geração de infraestrutura de análise, incluindo visitantes processamento gramatical adaptável via scripts Groovy interpretados; relatórios de logs e processos configuráveis; Freemarker modelo de motor para a geração de código fácil.

2.3.6 Modeling SDK for Visual Studio

Fazendo uso do software development kit (SDK) de modelação para Visual Studio (Microsoft, 2018), é possível criar poderosas ferramentas de desenvolvimento baseadas em modelos passiveis de serem integradas no Visual Studio. Da mesma forma, também é possível criar uma ou mais definições de modelo e integrá-las sobre a forma de um toolset para ser distribuído.

No coração de MSDK encontra-se a definição de um modelo que é criado para representar os conceitos de uma determinada área de negócios. Este modelo pode então ser complementado com uma variedade de ferramentas, como uma exibição diagramática, a capacidade de gerar código e outros artefactos, comandos para transformar o modelo e a capacidade de interagir com código e outros objetos no Visual Studio.

À medida que o modelo é desenvolvido, é possível combiná-lo com outros modelos e ferramentas para formar um poderoso conjunto de ferramentas que funciona como o centro do processo de desenvolvimento.

O MSDK permite desenvolver um modelo rapidamente na forma de uma linguagem específica do domínio (DSL), fazendo uso de um editor especializado para definir um esquema ou uma sintaxe de abstrata em conjunto com uma notação gráfica.

A partir dessa definição, VMSDK gera: uma implementação do modelo com uma API strongly-typed que corre sobre um armazenamento baseado em transações; um browser baseado em

árvores; um editor gráfico no qual os utilizadores podem editar o modelo ou partes dele; métodos de serialização que gravar os modelos em formato XML legível e funcionalidades para gerar código de programa e outros artefactos usando modelação de texto.

É também possível personalizar e estender todos estes recursos. Sendo que as extensões são integradas de forma que é possível atualizar a definição DSL e regerar os recursos sem destruir as extensões.

Depois de terminar o desenvolvimento de uma DSL, é possível distribui-la como parte de um pacote Visual Studio Integration Extension (VSIX), permitindo que os utilizadores finais trabalham com a DSL no Visual Studio.

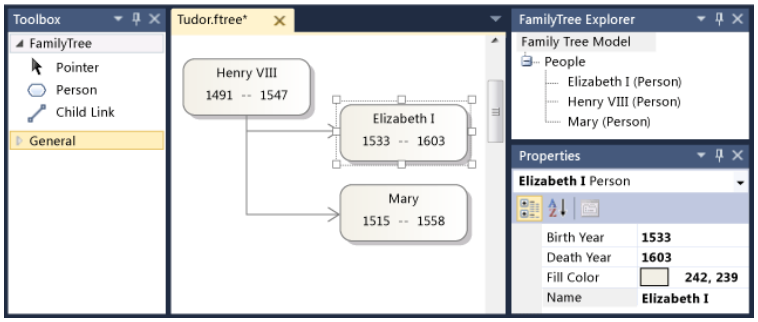

Figura 20 - Exemplo de uma DSL no Visual Studio

A notação é apenas uma parte de uma DSL. Juntamente com a notação, os pacotes VSIX incluem ferramentas que os utilizadores podem utilizar para ajudá-los a editar e gerar material a partir da DSL.

Para definir uma DSL, é necessário criar uma solução do Visual Studio a partir de um modelo. A parte chave da solução é o diagrama de definição DSL, que é armazenado num ficheiro. A definição DSL define as classes e formas suportadas pela DSL. Depois de modificar e adicionar a esses elementos, é possível adicionar código para personalizar a DSL em mais detalhes.

As DSL no Visual Studio são compostas por vários artefactos como por exemplo: uma classe de domínio, uma relação entre elementos do modelo, a forma necessária para exibir elementos dessa classe no diagrama, a ferramenta de elemento que permite aos utilizadores criar elementos.

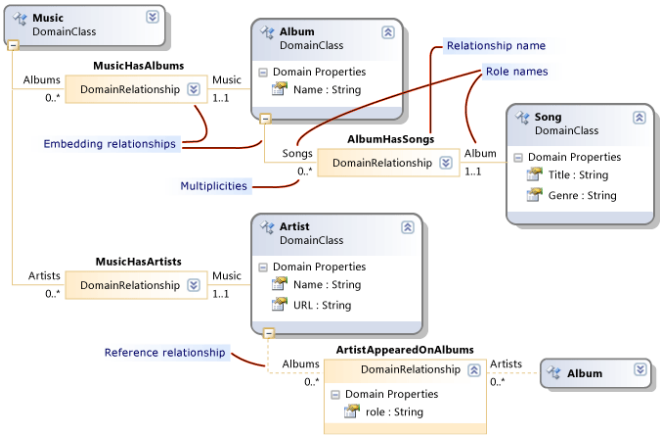

Figura 21 - Exemplo de um modelo de DSL no Visual Studio

Na Figura 21 é apresentado um exemplo de um diagrama de definição de um modelo de uma DSL. Neste diagrama encontram-se referenciados vários tipos de elementos que são suportados pela plataforma, para descrever os conceitos de negócio da DSL e as suas relações.

As classes de domínio representam os conceitos da DSL. As instâncias são elementos do modelo. Neste exemplo em concreto existem classes de domínio chamadas Album e Song.

Cada classe de domínio, exceto a classe raiz deve ser o destino de pelo menos uma relação de incorporação, ou deve herdar de uma classe que é o destino de uma relação de incorporação.

Num modelo, cada elemento é um nó numa única árvore de relacionamentos de incorporação. A origem e o destino de uma relação de incorporação são frequentemente chamados de pai e filho.

A Figura 22 ilustra um exemplo possível de uma DSL criada a partir do modelo definido anteriormente.

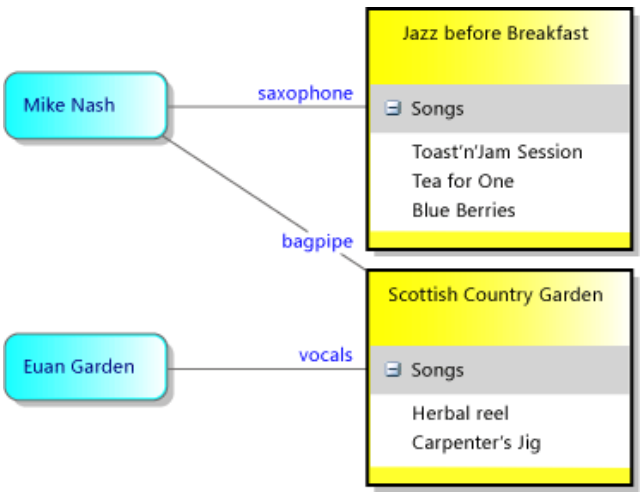

Figura 22 - Exemplo de instância de uma DSL

2.3.7 JetBrains MPS

JetBrains Meta Programming System (MPS) é uma plataforma (*Language Workbench*) open source para desenvolvimento de DSL (JetBrains, 2013).

O MPS evita completamente o formato de texto (JetBrains, 2019). Os programas são sempre representados por uma árvore de sintaxe abstrata (AST), que descreve completamente o código dos programas.

A Figura 23 apresenta uma representação visual das AST e pretende ilustrar o relacionamento entre os nós que a compõem.

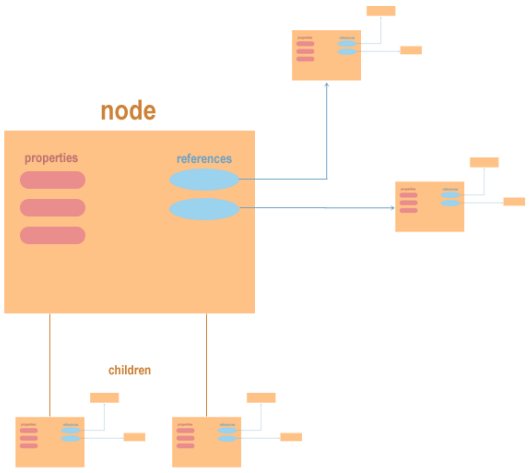

Figura 23 - AST

Cada nó tem um nó pai (exceto para nós raiz), nós filho, propriedades e referências a outros nós. Os nós que não têm um pai, são chamados nós raiz.

O MPS disponibiliza editores projecionais cuja finalidade é permitir visualizar e alterar a AST fonte representado nas mesmas, de forma fácil. O editor pode imitar o comportamento de um editor de texto para DSL com notações textuais, enquanto é suposto comportar-se como um editor de diagramas para linguagens gráficas.

O código fonte é armazenado de forma independe da sua sintaxe concreta (léxico e gramática) que é capturada por um modelo (estrutura da linguagem). Para fins de edição, o código fonte é projetado de acordo com a sintaxe num ambiente gráfico onde os utilizadores usam ações do rato e do teclado, adaptadas à edição projecional. A sintaxe concreta é especificada como parte da definição da linguagem e, portanto, conhecida pelo mecanismo de projeção, ferramentas de modelação geralmente também armazenam informações sobre o layout visual.

3 Análise de valor

Neste capítulo é descrito o processo de planeamento do negócio, realizada no âmbito do projeto dySMS.config, com o propósito de encontrar a melhor forma de combinar os meios disponíveis, para capturar e entregar o máximo de valor possível, às partes interessadas no referido projeto.

Com vista a atingir os propósitos previamente enunciados foram levadas a cabo um conjunto de atividades, cuja sequência e descrição é apresentada nas secções seguintes-

3.1 Partes interessadas

O projeto dySMS.config tem a coordenação científica do Dr. Nuno Silva em colaboração com do Dr. Paulo Maio e é desenvolvido em cooperação e coordenação com o líder do projeto, a empresa DigitalWind.

Os referidos elementos, na qualidade de gestores do projeto, têm uma participação transversal: realizando o planeamento para determinar o que deve ser feito e como fazê-lo; organizando o relacionamento entre as pessoas, a afetação de recursos, a definição das tarefas a realizar e a sua calendarização. Também dirigem a execução das referidas tarefas e controlam os resultados alcançados.

Uma vez apresentadas as duas entidades promotoras do projeto e a suas responsabilidades gerais, importa proceder à identificação das várias partes interessadas que se encontram em cada uma das organizações.

Tabela 1 - Partes interessadas Digital Wind

Parte interessada	Interesses
Administração	Reconvenção do software legado UEBE.Q. Atualização da base tecnológica do referido projeto. Aumento da eficiência nos processos da empresa. Diminuição dos custos de produção. Criação de uma rede de revendedores.
Equipa de desenvolvimento	Adoção de ferramentas e processos para agilizar e consolidar o processo de desenvolvimento. Incorporação de conhecimentos técnicos atualizados.
Equipa de implantação	Adoção de novas técnicas e ferramentas que permitam agilizar e consolidar o processo de implantação, nomeadamente software de gestão do processo de configuração.

Na Tabela 1 encontram-se identificadas as partes interessadas da empresa Digital Wind e os seus interesses relativamente ao projeto.

Tabela 2 - Partes interessadas ISEP

Parte interessada	Interesses
Coordenadores do projeto	Documentação do desenvolvimento e resultados atingidos. Disseminação de conhecimento através de publicações de caracter científico. Geração de novo conhecimento. Publicação de literatura cientifica.
Equipa de I&D	Aprofundar os conhecimentos técnicos em áreas como: o paradigma de desenvolvimento orientado a modelos, o desenvolvimento de DSL e o funcionamento de tecnologias especializadas nas áreas referidas. Adquirir conhecimentos e competências na área da investigação. Publicação de literatura cientifica. Aplicação prática dos conhecimentos adquiridos no mestrado, nomeadamente nas disciplinas de EDOM, ARQSOFT, LABDSOFT e EINOV. Utilização do projeto como base para a realização da dissertação de mestrado.

Na Tabela 2 encontram-se identificadas as partes interessadas do ISEP e os seus interesses relativamente ao projeto.

3.2 Desenvolvimento do conceito do produto

Numa fase inicial do processo de negócio, foi realizado um conjunto de atividades com o intuito de promover a inovação. Esta secção identifica e explica essas atividades, realizadas de acordo com o modelo New Concept Development - NCD (Koen, et al., 2001).

3.2.1 Identificação de oportunidades

A empresa Digital Wind, de acordo com os seus objetivos de negocio que identificavam como prioridade a reconversão do seu sistema de software UEBE.Q, identificou uma oportunidade que poderia querer prosseguir, na politica governamental, no âmbito do programa nacional P2020.

A identificação da referida oportunidade foi conseguida através do uso de técnicas de análise de tendências de tecnologia, análise de inteligência competitiva, planeamento de cenários assim como a analise dos fatores influenciadores, concretamente a politica governamental.

Como resultado deste esforço de analise do contexto atual e de previsão do futuro, foi possível perceber que o P2020 permitiria à empresa a criação de uma parceria com uma universidade, com o intuito de integrar novo conhecimento na empresa. Conhecimento este que seria a base para a inovação necessária para garantir, que a nova geração do software permitiria à empresa ganhar uma vantagem competitiva.

Desta forma também seria possível reagir à ameaça da concorrência, que tinha entrado no mesmo seguimento de mercado mais tarde e dispunha de software tecnologicamente mais evoluído

3.2.2 Análise de oportunidade

A partir do momento que a oportunidade se encontrava identificada, a empresa precisava realizar uma avaliação da mesma, para concluir se vale a pena prosseguir.

Para confirmar se a oportunidade identificada se pode transformar numa oportunidade de negocio, tornou-se necessário recolher informação adicional.

Neste elemento do processo NCD isso implica a realização de avaliações exploratórias, que no caso concreto da empresa Digital Wind em relação à oportunidade em questão, se concluiu que deveriam incindir sobre dois pontos: Parcerias com universidades e Programa da P2020.

O estudo do primeiro fator permitirá estimar a competência e a capacidade de apoio que poderá advir de uma parceria com uma universidade.

No que diz respeito ao programa P2020 importa perceber se este programa pode ser usado como base para a realização da referida parceria, que tipos de apoios se podem esperar, com que prazos funcionam e avaliar o esforço de gestão administrativa e de projeto requeridos por este programa.

Outro fator fundamental que é necessário começar a avaliar, é o financiamento necessário, tanto para a realização de um eventual projeto como para o trabalho adicional que será necessário no processo NCD.

No entanto, apesar de todo o esforço, importa perceber que nesta fase permanecerá um grau de incerteza significativo, pois importa manter os custos controlados, dada a natureza muito precoce do processo.

Nesse elemento, foram usados os mesmos métodos, ferramentas e técnicas usadas para determinar as oportunidades futuras (análise da concorrência, das tendências tecnológicas e dos fatores influenciadores), mas consideravelmente mais recursos foram gastos, fornecendo mais detalhes sobre a adequação e a atratividade da oportunidade selecionada.

Concretamente importa realçar as reuniões que foram realizadas com o ISEP e com a empresa de consultoria, para recolha de informação sobre as parcerias e sobre o P2020.

Esta reuniões tiveram que ser complementadas pelos esforços realizados por um grupo de discussão, no ceio da Digital Wind, que visaram a análise das informações recolhidas.

3.2.3 Geração e enriquecimento da ideia

Como a necessidade de upgrade do software existente, já tinha sido reconhecida pela empresa UEBE.Q, mesmo antes da identificação da oportunidade atual, o elemento de enriquecimento da ideia já se encontrava a decorrer na empresa a algum tempo.

Já tinham sido levados a cabo contatos diretos com os clientes e utilizadores do software, tinham sido auscultadas as opiniões dos elementos dos vários departamentos da empresa (técnico, desenvolvimento, marketing, etc.) e já tinham sido realizadas varias reuniões da direção onde foi abordado o assunto e foram sendo compilados e sistematizados os vários contributos:

4) Necessidade de adoção de novas metodologias de produção do software.

5) Um meio para simplificar as operações de desenvolvimento, implantação e configuração. Acelerá-las e reduzir os seus custos

6) Uma nova abordagem de vendas, através do recurso a parceiros, que implicava que o processo de configuração e personalização do software fosse automatizado.

7) Configuração e personalização, que poderia inclusive vir a ser considerado um novo serviço.

No entanto agora foi possível contar com a colaboração da outras empresas e instituições para aumentar esta atividade, nomeadamente o ISEP e a empresa de consultoria.

Neste elemento foram usadas técnicas de criatividade e brainstorming no decorrer dos contactos com os clientes e auscultação dos colaboradores.

Foi criado um banco de ideias e embora não tenha sido criado um cargo formal para alguém coordenar e gerir a recolha das mesmas este trabalho foi realizado em várias reuniões.

3.2.4 Seleção de ideias

O processo de seleção de ideias envolveu uma série iterativa de atividades que incluíram várias passagens pelo elemento de geração e enriquecimento da ideia, seguidas por reuniões ou discussões informais de avaliação do seu mérito e custos das ideias existentes, consultas aos eventuais novos parceiros (ISEP e empresa de consultoria) e novas diretivas do motor (direção).

Neste elemento foram usadas metodologias de portfólio baseadas em vários fatores: financeiro, probabilidade de sucesso técnico, probabilidade de sucesso comercial, mais valias que podem advir da ideia, enquadramento na estratégica da empresa, etc.

3.2.5 Definição de conceito

No elemento final do modelo NCD, a empresa Digital Wind, procedeu à compilação e sistematização das informações obtidas durante os elementos anteriores (tanto qualitativas como quantitativas) e confrontou a informação recolhida com um conjunto de critérios compilados pela direção.

Estes critérios incluíam âmbitos para variáveis como: tamanho da oportunidade, impacto financeiro na empresa, importância e urgência expressa pelos clientes (mercado), grau de enquadramento do conceito com estratégia da empresa e quantidade e dimensão dos fatores de risco comerciais e técnicos.

Também foi criado um esboço de plano de projeto, incluindo os recursos e o tempo estimados.

Embora o grau de formalidade dos métodos e técnicas utilizados não tenha sido muito elevado, foi feito um esforço no sentido de desenvolver critérios objetivos para suportar a decisão de aprovar o conceito, assim como para a criação de um relatório agregador da informação recolhida que fosse abrangente, estruturado e objetivo.

3.3 Conceito de negócio e análise de valor

Escolher a combinação de meios para satisfazer da melhor forma os interesses das partes interessadas é criar um modelo de negócio.

O artefacto adotado para resumir o modelo de negócio subjacente à realização do projeto atual foi o CANVAS (Osterwalder & Pigneur, 2010), pois é uma forma sistematizada, curta, sintética. gráfica e visual.

3.3.1 Análise SOWT

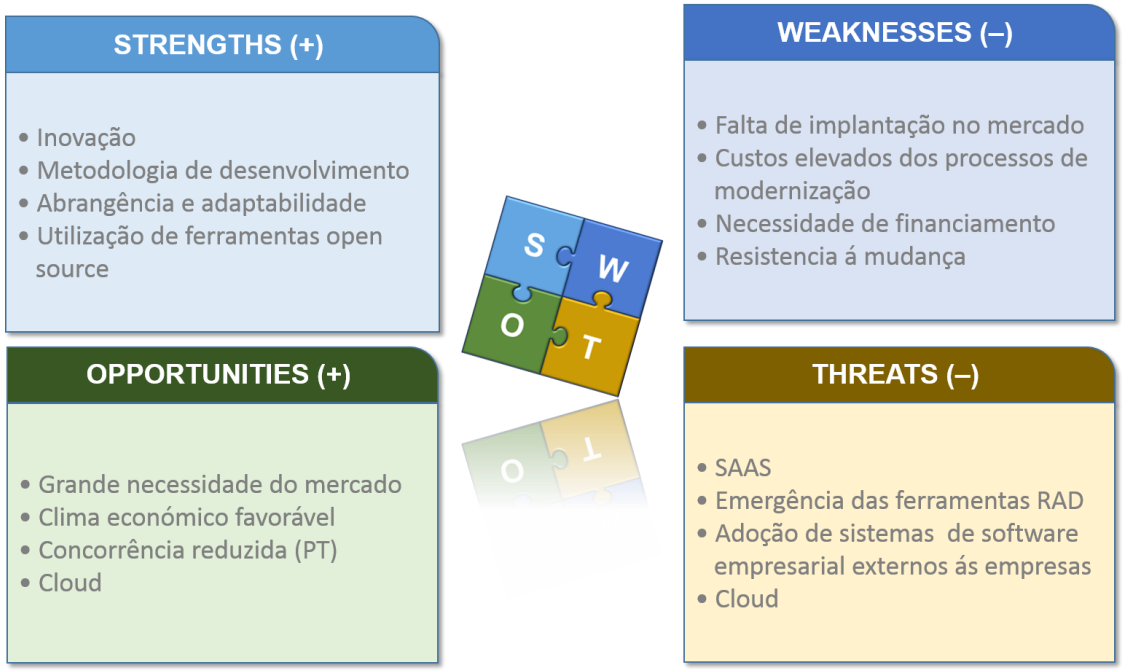

Figura 24 - Análise SWOT

3.3.1.1 Pontos fortes

O sistema dySMS introduz claramente uma forma inovadora de resolver os problemas da modernização dos sistemas legados, através da modelização dos processos de negócio e dos GUI, do uso de um motor de regras para gerir permissões, da utilização de um servidor de buscas de alta performance e de um ORM [2]que permitem a manipulação de forma fácil de dados, cuja vida útil ultrapassa a dos sistemas que os produziram.

A utilização da metodologia de desenvolvimento orientada a modelos permite a criação de modelos representativos do sistema legado, que podem ser transformados de forma automática em código. Desta forma é possível assegurar ao cliente que no futuro, quando for preciso adaptar o sistema a novas tecnologias, apenas será necessário criar uma nova transformação automática para os modelos existentes.

3.3.1.2 Oportunidades

De uma forma simplista é possível considerar a informática na cloud como sendo o fornecimento de serviços informáticos (servidores, armazenamento, bases de dados, rede, software, análises, entre outros) através da Internet ("a cloud"). A disseminação do uso destes serviços proporciona uma oportunidade clara para um sistema com as características do atual, pois oferece um ecossistema rico em funcionalidades, que facilitam a realização de muitas das tarefas subjacentes à modernização dos sistemas legados. Por outro lado, as empresas que fornecem os referidos serviços possuem, elas próprias, ferramentas cuja área de ação se pode sobrepor à do projeto atual.

Embora o custo do próprio sistema não seja muito elevado, o processo de modernização de sistemas legados pode representar custos substanciais para as empresas.

O projeto atual, que tem com finalidade o suporte à modernização de sistemas legados, irá sempre ter de lutar com a aversão natural à mudança, que irá encontrar nos clientes.

Nos cenários tipo, para os quais este projeto foi concebido, existe sempre a possibilidade de o cliente optar por descartar completamente o sistema antigo e apostar no desenvolvimento de um novo ou na aquisição de um sistema padrão, oferecido por empresas de desenvolvimento de software. No

[2] O mapeamento objeto-relacional é uma técnica para converter dados entre sistemas de tipos incompatíveis usando linguagens de programação orientadas a objetos.

primeiro caso as facilidades oferecidas pelas ferramentas RAD[3] podem-se revelar um fator decisivo, ao passo que na segunda hipótese o surgimento de modelos como o SAAS[4] aliado à grande flexibilidade soluções oferecidas podem fazer pender a decisão para a aquisição de um sistema externo.

XXX

[3] Termo usado para definir ferramentas que estimulam o design visual da interface gráfica, e alta reutilização de componentes, de forma a produzir uma aplicação no menor tempo possível

[4] Modelo de distribuição e comercialização de software onde o fornecedor do software se responsabiliza por toda a estrutura necessária ao funcionamento do sistema (servidores, conectividade, cuidados com segurança da informação) e o cliente utiliza o software via internet, pagando um valor pelo serviço oferto

3.3.2 CANVAS

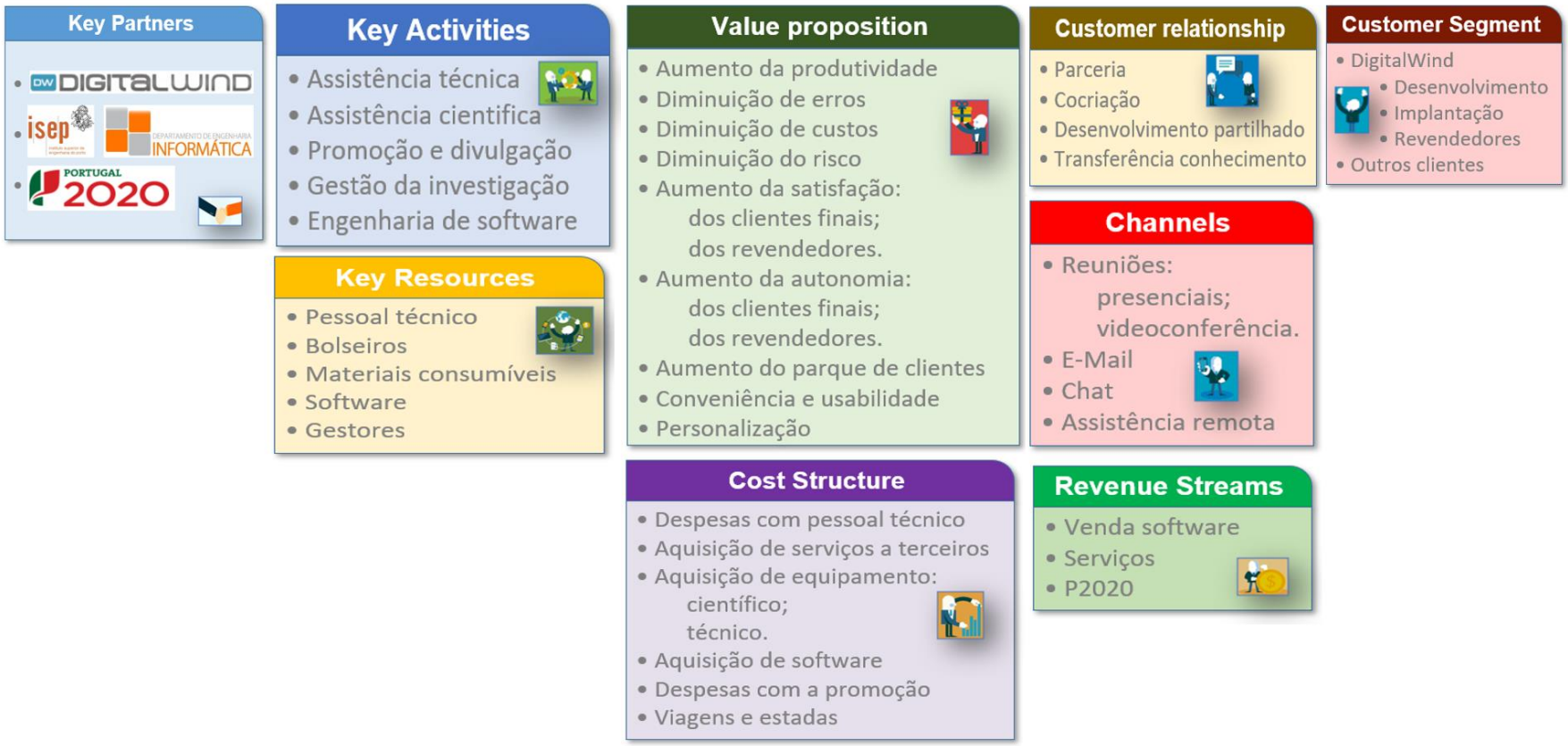

Figura 25 - Canvas

3.3.2.1 Parceiros chave

O projeto dySMS.config é parte integrante de um projeto em copromoção no âmbito do programa nacional P2020, designado dySMS. Este projeto de investigação e desenvolvimento foi proposto pela empresa DigitalWind em parceria com o ISEP.

3.3.2.2 Atividades chave

O projeto dySMS.config é composto essencialmente por dois tipos de atividades: investigação e engenharia de software.

No decorrer deste projeto é importante realizar atividades de investigação no âmbito da análise de problemas, pesquisa de informação, experimentação e avaliação complementadas pela produção de artefactos que documentem os resultados da investigação.

No que diz respeito à engenharia de software, as atividades a realizar são as atividades típicas deste processo, nomeadamente: levantamento de requisitos, design, implementação. teste, implantação e gestão do projeto.

3.3.2.3 Recursos chave

Tendo em conta as propostas de valor, o tipo de relacionamento entre os parceiros e os canais que suportam o referido relacionamento foram identificados vários perfis de recursos humanos, recursos físicos e financeiros que se coadunam com as atividades a realizar.

No que diz respeito a recursos humanos será necessário pessoal técnico, incluindo bolseiros contratados, dedicado a atividades de I&D, formação, assistência técnica e atividades de engenharia de software. Também é necessário contemplar pessoal para realizar gestão da investigação, desenvolvimento e inovação certificado segundo a NP 4457:2007.

Além dos recursos humanos também é necessário contemplar o software, materiais consumíveis e componentes necessárias para as atividades de I&D, engenharia de software ou experimentais e ou de demonstração.

Quanto aos recursos financeiros a afetar ao projeto, estes encontram-se cobertos na totalidade na subseção “Estrutura de custos”, pois a natureza do projeto requer um modelo de negócio cujos recursos financeiros e a estrutura de custos se sobrepõe.

3.3.2.4 Proposta de valor

Este projeto tem especial valor para as equipas de desenvolvimento e de implantação do sistema UEBE.Q. As referidas equipas irão obter um aumento obvio de produtividade, pois o software em questão acaba com a necessidade de duplicação de informações de configuração ao mesmo tempo que automatiza as tarefas nesta área.

Também garante uma diminuição drástica no número de erros, uma vez que os editores oferecidos pela ferramenta além de utilizarem uma linguagem mais próxima dos conceitos de

negócio, também oferecem funcionalidade de completar a introdução de dados e validação dos mesmos.

Este aumento de produtividade e diminuição de possibilidades de erros, nos departamentos nucleares da empresa, têm um impacto direto na performance dos outros departamentos que por sua vez leva a uma clara diminuição de custos e aumento de produtividade da globalidade da empresa.

Outro fator de grande importância e com enorme valor para a DigitalWind é o facto de que a elevação do nível de abstração que a ferramenta aporta ao processo de configuração, irá permitir que estas tarefas possam vir a ser desempenhadas por operadores com baixos conhecimentos técnicos.

Desta forma é possível a empresa expandir o seu negócio através das transferências das referidas tarefas de configuração para os seus revendedores, o que diminui os custos da DigitalWind e promove um aumento de autonomia dos revendedores.

Uma vez que estes últimos se encontram mais próximos do cliente final e têm um maior conhecimento das necessidades deste. Consequentemente será possível prestar um melhor serviço e o aumentar o grau de satisfação dos clientes.

Os clientes finais passam a dispor de um software mais adaptado às suas necessidades e eventualmente poderão eles próprios proceder à configuração do software.

O dySMS.config pode ser vendido/licenciado pela Digitl Wind como um módulo do UEBE.Q e isto também pode refletir-se em valor.

3.3.2.5 Relacionamento com o cliente

O desenvolvimento do projeto dySMS.config funciona numa relação de parceria entre a empresa DigitalWind e o ISEP.

O processo de criação do produto é partilhado pelas duas entidades nas áreas de planeamento, definição de requisitos, organização dos recursos humanos e outros e calendarização. No que diz respeito aos processos de engenharia de software as tarefas são da responsabilidade dos elementos do ISEP.

Embora o desenvolvimento do software dySMS.config tenha sido realizado pelo ISEP até a fase atual, a empresa DigitalWind também dispõe de um departamento de desenvolvimento (onde por exemplo foi desenvolvido o componente cliente do sistema) e eventualmente o futuro do projeto dySMS.config também será assegurado pelo referido departamento.

Uma das componentes mais importantes da parceria subjacente ao projeto é a transferência de conhecimento do ISEP, que é uma instituição de ensino superior publica e que encara esta atividade como fazendo parte da sua missão.

3.3.2.6 Fluxos de receita

Dada a natureza do projeto atual e o contexto do seu desenvolvimento, não é possível falar em receitas diretas imediatas, no sentido tradicional do termo. As receitas previstas têm de ser enquadradas no contexto mais alargado do projeto dySMS, que representa a nova geração do software a ser comercializado pela empresa DigitalWind.

Tendo em conta este âmbito mais alargado, os fluxos de receitas advêm da comercialização do referido software e das atividades complementares associadas ao mesmo, como por exemplo a assistência técnica.

Como foi previamente explicado o projeto dySMS.config é parte integrante de um projeto em copromoção no âmbito do programa nacional P2020.

O programa P2020 (Republica portuguesa, 2014) é um acordo de parceria adotado entre Portugal e a Comissão Europeia, que reúne a atuação dos 5 Fundos Europeus Estruturais e de Investimento - FEDER, Fundo de Coesão, FSE, FEADER e FEAMP - no qual se definem os princípios de programação que consagram a política de desenvolvimento económico, social e territorial para promover, em Portugal, entre 2014 e 2020.

No contexto deste programa existe o sistema de incentivos "Investigação e Desenvolvimento Tecnológico" (ANI – Agência Nacional de Inovação, 2014), que prevê apoiar projetos de empresas em copromoção com outras empresas ou restantes entidades do Sistema de I&I, alinhados com os domínios prioritários da Estratégia de Investigação e Inovação para uma Especialização Inteligente, que visem, designadamente através da realização de atividades de investigação industrial e desenvolvimento experimental, o reforço da sua competitividade e inserção internacional.

O financiamento do projeto é garantido por fundos próprios da empresa Digital Wind, associados ao sistema de incentivos previamente descrito.

Importa ainda acrescentar que existem perspetivas de o projeto dySMS.config poder vir a ser emancipado e transformar-se num produto independente. Caso estas perspetivas se confirmem o projeto atual será enquadrado no portfolio da Digital Wind e como tal irá gerar receitas diretas, de acordo com o modelo de negócio da referida empresa.

3.3.2.7 Segmento de clientes

O projeto dySMS.config foi produzido especificamente como um componente para o sistema dySMS que suporta a reconversão do software legado UEBE.Q. Nesta medida o segmento de mercado alvo do produto é constituído exclusivamente pelo cliente DigitalWind.

Como foi dito anteriormente dySMS.config tem como cliente alvo apenas a empresa DigitalWind, no entanto é possível prever a expansão para novos clientes de forma direta, pois as funcionalidades de manipulação num IDE, de ficheiros de configuração hibernate reveste-se claramente de grande utilidade.

Além do segmento de potencias clientes diretos identificados no paragrafo anterior, também é possível prever o interesse neste produto por parte de empresas com necessidades de um componente de gestão de configuração, para os seus sistemas de software.

4 Requisitos

O processo de elicitação dos requisitos funcionais foi realizado com base no modelo Use-Case 2.0 (secção 2.2.10) Nesta abordagem, a análise de um conjunto de cenários ilustrativos da utilização do sistema, permite identificar as várias funcionalidades a serem suportadas pelo sistema, perceber o seu encadeamento e a forma como os perfis de operadores (atores) fazem uso das mesmas.

No processo de levantamento de requisitos foram utilizadas várias técnicas das quais se podem destacar: workshops e análise de documentos. Na secção A.A.2 é possível consultar um exemplo de documento criado nesta fase, com o objetivo de documentar os resultados alcançados com a aplicação das referidas técnicas. No entanto, a principal fonte de informação usada para a compreensão dos requisitos foram os ficheiros de configuração já existentes.

Á partida o domínio da aplicação era definido pela sintaxe das várias linguagens usadas nos ficheiros de configuração. A análise dos referidos ficheiros permitiu identificar, de forma concreta, as partes das linguagens realmente necessárias. Permitiu também, compreender que as construções sintáticas em uso poderiam ser reduzidas mantendo a expressividade.

A geração de ficheiros equivalentes aos ficheiros preexistentes, demonstra inequivocamente se o sistema faz o que deve fazer e define o que significa implementar com sucesso cada um dos casos de uso.

Com base no modelo Use-Case 2.0 e na estrutura fornecida pelo padrão FURPS+ (Grady, 1992), este capitulo encontra-se divido em três secções que apresentam a informação obtida no arrolamento e caracterização dos perfis de utilizador, no levantamento de requisitos funcionais e na identificação das considerações não-funcionais ou de qualidade.

4.1 Perfis de utilizadores

Foram identificados dois perfis de utilizadores que devem ser suportados pelo software, com distintos níveis de acesso e funcionalidades (cf. **Erro! Fonte de referência não encontrada.**).

Tabela 3 - Perfis de operadores

Perfil	**Principais características**
Configurador Digital Wind	Este operador é responsável por criar todos objetos, que definem a configuração padrão de cada uma das versões do software UEBE.Q. Pode criar, alterar ou remover qualquer instância dos objetos existente no sistema.
Configurador Revendedor	Estes operadores recebem as versões padrão dos objetos que definem a configuração do sistema, fornecidas pela Digital Wind, e devem proceder à sua instalação no sistema.

	Pode estender a informação padrão através da criação de novos conceitos e mapeamentos. Apenas pode remover objetos da sua autoria. Apenas pode fazer alterações aos objetos padrão por via da sua extensão.

4.2 Funcionalidades

Esta seção começa por apresentar o resultado da elicitação de requisitos funcionais específicos do domínio de negócio, através do recurso a diagramas de caso de uso e a respetiva descrição.

Posteriormente são identificados os requisitos funcionais não específicos de domínio, que devido à sua natureza padronizada, não requerem grandes explicações e, portanto, apenas se procede à sua enumeração acompanhada de uma breve descrição.

4.2.1 Funcionalidades especificas de domínio

Na Figura 26 é apresentado o diagrama de casos de uso do sistema, onde os requisitos são apresentados sobre a forma de casos de uso oferecendo uma visão global do sistema a ser construído e dos atores que com ele interagem.

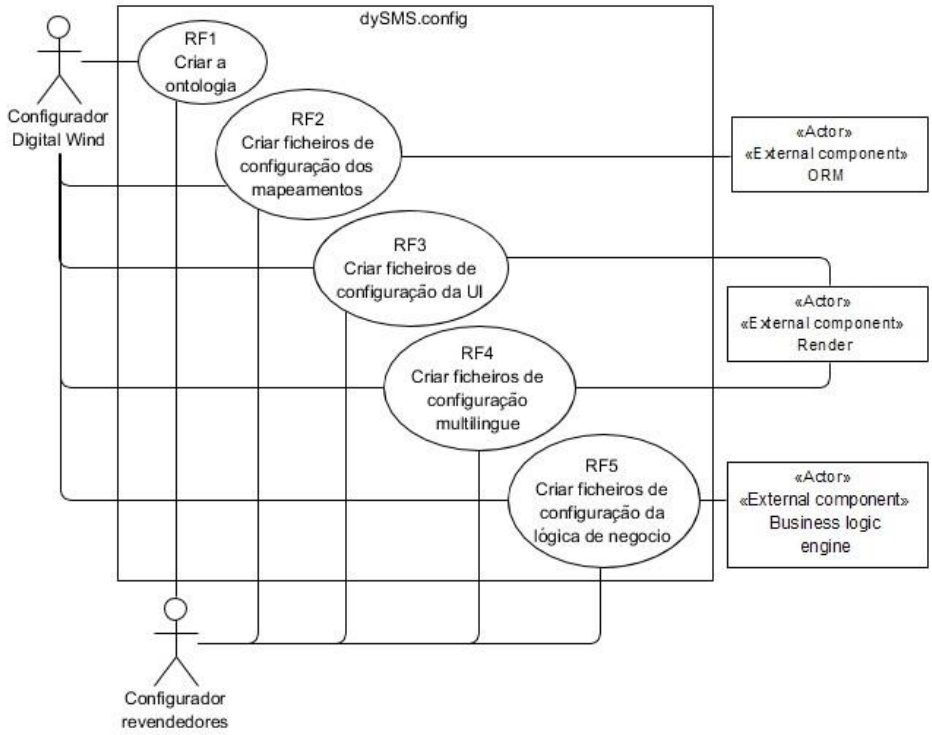

Figura 26 - Diagrama de casos de uso

A designação dos casos de uso foi escolhida de forma a permitir relacionar cada um deles com os objetivos específicos do projeto (secção 1.3), de forma quase direta. Importa esclarecer que

além da criação de novos elementos, deverá também ser possível alterar e remover elementos existentes.

Os últimos quatro casos de uso (RF2 a RF5) têm como pré-requisito o RF1, pois todos eles irão utilizar a informação da ontologia. Desta forma a informação é centralizada e é eliminada a redundância. A ontologia captura o processo e a linguagem do negocio e esconde as estruturas dos dados (técnicas, baixo nível e normalizadas), elevando o nível de abstração na criação dos ficheiros de configuração.

4.2.1.1 RF1 – Criar ontologia

Uma ontologia é um modelo de dados que representa um conjunto de conceitos dentro de um domínio e os relacionamentos entre estes (Ling & M. Tamer, 2009). Este requisito visa a criação uma ontologia, que estruture a informação da fonte de dados do programa UEBE.Q de uma forma mais adaptada ao domínio da aplicação.

Como é possível confirmar através da consulta da Tabela 4, o fluxo básico deste caso de uso é bastante simples e pode ser resumido como: criação, pela ordem correta, de cada um dos elementos que compõe a ontologia.

Tabela 4 - RF1 – Criar a ontologia

Fluxo básico	**Fluxos alternativos**
1. Criar tipos de dados 2. Criar conceitos 3. Criar propriedades dos conceitos 4. Criar ações dos conceitos 5. Criar relações entre os conceitos	

Os tipos de dados, que são usados para classificar as propriedades dos conceitos, existem independentemente dos conceitos, enquanto que todos os outros elementos são partes integrantes de cada conceito.

4.2.1.2 RF2 – Criar ficheiro de configuração dos mapeamentos

Este requisito, visa a definição de regras de rastreio entre a estrutura da ontologia e a estrutura da fonte de dados.

No lado esquerdo da Tabela 5, são enumerados os passos necessários para criar relações entre a fonte de dados e a ontologia. Estas relações são registadas, sobre a forma de ficheiros configuração. A importação dos mesmos pode ser realizada quando o dySMS.config é instalado pela primeira vez ou na eventualidade de estes terem sido alterados manualmente.

Tabela 5 - RF2 Criar ficheiro de configuração dos mapeamentos

Fluxo básico	**Fluxos alternativos**
1. Importar a estrutura da fonte de dados 2. Criar mapeamentos entre a fonte de dados e a ontologia 3. Gerar os ficheiros de configuração	A1. Importar ficheiros de configuração

4. Implantar o ficheiros de configuração no servidor

Para garantir que nos mapeamentos, não existem erros nas referências feitas aos vários elementos da estrutura da fonte de dados e para ser possível ao sistema oferecer ajuda na sua identificação, a designação de todos estes elementos deve existir num catalogo dentro do sistema. A criação do referido catalogo deve ser feita de forma automatizada a partir da meta-informação da fonte de dados. Depois desta informação ter sido recolhida, o dySMS.config pode funcionar de forma independente, apenas requer acesso á fonte de dados quando a estrutura da mesma sofrer alterações e a importação tiver que ser repetida.

Os mapeamentos devem permitir representar chaves primárias simples e complexas e relações com cardinalidades do tipo: um-para-um, um-para muitos, muitos-para-um e muitos-para-muitos.

Os processos de geração e transferência de ficheiros de configuração para o servidor devem ser automatizados. Antes de sobrepor os ficheiros existentes deve ser feita uma cópia de segurança.

4.2.1.3 RF3 – Criar ficheiro de configuração da UI

O componente UI permite recolher informação sobre a estrutura de cada UI para cada conceito/ação da ontologia. Esta informação permite gerar um ficheiro JSON que o componente cliente da aplicação UEBE.Q, consome em tempo de execução e usa para gerar o UI de cada ação de um determinado conceito de negócio.

Na Tabela 6 encontra-se a delineado o fluxo de execução para a geração de ficheiros de configuração da interface de utilizador da aplicação.

Tabela 6 - RF3 - Criar ficheiro de configuração da UI

Fluxo básico	**Fluxos alternativos**
1. Criar uma definição de ambiente de desenvolvimento 2. Criar as definições de widgets para o ambiente de desenvolvimento 3. Criar os formatos das UI 2. Gerar os ficheiros de configuração 3. Implantar o ficheiros de configuração no servidor	A1. Importar ficheiro de configuração

Os tipos de widgets (textbox, checkbox, radiobutton, etc.), disponíveis para manipulação de dados no ecrã, variam de acordo com o ambiente de desenvolvimento. O sistema de configuração dySMS.config deve estar preparado para se adaptar a vários tipos de ambientes de desenvolvimento.

Foi possível identificar dois tipos de UI: formulários para processamento de uma única entidade e formulários para processamento de um conjunto de entidades do mesmo tipo. No caso dos

formulários do primeiro tipo deve ser possível distribuir as propriedades do conceito por vários tabuladores.

4.2.1.4 RF4 – Criar ficheiro de configuração multilingue

Este requisito visa, facilitar o trabalho de tradução da aplicação para diferentes línguas. Embora existam muitas ferramentas que suportam este tipo de trabalho, a estruturação prévia da informação sobre a forma de uma ontologia, providencia um agregado de todos os termos a serem traduzidos, que facilita de forma significativa o trabalho de tradução. Pois, tendo por base o referido agregado de termos, torna-se possível capturar as suas traduções e gerar automaticamente os ficheiros de recursos para a ferramenta de tradução a ser usada.

Tabela 7 - RF4 - Criar ficheiro de configuração multilingue

Fluxo básico	**Fluxos alternativos**
1. Criar a declaração de uma língua	A1. Importar ficheiro de configuração
2. Criar um recurso de tradução	
3. Importar termos a serem traduzidos	
4. Inserir traduções	
5. Gerar os ficheiro de configuração	
6. Implantar ficheiro de configuração no servidor	

Na primeira coluna da Tabela 7 são descriminados os passos que descrevem a funcionalidade, sendo que através do fluxo A1 é possível automatizar a realização dos quatro primeiros passos, caso já existam ficheiros de configuração.

4.2.1.5 RF5 – Criar ficheiro de configuração da lógica de negócio

Este requisito identifica e descreve a funcionalidade que permite a definição de regras que, depois de agrupadas em função dos conceitos da ontologia a que se referem, permitem gerar os ficheiros usados como recursos pelo componente de gestão de regras de negócio da aplicação dySMS.

Tabela 8 - RF5 Criar ficheiro de configuração da lógica de negocio

Fluxo básico	**Fluxos alternativos**
1. Criar programa para um conceito 2. Criar as regras para o programa 3. Criar as funções para a regra 4. Criar os parâmetros para a função 5. Criar a expressão para cada parâmetro. 6. Gerar os ficheiros de configuração 7. Implantar o ficheiros de configuração no servidor	A1. Importar ficheiro de configuração

Na primeira coluna da Tabela 8 são descriminados os passos que descrevem a funcionalidade, sendo que através do fluxo A1 é possível automatizar a realização dos cinco primeiros passos, caso já existam ficheiros de configuração.

4.2.2 Funcionalidades não especificas de domínio

4.2.2.1 RF6 – Funcionalidades de impressão

Constata-se que é natural que os operadores necessitem imprimir as configurações, para usarem como suporte para reflexões e disclusões relativas às mesmas.

4.2.2.2 RF7 – Funcionalidade de procura

Constata-se que é natural que os operadores necessitem de procurar alguma instância de um dos tipos de objetos ou o conteúdo das mesmas.

O número de instâncias de objetos de configuração aumenta rapidamente e encontrar uma delas pode ser moroso.

4.3 Requisitos não funcionais

Nesta seção são identificados os requisitos que representam os atributos de qualidade de software relevantes para o sistema.

4.3.1 Usabilidade

Requisitos relacionados com as características e consistência dos UI.

4.3.1.1 RNF1 – Edição assistida

A utilização de um mecanismo de configuração implica o manuseamento de um volume grande de dados, num formato pouco amigável por operadores com poucos conhecimentos técnicos.

Para resolver estes problemas, o dySMS.config deve fornecer ao configurador um ambiente de edição da informação de configuração, que: (i) eleve o nível de abstração da linguagem utilizada; (ii) ofereça funcionalidades de apoio ao preenchimento dos dados e (iii) que permitam a validação dos dados inseridos.

4.3.2 Interface

4.3.2.1 RNF2 – Integração com o dySMS

O dySMS.config deve ser capaz de gerar os ficheiros de configuração que são consumidos pelo sistema integrador respeitando as estruturas de cada um deles, respetivamente: XML formato hibernate, ficheiros de configuração de UI em JSON em formato próprio (ver a seguir), ficheiros de regras de acesso a dados em formato DROOLS assim como os ficheiros de tradução multilingue no formato suportado pelo framework Angular (i18n).

O formato dos ficheiros de configuração de UI é um formato não padrão definido especificamente para o respetivo componente do sistema dySMS.

Na **Erro! Fonte de referência não encontrada.**,encontra-se a definição da estrutura para definição de UI para a manipulação de uma coleção de instâncias de um dado conceito.

Tabela 9 - Estrutura configuração UI - Coleção de instâncias

Objetos	Atributos	Descrição
columns	key	Identificação do atributo no conceito com o caminho completo separado por pontos
	order	Qual o ordinal na apresentação em colunas
	allowSort	Booleano com a definição se é possível ordenar a coluna
	type	Qual a representação do campo (ex.: text, date, etc.)
	exportable	Booleano com a definição se a coluna deve ser exportável para CSV, PDF, etc.
defaultSort	column	Identificação com o ordinal de qual a coluna que deve ser usada para fazer a ordenação. O ordinal começa em zero (0)
	type	Desc ou Asc
actions	action	Nome da ação de negócio
	label	Resource key que será utilizada na contextualização
	order	Qual o ordinal na apresentação das ações

Na **Erro! Fonte de referência não encontrada.**, encontra-se a definição da estrutura que permite definir um UI para a edição de uma instância única de um dado conceito.

Tabela 10 - Estrutura de configuração UI - Instância única

Objetos	**Atributos**	**Descrição**
properties	key	Identificação do atributo no conceito com o caminho completo separado por pontos
	order	Qual o ordinal na apresentação em colunas
	type	Qual a representação do campo (ex.: text, date, etc.)
	concept	Elemento de ligação a outro conceito
actions	action	Nome da ação
	label	Resource key que será utilizada na contextualização
	order	Qual o ordinal na apresentação das ações

4.3.3 Implementação

4.3.3.1 RNF3 – Processo de desenvolvimento

O processo de desenvolvimento deve seguir o modelo Iterativo e Incremental adotando os princípios das metodologias ágeis, com o intuito de promover uma prototipagem rápida da aplicação de modo a evidenciar desde cedo as funcionalidades implementadas correspondem aos requisitos previstos inicialmente.

4.3.3.2 RNF4 - Paradigma

Relativamente aos paradigmas de programação escolhidos para o desenvolvimento da aplicação, deve ser adotada a metodologia orientada a modelos dando enfase especial à utilização de DSL.

No que diz respeito ao código complementar usado para desenvolvimento de operações mais complexas e especificas, deve ser adotada uma metodologia orientada a objetos (OO), procurando seguir princípios GRASP e SOLID na definição dos diversos componentes da aplicação e na sua interação.

4.3.3.3 RNF5 - Plataforma

Preferencialmente deve ser adotada uma plataforma tecnológica que proporcione uma base para o processo de desenvolvimento, para não ser necessária a implementação de raiz de todas as funcionalidades padrão inerentes à metodologia adotada.

Tendo em conta os objetivos propostos, o contexto do projeto dySMS.config e a análise tecnológica, foi realizada uma avaliação às várias ferramentas identificadas. Esta análise permitiu concluir que a plataforma mais indicada para suportar o desenvolvimento do projeto seria a JetBrains MPS.

Na base da decisão tomada encontram-se vários critérios dos quais importa salientar os que são descritos de seguida:

1) Qualidade e maturidade das ferramentas: foram descartadas ferramentas académicas (SDF, MontiCore) pois não dispunham das funcionalidades, documentação e maturidade necessárias para suportar o desenvolvimento do projeto atual.

2) Racionalidade económica: foram descartadas todas as ferramentas que tivessem custos associados (Intentional plataform, MetaEdit+, Modeling SDK for Visual Studio), visto existirem alternativas com grande qualidade sem custos.

Nesta fase do processo de avaliação restavam o Eclipse Xtext e o JetBrains MPS. O eclipse Xtext foi preterido em função do JetBrains MPS, pois as suas funcionalidades de extensão por herança de linguagens, não foi considerada suficiente. Na prática; seria como programação orientada a objetos com apenas herança e sem delegação ou “traits”.

Outro fator que fez pender a escolha para o MPS foi o facto de este ser especializado em DSL e oferecer uma abordagem projecional para a edição de linguagens.

4.3.3.4 RNF6 – DevOps e Gestão de projeto

Dado que o projeto vai ser desenvolvido por uma única pessoa não se prevê a necessidade de adoção de quaisquer artefactos, ferramentas ou plataformas de suporte a operações de integração continua, distribuição continua ou de gestão de projeto.

5 Design

Este capítulo descreve as partes arquitetonicamente significativas do modelo de design e a decomposição do mesmo em componentes e pacotes. Para cada pacote arquitetonicamente significativo, é apresentada a sua decomposição em classes. Para as classes arquitetonicamente significativas são descritas as suas responsabilidades, bem como as suas relações e os atributos mais importantes.

A transposição dos requisitos para a fase de design foi realizada, em conformidade com o modelo adotado (2.2.10 Use-Case 2.0), através da organização dos mesmos em *Slices*. Na Tabela 11 é apresentado o resultado dessa divisão.

Tabela 11 - Divisão de requisitos em "slices" para o design

Slices	**Requisitos**
S1 Criar ontologia	RF1
S2 Importar a estrutura da fonte de dados	RF2
S3 Criar mapeamentos entre a fonte de dados e a ontologia	RF2
S4 Criar configuração da UI	RF3
S5 Criar configuração da lógica de negócio	RF4
S6 Criar traduções	RF5
S7 Gerar ficheiros de configuração	RF2 a RF5
S8 Implantar o ficheiros de configuração no servidor	RF2 a RF5

O requisito 2 foi dividido em dois *slices* porque cada dum dos fluxos correspondentes requer a definição de modelo de dados próprio (DSL). Os *slices* S7 e S8 representam funcionalidades comuns a vários casos de uso, que a nível de design podem ser agrupadas.

5.1 Visão Geral

No nível mais elevado de abstração o projeto dySMS.config é ele próprio um componente de software.

O componente representativo do projeto foi desenhado tendo em conta os princípios da abordagem DDD. Neste sentido o foco principal neste projeto de software foi o próprio domínio e não os detalhes técnicos.

Desta forma, como se pode ver na Figura 27Figura 27 - Diagrama de componentes - Visão geral, no nível mais elevado de abstração, o sistema começou por ser dividido, a nível logico, em dois componentes.

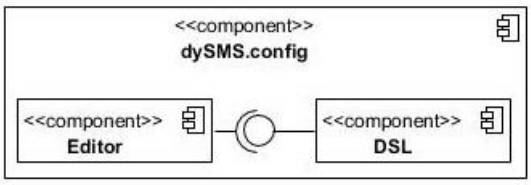

Figura 27 - Diagrama de componentes - Visão geral

O componente DSL pretende enfatizar a importância dada a uma representação adequada e efetiva do domínio do problema, com o intuito de ajudar os programadores e os especialistas do domínio a partilharem os seus conhecimentos, através da representação dos mesmos sobre a forma de modelos (DSL). No que diz respeito ao componente Editor, este tem como objetivo agregar as funcionalidades da UI para manipular os conceitos da DSL, garantindo assim a separação das mesmas do foco principal que é o domínio.

Estes dois componentes, foram por sua vez, divididos em subcomponentes, que são descritos em maior detalhe nas secções seguintes, por forma a diminuir acoplamento e aumentar a coesão, de acordo com as boas práticas de design comumente aceites. (Martin, 2012). A descrição dos estereótipos não padronizados e cores, utilizados nos diagramas, pode ser consultada na secção A.A.1 Catalogo de elementos arquiteturais.

5.2 DSL

Como ilustra a Figura 28, o componente DSL agrega todas as linguagens especificas de domínio utilizadas no sistema (Slices S1 a S5), que podem ser acedidas através de uma interface única e padronizada, implementada no componente Factories.

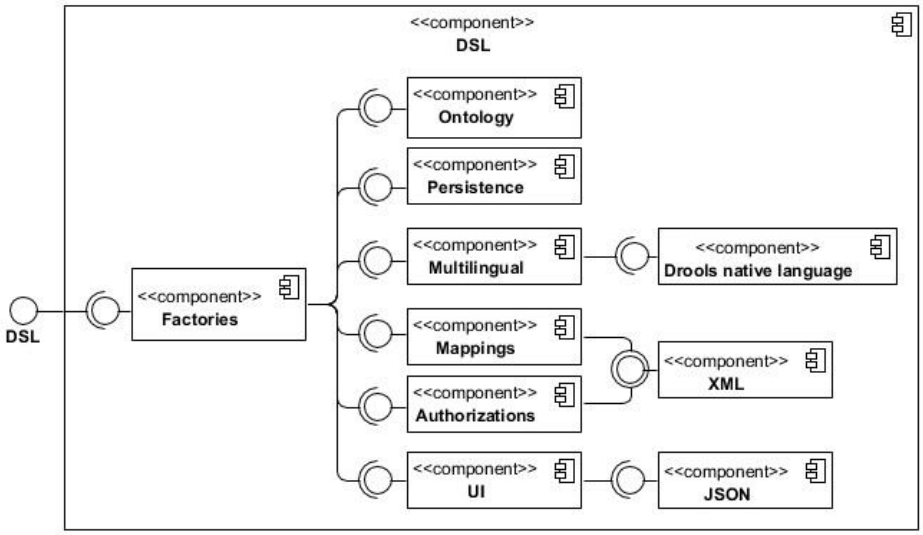

Figura 28 - Diagrama de componentes – DSL

O componente Factories permite encapsular a complexidade inerente à lógica de criação das DSL principais (Ontology, Persistence, Mappings, Multilingual, Authorizations e UI), que modelam os conceitos de negócio e as suas relações. As DSL secundárias (XML, JSON e Drools native language) modelam as várias linguagens usadas nos ficheiros de configuração.

Em conformidade com os princípios da abordagem orientada a modelos, as DSL secundárias permitem intermediar o processo de serialização que é implementado em duas fases, conforme a seguir descrito.

1) Primeiro são realizadas as transformações M2M das DSL principais nas DSL secundárias. Estas transformações permitem estruturar a informação das entidades de domínio de acordo com o formato da linguagem adotada para os respetivos ficheiros de configuração.

3) Seguidamente serão gerados ficheiros de texto através de transformações M2T das DSL secundárias.

De acordo com as considerações teóricas apresentadas na seção 2.2.8 DSL, o componente DSL, além de albergar a estrutura das DSL também é responsável pelas funcionalidades relativas à configuração de editores, definição de restrições e geração de transformações.

A complexidade, relevância e dimensão dos objetos de domínio que constituem o componente atual, justificam uma abordagem detalhada a nível de design, que é apresentada nas seções seguintes.

Relativamente à implementação das funcionalidades complementares aos objetos de domínio (estruturação dos editores, definição de restrições e transformações) não são apresentados pormenores relativos ao design, pois está previsto que a fase de desenvolvimento seja realizada com recurso a um *Language Workbench* que implementa estas funcionalidades de forma nativa.

5.2.1 Ontology

O componente Ontology encontra-se divido em dois subcomponentes: Types e Concepts. Embora estes dois componentes sejam interdependentes, a sua separação permite uma diminuição do acoplamento, pois possibilita que os outros componentes dependam unicamente do Types ou do Concepts e não obrigatoriamente dos dois.

5.2.1.1 Types

A Figura 29 apresenta as classes e relações usadas para modelar as estruturas de dados utilizadas na ontologia: Dados simples, Coleções e Conceitos.

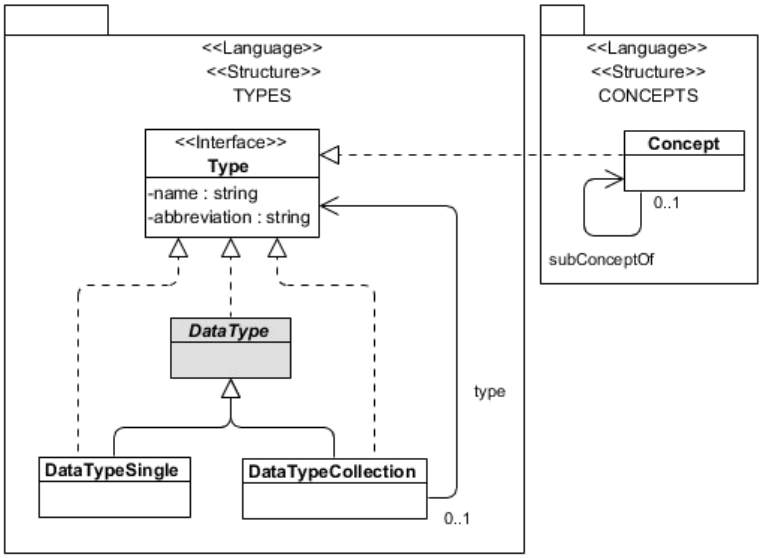

Figura 29 - Diagrama de classes dos Tipos

A classe DataTypeSingle abstrai o conceito de tipo de dados primitivos, como por exemplo string ou integer. Através da classe DataTypeCollection é definida a estrutura de dados necessária para representar conjuntos agregadores de dados simples, como por exemplo Set ou List. O tipo de dados simples agregados pela coleção é identificado pela propriedade type. A classe DataType permite classificar os tipos simples e as coleções como sendo extensões de um conceito único. Desta forma é possível utilizar os dois de forma indiferenciada em situações onde tal é necessário.

A interface Type contém as propriedades comuns a todas as categorias de tipos de dados e cria uma generalização que estende o conceito de tipo para abranger os Concept. A classe Concept representa tipos de dados complexos compostos por atributos e relações. O pacote Concepts, apresentado de forma simplificada, é descrito em detalhe na secção seguinte.

5.2.1.2 Concepts

O diagrama de classes apresentado na Figura 30 ilustra o modelo adotado para estruturar a representação de conceitos. Os conceitos representam classes de entidades do domínio da aplicação. Como implementam a interface Type também eles podem ser usados como tipos de dados.

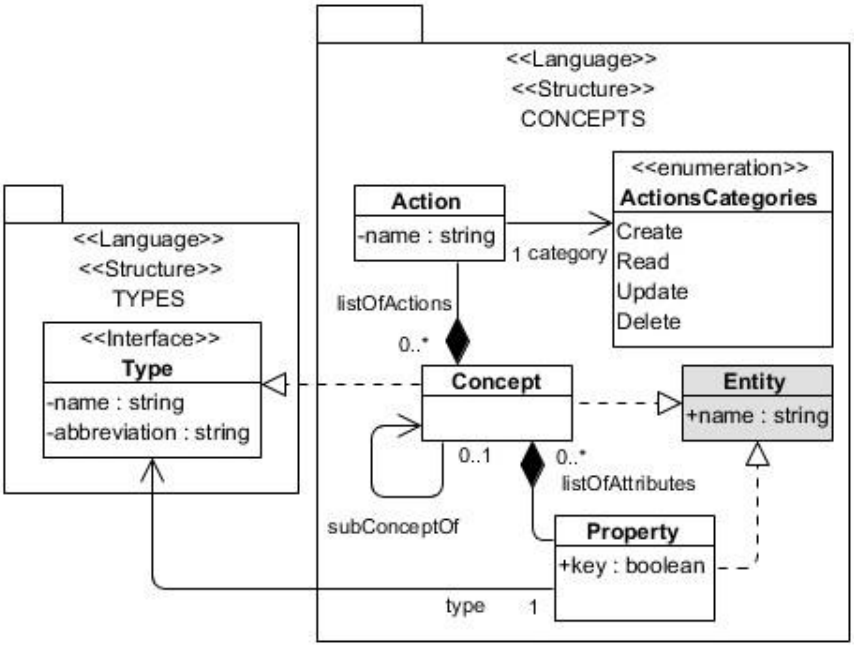

Figura 30 - Diagrama de classes dos Conceitos

Os conceitos (Concept) são compostos por propriedades (Property) e ações (Action) e podem ser extensões de outros conceitos (subConceptOf). Cada uma das Property é identificada pelo seu nome (name) e a informação que contém é definida pelo tipo de dados que a caracteriza (type). Cada uma das instâncias do conceito pode ser identificada univocamente através de uma ou mais propriedades, que são identificadas como sendo a sua chave - key.

As Action representam as ações de negócio que se encontram associadas a um determinado conceito e têm como propósito permitir identificar um contexto concreto para o qual existe uma UI. As Action são categorizadas através das quatro operações básicas utilizadas nas bases de dados relacionais para acrescentar informação semântica necessária à geração dos referidos UI.

Os conceitos e as propriedades são classificados como Entity, para permitir ao componente Multilingual referenciar cada um deles independentemente da sua natureza concreta.

5.2.2 Persistence

O modelo de domínio do componente Persistence, apresentado na Figura 31, representa a estrutura simplificada de uma fonte de dados. Esta informação é utilizada pelo componente Mappings para relacionar a informação contida na fonte de dados, com a sua representação na ontologia definida no componente Ontology.

Embora presentemente o projeto dySMS.config apenas esteja preparado para funcionar com fontes de dados relacionais, é previsível que futuramente seja necessário suportar outros tipos.

Tendo em conta este requisito, foi criada a classe abstrata DataSource que permite abstrair vários tipos de fontes de dados.

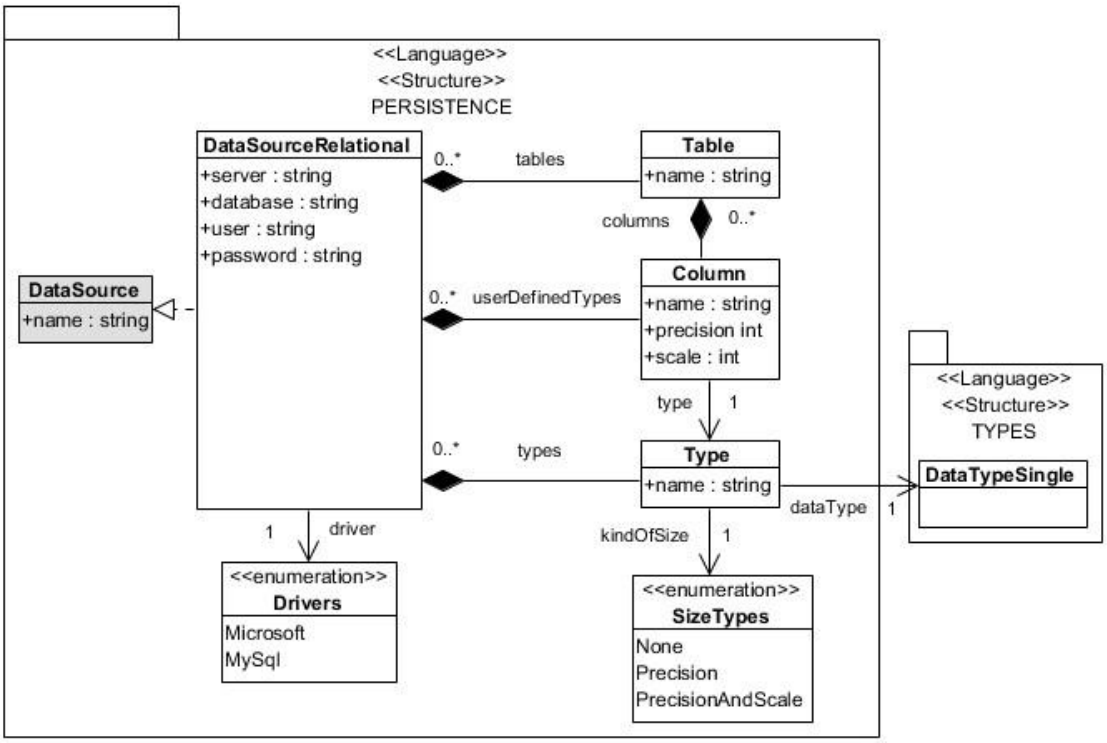

Figura 31 - Diagrama de classes da persistência

A classe DataSourceRelational é o objeto principal, que contém todos os dados necessários para aceder a uma base de dados relacional: o nome do servidor, o nome da base de dados, o utilizador e a respetiva chave de acesso. O campo driver permite definir o tipo de base de dados a que se pretende aceder, por exemplo Microsoft ou MySQL. Agregada na referida classe encontra-se a informação sobre a estrutura dos dados usada na fonte de dados relacional: tipos de dados nativos, tipos de dados personalizados e tabelas com os seus atributos.

Cada tabela da fonte de dados é representada por uma instância da classe Table, que agrega várias instâncias da classe Column. As instâncias da classe Column permitem registar os dados sobre os atributos da tabela. Cada atributo têm um tipo na fonte de dados, que é identificado pela propriedade type.

Cada um dos tipos da base de dados é representado numa instância da classe Type, onde é guardada a designação do tipo (name), a classificação do seu tamanho (kindOfSize) e a correspondência deste com um dos tipos da ontologia (dataType). A classificação do tamanho de um tipo (SizeTypes) indica quais os atributos necessários para caracterizar a dimensão do referido tipo. Existem tipos sem tamanho, com tamanho e com casas decimais

Os tipos de dados personalizados (UserDefinedTypes), são criados na base de dados a partir de um tipo básico, ao qual é atribuído um novo nome e uma precision e scale especificas. Estes

novos tipos pretendem representar conceitos do domínio aplicacional. Dado que a informação necessária para representar UserDefinedTypes é similar á informação relativa aos atributos de uma tabela, foi reaproveitada a classe Columns.

5.2.3 Mappings

A componente Mappings, visa a definição de regras de rastreio entre a estrutura da ontologia e a estrutura da fonte de dados. Na Figura 32 é apresentado o diagrama que modela o domínio deste componente.

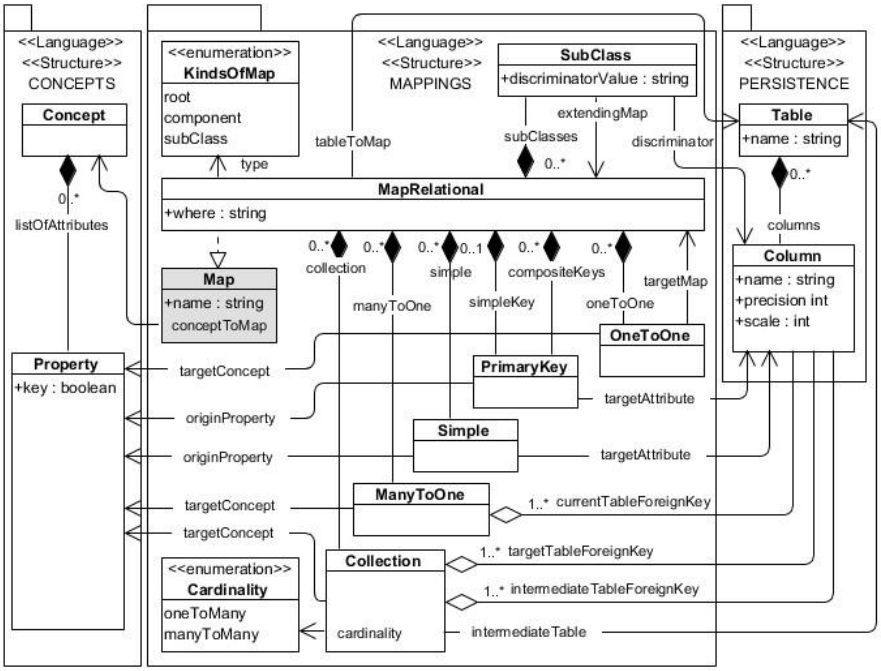

Figura 32 - Diagrama de classes dos mapeamentos

O projeto dySMS.config está preparado para funcionar apenas com fontes de dados relacionais, mas é previsível que futuramente venha a trabalhar com outros tipos. Por esta razão foi criada a classe abstrata Map, que permite agregar um conjunto de regras de mapeamento relativas a um conceito da ontologia (conceptToMap), independentemente do tipo de fonte de dados.

No caso concreto das bases de dados relacionais a classe MapRelacional representa um conjunto de regras de mapeamento de um conceito (conceptToMap) a uma tabela (tableToMap). Como a uma tabela podem corresponder a vários conceitos, podem existir várias instâncias da referida classe para a mesma tabela e conceitos diferentes. A instância principal é identificada como root na propriedade type. Nesta mesma classe, através da propriedade where, é possível filtrar os registos da tabela relevantes para o conceito em questão.

As regras do tipo subClass permitem dividir os registos da tabela atual (tableToMap) em subconjuntos. Esta divisão é realizada com base na classe SubClass cuja propriedade discriminator referencia o atributo da tabela atual, que identifica os subconjuntos. Para cada

valor distinto do referido atributo (discriminatorValue) é criada uma nova instância da classe MapRelacional com o tipo subClass. Esta nova instância é referenciada pela propriedade extendingMap e identifica o conceito que representa o subconjunto de registos da tabela atual.

Os três tipos de regras simpleKey, compositeKey e simple aplicam-se a propriedades do conceptToMap que correspondem diretamente a atributos da tabela tableToMap. Para cada uma destas propriedades, têm que ser criada uma regra do tipo Simple ou PrimaryKey onde é identificado o atributo (targetAttribute) correspondente á propriedade (originProperty). Para distinguir entre chaves primárias com apenas um atributo e chaves primárias com vários atributos, existem as relações simpleKey e compositeKey. Para identificar os atributos que não fazem parte da chave existe a relação simple.

As regras do tipo oneToOne são usadas para dividir os atributos da tabela atual por vários conceitos. Para relacionar o MapRelacional atual com outro conceito, é necessário criar uma instância da classe OneToOne. Nesta instância é identificada a propriedade do conceito atual (targetConcept) e é definida uma referência (targetMap) para o MapRelacional de outro conceito. Embora existam dois conceitos, a tabela é uma só, cujos atributos são divididos em grupos. Cada um dos conceitos contém propriedades que representam os atributos de um dos grupos. O targetMap referenciado nestas regras tem que ser do tipo Component.

No tipo de regras manyToOne permite relacionar o conceito atual com um segundo conceito, que representa uma tabela distinta da atual. A tabela atual contém uma chave estrangeira, que corresponde á chave primária da tabela representada pelo segundo conceito. Na definição de uma regra de mapeamento ManyToOne, deve ser referenciada a propriedade do conceito atual (targetConcept) e todos os atributos da tabela atual que formam a respetiva chave estrangeira (currentTableForeignKey). A tabelas destino, para a qual se pretende criar a relação, é obtida a partir do tipo de dados da propriedade referenciada em targetConcept.

O subtipo de regras collection permite relacionar um conjunto de registos de outra tabela (tabela destino), com um ou mais registos da tabela atual. A classe Colection define a estrutura de dados para este tipo de regras. A propriedade cardinality indica se na tabela atual são considerados vários registos (manyToMany) ou apenas um (oneToMany).

A tabela atual encontra-se identificada no MapRelational atual, pela propriedade tableToMap. A tabela destino é identificada de forma indireta: a referência targetConcept identifica a propriedade do conceito atual, cujo tipo identifica o conceito destino. A partir deste conceito é possível obter a tabela correspondente.

O tipo de cardinalidade oneToMany requerer a existência de uma chave estrangeira na tabela destino, que se relaciona com a chave primária da tabela atual. Esta chave estrangeira é representada pela relação targetTableForeignKey, que pode conter um ou mais referências a atributos (Column) da tabela secundária.

Nas regras de cardinalidade manyToMany, a relação implica a existência de uma tabela intermediária (intermediateTable). A relação targetTableForeignKey identifica os atributos da

tabela destino que se relacionam com a chave primária da tabela intermediaria. A relação intermediateTableForeignKey identifica os atributos da tabela intermediaria que se relacionam com a chave primária da tabela atual.

5.2.4 UI

Na Figura 33 é apresentado o diagrama que modela o domínio deste componente.

Neste momento a aplicação dySMS.config apenas está preparada para gerar ficheiros de configuração de UI, para o ambiente ANGULAR, no entanto está previsto que no futuro seja possível usar a aplicação com outros ambientes de desenvolvimento. Tendo em conta este fator o design da estrutura de dados para o componente atual já comporta a possibilidade de subordinar as configurações dos UI a um determinado ambiente de desenvolvimento. A parte da estrutura de dados com este propósito é composta por duas classes: EnvironmentUI e Widget. A primeira serve para identificar o ambiente de desenvolvimento de cada UI e agrega uma lista dos controlos de ecrã (listOfWidget) suportados pelo ambiente em questão. A segunda classe (Widget) identifica cada um dos referidos controlos de ecrã e indica para cada um deles o tipo de opções adicionais necessárias para o descrever.

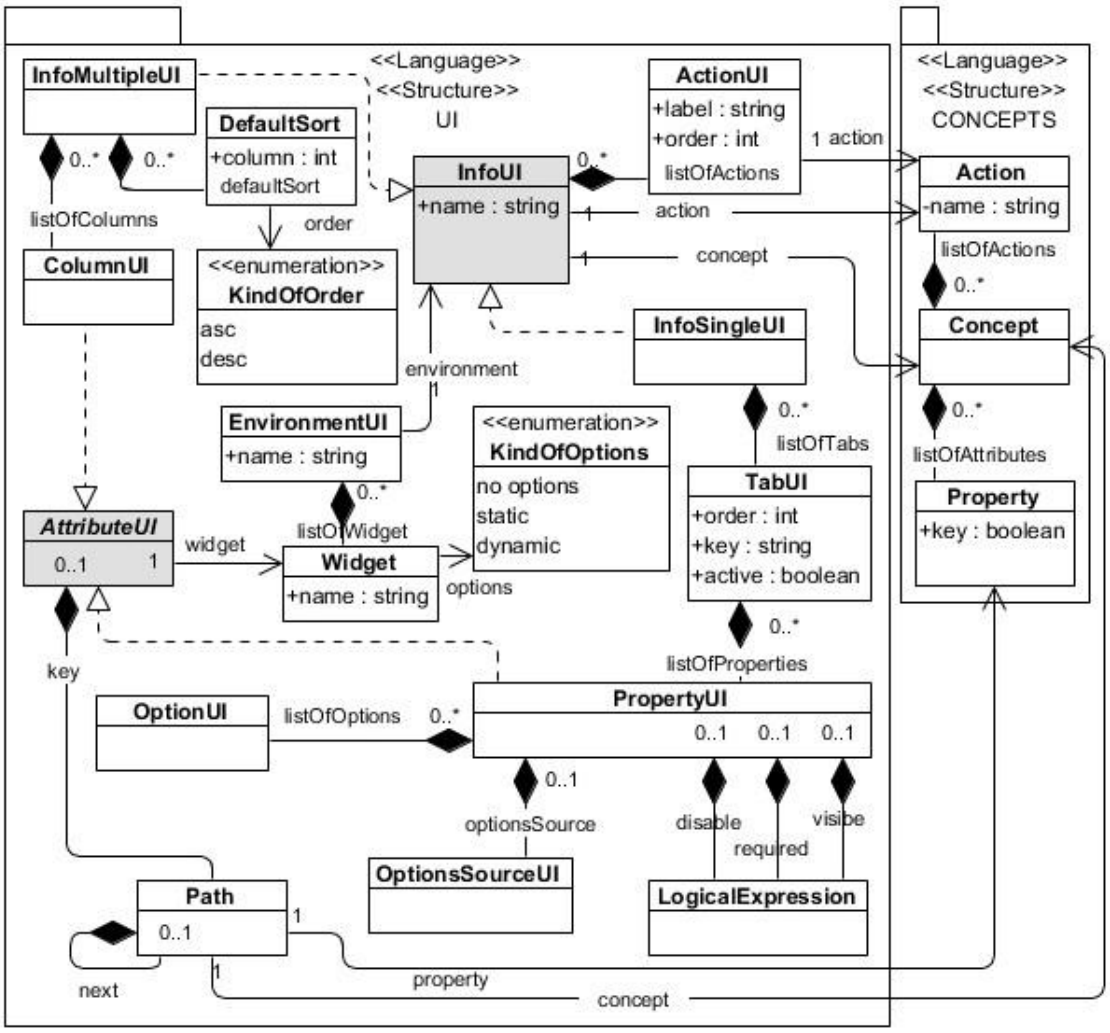

Figura 33 - Diagrama de classes do UI

Durante o levantamento de requisitos foi possível identificar dois tipos de UI: formulários para processamento de uma única entidade (InfoSingleUI) e formulários para processamento de um conjunto de entidades do mesmo tipo (InfoMultipleUI). As duas classes representativas de cada um destes tipos são extensões da superclasse (InfoUI) que contém referencias para um conceito e uma ação de negócio, que definem o contexto de utilização do UI. Estas propriedades são necessárias para todos os tipos de UI pois identificam os dados (concept) a serem editados e a funcionalidade (action) onde a edição se enquadra. Além do contexto de execução esta classe ainda contém uma lista de ações do conceito atual (listOfAction), que podem ser evocadas a partir do UI em questão.

Um UI do tipo infoSingleUI permite distribuir as propriedades do conceito por vários separadores, para proporcionar um agrupamento lógica dos mesmos. Esta separação em separadores distintos é representada pela classe TabUI, que agrega um conjunto de PropertyUI (listOfProperties). Cada instância da classe PropertyUI relaciona uma propriedade do conceito (key) com um widget. Este relacionamento pode ser caracterizado através das propriedades: disable, visible, required, listOfOptions e optionSource. As primeiras três são expressões lógicas

que ativam ou desativam o comportamento que o seu nome identifica. As últimas duas são informações complementares associadas ao tipo de controlo de ecrã escolhido para editar a propriedade em questão. Como foi referido anteriormente, a recolha desta informação adicional esta sujeita à definição options associada ao tipo de controlo de ecrã.

Um UI tipo infoMultipleUI pode ser visto como a definição de um ecrã que contém uma lista de instâncias do conceito em questão. Nesse ecrã o operador pode selecionar uma ou várias instâncias e depois escolher uma a ação ActionUI a aplicar à seleção. A definição de um UI deste tipo agrega uma lista de ColumnUI (listOfColumns). Cada instância de ColumnUI identifica as propriedades do conceito atual a serem apresentadas no ecrã, assim como o tipo de controlo de ecrã que deve ser utilizado para apresentar cada uma das referidas propriedades. Com o intuito de identificar a ordem subjacente à apresentação das várias instâncias do conceito corrente deve ser usada a propriedade defaultSort, que permite indicar para cada coluna, identificada pelo seu número, um tipo de ordenação: ascendente ou descendente.

Tanto as PropertyUI como as ColumnUI são extensões da classe AttributeUI. Esta classe serve o propósito de relacionar uma propriedade de um conceito com um tipo de controlo de ecrã. A definição do controlo de ecrã é linear, mas a definição da propriedade do conceito requer a composição de um caminho. Caminho este, cujo inicio é o conceito atual e se propaga através de vários conceitos e respetivas propriedades, até terminar numa propriedade cujo tipo é um tipo de dados simples. As partes intermédias do referido caminho representam a enumeração de propriedades dos respetivos conceitos cujo tipo de dados é outro conceito. Quando isto se verifica a propriedade na realidade funciona como uma relação entre conceitos, indicando que o conceito origem, além das propriedades para registo de dados simples, também contém referencias a dados complexos, agrupados sobre a forma de um conceito diferente do conceito origem.

5.2.5 Multilingual

A estrutura de dados deste componente, visa capturar as várias traduções para os conceitos, propriedades e a ações existentes na ontologia. Com a informação recolhida é possível gerar, de forma automática, ficheiros de recursos multilingue para serem usados pelo componente cliente da aplicação UEBE.Q no desenho do UI. Na Figura 34 é apresentado o diagrama que modela o domínio deste componente.

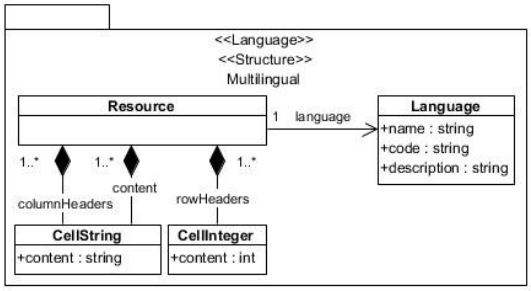

Figura 34 - Diagrama de classes do componente multilingue

Para cada língua que se pretenda fazer uma tradução é necessário começar por criar uma instância da classe Language. Nas instâncias desta classe, além de identificar e descrever a língua em questão, também é necessário definir o código associado à mesma de acordo com o conjunto de normas ISO 639.

Para cada uma das línguas previamente definidas é necessário criar uma instância da classe Resource, que associa á língua em questão a representação de uma tabela de duas colunas. A primeira coluna é preenchida com a designação dos conceitos, propriedades e ações da ontologia. A segunda coluna deve ser preenchida com as traduções, dos termos que se encontram na primeira coluna.

A estrutura de classes que suportam a informação de um recurso reflete a organização em tabela, subjacente à sua manipulação. O agregado columnHeaders é composto por duas instâncias da classe CellString, que apenas contém o texto que identifica cada uma das colunas: "Termo original" e "Tradução". O agregado rowHeaders é composto por um número de instâncias, da classe CellInteger, igual ao número total de conceitos, somado com o número total de propriedades de todos os conceitos, somado com o número total de ações de todos os conceitos. Em cada uma das referidas instâncias é guardado o número da linha. Por fim no agregado content encontram-se todos os termos a traduzir e as respetivas traduções. A dimensão deste último é igual ao número de colunas multiplicado pelo número de linhas. Dado um termo de índice n, é possível aceder à sua tradução através da fórmula 2n-1, sendo que o valor de n começa em 0.

A escolha desta estrutura foi feita por forma a ir de encontro às restrições impostas pelo editor projecional nativo da plataforma MPS, com o intuito de explorar as funcionalidades que o referido editor oferece no que diz respeito à edição de informação sobre várias formas, i.e., todas as outras DSL são editadas em forma de texto e nesta foi decidido explorar a representação da informação sobre a forma de tabela.

As classes CellString e CellInteger, apenas existem porque o referido editor exige que cada célula de uma tabela seja representada por uma classe. Esta abordagem visa permitir que em cada uma das células da tabela (cabeçalhos incluídos) possa conter dados complexos, compostos por vários atributos e até formulas. No caso concreto da nossa aplicação, como caso de uso atual apenas requer a edição de dados de tipo simples, a obrigação de utilizar classes distintas para os vários tipos de dados a editar é manifestamente exagerada.

5.2.6 Authorizations

A estrutura de dados deste componente visa capturar a definição de regras de configuração do funcionamento das ações de negócio.

Como é possível constatar no diagrama apresentado na Figura 35 as regras são agrupadas na classe Program através da propriedade listOfRules. Esta classe permite relacionar um conjunto de regras com o conceito da ontologia (concept).

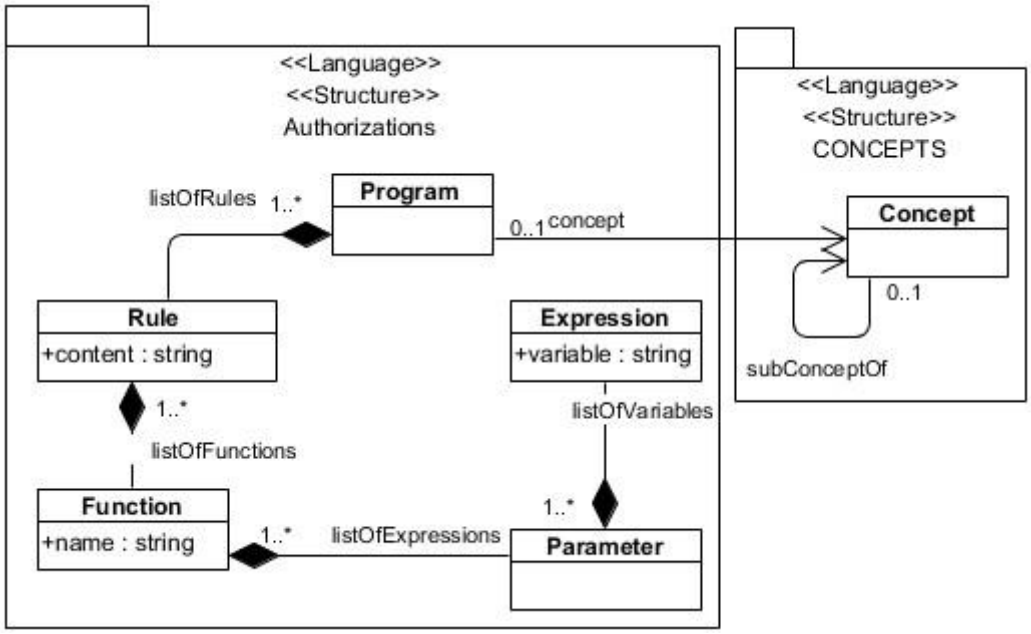

Figura 35 - Diagrama de classes do componente authorizations

Cada regra (Rule) é constituída por chamadas a funções do motor de regras. Cada função pode ter vários parâmetros (Parameter). Cada um dos parâmetros é o resultado de uma expressão (Expression) que contém a propriedade váriable onde é definda a expressão que opera dados disponíveis no motor de regras.

5.3 Editor

Outra consideração importante no design do projeto foi complementar a abordagem orientada ao domínio com os princípios de MDD, para permitir que uma compreensão profunda do domínio, possa passar das mentes dos especialistas de domínio e dos utilizadores para os programadores, através do uso de uma linguagem comum que é oferecida por este último.

Como foi referido na seção 2.2.6 Model driven design, o MDD advoga a adoção de um estilo arquitetural estruturado em camadas. Sendo que dentro de cada camada deve ser adotado um design coeso, que idealmente apenas gere dependências para as camadas inferiores.

Esta separação em camadas promove a separação de responsabilidades que vai de encontro aos princípios GRASP, que postulam uma alta coesão e um baixo acoplamento.

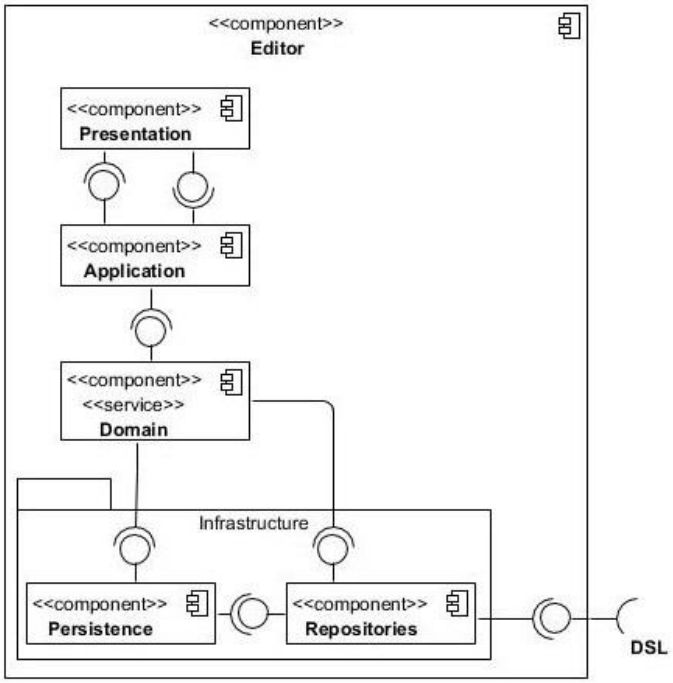

Figura 36 - Diagrama de componentes – Editor

Na Figura 36 - Diagrama de componentes – Editor, é possível constatar a aplicação direta dos princípios do estilo arquitetural de camadas no design deste componente. Os nomes dados aos componentes foram propositadamente escolhidos para evidenciar este facto.

Não são apresentados pormenores relativos ao design dos componentes Presentation, Persistence e Repositories, porque está previsto que a fase de desenvolvimento seja realizada com recurso a um *Language Workbench* (secção 2.2.9), que implementa estas funcionalidades de forma nativa.

No que respeita aos componentes Application e Domain, a sua implementação também será muito condicionada pela adoção da referida ferramenta. Este facto torna contra procedente quaisquer tentativas de evoluir na definição da sua arquitetura.

5.3.1 Presentation

O componente Presentation é responsável por fornecer interfaces de utilizador(UI) para realizar todas as tarefas, da aplicação.

Este componente é uma coleção de ecrãs; cada ecrã é preenchido por um conjunto de dados obtido a partir do componente Application sobre a forma de um modelo de visualização. Qualquer ação, que começa a partir do ecrã, encaminha outro conjunto de dados para a o componente Application, sobre a forma de um modelo de leitura.

Tendo em conta a abordagem orientada a modelos que foi adotada neste projeto. Considera-se que este componente (Presentation), também deve conter funcionalidades de geração

automatizada e execução de editores, para cada uma das DSL do domínio. Estas funcionalidades, são suportadas pela meta-informação das DSL: estrutura, estrutura de editores e restrições.

Toda a informação necessária é fornecida pelo componente Application que a solicita ao serviço Domain, que por sua vez tem a responsabilidade de a obter a partir da camada Infrastructure. Assim que o operador acaba a edição os dados seguem o caminho inverso para serem persistidos.

5.3.2 Application

O propósito do componente Application é agregar um conjunto de classes que funcionam como Controllers, cada uma encarregue de executar um fluxo de trabalho complexo correspondente a um dos casos de uso identificados no capítulo 4. Requisitos. A gestão do estado de cada um destes fluxos também é responsabilidade deste componente.

No caso das funcionalidades especificas de domínio, estes fluxos de trabalho permitem orquestrar a execução de tarefas que são delegadas nos serviços de domínio particulares de cada entidade, existentes na camada subjacente (componente Domain). Por sua vez este componente delega (através da camada de Infrastruture) a execução das operações de persistência necessárias ás referidas tarefas, nas entidades do modelo de domínio que se encontram no componente DSL.

Relativamente às funcionalidades não especificas de domínio, este componente é diretamente responsável pela sua implementação.

Por fim as funcionalidades especificas de domínio, que não se encaixam em nenhuma entidade, são delegadas no componente Domain.

O componente Application é o ponto de arranque do sistema com a responsabilidade de obter e fazer aplicar os parâmetros de configuração, que permitem personalizar o comportamento dos fluxos de trabalho.

Também é este componente que contém informação sobre as funcionalidades disponibilizadas pela aplicação nos vários contextos de execução. Informações estas que permitem ao componente Presentation gerar menus e outros controlos gráficos de interação com o utilizador. Como esta responsabilidade não se encontra codificada de forma rígida no componente Presentation, isto garante que na eventualidade deste ter de ser substituído, a sua substituição não implica alterações nos componentes das outras camadas.

5.3.3 Domain

O componente Domain é responsável pela implementação dos serviços de domínio (S6 e S7), que são parte da lógica de domínio que não se encaixam em nenhuma das entidades existentes, pois operam sobre várias entidades.

Este componente agrega as classes que implementam funcionalidades operacionais complementares aos processos de negócio e podem ser agrupadas em duas categorias: população automática das entidades de negócio e implantação dos dados de configuração.

Estas funcionalidades implicam a leitura e gravação de recursos externos à aplicação o que é feito diretamente por este componente. No que diz respeito às operações de leitura e gravação de informação do domínio da aplicação o componente atual delega esta responsabilidade nos componentes da camada Infrastructure.

5.3.4 Persistence

O componente Persistence, como o seu nome indica é responsável pelas funcionalidades relacionadas com a gravação e recuperação da informação relativa às instâncias dos vários objetos do domínio.

Na realização das suas tarefas recorre ao componente Repositories para obter meta-informação sobre a estrutura das entidades do domínio.

5.3.5 Repositories

O componente Repositories forma uma camada intermedia entre o modelo de domínio e a persistência de estados em fontes de dados.

Permite comunicar com o componente DSL para obter informações sobre os modelos de domínio, nomeadamente a estrutura das DSL e dos seus editores assim como as restrições impostas à informação passível de ser registada nas mesmas.

Este componente também implementa funcionalidades que permitem estruturar e manipular a informação recolhida, de maneira a que esta seja disponibilizada de forma adequada ao seu processamento pelos outros componentes.

6 Implementação

Este capítulo permite apresentar evidencias demonstrativas da utilização do *Language WorkBench* da JetBrains designado MPS, na implementação dos requisitos, em consonância com a decisões arquiteturais apresentadas na seção 5 Design,

O capítulo começa por explicar alguns aspetos estruturantes subjacentes á plataforma MPS (secção **Erro! Fonte de referência não encontrada.**). De seguida é explicada a forma como as a decisões arquiteturais foram transportadas para a plataforma (secção 6.2). Por fim, nas duas ultimas secções, é descrita a implementação dos dois componentes de alto nível da solução: DSL (secção 6.3) e Editor (secção 6.4).

Na Tabela 12 é apresentada a lista de *slices* criados em função das premissas enunciadas no nos parágrafos anteriores.

Tabela 12 - *Slices* adotados na fase de implementação

Slices	**Requisitos**
S1 Criar ontologia	RF1
S2 Importar a estrutura da fonte de dados	RF2
S3 Criar mapeamentos entre a fonte de dados e a ontologia	RF2
S4 Criar configuração da UI	RF3
S5 Criar configuração da lógica de negócio	RF4
S6 Criar traduções	RF5
S7 Importar ficheiros de configuração	RF2 a RF5
S8 Gerar ficheiros de configuração	RF2 a RF5
S9 Implantar o ficheiros de configuração no servidor	RF2 a RF5
S10 Criar projeto de configuração	
S11 Personalizar projeto de configuração	

Uma vez que era impraticável e contra procedente apresentar uma descrição detalhada para cada *slice,* estes são usados como exemplos nos vários processos descritos nas secções deste capítulo. Os *slices* S10 e S11, que não correspondem a nenhum requisito, representam funcionalidades que emergem no processo de desenvolvimento a partir da modalidade de implantação adotada para o sistema (seção 6.1.4) .

6.1 MPS

Nesta secção é descrita a forma como o MPS aborda as questões fundamentais da implementação de software.

6.1.1 Estrutura de um projeto

A Figura 37 mostra um diagramas UML de pacotes da estrutura simplificada de um projeto MPS (JetBrains, 2019). A partir desta imagem é possível perceber a estrutura hierárquica dos componentes, sendo que os componentes que não agregam filhos, são representados em forma de classes. Os componentes encontram-se classificados pelos respetivos estereótipos, que se encontram por cima dos nomes.

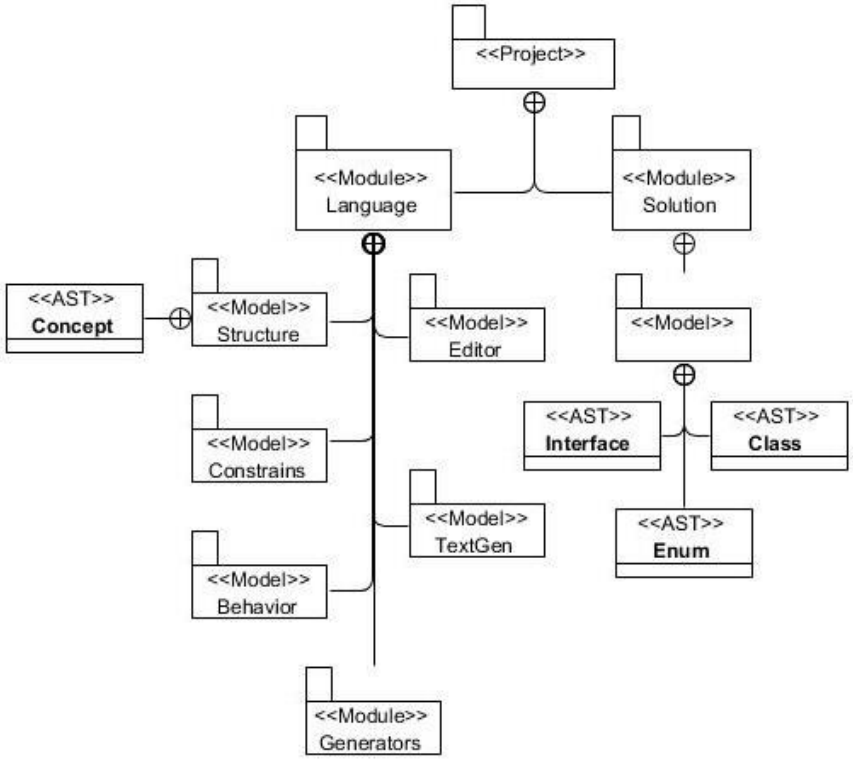

Figura 37 - Estrutura de um projeto MPS

Os componentes de um projeto MPS e a sua classificação permitem ilustrar a transposição dos conceitos teóricos da metodologia orientada a modelos, para uma implementação concreta orientada a objetos. Como este processo contribui para uma clarificação importante da teoria, os vários componentes e tipos são descritos a seguir.

1) Projeto: principal unidade organizacional do MPS. Os projetos são constituídos por um ou mais módulos, que por sua vez são compostos por modelos.

2) Modelo: elemento organizacional de nível inferior que funciona como unidade de compilação código de um programa ou como agregador de definições de um aspeto de uma DSL.

3) Módulos: elemento de organização de nível superior de um projeto MPS, que pode ser de dois tipos: linguagem ou solução.

4) Linguagem: tipo especial de modulo que agrega os vários modelos que definem uma DSL, como se pode ver a seguir.

    a) Estrutura: modelo composto por conceitos, onde se encontram descritos os nós e a estrutura da AST da DSL.

        i) Conceito: especificação da estrutura de um nó. Como cada nó pode ser muito diferente dos outros, cada um deles contém uma referência para o conceito, que define o seu "tipo", a sua classe. O conceito especifica quais filhos, propriedades e referências uma instância de um nó pode ter.

5) Editor: modelo onde se encontra descrita a forma como uma DSL será apresentada no editor projecional. Desenho do formulário de edição dos conceitos complementado com algumas regras de comportamento para o editor.

    a) Restrições: modelo onde se encontram definidas restrições adicionais a aplicar aos conceitos, às suas propriedades e relacionamentos.

    b) Comportamento: modelo onde se encontra descrito o aspeto comportamental da AST. Permite definir um construtor para a AST, assim como métodos.

    c) Geradores: modelos que definem possíveis transformações de uma DSL em algo mais, tipicamente noutras DSL.

    d) TextGen: modelo onde é possível definir uma transformação de uma DSL em texto.

6) Solução: tipo mais simples possível de módulo em MPS. É apenas um conjunto de modelos, que contêm código, agrupados através de um nome comum.

    a) Soluções sandbox: estas soluções permitem aos programadores fazer ensaios e experiências sobre as linguagens e sobre o código das soluções runtime. Normalmente não são consideradas quando se gera o executável do projeto.

    b) Soluções runtime: estas soluções contêm código, do qual dependem outros módulos (soluções, linguagens ou geradores). O código pode consistir de modelos MPS, bem como de classes Java, fontes ou arquivos JAR.

    c) Soluções plugin: permitem estender a funcionalidade do IDE (acrescentar novos menus, adicionar janelas ao painel de ferramentas laterais, definir janelas de preferências personalizadas, etc.).

6.1.2 Integração de bibliotecas externas

Quando existem funcionalidades na aplicação que necessitam de funcionalidades pré-existentes em java, a solução obvia é utilização de bibliotecas dinamicamente carregáveis, que

aplicativos Java podem chamar em tempo de execução. Como a plataforma Java não depende de um sistema operativo específico, as aplicações não podem depender de nenhuma das bibliotecas nativas dos mesmos. Em vez disso, a plataforma Java e inúmeros fabricantes fornecem um conjunto abrangente de bibliotecas de classes, contendo funções para a realização de tarefas comuns.

A plataforma MPS apresenta como solução para este problema a criação de *stubs*, esboços das interfaces das bibliotecas externas, que na prática são pedaços de código usados para simular o comportamento do código existente.

Dadas as especificidades associadas a esta abordagem assim como a grande frequência com que ela é necessária, nesta Seção é apresentada uma descrição dos vários passos que compõe este processo. Para tal será usado como exemplo a integração dos drivers de acesso a bases de dados SQL da Microsoft. Estes drivers foram usados no caso de uso 7- CRUD persistência, onde os objetos fontes de dados tem necessidade de aceder às referidas bases de dados com o propósito de importar a estrutura das tabelas e de outros componentes existentes nas mesmas.

Para iniciar o processo é necessário abrir as propriedades da solução onde será adicionado o stub das bibliotecas.

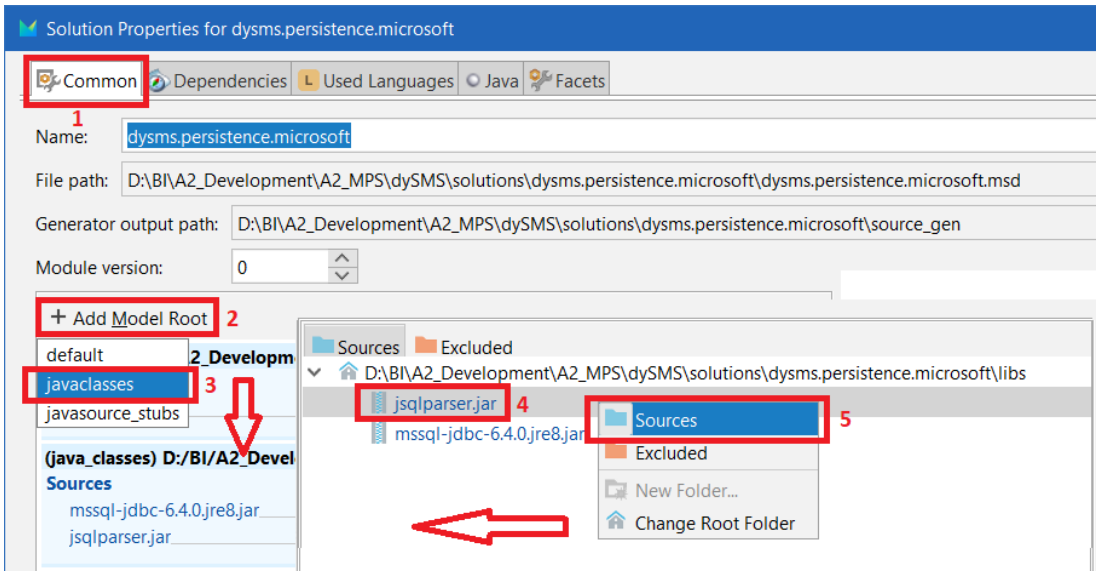

Figura 38 - Integração de classes JAVA, passo 1

Na Figura 38 é possível visualizar os passos necessários á realização do processo, que são descritos a seguir:

1) Selecionar tabulador

2) Acrescentar uma nova pasta raiz ao modelo

3) Escolher o formato “javaclasses”

4) Aceder ao menu de contexto do ficheiro JAR que se encontra na pasta

5) Marcar o referido ficheiro como fazendo parte das fontes do modelo

De seguida, ainda nas propriedades da solução é necessário configurar definições relativas ao JAVA.

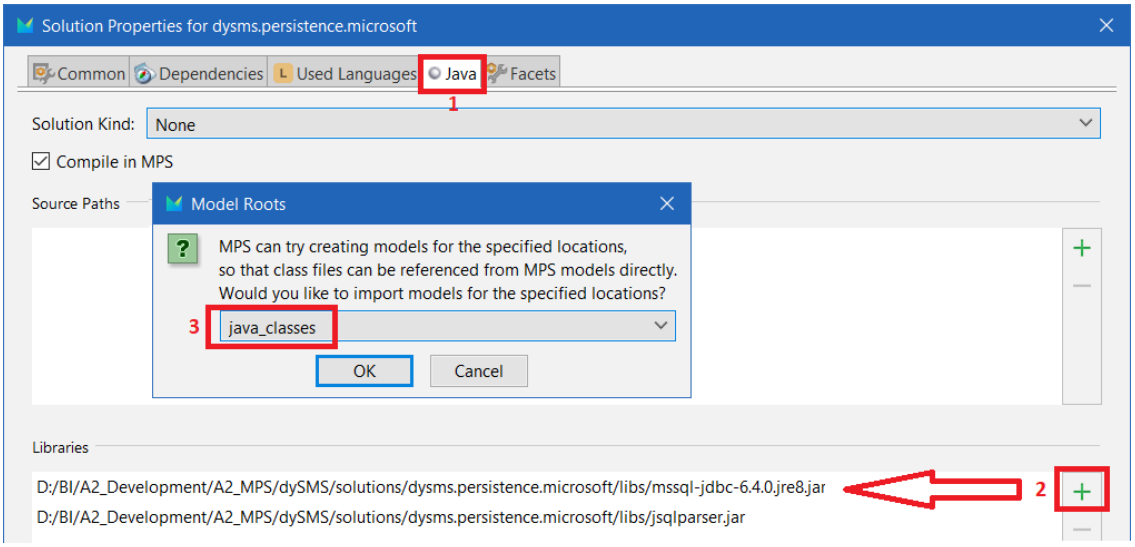

Figura 39 - Integração de classes JAVA, passo 2

Na Figura 39 é possível visualizar os passos necessários á realização do processo, que são descritos a seguir:

1) Selecionar tabulador

2) Adicionar referências aos ficheiros JAR

3) Para cada ficheiro JAR selecionar o tipo “java_classes”

Assim que os dois passos anteriores forem realizados, é possível fazer referência aos drivers usando as instruções JAVA que permitem aceder a uma classe pelo nome, como se demonstra na Figura 40.

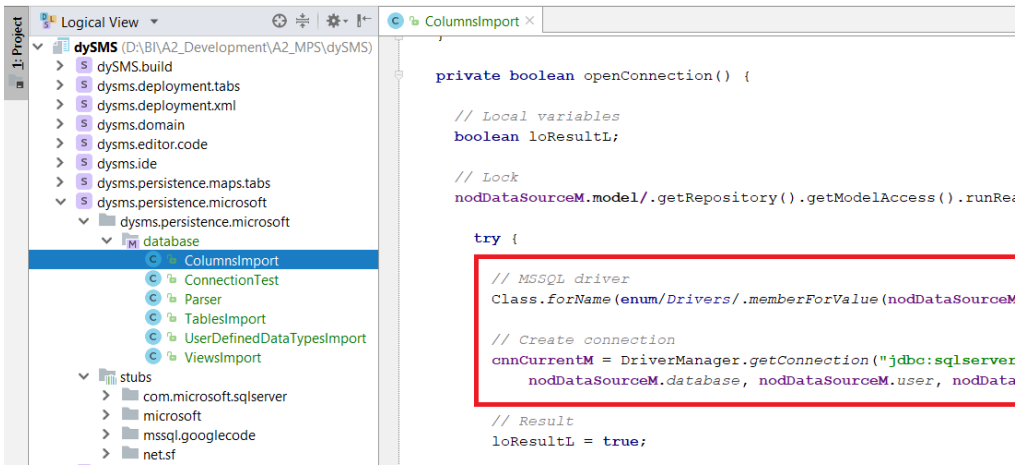

Figura 40 - Usar classes JAVA integradas

6.1.3 Compilação do projeto

A plataforma MPS disponibiliza recursos específicos para a criação dos executáveis dos projetos, nomeadamente um tipo de solução especifica e uma linguagem (DSL) para definir os passos do processo de compilação de uma forma declarativa.

6.1.3.1 Projeto de compilação

Na Figura 41 é possível visualizar o processo de criação do projeto de compilação, que é realizado com o apoio de um assistente, que automatiza grande parte do processo, bastando identificar o tipo de executável (1) e as linguagens e soluções as serem compiladas (2).

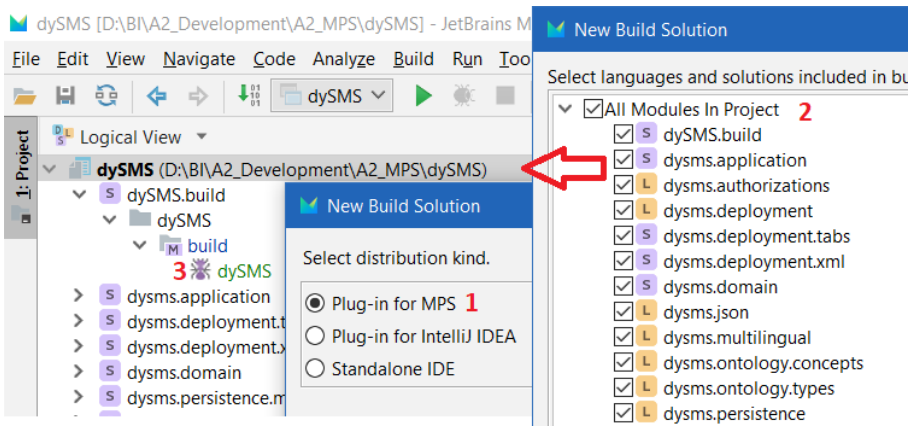

Figura 41 - Projeto de compilação

Para dar início à compilação basta aceder ao menu de contexto do script (3) e escolher a opção Run, que depois de transformar o script de compilação num script ANT, realiza a compilação de forma automática.

6.1.3.2 Script de compilação

A linguagem de build nativa da plataforma é uma DSL extensível de automação do processo de build, que é transformada de forma automática num script ANT. Desta forma torna-se possível aproveitar o poder de execução do ANT, mantendo o código fonte limpo e livre de desordem e detalhes irrelevantes.

O facto de esta DSL estar organizada como uma pilha de linguagens MPS, com a linguagem do ANT na base, permite que cada parte do procedimento de compilação seja expresso num nível de abstração diferente. A compilação de um artefacto complexo, como o plug-in do projeto dySMS.config, pode ser especificada de uma forma simples através do aproveitamento do template que é gerado automaticamente, ao qual apenas é necessário fazer algumas alterações.

O script de build para o projeto atual pode ser decomposto em três partes.

```
build dySMS generates build.xml 1
base directory: ../../
use plugins:
  java
  mps
macros:
  folder mps_home = ./../../../../A1_IDEs/A2_MPS-2018.1 2
dependencies:
  mps (artifacts location $mps_home)
```

Figura 42 - Cabeçalho do script de build

Nesta parte, como se pode ver na Figura 42, são declaradas as características gerais do processo de compilação como por exemplo: (1) nome do ficheiro ANT a ser gerado e (2) a pasta onde se encontra a plataforma MPS, para permitir aceder às várias livrarias da mesma.

```
project structure:
  idea plugin dySMS from generated by default 'plugin.xml' descriptor file
    name dySMS
    short (folder) name dySMS
    description <no description>
    version 1.0
    content:
      dySMS
    dependencies:
      jetbrains.mps.core
      com.intellij.modules.mps 1
  mps group dySMS
    language dysms.persistence.maps
      load from ./languages/dysms.persistence.maps/dysms.persistence.maps.mpl
    language dysms.system.styles
      load from ./languages/dysms.system.styles/dysms.system.styles.mpl
    solution dysms.deployment.tabs
      load from ./solutions/dysms.deployment.tabs/dysms.deployment.tabs.msd
    solution dysms.system
      load from ./solutions/dysms.system/dysms.system.msd
    language dysms.ontology.concepts
      load from ./languages/dysms.ontology.concepts/dysms.ontology.concepts.mpl
    language dysms.multilingual
      load from ./languages/dysms.multilingual/dysms.multilingual.mpl
    solution dysms.resources
      load from ./solutions/dysms.resources/dysms.resources.msd
    language dysms.persistence 2
```

Área 1

Área 2

Figura 43 - Estrutura do projeto no script de build

Na Figura 43, na área 1 encontram-se a identificação do projeto e as referências às várias livrarias da plataforma, que são usadas pelo dySMS. Por omissão o assistente não adiciona uma das livrarias (1), que tem de ser adicionada manualmente. Na área 2 é criada uma relação de todos os componentes constituintes do projeto a serem incluídos no processo de compilação. Manualmente é necessário aceder ao menu de contexto de um dos componentes (2) e recarregar os módulos do mesmo do disco. Esta operação permite refrescar internamente as definições de todos os módulos e resolve o erro que por amissão aparece ao abrir o script.

Figura 44 - Livrarias JAVA eternas no script de compilação

A última parte do script, representada na Figura 44, contém uma lista com as livrarias JAVA externas à plataforma MPS (1), que tem de ser adicionada manualmente, pois o assistente não as define.

6.1.4 Implantação

A plataforma MPS permite a criação de três formas diferentes de artefacto de implantação. Nesta Seção serão apresentadas breves explicações acerca de cada uma delas e sobre a razões que suportaram a decisão de optar pelo formato de plugin para o MPS. Na subseção referente à opção escolhida encontram-se os detalhes de design concretos do projeto atual.

6.1.4.1 Versão personalizada e autónoma do IDE

O termo IDE autônomo refere-se a uma versão minimalista do IDE que contém apenas os artefactos necessários para uma determinada finalidade comercial. IDE autônomos fornecem uma maneira conveniente de distribuir DSL aos utilizadores finais, que pretendam usar linguagens, incluindo todas as nativas do IDE, reformulação e análise de código preparado pelos designers de linguagens no conforto de um IDE dedicado. Todas as funcionalidades relacionadas com o design de novas linguagens-assim como linguagens nativas desnecessárias podem ser removidas.

Depois de projetar as linguagens especificas de um projeto, o MPS permite criar uma versão reduzida do próprio IDE, que contém apenas os artefactos necessários para usar essas linguagens específicas de domínio. Vários aspetos do MPS podem ser removidos ou personalizados para fornecer uma experiência personalizada a um determinado grupo de utilizadores.

Esta opção faz sentido ser usada quando o conjunto de novas linguagens e funcionalidades de um dado projeto têm uma dimensão tal que requerem uma transfiguração completa do ambiente, que não é o caso. Relativamente à personalização do aspeto, logos e outros aspetos relacionados com a imagem de marca, não se aplicam neste projeto visto não existirem fins comerciais. Tendo em conta as razões anteriores, o aumento da dimensão do executável gerado, o aumento de complexidade na distribuição e o aumento da entropia em geral que esta opção aportaria ao dySMS.config, justificam claramente que ela seja descartada.

6.1.4.2 Plugin para o InteliJ IDEA

Esta forma de distribuição tem como finalidade disponibilizar um conjunto de linguagens criadas em MPS a programadores JAVA dentro do IntelliJ IDEA.

Como claramente este não é o propósito do projeto atual, naturalmente esta opção também foi preterida.

6.1.4.3 Plugin para o MPS

A adoção deste formato de distribuição, onde as linguagens de domínio junto com as funcionalidades adicionais necessárias ao projeto são agrupadas num pequeno ficheiro em formato ZIP que pode facilmente ser acoplado ao IDE MPS, previamente instalado no cliente, tornou-se a escolha obvia desde o inicio.

Além da simplificação da distribuição e do processo de compilação e build, este formato oferece ainda todas as funcionalidades de personalização do IDE necessárias ao projeto, garante uma independência relativamente à versão do mesmo e do sistema operativo e ainda assegura a extensibilidade futura das funcionalidades do dySMS.config, tanto por parte do cliente como dos seus parceiros.

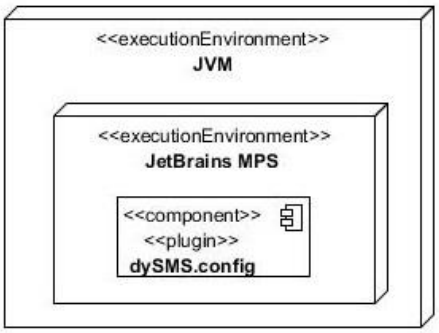

Figura 45 - Diagrama de implantação

O diagrama de implantação, apresentado na Figura 41, evidencia claramente a simplicidade, abstração e versatilidade do formato de implantação adaptado.

6.2 Design orientado a modelos

Nesta secção pretende-se mostrar evidencias da aplicação da teoria subjacente à metodologia orientada a modelos e descrever o suporte tecnológico fornecido pela ferramenta adotada, na aplicação desta metodologia.

6.2.1 Componentes

Como foi explicado na secção 6.1.1, a plataforma MPS permite a criação de dois tipos diferentes de componentes: soluções e linguagens. Fazendo uso destes recursos, o design da aplicação foi transposto para a plataforma como se pode ver na Figura 46, onde o 2º estereótipo de cada componente representa o tipo de componente da plataforma.

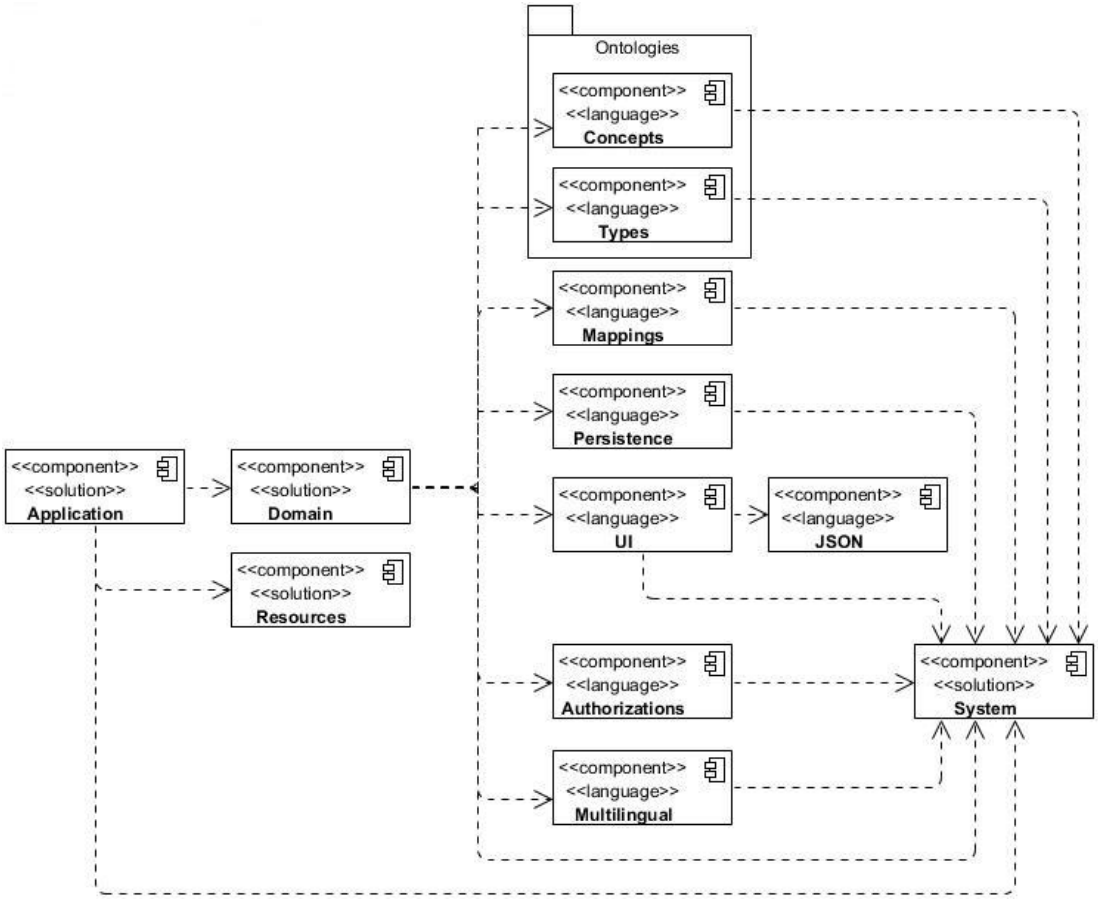

Figura 46 - Diagrama de componentes

O diagrama, apresentado na Figura 54, evidencia a clara separação de responsabilidades de cada componente assim como a focalização no domínio do problema, que o uso de uma plataforma especializada permite aportar ao design das aplicações, nomeadamente através do uso de DSL («*language*»). O componente System, que não existia na fase de design, foi criado para servir de interface entre os outros componentes e o MPS.

A arquitetura do projeto encontra-se também em conformidade com os padrões inerentes á metodologia adotada (secção 2.2.6.1), como se explica a seguir.

1) Serviços: no caso concreto do dySMS.config, foi necessário criar vários serviços que abrangem funcionalidades transversais à aplicação e funcionalidades auxiliares para alguns dos outros componentes. No primeiro caso temos por exemplo os componentes APPLICATION e SYSTEM, que gerem o acesso às funcionalidades internas da plataforma.

No que diz respeito a serviços auxiliares podemos enunciar por exemplo o PERSISTENCE que agrega vários componentes de suporte para as funcionalidades relacionadas com o mapeamento de conceitos da ontologia com as respetivas fontes de dados.

2) Entidades: em conformidade as regras do MPS nesta categoria encontram-se todos os modelos que constituem o domínio da aplicação. São exemplos de modelos nesta categoria conceitos como, *Types*, *Resources*, etc.

3) Objetos valores: nesta categoria encontram-se conceitos como as *languages*, que permitem qualificar os recursos multilingue ou os *widgets* que definem os vários controlos possíveis existentes num ambiente de desenvolvimento.

4) Agregados: Alguns dos conceitos do domínio são compostos por um modelo raiz e por vários outros modelos agregados ao primeiro. Um exemplo destes conceitos é o *mapping*, além dos atributos próprios contém coleções de outros conceitos como *foreign keys* ou *relations one* to *one*. A manipulação dos objetos secundários apenas faz sentido quando estes se encontram associados ao conceito raiz.

5) Repositórios e Fabricas: no caso destes padrões de design a plataforma MPS dispões de mecanismos nativos que realizam estas operações, sem haver necessidade qualquer desenvolvimento adicional. Esta é uma das muitas funcionalidades incorporadas na referida plataforma, que a tornam ideal para a metodologia orientada a modelos.

6.2.2 *Layers*

Devido a restrições impostas pela própria plataforma nem sempre foi possível respeitar completamente os princípios associados a este padrão. Os componentes estruturais de um projeto MPS pretendem evidenciar e facilitar o desenvolvimento orientado a modelos. Por esta razão existem várias situações em que componentes, que supostamente se deveriam encontrar em camadas diferentes se encontram juntos, quebrando os princípios deste padrão. Um exemplo flagrante pode ser constatado na agregação dos componentes de edição das DSL (camada de apresentação) com os modelos das mesmas (camada de negócio).

Apesar das considerações referidas foi feito um esforço para ultrapassar restrições impostas pela plataforma e adotar os princípios deste padrão (seção 2.2.6.2), como se pretende demonstrar na Figura 47 - Arquitetura em camadasFigura 47, onde se encontram representados a vermelho os vários componentes do projeto, distribuídos pelas várias camadas. A azul foi introduzida a indicação das funcionalidades implementadas de forma nativa pela aplicação.

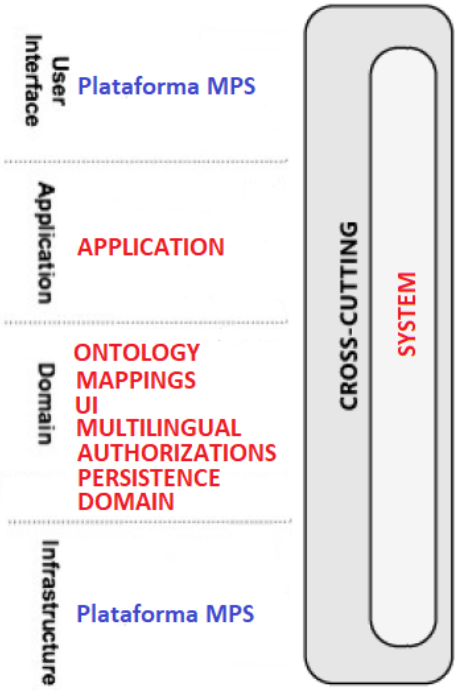

Figura 47 - Arquitetura em camadas

A camada Domain é composta por todas as DSL, que representam as entidades de domínio (componente DSL) e pelo componente Domain, que é responsável pela implementação dos serviços de domínio (S7, S8 e S9), que são parte da lógica de domínio, mas não se encaixam em nenhuma das entidades existentes, pois operam sobre várias entidades.

6.3 DSL

Como foi explicitado no processo de design, o componente DSL agrega os subcomponentes necessários á implementação de todas as DSL. Em cada um dos subcomponentes são implementadas as funcionalidades relativas a uma DSL em concreto, que representa uma entidade no domínio.

O processo mais básico de criação de uma DSL requer a execução de quatro passos nomeadamente: (i) definição dos conceitos da linguagem (sintaxe abstrata); (ii) definição de restrições; (iii) definição do editor para os conceitos e (iv) definição de um gerador.

Cada um dos subcomponentes referidos no primeiro paragrafo, encontra-se dividido em pacotes, que correspondem aos passos necessários para a criação de uma DSL.

6.3.1 Esquema

O esquema da DSL permite definir a sua estrutura, composta por conceitos relacionados entre si. Na Figura 48 é possível visualizar a definição de um conceito no editor projecional do MPS.

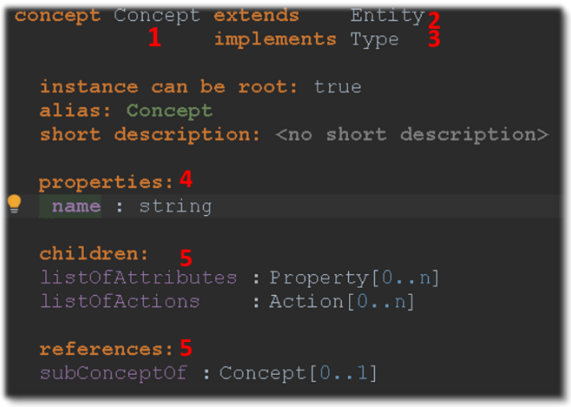

Figura 48 – Esquema de uma DSL

Na Figura 48 encontram-se assinalados os elementos relevantes para a definição de um de um conceito: (1) Nome do conceito, (2) Um conceito pode estender outro conceito, (3) Um conceito pode implementar um ou mais interfaces, (4) Um conceito pode conter atributos com dados simples; (5) Um conceito pode agregar conjuntos de tipos complexos (outros conceitos) e (6) Um conceito pode conter referências a outros conceitos.

A estrutura do componente Ontology (seção 5.2.1) foi desenhada tendo por base a estrutura dos esquemas das linguagens do MPS, sendo assim, é possível transmitir uma descrição mais detalhada do funcionamento da definição de esquemas no MPS, fazendo uma analogia com o referido componente. Na Ontology, tal como aqui, as entidades são representadas por Concept e cada Concept é composto por Property, que têm um name e um type. Os type das Property podem ser DataTypeSingle, DataTypeColletion e Concept, que o MPS usa para agrupar as Property em três categorias respetivamente: properties, children e references. Os Concept do MPS têm atributos que não existem na Ontology e vice-versa, que são específicos de cada contexto, como por exemplo: “instance can be root”, “alias” ou as Action na Ontology.

6.3.2 Restrições

As restrições são regras que acrescentam significado semântico aos atributos dos conceitos que definem estrutura da DSL. Estas regras permitem definir os valores válidos numa instância de um determinado atributo recorrendo a código em java (Código 1)

```
public class cardinality {
 private dysms_system_Utils sysCurrentM;
 private node<Property> nodCurrentM;
 private int inNew_CardinalityM;
 private list<enummember<KindsOfCardinalities>> lstCardinalitiesM;
 public cardinality(node<Property> nodCurrentP, int inNew_CardinalityP) {}
 public boolean isValid() {
   identifyPossibleCardinalities();
   return verify();}
 private void identifyPossibleCardinalities() {...}
 private boolean verify() {...} }
```

Código 1 – Exemplo de código Java para definição de restrições de uma DSL

As classes do código Java, podem então ser usadas para definir uma regra e associa-la a um conceito como se ilustra na Figura 49.

```
concepts constraints Property {
  can be child <none>

  can be parent <none>

  can be ancestor <none>

  property {cardinality}
    get <default>
    set <default>
    is valid (propertyValue, node)->boolean {

      cardinality carCurrentL;

      carCurrentL = new cardinality(node, propertyValue);

      return carCurrentL.isValid();

    }
```

Figura 49 - Restrições de uma DSL

No caso concreto do exemplo apresentado, para o atributo cardinality do concept Property da ontologia é criada uma classe Cardinality (Código 1) que é usada numa regra que verifica se o valor atribuído a esse atributo é válido.

6.3.3 Editor

Sendo a plataforma MPS uma plataforma especializada produção de linguagens de domínio específico, oferece uma abordagem inovadora no que diz respeito ao uso de editores projecionais. Estes tipos de editores permitem que os utilizadores de aplicações baseadas na referida plataforma, atuem sobre os dados através de interfaces que emulam o funcionamento de um editor de texto. Esta abordagem agiliza e simplifica significativamente a operacionalidade e ao mesmo tempo permite proporcionar assistência, sugestões e validação à medida que o utilizador vai evoluindo na introdução e alteração dos dados. Além destas facilidades permite ainda juntar no mesmo editor múltiplas formas de manipulação de dados, como tabelas, expressões matemáticas, *widgets*, etc.

Editar a AST e não o texto real requer alguma habituação. Sem personalização específica, cada elemento do programa deve ser selecionado de uma lista suspensa para ser "instanciado". No entanto, o MPS fornece personalizações dos editores para permitir uma edição que se assemelha às IDE modernas, que usam modelos de código automaticamente expandindo. Isto torna a edição muito conveniente e produtiva.

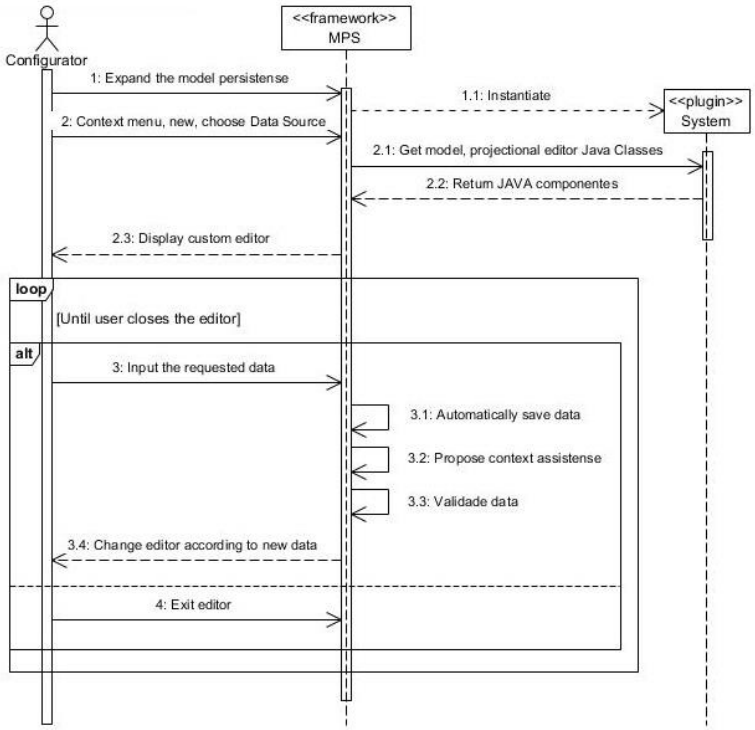

Figura 50 – Diagrama de sequência para o editor

No diagrama representado na Figura 50 é apresentada a sequencia de passos que descrevem o processo de funcionamento de um editor projecional na plataforma MPS. Para implementar um editor para uma DSL é necessário definir uma estrutura que informe a plataforma acerca do posicionamento das propriedades do conceito correspondente no ecrã. A Figura 51 apresenta um exemplo onde é defenda a estrutura do editor para o conceito Concept da ontologia.

```
<default> editor for concept Concept
  node cell layout:
    [/
      [> Concept $swing component$ <]
      <constant>
      [/
        [> Name { name } <constant> <]
        [> Subconcept of ( % subConceptOf % -> { name } ) [-] <]
      /]
      <constant>
      Properties
      <constant>
      (/ % listOfAttributes % /)
        /empty cell: <default>
        /folded cell: <default>
      <constant>
      Actions
      <constant>
      (/ % listOfActions % /)
        /empty cell: <default>
        /folded cell: <default>
    /]
```

Figura 51 - Editor de uma DSL

A definição da estrutura pode ser complementada com código associado a um elemento especifico do editor. No estrato de Código 2 é apresentado um exemplo usado no editor anteriormente referido.

```
component provider: (node, editorContext)->JComponent
{
  //Parameters
  string[] teOptionsM;
  teOptionsM = new string[1];
  teOptionsM[0] = "Import fields";
  // Instanciate
  MenuExecutor mnuExecutorL = new MenuExecutor(node, teOptionsM);
  Menu mnuGenericL = new Menu(mnuExecutorL, teOptionsM);
  return mnuGenericL.getPanel();
}
```

Código 2 - Definições complementares do editor de uma DSL

Em Código 2 encontra-se o código afeto ao componente do editor designado "swing component". Este código permite que no editor apareça uma imagem usada para aceder ao menu de contexto do conceito. Neste menu é possível acionar uma operação que permite transformar o conceito.

6.3.4 Gerador

Um dos principais princípios/objetivos da metodologia de desenvolvimento de software MDE é a geração automatizada de código a partir de modelos abstratos (Marco Brambilla, 2017). De acordo com a regra Modelos + Transformações = Software é possível aumentar de forma significativa a produtividade e eficiência.

Em MDE as transformações designam os processos automatizados de transformação de modelos noutros modelos até que no final é possível a geração de código, em forma de texto.

No caso concreto da plataforma adotada para suportar o desenvolvimento deste projeto, a implementação das referidas transformações implica a utilização de elementos arquiteturais, intrínsecos aos projetos MPS.

O primeiro destes elementos a ser implementado é um "Gerador", que permite transformar um modelo noutro, que tem de ser complementado pelo "TextGen" que finaliza o processo e transforma o modelo final em código.

Para uma linguagem cujos programas devem ser processados com uma ferramenta orientada a texto o "TextGen" gera texto. Para o processamento de linguagens de alto nível é usado o "Gerador", que transforma um modelo expresso em L_d (uma linguagem para o domínio d) num modelo em L_{d1} (uma linguagem de um domínio inferior).

Neste último caso, os geradores não são geradores de texto, mas sim processos que realizam a conversão entre árvores de sintaxe abstrata (AST).

No diagrama representado na Figura 52 é apresentada a sequencia de passos que descrevem o processo de funcionamento de um gerador na plataforma MPS.

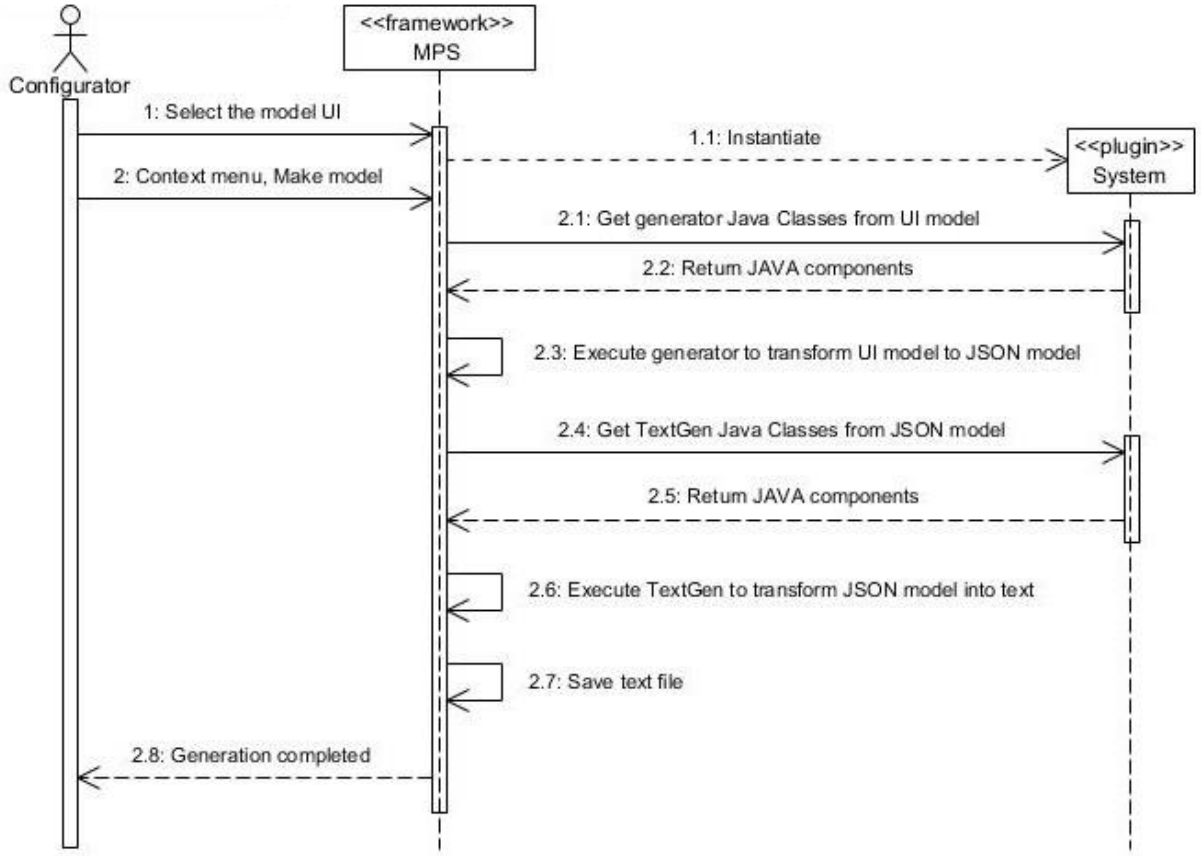

Figura 52 - Diagrama de sequência para o gerador

6.4 Editor

Nesta secção são descritas

6.4.1 Criar projeto

O primeiro passo que permite dar inicio à utilização do dySMS.config é a criação de um novo projeto. Um projeto dySMS.config é uma adaptação do conceito genérico de projeto do IDE MPS ao domínio do dySMS. O processo de criação do projeto, Figura 53, encontra-se plenamente integrado no referido IDE seguindo os mesmos passos que seriam necessários para criar um projeto padrão. Apenas sendo necessário que o utilizador escolha de entre os tipos de projetos disponíveis o tipo dySMS. Esta integração é conseguida recorrendo às capacidades de extensão do próprio MPS. Neste caso concreto o recurso utilizado designa-se *Application* plugin, que permite estender os grupos de projetos pré-existentes e adaptá-los às necessidades concretas do domínio da nossa aplicação, codificando um novo *template* de criação de projeto.

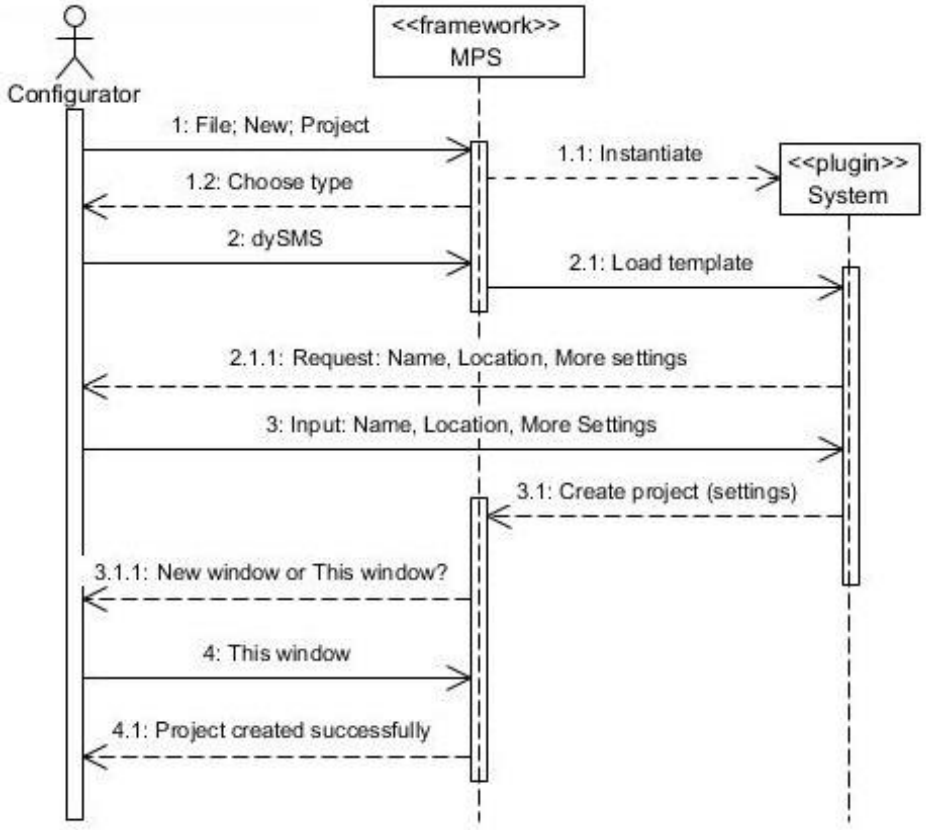

Figura 53 – Diagrama de sequência da criação de um projeto

O projeto dySMS.config funciona como um plugin para a plataforma MPS, que permite que o configurador crie vários projetos de configuração, Figura 54, com uma estrutura e funcionalidades personalizadas (1).

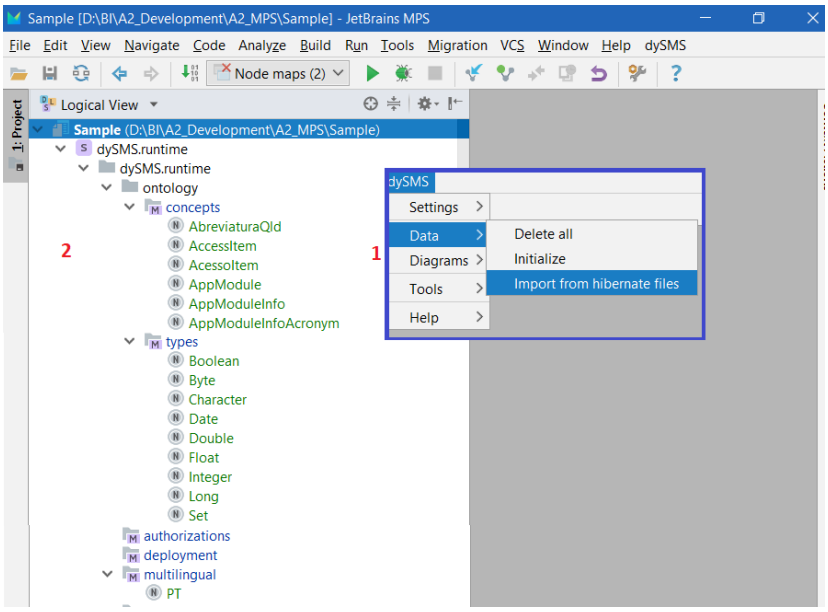

Figura 54 - Projeto em tempo de execução

Os projetos dySMS.config são compostos por várias pastas, que permitem agrupar as instâncias dos vários objetos específicos da aplicação (2). Cada uma destas instâncias representa um elemento de configuração do sistema e a estrutura de pastas, Figura 55, proporciona ao utilizador uma visão organizada dos vários aspetos a serem configurados.

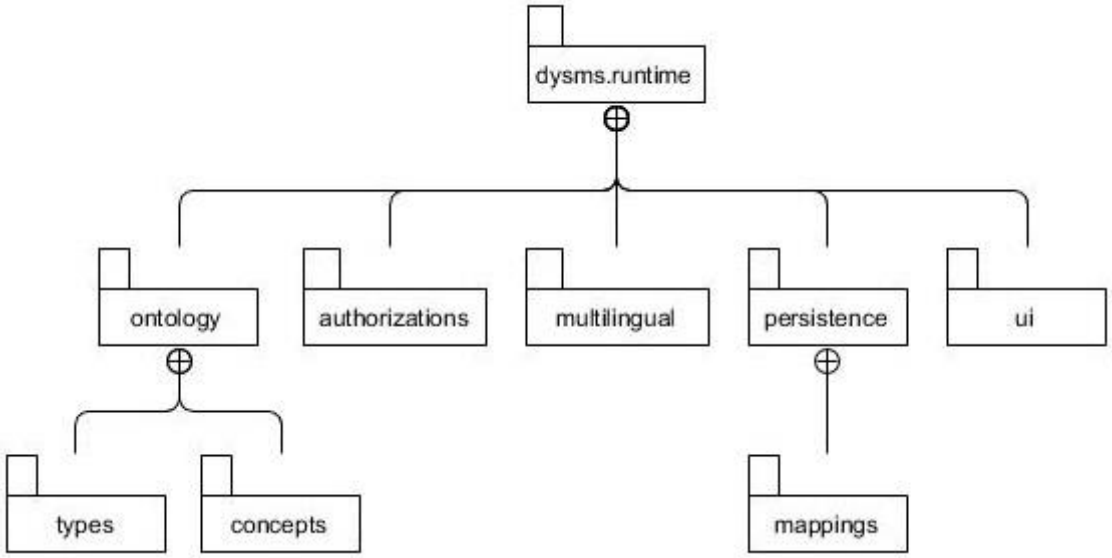

Figura 55 - Diagrama de pacotes do projeto em tempo de execução

6.4.2 Definir configuração

Os projetos do tipo dySMS têm uma estrutura própria, composta por uma solução de runtime que agrega um conjunto de modelos. Os objetivos estruturais subjacentes a esta organização pretendem proporcionar ao utilizador o acesso às várias funcionalidades de forma que a sua disposição se encontre em conformidade com a realidade do trabalho a ser realizado. Também foi tida em conta a necessidade de evidenciar a sequência lógica da realização das várias operações e a agregação lógica dos vários artefactos gerados.

Apesar de todas estas considerações, foi acrescentada esta opção com o intuito de flexibilizar a referida estrutura por forma a permitir aos operadores organizar os projetos de acordo com as suas preferências.

Como é possível constatar na Figura 56, além da estrutura do projeto, também é possível personalizar vários comportamentos da aplicação indicando alterativas para a localização das pastas de *input* e *output* e das ferramentas externas utlizadas.

Assim como na funcionalidade anterior, esta encontra-se plenamente integrada no IDE fazendo uso do recurso designado como componente de preferências.

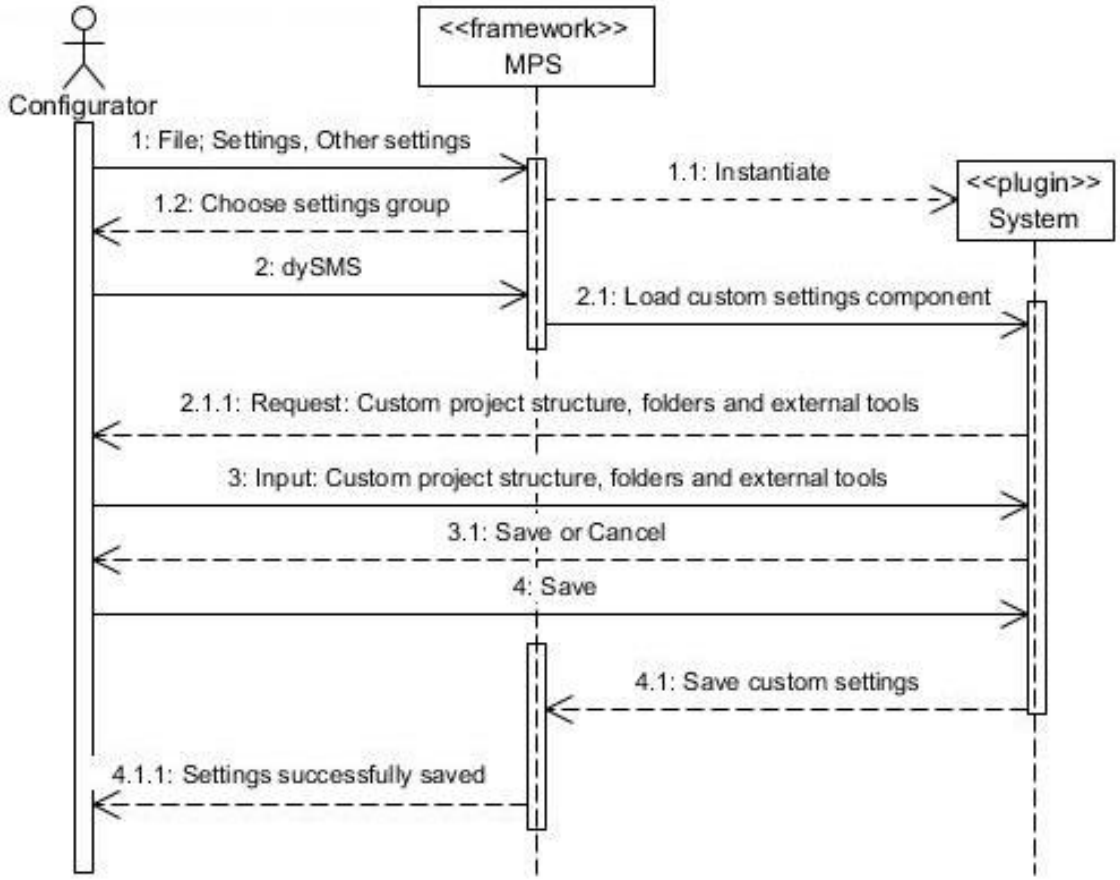

Figura 56 – Diagrama de sequencia para definição da configuração de um projeto

6.4.3 Importar ficheiros de configuração

A descrição dos pormenores arquiteturais desta funcionalidade permite explicar a forma como é implementada a extensão do IDE, através do acrescento de menus personalizados. Também permite abordar as especificidades inerentes à integração de código Java nativo, num projeto que funciona sobre a plataforma MPS.

O mecanismo que suporta o acréscimo de novas opções ao menu do IDE funciona através da criação de dois tipos de componentes: Grupos de opções e Ações. O primeiro tipo permite agrupar opções de menu num componente único, que representa um menu. Enquanto que o segundo tipo, é utilizado para especificar as ações concretas a serem executadas quando o operador seleciona um item de menu. Associado aos dois, existem ainda várias propriedades que permitem configurar aspetos específicos de cada um, como por exemplo a posição ou a ordem pela qual cada um deve ser apresentado no ecrã.

No que concerne à integração de código Java nativo importa salientar que a referida plataforma trata este código em forma de AST e não em forma de texto, no entanto através da utilização de um editor projecional, especificamente adaptado à AST da linguagem Java, que é fornecido

com o IDE de forma nativa, a edição do código processa-se de uma forma parecida com a edição tradicional em formato de texto.

No caso concreto apresentado na Figura 57, foi adicionada uma nova opção de menu que faz parte do grupo (menu) designado por dySMS. Esta opção permite executar um conjunto de classes java, que têm por finalidade realizar a interpretação de ficheiros em formato hibernate e transformar a informação neles contida num agregado de objetos. Tendo por base esta estruturação prévia dos dados, procede-se de seguida à criação automatizada de instâncias dos modelos constituintes do domínio da aplicação.

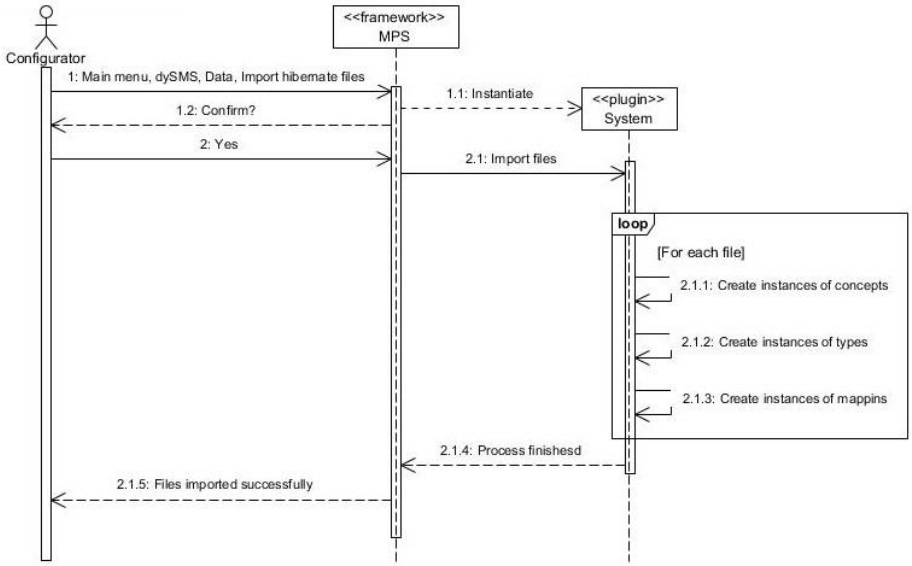

Figura 57 – Diagrama de sequência para a importação de ficheiros

6.4.4 Biblioteca system

Esta livraria tem por finalidade agregar funcionalidades transversais ao projeto, nomeadamente funções onde se encontra encapsulada a complexidade inerente ao API de comunicação com o IDE (Anon., 2018).

Nesta Seção serão apresentadas algumas explicações sobre as funcionalidades mais relevantes.

6.4.4.1 Obtenção de referências ao projeto em tempo de execução

De acordo com o local e com a informação necessária o API do MPS oferece vários objetos, no entanto para que estes objetos sejam instanciados é necessário obter o contexto de execução, cuja instância é acedida através do objeto base designado DataManager. O contexto de execução permite distinguir em vários projetos, que se podem encontrar abertos em simultâneo.

```
DataContext dtcCurrentL = DataManager.getInstance().getDataContext();
mppProjectM = MPSDataKeys.MPS_PROJECT.getData(dtcCurrentL);
oapProjectM = MPSDataKeys.PROJECT.getData(dtcCurrentL);
```

```
teProject_NameM = oapProjectM.getName();
```

Código 3 - Acesso o projeto em tempo de execução

A partir da referência ao projeto, no formato Open API, torna-se possível obter o nome do projeto ativo.

```
List<Project> lstOpen_ProjectsL =
ProjectManager.getInstance().getOpenedProjects();

Project prjReturnL = lstOpen_ProjectsL.get(0);

foreach prjCurrentL in lstOpen_ProjectsL {
  prjReturnL = prjCurrentL;
  if (prjCurrentL.getName().equalsIgnoreCase(teProject_NameL)) { break; }
}

return prjReturnL;
```

Código 4 - Acesso ao projeto ativo

Através do objeto que gere todas a janelas do MPS abertas (ProjectManager) é possível obter uma lista dos projetos abertos nas referidas janelas. Fazendo uso do nome do projeto ativo, torna-se possível identificar na referida lista a referência para a instância onde o operador está a trabalhar no momento.

6.4.4.2 Obter referências a componentes e informações do projeto ativo

Como foi explicado no capítulo de design, todos os projetos MPS têm são compostos por um conjunto de componentes: módulos, modelos, nós, etc. Nesta Seção são apresentados excertos de código que demonstram como é possível manipular cada um dos componentes em tempo de execução, assim como obter informações relevantes sobre eles.

```
pbdCurrentL = ProjectBaseDirectory.getInstance(getOAPProject());
```

Código 5 - Obter pasta do projeto em tempo de execução

```
Iterable<? extends SModule> smdAllL = getProject().getModules();
foreach modCurrentL in smdAllL
{
  if (modCurrentL.getModuleName().equalsIgnoreCase(teModule_ValueL))
  {
    Iterable<SModel> mdlAllL = modCurrentL.getModels();
    foreach mdlCurrentL in mdlAllL
    {
      if (mdlCurrentL.getModelName().equalsIgnoreCase(teModule_ValueL + "."
+ enuModelP.getValue()))
      {
        mdlReturnL = mdlCurrentL;
        return mdlReturnL;
      }
    }
  }
}
```

Código 6 - Acesso aos mudulos do projeto em tempo de excução

No componente sistema foi criado um ENUM (dySMS_system_Structure) que contém uma lista das designações dos módulos e modelos de um projeto do tipo dySMS.config.

```
private nlist<> getRoots(dysms_system_Structure enuModuleP,
                         dysms_system_Structure enuModelP)
{
  nlist<> lstReturnL;
  model mdlReturnL = getModel(enuModuleP, enuModelP);
  getProject().getModelAccess().runReadAction({ =>
    lstReturnL = mdlReturnL.roots(<all>);
    return lstReturnL;
  });
  return lstReturnL;
}
```

Código 7 - Acesso à lista de modelos de um modulo

```
public SNode getRootElement(dysms_system_Structure enuCurrentP,
                            int inIndexP)
{
  SNode sndReturnL;
  nlist<> lstRootsL = getRoots_RunTime(enuCurrentP);
  getProject().getModelAccess().runReadAction({ =>
    sndReturnL = lstRootsL.get(inIndexP);
    return sndReturnL;
  });
  return sndReturnL;
}
```

Código 8 - Acesso ao nó raiz de um modelo

6.4.4.3 Execução sequencial de ações

Inúmeras funcionalidades do projeto são implementadas através da execução sequencial de vários passos consecutivos, cuja execução deve bloquear o acesso a outras operações do IDE e que para cada passo deve ser apresentado uma caixa de diálogo com o estado.

Para a implementação deste tipo de processos foi usado o padrão de design Pipeline, devidamente integrado com UI e funcionalidades da plataforma.

```
public void execute(final dysms_system_ModalTask mdtCurrentP) {

  mdtCurrentP.setSystem(this);

  Task.Modal modalTask = new Task.Modal(getOAPProject(),
                                        mdtCurrentP.getWindowName(), true)
  {

    @Override
    public void run(@NotNull() final ProgressIndicator indicator) {

      final ProgressMonitorAdapter adapter = new
            ProgressMonitorAdapter(indicator);
```

```
SRepository repository = getMPSProject().getRepository();

int stepValue = 1;

adapter.start(mdtCurrentP.getProgressName(),
              mdtCurrentP.getNumberSteps());

for (int inStepL = 0; inStepL < mdtCurrentP.getNumberSteps();
     inStepL++) {

  adapter.step(mdtCurrentP.getStepName());

  switch (mdtCurrentP.getStepType()) {
    case dysms_system_StepType.NORMAL :

      mdtCurrentP.executeStep();

    case dysms_system_StepType.READ_ACTION :

      WaitForProgressToShow.runOrInvokeAndWaitAboveProgress({ =>
        repository.getModelAccess().runReadAction(new Runnable() {
          public void run() {
            mdtCurrentP.executeStep();
          }
        });
      });

    case dysms_system_StepType.WRITE_ACTION :

      WaitForProgressToShow.runOrInvokeAndWaitAboveProgress({ =>
        repository.getModelAccess().runWriteAction(new Runnable() {
          public void run() {

            mdtCurrentP.executeStep();

          }
        });
      });

    case dysms_system_StepType.COMMAND :

      // The correct way to call command with progress is as follows
      // The dialog might not show up if the method for the usual
         read & write locks are used
      WaitForProgressToShow.runOrInvokeAndWaitAboveProgress({ =>
        repository.getModelAccess().executeCommand({ =>

          mdtCurrentP.executeStep();

        }); }, ModalityState.defaultModalityState());

  }

  adapter.advance(stepValue);

  // Next step
  mdtCurrentP.nextStep();

  // Check if progress is canceled
  if (adapter.isCanceled()) { return; }
```

```
      }

      adapter.done();

    }

  };

  // The execute() method of actions must be very quick
  // so every long calculation must be invoked outside of this method like
this:
  ApplicationManager.getApplication().invokeAndWait(
  { => ProgressManager.getInstance().run(modalTask);
  });
}
```

Código 9 - Execução sequencial de ações

Esta função faz uso do objeto nativo Task_Modal para garantir que o IDE não permite interação com o utilizar até a terminar o processo. Para indicar o estado de execução e permitir a interrupção do processo recorre ao objeto ProgressMonitor.

Para suportar a implementação do referido padrão foi criado na livraria atual uma interface designada dysms_system_ModalTask. Todos os passos de execução de um determinado processo implementam esta interface, conseguindo desta forma a abstração necessária para a criação de uma função genérica de execução.

6.4.4.4 Comunicação com o utilizador

Apesar de ser possível a utilização de funções nativas do JAVA para comunicar com o utilizador, na implementação do projeto atual foram adotadas as funcionalidades proporcionadas pelo IDE com esta finalidade.

```
public void showError(string teMessageP) {

  ToolWindowManager manager =
    ToolWindowManager.getInstance(getOAPProject());

  manager.notifyByBalloon("Messages", MessageType.ERROR, teMessageP);

}
```

Código 10 - Apresentação de mensagens fazendo uso dos recursos do IDE

A utilização de métodos nativos de I/O garante um funcionamento dos mesmos de uma forma mais segura, devidamente integrada com as funcionalidades de multiprocessamento e com um aspeto no ecrã adequado ao aspeto global do IDE.

7 Testes

Neste capítulo é apresentada a descrição dos vários processos realizados com o intuito de verificar que o sistema apresenta os comportamentos esperados, dado um conjunto de casos de uso, adequadamente selecionados a partir do domínio de execução.

7.1 Casos de uso

Na secção **Erro! Fonte de referência não encontrada. Erro! Fonte de referência não encontrada.** encontram-se descritos os casos de uso, que abrangem todas as funcionalidades existentes no projeto de software atual.

Dada a quantidade não muito extensa de casos de uso e linearidade subjacente ao funcionamento do sistema, é possível criar um fluxo de execução base e a partir deste derivar todas as conjugações necessárias, para garantir uma cobertura total das funcionalidades do sistema.

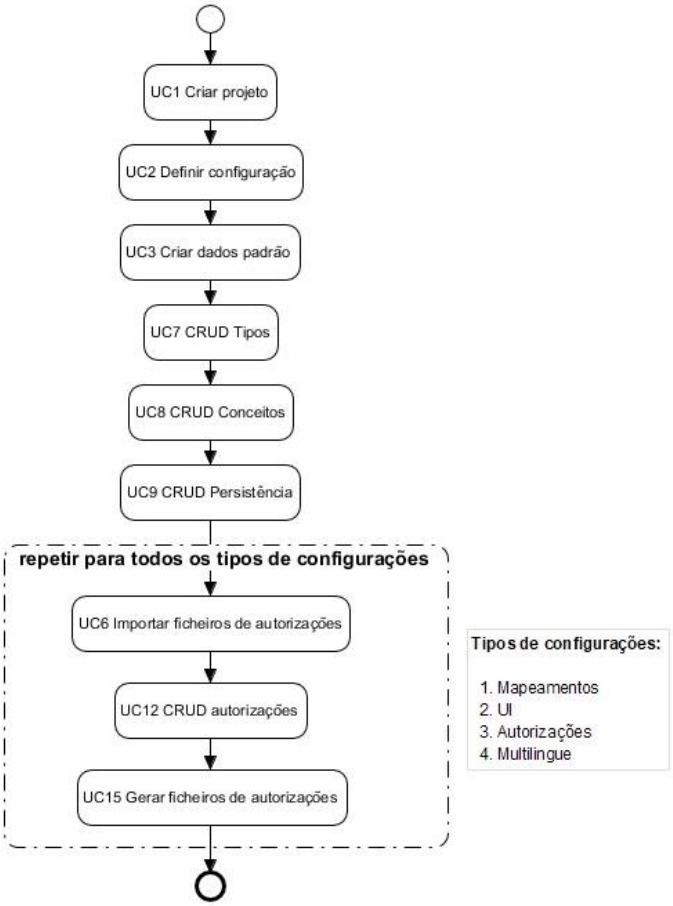

Figura 58 –Fluxo de execução base do sistema

7.2 Testes unitários

7.2.1 Desenvolvimento

Para que seja possível compreender os testes unitários é necessário apresentar uma breve descrição do processo de desenvolvimento.

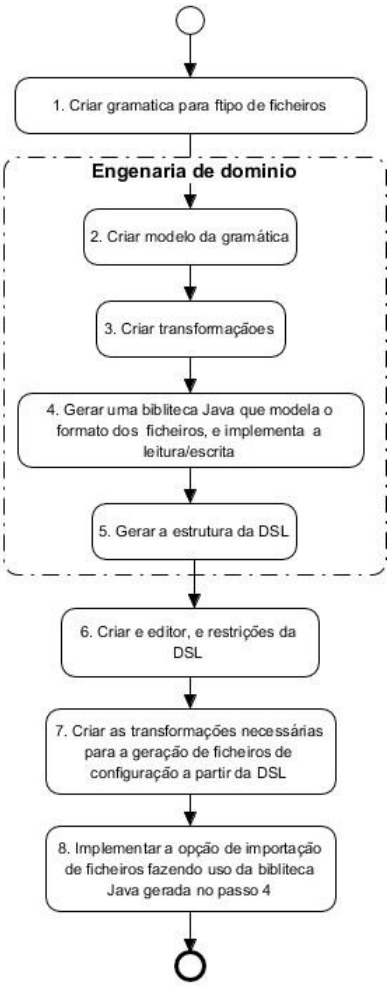

Figura 59 - Processo de desenvolvimento

Como é possível constatar na Figura 59 - Processo de desenvolvimento, o desenvolvimento de cada uma das funcionalidades de processamento, dos vários tipos de ficheiros configuração, tem inicio com a criação de uma gramatica para a respetiva linguagem.

Este artefacto é a base para a definição do modelo da engenharia de domínio, do meta-modelo da DSL e permite simplificar todo o processo, através da redução da linguagem original para um subconjunto da mesma. Este subconjunto apenas precisa da expressividade mínima suficiente para o sistema atual. Por exemplo, da totalidade da linguagem dos ficheiros de regras Drools apenas é necessária uma pequena parte para expressar as autorizações.

Além desta simplificação natural, podem e devem ser aplicadas restrições com vista a reduzir ainda mais a expressividade da mesma o que torna muito mais fácil o processo de configuração.

Outra das grandes vantagens é que com o auxilio da biblioteca Antlr aplicada á nossa gramática é possível automatizar o processo de serialização dos ficheiros e da criação da estrutura da DSL.

7.2.2 Testes

Com base no processo de desenvolvimento é possível identificar facilmente os vários testes a serem realizados e a incidência dos mesmos.

7.2.2.1 Gramática

A verificação da gramatica criada no ponto 1 do processo de desenvolvimento foi facilmente realizada através da criação de um conjunto de testes que realizavam o parsing de todos os ficheiros de configuração já existentes, que tinham sido criados manualmente.

```
@Test
void AccaoCP()
  throws Exception
{

  // 0. local variables
  String fileName;
  droolsLexer lexer;
  droolsParser parser;
  syntaxErrorListener listener;

  // 1. initialize
  fileName = "D:/bi/A1_eubeq/rules/AccaoCP.drl";

  // 2. create a lexer that feeds off of input CharStream
  lexer = new droolsLexer(CharStreams.fromFileName(fileName));

  // 3. create a buffer of tokens pulled from the lexer
  CommonTokenStream tokens = new CommonTokenStream(lexer);

  // 4. Create a parser that feeds off the tokens buffer
  parser = new droolsParser(tokens);

  // 5. create error listener
  listener = new syntaxErrorListener();

  // 6. assign listener
  parser.addErrorListener(listener);

  // 7. execute parser
  parser.program();

  // 8. verify
  assertTrue( listener.getSyntaxErrors().isEmpty() );

}
```

Código 11 - Exemplo de teste unitário

7.2.2.2 Biblioteca java

Para testar o funcionamento deste componente procedeu-se á criação de um novo conjunto de testes unitários, que na realidade são apenas uma variação dos testes anteriores. Estes novos testes usam a biblioteca para ler todos os ficheiros de regras, povoar as classes e gerar novos ficheiros. Para verificar o correto funcionamento da biblioteca vasta comparar os novos ficheiros gerados com os pré-existentes.

7.2.2.3 Modelo da DSL

No caso deste artefacto não foi necessário criar testes unitários, pois durante a criação do editor da DSL procedesse, naturalmente, a realização de uma verificação heurística da estrutura da DSL.

7.2.2.4 Transformações da DSL

Como nesta fase a biblioteca java já se encontra testada, os testes unitários, criados agora, apenas têm que comparar os ficheiros criados pelas transformações com os ficheiros criados pela biblioteca.

7.2.2.5 Importação dos ficheiros existentes

Esta funcionalidade usa a biblioteca Java para ler os dados dos ficheiros, que são usados para povoar os modelos das DSL. Como os dois modelos de dados são iguais o processo de povoamento é direto. No entanto para garantir a sua validação devem ser executadas novamente as transformações e os respetivos testes.

7.3 Testes funcionais

7.3.1 Execução automatizada

O sistema atual dispõe de um processo que permite a execução automatizada de todas as opções existentes, criada para demonstrar o produto e para servir de ajuda interativa para os utilizadores.

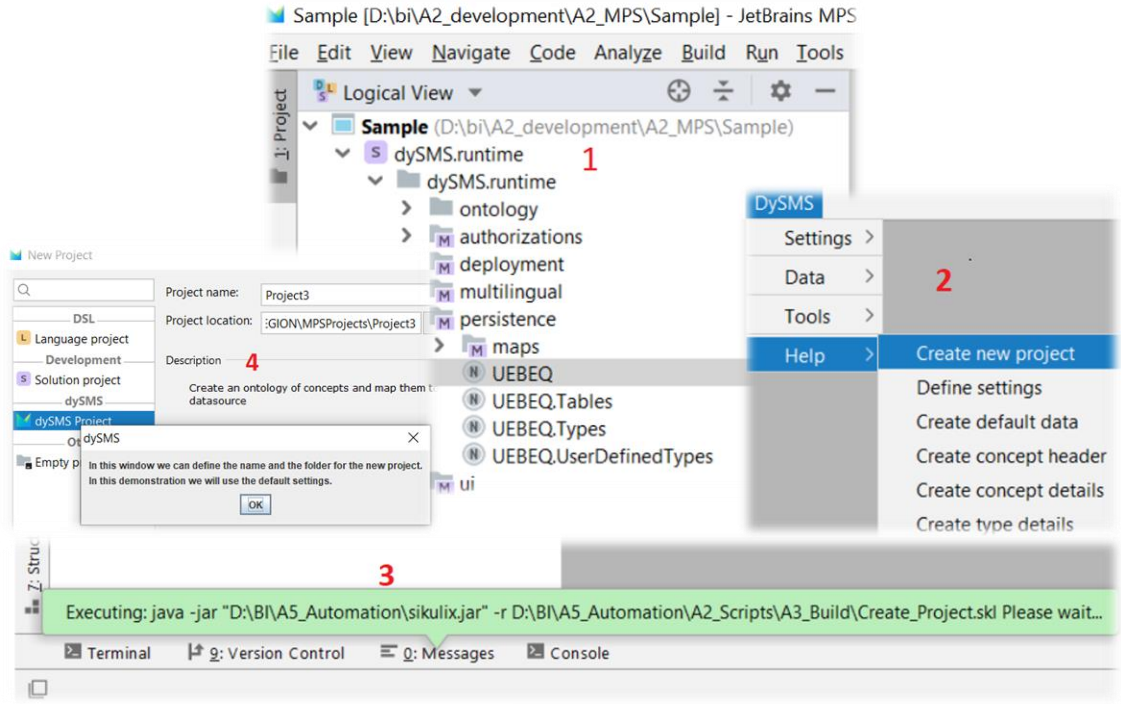

Figura 60 – Processo automatizado de execução

Dentro do IDE (1) onde são criados projetos de configuração, o operador acede á opção do menu principal designada dySMS (2), seleciona a opção ajuda e de seguida escolhe a funcionalidade que pretende executar de forma automatizada.

Assim que a automação tem início é apresentada uma mensagem informativa (3), no decorrer da execução são apresentadas mensagens explicativas (4).

7.3.2 Testes

O processo descrito na secção 7.3.1 Execução automatizada, foi implementado com recurso a uma biblioteca Java de automação, designada Sikulix (RainMan, 2019). Esta biblioteca dispõe de um editor que permite criar as automações e grava-las em forma de ficheiros. Em tempo de execução do sistema o referido ficheiro é carregado e executado pela mesma biblioteca.

Dado que o processo de automação se encontra integrado na aplicação, foi criado um conjunto de testes funcionais (JUnit5) que executam cada uma das automações procedem à aferição dos dados registados no sistema, para comprovar o correto funcionamento da funcionalidade correspondente.

8 Avaliação

Qualidade é um termo que pode ter diferentes interpretações e para se abordar a qualidade de software de maneira objetiva é necessário, obter um consenso em relação à definição de qualidade de software que está a ser utilizada (secção 2.2.11.1).

Uma vez estabelecidas as características relevantes para a qualidade no contexto do projeto atual, importa também salientar que a qualidade é interpretada de acordo com os diferentes pontos de vista das diferentes partes interessadas (secção 3.1). A administração da Digital Wind avalia os aspetos de conformidade em relação aos requisitos, aspetos comerciais, económicos e de planeamento. A equipa de desenvolvimento e os gestores de projeto do ISEP avaliam os aspetos de conformidade em relação á qualidade do design, da implementação e dos testes. Por fim as equipas de desenvolvimento e implementação da Digital Wind, na qualidade de utilizadores do sistema, avaliam o software sem conhecer seus aspetos internos e estão principalmente interessados na facilidade de utilização, no desempenho e na confiabilidade dos resultados.

Para que a avaliação seja mais efetiva é importante que o processo de avaliação (secção 2.2.11.2) seja bem definido, estruturado e que utilize um modelo de qualidade que permita estabelecer e avaliar requisitos de qualidade. Para tal foram adotadas as normas da série ISO/IEC 14598 em conjunto com a série ISO/IEC 9126. O projeto dySMS.config foi desenvolvido por uma equipa temporária e especificamente criada para o projeto. Devido a este facto não faz sentido avaliar a maturidade organizacional ou capacidade de áreas de processo para estabelecer prioridades para melhoria. Logo não foi realizada avaliação na área da qualidade do processo de software.

As secções seguintes descrevem cada uma das fases do processo de avaliação de acordo com o padrão ISO/IEC 14598-1.

8.1 Estabelecimento de requisitos de avaliação

Nesta fase é necessário garantir que os requisitos sejam transformados em características de qualidade em concordância com o modelo de qualidade da ISO/IEC 9126-1. Como se explica de seguida.

1) Propósito da avaliação: define a parte do padrão a aplicar (14598-3, 14598-4, 14598-5), no caso concreto do projeto atual como o produto vai ser desenvolvido (não comprado) aplica-se a parte: 14598-3.

2) Identificar o tipo de produto a ser avaliado: esta aplica-se ao dySMS.config, que é uma aplicação de desktop, mono utilizador cujos dados serão persistidos sobre a forma de ficheiros.

3) Especificar modelo de qualidade: norma ISO/IEC 9126-1. Esta norma postula que o modelo deve ser dividido em duas partes: (1) modelo de qualidade para características externas e internas e (ii) modelo de qualidade para qualidade aquando da sua utilização em ambiente real. A primeira parte do modelo deve ser hierarquicamente decomposto por meio de características e subcaracterísticas, as quais podem ser usadas como uma lista de verificação de tópicos relacionados com qualidade. Na Tabela 13, encontra-se a lista de características de qualidade relevantes para o projeto dySMS.config. Esta lista é um subconjunto das características propostas pela norma ISO/IEC 9126-1, das quais foram retiradas as características que não tem aplicação no projeto atual.

Tabela 13 - Modelo de qualidade adotado

Característica	Descrição
Funcionalidade	
Adequação	Fazer o que é apropriado
..Acurácia	Fazer o que foi proposto de forma correta
Interoperabilidade	Interage bem com o dySMS
Confiabilidade	
Maturidade	Não apresentar falhas frequentes
Tolerância a Falhas	Quando ocorrer falhas, o software deve reagir bem
..Recuperabilidade	Ser capaz de recuperar dados em caso de falhas
Usabilidade	
..Inteligibilidade	Ser fácil de entender o conceito lógico e sua aplicabilidade
..Apreensibilidade	Ser fácil de aprender a usar
..Operacionalidade	Ser fácil de usar e controlar
Eficiência	
Comportamento em relação ao tempo	Na execução das funções, ser pequeno o tempo de resposta
Manutenibilidade	
..Modificabilidade	Ser fácil de modificar e adaptar
Estabilidade	Não existir risco de efeitos inesperados quando se fizer alterações
Portabilidade	
Adaptabilidade	Interagir com Windows, Mac e Linux
Instalação	Ser fácil de instalar

8.2 Especificação da avaliação

Segundo a norma ISO/IEC 9126-1 nesta fase é necessário definir métricas para as características de qualidade, definir os níveis de pontuação para as métricas e estabelecer de critérios para julgamento.

8.2.1 Selecionar métricas

Em consonância com o modelo de avaliação previamente elaborado, foi desenvolvido um questionário com perguntas chave que permitem avaliar cada uma das características identificadas.

Questionário de avaliação de qualidade

	Questão	Valor
1	Os requisitos identificados abrangem a totalidade das funcionalidades desejadas?	
2	As funcionalidades implementadas funcionam com exactidão?	
3	O dySMS.config interage de forma correcta com o dySMS?	
4	Com que frequência apresenta falhas por defeitos no software?	
5	Quando ocorrem falhas, o software reage graciosamente?	
6	O dySMS.config é capaz de recuperar dados em caso de falhas?	
7	É fácil entender a lógica subjacente ao dySMS.config?	
8	O dySMS.config é fácil de aprender a usar?	
9	O dySMS.config é fácil operar e controlar?	
10	As funcionalidades têm um bom tempo de resposta?	
11	Dado o tempo de resposta aos pedidos de alterações, acha que o software é facil de modificar?	
12	Quando existem atualizações, com que frequência detectou efeitos inesperados?	
13	O dySMS.config funciona bem em Windows, Mac e Linux?	
14	O dySMS.config é facil de instalar?	

Figura 61 - Questionário de avaliação de qualidade

8.2.2 Níveis de pontuação para as métricas

Os valores aceites nas respostas são valores inteiros no intervalo de 0 a 5, considerando a escala: 0 – Não concordo até 5 – Concordo plenamente.

Tendo em conta a importância relativa atribuída a cada um dos critérios, de acordo com os requisitos funcionais e com as espectativas que foram demonstradas pelas várias partes interessadas, foi criada a Tabela 14, de ponderação das várias questões.

Com base nesta tabela será utilizada a seguinte formula para calcular a nota final:

$$nota\ final = (v1 * p1) + \cdots + (v14 * p14) \quad (1)$$

onde v_n representa os valores das notas das respetivas perguntas e o p_n representa o valor de ponderação para a pergunta.

Tabela 14 -Ponderação das questões

Característica	Questão	Ponderação
Funcionalidade		
Adequação	1	2
Acurácia	2	2
Interoperabilidade	3	1
Confiabilidade		
Maturidade	4	1
Tolerância a Falhas	5	1
Recuperabilidade	6	2
Usabilidade		
Inteligibilidade	7	1
Apreensibilidade	8	1
Operacionalidade	9	2
Eficiência		
Comportamento em relação ao tempo	10	1
Manutenibilidade		
Modificabilidade	11	1
Estabilidade	12	2
Portabilidade		
Adaptabilidade	13	1
Instalação	14	1

Quando a avaliação for realizada, a nota total de cada uma das características deve ser normalizada utilizando a fórmula:

$$\left(\frac{nota\ total\ da\ caracteristica_n}{nota\ maxima\ da\ caracteristica_n}\right) \times 10 \quad (2)$$

, onde a nota total da característica n representa a soma ponderada das notas de todas as subcategorias da categoria n, e a nota máxima é obtida pela fórmula (1) assumindo o valor 5 para as notas de todas as subcategorias da categoria n.

O grau de satisfação das características está sintetizado num gráfico em faixas.

Figura 62 - Escala das características

8.2.3 Critérios para julgamento

Para o apurar o resultado da avaliação, cada característica deve ser considerada individualmente e não se aceita nenhum valor inferior a REGULAR. A partir da média dos resultados de todas as perguntas é possível obter uma avaliação global.

8.3 Projetar a avaliação

A fase de projeto da avaliação consiste na documentação dos procedimentos que serão utilizados pelo avaliador para executar a medição.

Nesta fase os recursos necessários como, por exemplo, pessoas e técnicas, bem como a sua alocação devem ser especificados para as diferentes atividades durante a fase de execução da avaliação.

Como foi explicado na seção 1.1 Contexto, o projeto dySMS.config é desenvolvido no âmbito de um projeto I&D cuja data de termino é 30/06/2019. O processo de avaliação do referido projeto será realizado durante o ultimo mês do referido projeto de acordo com as informações apresentadas na Tabela 15..

Tabela 15 - Planeamento da avaliação

Tarefa	Inicio	Fim	Artefactos
Obter as medidas	15/06/2019 Técnico do ISEP Gestor Digital Wind Equipa implantação digital Wind Equipa de desenvolvimento Digita Wind	15/06/2019	Inquérito
Comparar com critérios	16/06/2019 Técnico do ISEP	20/06/2019	Folha de Excel
Julgar os resultados	21/06/2019 Técnico do ISEP	30/06/2019	Folha de Excel

Importa explicar que as datas apresentadas pretendem apenas delimitar os períodos de realização de cada tarefa. O facto de algumas delas apontarem para dias não uteis não é significativo.

8.4 Execução da Avaliação

Na fase de execução da avaliação, as métricas selecionadas são aplicadas ao produto de software, obtendo-se os valores nos níveis de pontuação.

Estes valores medidos são comparados com os critérios para julgamento determinados anteriormente.

Durante esta fase do processo de avaliação foram realizados 23 inquéritos, abrangendo a totalidade das partes interessadas, cujas médias dos resultados para cada uma das perguntas se encontra na Tabela 16..

Tabela 16 - Resultados da avaliação

Característica	Questão	Resultado
Funcionalidade		
Adequação	1	20
Acurácia	2	18
Interoperabilidade	3	20
Confiabilidade		
Maturidade	4	14
Tolerância a Falhas	5	13
Recuperabilidade	6	18
Usabilidade		
Inteligibilidade	7	17
Apreensibilidade	8	15
Operacionalidade	9	15
Eficiência		
Comportamento em relação ao tempo	10	12
Manutenibilidade		
Modificabilidade	11	18
Estabilidade	12	16
Portabilidade		
Adaptabilidade	13	14
Instalação	14	18

A partir dos resultados apurados é possível constatar que em todas as características o resultado nunca foi abaixo de regular e dada a media global de 16,58 o resultado é considerado Bom.

9 Conclusão

Neste capítulo são analisados os resultados atingidos na realização do projeto, tendo em conta os objetivos definidos.

Tendo em conta à análise dos resultados atingidos, a informação e comentários recolhidos na fase de testes de aceitação e de avaliação, na segunda secção deste capítulo são apresentados vários pontos onde foram identificadas necessidades de melhoria assim como são identificadas propostas de novas funcionalidades que não foram consideradas no levantamento inicial de requisitos.

9.1 Resultados

Os resultados do trabalho realizado podem ser avaliados da perspetiva prática e académica.

9.1.1 Prática

A parte prática corresponde ao software desenvolvido e tendo em conta que foi possível implementar todos os requisitos identificados, que foram aplicadas com sucesso as metodologias e padrões escolhidos e que o trabalho foi realizado dentro dos prazos é possível concluir de forma empírica que o trabalho foi realizado com sucesso.

Formalmente foi realizado o processo de avaliação descrito no capitulo 8, cujo resultado atesta a boa qualidade do trabalho realizado no que respeita tanto aos requisitos funcionais como aos não funcionais.

9.1.2 Académicos

O bom resultado do trabalho nesta área pode ser atestado pela duas publicações cientificas que foram publicas tendo por base o processo de investigação realizado.

9.2 Considerações finais

Nesta secção são apresentadas conclusões de natureza pessoal, baseadas na experiência resultante da aplicação prática, no projeto, dos conceitos teóricos adquiridos no processo investigação.

Embora abordem temáticas que constam de outras secções, importa salientar que aqui, ao contrário das outras seções, é apresentada uma analise qualitativa e subjetiva.

9.2.1 Model driven design

Como se pode constatar, tanto os padrões adotados como as relações entre eles, são na sua maioria comuns aos utilizados a outras metodologias. No entanto ao fazermos uso destes elementos tendo por base os modelos, a estrutura e formalização que estes aportam ao processo, torna possível uma maior automatização de processos, o que por sua vez permite ganhos significativos de eficiência e produtividade.

9.2.2 Qualidade

O desenvolvimento da aplicação procura aportar à mesma a capacidade de ser estendida de um modo ágil e sem impactar as funcionalidades já implementadas e o respetivo fluxo de execução. Para tal contribui a utilização de vários padrões de design como o Pipeline, integrados numa arquitetura em camadas com uma divisão de responsabilidades consistente, que resultam numa aplicação confiável e de fácil manutenção.

O encapsulamento das funcionalidades de interação com o API da plataforma MPS, no serviço System, garante a separação e isolamento deste crosscutting concern com o intuito de garantir uma fácil adaptação futura a novas versões da plataforma.

A utilização de DSL e MDE garante que o foco do seu design e implementação se encontra no domínio do negócio. Esta abordagem oferece garantias de uma grande flexibilidade e tolerância a alterações futuras relacionadas com a esfera de conhecimento da área ou das regras de negócio. Permitindo inclusive que o software atual possa ser adaptado com facilidade a novas áreas de negócio onde seja necessária a gestão de configurações.

Complementarmente, a implementação em JAVA da plataforma onde se integra o projeto atual, acresce valor à aplicação, no sentido de permitir a sua execução em diferentes sistemas operativos com o mesmo código.

O caráter standalone da aplicação leva a que qualquer correção/atualização da aplicação envolva novo processo de instalação no dispositivo.

Em termos de disponibilidade, uma vez que a aplicação é executada na máquina do Utilizador e não possui dependências externas, esta está apenas dependente do estado respetivo do dispositivo.

9.2.3 JetBrains MPS

MPS é um ambiente poderoso para a engenharia linguística. Embora nem todas as suas características sejam únicas, a combinação de composição flexível e da liberdade de notação como consequência da abordagem projecional é certamente convincente. Também é importante enfatizar que a ferramenta é escalável para tamanhos de programa realistas, o editor é muito utilizável, e integra-se bem com as ferramentas de controlo de versão existentes

(diff e Merge é fornecido ao nível da sintaxe concreta). No mínimo, a ferramenta é um ambiente muito bom para a experimentação com DSL no contexto da pesquisa acadêmica e industrial.

A maior desvantagem do MPS é a sua curva de aprendizagem não-trivial. Porque funciona de forma tão diferente da maioria dos ambientes de engenharia de linguagem tradicionais, e porque aborda tantos aspetos das linguagens (inclui sistemas de tipo, fluxo de dados e refactoring do código) dominar as ferramentas requer um investimento significativo em termos de tempo. É essencial, que este investimento seja reduzido por uma melhor documentação e padrões melhores, para manter as coisas simples e simples e as coisas complexas geriveis.

O formato dos ficheiros de dados utiliza o padrão XML mas os elementos e as propriedades são bastante complexos o que dificulta a compreensão e eventual alteração dos mesmos.

Nos referidos ficheiros são usados apontadores para os conceitos registados num formato incompreensível.

Quando um objeto é apagado e um novo é criado para o substituir, todas a referências ao objeto apagado noutros objetos tornam-se inválidas e é necessário percorrer todo o projeto e corrigir cada uma das referências manualmente.

Não é possível gerar novas instâncias de modelos fazendo uso das transformações de modelos. As transformações M2M apenas geram novas instâncias como artefactos intermédios do processo de geração de texto.

O editor projecional fica muito lento à medida que o conteúdo editado escala. Com 1000 tabelas associadas a uma fonte de dados o tempo de resposta é sofrível.

O debug é rudimentar.

O processo de geração automática de ficheiros de build não consegue encontrar todas as referências necessárias e obriga a intervenção humana.

Apenas permite definir restrições personalizadas para campos que contém referências. Deveria ser possível faze-lo também para campos que contém elementos agregados.

O processo de reengenharia do código é muito bom, muito superior ao de um editor normal.

9.3 Trabalho futuro

Cada um dos itens apresentado é classificado de acordo com a sua natureza com o intuito de agrupar e caracterizar todos os itens.

9.3.1 Melhoria 1 – Estrutura dos tipos coleções

Neste momento a estrutura de dados que suporta a representação dos vários tipos de coleções do Java, obriga a que seja criado um tipo de dados para cada coleção que agrega dados de tipos diferentes, apesar de ser o mesmo tipo de coleção (1).

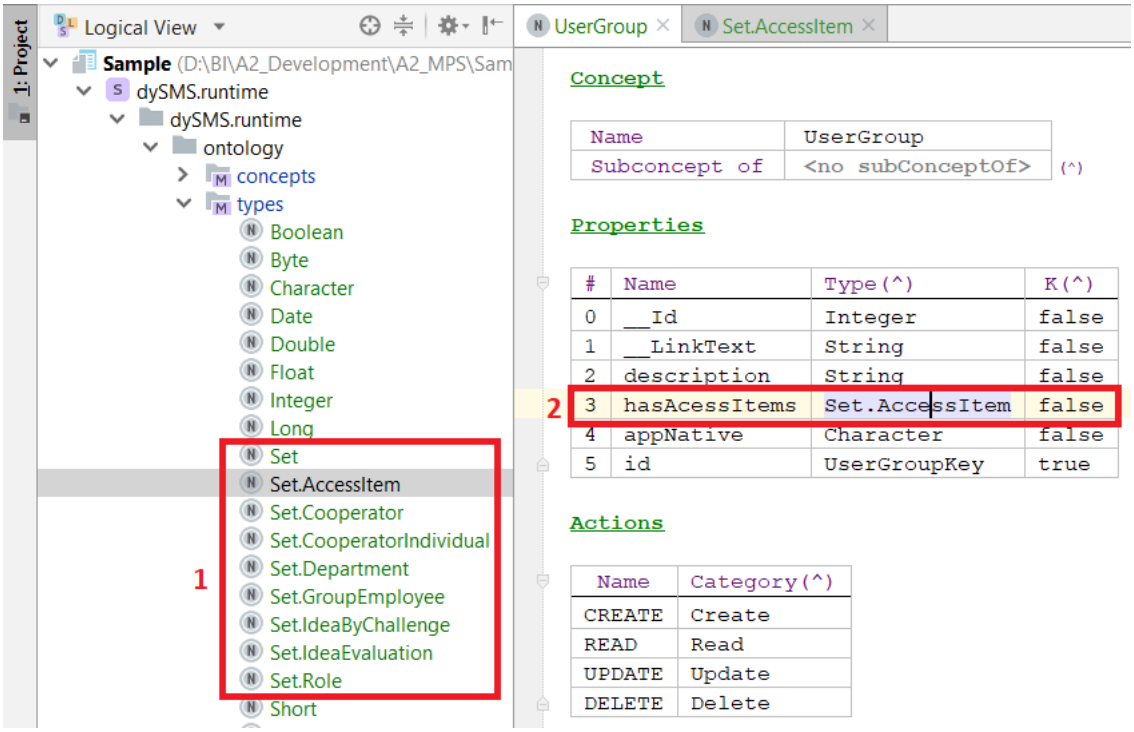

Figura 63 - Tipos de dados coleção

Talvez fizesse mais sentido apenas definir um tipo de dados para cada tipo de coleção e na definição das propriedades dos conceitos (2) acrescentar uma nova propriedade que identificasse a coleção além do tipo.

9.3.2 Melhoria 2 – Acrescentar cardinalidade às propriedades dos conceitos

Neste momento quando o operador cria mapeamentos, existe uma funcionalidade que permite automatizar parcialmente o preenchimento dos dados a partir dos dados previamente definidos no conceito. No entanto esta opção apenas consegue identificar a chave primária e as propriedades do conceito que têm relações simples com os campos da tabela.

Se na definição dos conceitos fosse possível acrescentar informação sobre a cardinalidade de cada uma das propriedades, esta informação iria permitir preencher automaticamente bastantes mais dados do mapeamento.

9.3.3 Melhoria 3 – Importação das relações entre tabelas da BD relacional

Se a aplicação dySMS.config fosse capaz de importar a informação acerca das chaves primárias e estrangeiras a partir da BD, o âmbito da opção de preenchimento automático dos mapeamentos, referida no ponto anterior poderia ser alargado para abranger não só as propriedades do conceito, mas também os campos das tabelas. Previamente é necessário apurar se a base de dados da aplicação UEBE.Q contém este tipo de informação.

9.3.4 Melhoria 4 – Criar conceitos a partir da informação de uma tabela

Foi possível constatar a partir da observação do uso da aplicação, que antes de iniciar o processo de criação de novos conceitos e respetivos mapeamentos são criadas na BD as tabelas respetivas. Neste contexto operacional faz sentido acrescentar uma opção que permita automatizar a criação de conceitos a partir da informação das tabelas.

9.3.5 Domínio aplicacional 1 – Suporte para novos tipos de fontes de dados

Embora tenha existido consenso entre as partes interessadas acerca de que nesta fase do projeto o âmbito da aplicação se restringisse apenas ao processamento de dados sobre a forma de BD relacionais, foi possível notar que no futuro será conveniente alargar o âmbito de ação a outras estruturas de dados.

Prevê-se que será necessário a obtenção de informação de formatos distintos para permitir à DigitalWind oferecer aos seus clientes a integração de informação pré-existente nos referidos clientes com os dados da aplicação UEBE.Q. Como esta informação pode não se encontrar estruturada sobre a forma de BD relacionais, eventualmente será necessário tratar outras fontes de dados como: Ficheiros Excel, Documentos de texto, etc.

Esta extensão ao domínio aplicacional requer, no entanto a constatação de casos concretos que permitam justificar os custos de desenvolvimento e definir especificamente os tipos de ficheiros a serem abrangidos.

9.3.6 Melhoria 5 – Acrescentar novos tipos de drivers para BD relacionais

Atualmente a aplicação apenas se encontra preparada para aceder a BD relacionais da Microsoft, no entanto facilmente se percebe que à medida que o número de utilizadores aumentar é bastante provável que se venham a encontrar situações em que será necessário aceder a dados em BD de outros fornecedores.

Como no caso anterior esta melhoria também requer a constatação de casos concretos que permitam justificar os custos de desenvolvimento e definir especificamente os fornecedores de BD a serem abrangidos.

9.3.7 Melhoria 6 – Alargar o subconjunto de especificações dos ficheiros hibernate suportadas pela aplicação

Apesar do pouco tempo de funcionamento da aplicação em ambiente real, já surgiu a necessidade de adaptar os mapeamentos para suportarem novas potencialidades do Hibernate, como por exemplo a utilização de fórmulas na especificação das propriedades dos conceitos.

Esta constatação permite prever que à medida que a utilização do projeto se intensificar, irá naturalmente ser necessário aumentar a capacidade de processar nova regras de configuração do Hibernate, para fazer face a novos problemas.

No entanto, estes tipos de melhorias apenas fazem sentido serem consideradas em face a problemas concretos, à medida que estes forem surgindo, visto que a funcionalidades mais importantes do hibernate já se encontram abrangidas no âmbito atual do funcionamento do software.

9.3.8 Melhoria 7 – Melhorar o mapeando de tipos

No estado atual da aplicação o suporte para o mapeamento dos tipos de dados Java e os tipos de dados do SQL é muito rudimentar e pode resultar em situações ambíguas. Este facto faz antever a necessidade de melhorias nesta área. No entanto por agora as capacidades existentes permitem fazer face a todas as situações encontradas, portanto esta melhoria não é considerada urgente.

9.3.9 Domínio aplicacional 2 – Acrescentar perfil para os distribuidores

Existe a pretensão de permitir que os distribuidores do software da DigitalWind, possam vir a poder estender as configurações originais, por forma a aumentar a sua autonomia.

Este requisito foi identificado oportunamente, como se pode constatar no capítulo de requisitos, e as potencialidades nativas da plataforma MPS no que respeita à extensão de componentes, a possibilidade de estender os conceitos já implementada e o método de implantação escolhido oferecem garantias para a futuro desenvolvimento pleno desta funcionalidade. No entanto a

aplicação ainda não contempla a gestão de operadores nem a possibilidade de estender os mapeamentos.

Tendo em conta estes pressupostos, será necessário fazer uma nova avaliação desta necessidade assim que a aplicação for considerada suficientemente amadurecida pela DigitalWind.

9.3.10 Domínio aplicacional 3 – Novos ambientes de desenvolvimento para os UI

Atualmente a aplicação apenas se encontra preparada para produzir ficheiros de configuração de UI para o ambiente de desenvolvimento ANGULAR, que é o ambiente em que o componente clientes da aplicação UEBE.Q se encontra desenvolvido. No entanto o design do software teve em consideração a possibilidade de virem a ser suportados outros ambientes.

Referências

ANI – Agência Nacional de Inovação, S., 2014. *IDT EM COPROMOÇÃO.* [Online]
Available at: https://ani.pt/incentivos/idt-em-co-promocao/
[Acedido em 12 10 2018].

Anon., 2018. *Open API - accessing models from code.* [Online]
Available at: https://confluence.jetbrains.com/display/MPSD20182/Open+API+-+accessing+models+from+code
[Accessed 10 09 2018].

Atkinson, C. & Kuhne, T., 2003. Model-driven development: a metamodeling foundation. *IEEE Software,* 20(5), pp. 36 - 41.

Booch, g., 1994. *Object-Oriented Analysis and Design with Applications.* 2nd ed. s.l.:Addison Wesley.

Bourque, P. & Fairley, R., 2014. *Guide to the Software Engineering Body of Knowledge,* s.l.: IEEE Computer Society.

Brambilla, M., Cabot, J. & Wimmer, M., 2017. *Model-Driven Software Engineering in Practice.* 2ª ed. s.l.:Morgan & Claypool.

Chang, C. & Keisler, H., 1990. *Model theory.* s.l.:ElseVier.

Chef, 2019. *CHEFF.* [Online]
Available at: https://www.chef.io/
[Acedido em 18 08 2019].

CMMI, 2018. *CMMI,* s.l.: CMMI Institute.

ConfigApp, 2019. *ConfigApp.* [Online]
Available at: www.configapp.com
[Acedido em 18 08 2019].

Consortium, W. W. W., 2004. *Web Services Technologies.* [Online]
Available at: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest
[Acedido em 09 10 2018].

Czarnecki, K., 2004. Unconventional Programming Paradigms. In: Heidelberg, ed. *Overview of Generative Software Development.* Berlin: Springer, pp. 326-341.

Czarnecki, K. & UW, E., 2000. *Generative programming: methods, tools, and applications.* s.l.:Addison-Wesley.

Deursen, A., Klint, P. & W, V. J. M., 2000. Domain-specific languages: an annotated bibliography. *Newsletter ACM SIGPLAN Notices,* 35 (6), pp. 26 - 36 .

Eclipse foundation, 2018. *LANGUAGE ENGINEERING FOR EVERYONE.* [Online]
Available at: http://www.eclipse.org/Xtext/
[Acedido em 02 10 2018].

Esposito, D. & Saltarello, A., 2014. *Discovering the Domain Architecture - The layered architecture,* s.l.: Microsoft Press.

Evans, E., 2003. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* s.l.:Addison-Wesley Professional.

Evans, E., 2004. *Domain driven vs Model Driven.* [Online]
Available at: http://dddcommunity.org/uncategorized/evans_2004/
[Acedido em 02 10 2018].

Favre, J.-M., 2005. *Megamodelling and Etymology.* [Online]
Available at: https://www.researchgate.net/publication/30815076_Megamodelling_and_Etymology
[Acedido em 02 10 2018].

Fowler, M., 2005. *Language Workbenches: The Killer-App for Domain Specific Languages?.* [Online]
Available at: https://www.martinfowler.com/articles/languageWorkbench.html
[Acedido em 26 09 2018].

Fowler, M., 2010. *Domain-Specific Languages.* 1st Edition ed. s.l.:Addison-Wesley.

Grady, R., 1992. *Practical Software Metrics for Project Management and Process Improvement.* s.l.:Prentice-Hall.

HAT, R., 2019. *Ansible.* [Online]
Available at: https://www.ansible.com/
[Acedido em 18 08 2019].

Intentional Software, l., 2017. *Technology.* [Online]
Available at: http://www.intentsoft.com/intentional-technology/
[Acedido em 2 10 2018].

ISO/IEC12207, 2008. *Systems and software engineering -- Software life cycle processes,* s.l.: International Organization for Standardization.

ISO/IEC14598-1, 1999. *Information technology -- Software product evaluation,* s.l.: International Organization for Standardization.

ISO/IEC25010, 2011. *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models,* s.l.: International Organization for Standardization.

Jacobson, I., Spence, I. & Bittner, K., 2011. *USE-CASE 2.0 EBOOK.* [Online]
Available at: https://www.ivarjacobson.com/sites/default/files/field_iji_file/article/use-case_2_0_jan11.pdf
[Acedido em 20 09 2019].

JetBrains, 2013. *Main page.* [Online]
Available at: https://www.jetbrains.com/mps/
[Acedido em 01 09 2019].

JetBrains, 2019. *How Does MPS Work?.* [Online]
Available at: https://www.jetbrains.com/mps/concepts/
[Acedido em 02 10 2018].

JetBrains, 2019. *http://usna86-techbits.blogspot.com/2012/11/uml-class-diagram-relationships.html.* [Online]
Available at: https://www.jetbrains.com/help/mps/mps-project-structure.html
[Acedido em 20 09 2019].

Kent, S. J., 2002. *IFM '02 Proceedings of the Third International Conference on Integrated Formal Methods.* London, Springer-Verlag.

Koen, P. et al., 2001. Providing Clarity and A Common Language to the “Fuzzy Front End”. *Research-Technology Management*, pp. 46-55.

Lakatos, E. M. & Marconi, M. d. A., 2010. Em: *Fundamentos de Metodologia Científica.* São Paulo: Atlas, p. 139.

Ling, L. & M. Tamer, Ö., 2009. *Encyclopedia of Database Systems.* s.l.:Springer-Verlag.

Ludewig, J., 2003. Models in software engineering – an introduction. *SoSyM,* 2(5).

Luz, M. & Silva, A., 2004. *Proceedings of the 3rd workshop in software model engineering.* s.l., IEEE Computer Society.

Marco Brambilla, J. C. a. M. W., 2017. *Model-Driven Software Engineering in Practice.* 2ª ed. s.l.:Morgan & Claypool.

Martin, R., 2012. *Código Limpo. Habilidades Práticas Do Agile Software.* 1 ed. s.l.:Alta Books.

mediaBOLD, P. A., 2017. *Tecnologia Obsoleta é o Principal Obstáculo à Transformação Digital.* [Online]
Available at: http://www.fujitsu.com/pt/about/resources/news/press-releases/2017/tecnologia-obsoleta-o-principal-obst-culo-transforma-o.html

Metacase, 2018. *Metacase.* [Online]
Available at: http://metacase.com/
[Acedido em 02 10 2018].

Microsoft, 2018. *SDK de Modelagem para Visual Studio - linguagens específicas ao domínio.* [Online]
Available at: https://docs.microsoft.com/pt-br/visualstudio/modeling/modeling-sdk-for-visual-studio-domain-specific-languages?view=vs-2017
[Acedido em 02 10 2018].

Monticore, 2018. *The MontiCore Language Workbench.* [Online]
Available at: http://www.monticore.de/
[Acedido em 02 10 2018].

Northern.tech, I., 2019. *CFEngine.* [Online]
Available at: https://cfengine.com/
[Acedido em 17 08 2019].

OMG, 2015. *UML Superstructure Specification, v2.5.* [Online]
Available at: https://www.omg.org/spec/UML/2.5/PDF
[Acedido em 01 09 2019].

Osterwalder, A. & Pigneur, Y., 2010 . *Business Model Generation;.* First ed. s.l.:John Wiley & Sons, Inc.

Pereira, N. & Baltarejo, P., 2016. *TMDEI Thesis Template.* [Online]
Available at: https://www.overleaf.com/latex/templates/tmdei-thesis-template-dei-slash-isep/dtvkwwtwzccc
[Acedido em 18 Setembro 2019].

Pressman, R. S., 2001. *Software Engineering: A Practitioner's Approach.* 5th ed. s.l.:McGraw-Hill Higher Education.

puppet, 2019. *puppet.* [Online]
Available at: https://puppet.com/
[Acedido em 18 08 2019].

RainMan, 2019. *Sikulix.* [Online]
Available at: http://sikulix.com/
[Acedido em 05 08 2019].

RedHat, 2019. *Hibernate Tools.* [Online]
Available at: https://hibernate.org/tools/
[Acedido em 18 08 2019].

RedHat, 2019. *Rule Workbench (IDE).* [Online]
Available at: http://www.jbug.jp/trans/jboss-rules3.0.2/ja/html/ch05.html
[Acedido em 18 08 2019].

Republica portuguesa, X. G., 2014. *O que é o Portugal 2020.* [Online]
Available at: https://www.portugal2020.pt/Portal2020/o-que-e-o-portugal2020
[Acedido em 12 10 2018].

Sanders, E. J. C., 1994. *Software Quality.* 1st ed. s.l.:ACM Press.

Silva, A. R. d., 2015. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, pp. 139 - 155.

StrateGeoxT, 2018. *SDF Home.* [Online]
Available at: http://strategoxt.org/Sdf
[Acedido em 02 10 2018].

Tolvanen, S. K. J., 2007. *Domain-Specific Modeling: Enabling Full Code Generation.* s.l.: John Wiley & Sons, Inc..

Vangheluwe, H., De Lara, J. & Mosterman, P., 2002. *An introduction to multi-paradigm modelling and simulation. In: Proceedings of the AIS'2002 conference.* s.l., s.n.

Wimmer, T., KühneGergely Mezei, E. & SyrianiHans Vangheluwe, M., 2009. *International Conference on Model Driven Engineering Languages and Systems.* Berlim, Heidelberg.

Zoho, 2019. *Configuration file management in Network Devices.* [Online]
Available at: https://www.manageengine.com/network-configuration-manager/configuration-file-management.html
[Acedido em 18 08 2019].

A. Anexos

A.1 Catalogo de elementos arquiteturais

A.1.1 Estereótipos

Os estereótipos são um dos mecanismos de extensibilidade UML. Permitem estender o vocabulário UML para criar novos elementos de modelo, derivados dos existentes, mas que possuem propriedades específicas adequadas para um domínio específico ou uso especializado (OMG, 2015).

Na Tabela 17 encontra-se a descrição dos estereótipos, que não fazem parte do padrão UML, usados nos vários diagramas contidos neste documento.

Tabela 17 – Estereótipos personalizados

Estereótipo	Descrição
Language	Este estereótipo identifica componentes que implementam uma DSL (Secção 2.2.8). Tipicamente um componente deste tipo agrega a estrutura da DSL, restrições, configurações de editores e processos de transformação M2M e M2T.
Structure	Este estereótipo identifica os elementos das DSL que são usados especificamente para definir a sua estrutura. Permitindo fazer a separação entre estes e os elementos usados para funcionalidades complementares como configuração de editores, definição de restrições ou definição de transformações.
Plugin	Este estereótipo identifica componentes que funcionam integrados em aplicações pré-existentes com o intuito de estender as funcionalidades existentes. A aplicação hospedeira provê serviços que o plugin pode usar, incluindo uma forma para a extensão se registrar a si mesma e um protocolo para a troca de informações. Os plugins dependem de tais serviços, e não trabalham de forma autónoma. Em contrapartida, a aplicação hospedeira é independente, de forma que é possível adicionar e atualizar os plugins dinamicamente, sem a necessidade de efetuar alterações na própria aplicação.

A.1.2 Cores

As classes abstratas são representadas com o fundo cinzento, para distinguir das classes normais que tem fundo branco.

A.2 Documento de definição de requisitos

O documento que se segue foi produzido pela empresa DigitalWind, durante a fase de levantamento de requisitos. Neste documento encontra-se uma descrição feita pela referida empresa das funcionalidades a serem implementadas no projeto dySMS.config.

dySMS

Dynamic Standards Management System

Considerações MetaInformação

1) PROJETO N.º	
2) TÍTULO	
3) AUTORES DIGITALWIND	
4) PALAVRAS-CHAVE	

5) PROJETO DYSMS - DYNAMIC STANDARDS MANAGEMENT SYSTEM	6) DATA ORIGINAL: DATA DA ÚLTIMA REVISÃO:

Cofinanciado por:
COMPETE 2020 PORTUGAL 2020 UNIÃO EUROPEIA Fundos Europeus Estruturais e de Investimento

Tabela de Conteúdos

1. INTRODUÇÃO 3
1.1. Componentes de Configuração 3
2. FICHEIROS DE CONFIGURAÇÃO 4
2.1. Mapeamento de entidades 4
2.2. Meta informação 5
2.2.1. JsonMetaInfo - ReadList 6
2.2.2. JsonMetaInfo - ReadInstance 7
2.2.3. JsonMetaInfo - Create 8
2.2.4. JsonMetaInfo - Update 10
2.3. Ações UI legada e nova 12
2.4. Resources (multilíngue) 12

1. INTRODUÇÃO

1.1. Componentes de Configuração

Um dos pontos essenciais em que assenta a abordagem de desenvolvimento do projeto DySMS é alta capacidade de configuração. Nesse sentido deverão ser utilizados uma série de ficheiros de configuração, gerados pela componente (DSL), nomeadamente ficheiros de mapeamento entre conceitos sistema legado vs entidades API de autorização e meta informação necessária à UI para uma implementação dinâmica das interfaces.

A lógica adotada será a existência de um ficheiro de meta informação por cada conceito definido.

2. FICHEIROS DE CONFIGURAÇÃO

2.1. Mapeamento de entidades

Esta componente tem como função de converter instâncias de conceitos provenientes da API de autorização (json) em instâncias de conceitos legados (classe, conhecidas pelo uebe.Q e respetivas dlls). Estes ficheiros apenas são necessários se se pretender inserir, atualizar ou eliminar instâncias do conceito na BD do uebe.Q.

Esta implementação tem em conta que:

1. Existe um ficheiro xml por conceito: *<nome do conceito>.xml*
2. A estrutura do ficheiro consiste em:

```
<?xml version="1.0" encoding="utf-8" ?>
<class name="<nome do conceito>">
   <property name="<prop. legada>" jsonproperty="caminho prop. json"/>
   <property name="<prop. legada>" jsonproperty="caminho prop. json"/>
   <property name="<prop. legada>" jsonproperty="caminho prop. json"
   complex="true">
      <property name="<prop. legada>" jsonproperty="caminho prop.
      json"/>
      …
   </property>
   …
</class>
```

3. Na estrutura definida temos:

3.1. Elemento “Class”

3.1.1. Atributo “Name” – Nome do conceito definido

3.2. Elemento “Property”

3.2.1. Atributo “Name” – Nome da propriedade do sistema legado, tipicamente o nome da coluna na tabela

3.2.2. Atributo “JsonProperty” - Nome do atributo definido através da definição de conceito na camada de autorização. Deve ter o caminho completo separado por pontos(.)

3.2.3. Atributo “Complex” – Booleano com a definição de que se trata de um parâmetro composto por outros parâmetros/atributos.

2.2. Meta informação

Componente que efetua a consulta aos ficheiros de configuração (gerados pela DSL) e que devolve a meta informação relativa a determinados conceito e ação. Ficheiros xml que definem para cada conceito e cada ação qual a meta informação associada (formulários, listagens, detalhes, …)

Esta implementação tem em conta que:

1. Existe um ficheiro xml por conceito: <*nome do conceito*>.*xml*
2. A estrutura do ficheiro consiste em:

```
<?xml version="1.0" encoding="utf-8" ?>
<concept name="<nome do conceito>">
   <action name="<acaoA>" jsonMetainfo="<metainformacao em json>"/>
   <action name="<acaoB>" jsonMetainfo="<metainformacao em json>"/>
   …
</concept>
```

3. Na estrutura definida temos:

3.1. Elemento “Concept”

3.1.1. Atributo “Name” – Nome do conceito definido

3.2. Elemento “Action”

3.2.1. Atributo “Name” – Nome da ação sobre o conceito. Atualmente pode assumir os seguintes valores base:

   a. READLIST
   b. READINSTANCE
   c. CREATE
   d. UPDATE

No conceito ideia foi alem destas ações definido tb a ação:

   a. UPDATESTATUS

3.2.2. Atributo “JsonMetaInfo” – Elemento JSON com a definição específica da ação para implementação na UI

2.2.1. JsonMetaInfo - ReadList

A informação presente neste bloco de Json é usada nas listagens dos diferentes conceitos. Tem como objetivo definir quais as colunas que devem ser apresentadas na listagem e as ações globais inerentes ao conceito que representam.

É apresentado em seguida o exemplo de json para listagem de ideias.

```
{
"columns":[
        {"key":"code", "order":1, "allowSort":true, "type":"text", "exportable":true},
        {"key":"title", "order":2, "allowSort":true, "type":"text", "exportable":true},
        {"key":"innovationType.description",    "order":3,    "allowSort":true,    "type":"text",
"exportable":true},
        {"key":"description", "order":4, "allowSort":false, "type":"text", "exportable":true},
        {"key":"entryDate", "order":5, "allowSort":true, "type":"date", "exportable":true},
        {"key":"status.status.description",    "order":6,    "allowSort":true,    "type":"text",
"exportable":true}
],
"defaultSort":{"column":4, "type":"desc"},
"actions":[ {"action":"CREATE", "label":"addNew", "order":1} ]
}
```

Esta estrutura Json tem em conta o seguinte:

1. Elemento “Columns” – Array com as colunas a apresentar. Cada elemento deste array pode ter os seguintes elementos
   1.1. Elemento “key” – identificação do atributo no conceito com o caminho completo separado por pontos(.)
   1.2. Elemento “order” – qual o ordinal na apresentação em colunas
   1.3. Elemento “allowSort” – booleano com a definição se é possível ordenar a coluna
   1.4. Elemento “type” – qual a representação do campo (ex. text, date, etc)
   1.5. Elemento “exportable” – booleano com a definição se a coluna deve ser exportável para CSV, PDF, etc
2. Elemento “defaultSort” – identificação de como deve ser efetuada a ordenação por omissão.
   2.1. Elemento “column” – identificação com o ordinal de qual a coluna que deve ser usada para fazer a ordenação. O ordinal começa em zero (0).
3. Elemento “Actions” – Array com as possíveis ações a realizar sobre o conceito. Cada elemento deste array pode ter os seguintes elementos
   3.1. Elemento “Action” – Nome da ação
   3.2. Elemento “Label” – Resource key que será utilizada na contextualização
   3.3. Elemento “Order” – qual o ordinal na apresentação das ações

2.2.2. JsonMetaInfo - ReadInstance

A informação presente neste bloco de Json é usada na visualização de instancias dos diferentes conceitos. Tem como objetivo definir quais os campos que devem ser apresentadas, qual a sua ordem, e a sua representação.

É apresentado em seguida o exemplo de json para visualização de ideia.

```
{'properties':[
        {'key':'code', 'order':1, 'type':'text'},
        {'key':'entryDate', 'order':2, 'type':'date'},
        {'key':'status.status.description', 'order':3, 'type':'text'},
        {'key':'title', 'order':4, 'type':'text'},
        {'key':'author.otherAuthors',              'order':5,              'type':'objectArray',
'concept':{'name':'Cooperator', 'idKey':'id', 'textKey':'name'}},
        {'key':'author.externalAuthor.externalAuthorName', 'order':5, 'type':'text'},
        {'key':'author.externalAuthor.externalAuthorContact', 'order':5, 'type':'text'},
        {'key':'description', 'order':6, 'type':'text'},
        {'key':'note', 'order':7, 'type':'text'},
        {'key':'source.description', 'order':8, 'type':'text'},
        {'key':'challenge',    'order':9,    'type':'object',    'concept':{'name':'Challenge',
'idKey':'id', 'textKey':'title'}},
        {'key':'innovationType.description', 'order':10, 'type':'text'},
        {'key':'documents', 'order':11, 'type':'documents'}
],
'actions':[
        {'action':'UPDATE', 'label':'edit', 'order':2},
        {'action':'UPDATESTATUS', 'label':'editStatus', 'order':3},
        {'action':'DELETE', 'label':'delete', 'order':4}
]}
```

Esta estrutura Json tem em conta o seguinte:

1. Elemento “Properties” - Array com os campos/colunas a apresentar. Cada elemento deste array pode ter os seguintes elementos
   - 1.1. Elemento “key” – identificação do atributo no conceito com o caminho completo separado por pontos(.)
   - 1.2. Elemento “order” – qual o ordinal na apresentação em colunas
   - 1.3. Elemento “type” - qual a representação do campo (ex. text, date, objectArray, documents, etc)
   - 1.4. Elemento “concept” – elemento de ligação a outro conceito que pode ter os seguintes elementos
     - 1.4.1. Elemento “name” – qual o nome do conceito a ligar
     - 1.4.2. Elemento “idkey” – qual o atributo do conceito a ligar que é chave
     - 1.4.3. Elemento “textKey” – qual o atributo do conceito a ligar que tem a representação em texto

2. Elemento “actions” – Array com as possíveis ações a realizar sobre cada instância do conceito. Cada elemento deste array pode ter os seguintes elementos
   2.1. Elemento “Action” – Nome da ação
   2.2. Elemento “Label” – Resource key que será utilizada na contextualização
   2.3. Elemento “Order” – qual o ordinal na apresentação das ações

2.2.3. JsonMetaInfo - Create

A informação presente neste bloco de Json é usada no formulário de criação de instância dos diferentes conceitos. Tem como objetivo definir quais os campos que devem ser apresentadas, qual a sua ordem, e a sua representação bem como regras de validação de visualização e preenchimento obrigatório.

É apresentado em seguida o exemplo de json para criação de ideia.

```
{'tabs':[
    {'order':1, 'key':'Idea', 'active':true},
    {'order':2, 'key':'Challenge', 'active':false},
    {'order':3, 'key':'Document', 'active':false}
],'form':[
    {'order': 1, 'tab': 1, 'key':'title', 'required': true, 'type': 'text'},
    {'order': 2, 'tab': 1, 'key':'entryDate', 'required': true, 'type': 'dateToday'},
    {'order': 3, 'tab': 1, 'key':'author.type', 'type': 'radioButton',
'options':[{'key':'i', 'value':'Internal'},{'key':'e', 'value':'External'},{'key':'a',
'value':'Anonymous'}]},
    {'order': 4, 'tab': 1, 'key':'author.otherAuthors', 'required': true, 'type':
'multipleSelectSticky', 'requiredCondition':{'field': 'author.type','operator':'==',
'condition':'i'}, 'condition':{'field': 'author.type','operator':'==', 'condition':'i'},
    'optionsSource':{'concept':'Cooperator', 'valueKey':'id', 'textKey': 'name'}},
    {'order': 5, 'tab': 1, 'key':'author.externalAuthor.externalAuthorName', 'required':
true, 'type': 'text', 'requiredCondition':{'field': 'author.type','operator':'==',
'condition':'e'}, 'condition':{'field': 'author.type','operator':'==', 'condition':'e'}},
    {'order': 6, 'tab': 1, 'key':'author.externalAuthor.externalAuthorContact', 'required':
true, 'type': 'text', 'requiredCondition':{'field': 'author.type','operator':'==',
'condition':'e'}, 'condition':{'field': 'author.type','operator':'==', 'condition':'e'}},
    {'order': 7, 'tab': 1, 'key':'author.confidential', 'type': 'checkbox'},
    {'order': 8, 'tab': 1, 'key':'confidential', 'type': 'checkbox'},
    {'order': 9, 'tab': 1, 'key':'description', 'required': true, 'type': 'textarea'},
    {'order': 10, 'tab': 1, 'key':'note', 'type': 'textarea'},
    {'order': 11, 'tab': 1, 'key':'challenge.id', 'type': 'singleSelect',
'optionsSource':{'concept':'Challenge', 'valueKey':'id', 'textKey': 'title'}},
    {'order': 12, 'tab': 1, 'key':'source.id', 'type': 'singleSelect',
'optionsSource':{'concept':'Source', 'valueKey':'id', 'textKey': 'description'}},
    {'order': 13, 'tab': 1, 'key':'innovationType.id', 'type': 'singleSelect',
'optionsSource':{'concept':'InnovationType', 'valueKey':'id', 'textKey': 'description'}},
    {'order': 14, 'tab': 3, 'key':'documents', 'type': 'documents'},
    {'order': 15, 'tab': 2, 'key':'ChallengeOthers', 'required': true, 'type':
'radioButton', 'options':[{'key':'0', 'value':'DontChallenge'},{'key':1,
'value':'SendToAll'},{'key':2, 'value':'SelectCooperators'}]},
    {'order': 16, 'tab': 2, 'key':'Message', 'type': 'textarea', 'condition':{'field':
'ChallengeOthers','operator':'!=', 'condition':'0'}},
    {'order': 17, 'tab': 2, 'key':'ChallengedCooperators', 'required': true, 'type':
'multipleSelect', 'requiredCondition':{'field': 'ChallengeOthers','operator':'==',
'condition':'2'}, 'condition':{'field': 'ChallengeOthers','operator':'==', 'condition':'2'},
    'optionsSource':{'concept':'Cooperator', 'valueKey':'id', 'textKey': 'name'}}
]}
```

Esta estrutura Json tem em conta o seguinte:

1. Elemento “Tabs” - Array com os tabs a apresentar. Este elemento pode ser facultativo. Cada elemento deste array pode ter os seguintes elementos
   1.1. Elemento “order” – qual o ordinal na apresentação das tabs

1.2. Elemento “key” – resource key que será utilizada na contextualização

1.3. Elemento “active” – booleano com indicação se a tab está ativa por omissão.

2. Elemento “Form” – Array com os campos a apresentar. Cada elemento deste array pode ter os seguintes elementos

2.1. Elemento “order” - qual o ordinal na apresentação dos campos

2.2. Elemento “tab” – qual o ordinal da tab em que o campo deve ser apresentado

2.3. Elemento “key” - resource key que será utilizada na contextualização e que deve coincidir com o nome do atributo no conceito (separado por pontos)

2.4. Elemento “required” – booleano com indicação se o campo é obrigatório

2.5. Elemento “type” - qual a representação do campo (ex. text, date, dateToday, radioButton, multipleSelectSticky, checkbox, textarea, singleSelect, documents, multipleSelect, etc)

2.6. Elemento “requiredCondition” – Elemento composto que indica qua a condição de obrigatoriedade. Este elemento é constituído pelos seguintes elementos

2.6.1. Elemento “field” – qual o campo do qual depende no formulário

2.6.2. Elemento “operator” – qual o operador de comparação de valor com o campo identificado

2.6.3. Elemento “condition” – qual o valor que torna a condição verdadeira

2.7. Elemento “condition” – Elemento composto que determina as condições de visibilidade do campo, ou seja, só é visivel se determinado campo assumir determinado valor. Este elemento é constituído pelos seguintes elementos

2.7.1. Elemento “field” – qual o campo do qual depende no formulário

2.7.2. Elemento “operator” – qual o operador de comparação de valor com o campo identificado

2.7.3. Elemento “condition” – qual o valor que torna a condição verdadeira

2.8. Elemento “optionsSource” – Elemento composto que indica como deve ser alimentado as opções de select. Este elemento é é composto pelos seguintes elementos

2.8.1. Elemento “concept” – qual o nome do conceito a ligar

2.8.2. Elemento “valueKey” – qual o atributo do conceito a ligar que é chave

2.8.3. Elemento “textKey” – qual o atributo do conceito a ligar que tem a representação em texto

2.9. Elemento “options” – Array com as opções possíveis para um elemento do tipo “radioButton”. Cada elemento deste array pode ter os seguintes elementos

2.9.1. Elemento “Key” – valor/chave da opção

2.9.2. Elemento “value” - resource key que será utilizada na contextualização

2.2.4. JsonMetaInfo - Update

A informação presente neste bloco de Json é usada no formulário de edição de instância dos diferentes conceitos. Tem como objetivo definir quais os campos que devem ser apresentadas, qual a sua ordem, e a sua representação bem como regras de validação de visualização e preenchimento obrigatório.

É apresentado em seguida o exemplo de json para edição de ideia.

```
{'tabs':[
    {'order':1, 'key':'Idea', 'active':true},
    {'order':2, 'key':'Challenge', 'active':false},
    {'order':3, 'key':'Document', 'active':false}
],'form':[
    {'order': 1, 'tab': 1, 'key':'code', 'type': 'text', 'disabled': true},
    {'order': 2, 'tab': 1, 'key':'title', 'required': true, 'type': 'text'},
    {'order': 3, 'tab': 1, 'key':'entryDate', 'required': true, 'type': 'dateToday'},
    {'order': 4, 'tab': 1, 'key':'author.type', 'type': 'radioButton', 'options':[{'key':'i',
'value':'Internal'},{'key':'e',      'value':'External'},{'key':'a',      'value':'Anonymous'}],
'disabled': true},
    {'order': 5, 'tab': 1, 'key':'author.otherAuthors', 'type': 'multipleSelectSticky',
'condition':{'field': 'author.type','operator':'==', 'condition':'i'},
    'optionsSource':{'concept':'Cooperator', 'valueKey':'id', 'textKey': 'name'}, 'disabled':
true},
    {'order': 6, 'tab': 1, 'key':'author.externalAuthor.externalAuthorName', 'type': 'text',
'condition':{'field': 'author.type','operator':'==', 'condition':'e'}, 'disabled': true},
    {'order': 7, 'tab': 1, 'key':'author.externalAuthor.externalAuthorContact', 'type':
'text', 'condition':{'field': 'author.type','operator':'==', 'condition':'e'}, 'disabled': true},
    {'order': 8, 'tab': 1, 'key':'author.confidential', 'type': 'checkbox'},
    {'order': 9, 'tab': 1, 'key':'confidential', 'type': 'checkbox'},
    {'order': 10, 'tab': 1, 'key':'description', 'required': true, 'type': 'textarea'},
    {'order': 11, 'tab': 1, 'key':'note', 'type': 'textarea'},
    {'order':   12,   'tab':   1,   'key':'challenge.id',   'type':   'singleSelect',
'optionsSource':{'concept':'Challenge', 'valueKey':'id', 'textKey': 'title'}},
    {'order':   13,   'tab':   1,   'key':'source.id',   'type':   'singleSelect',
'optionsSource':{'concept':'Source', 'valueKey':'id', 'textKey': 'description'}},
    {'order':   14,   'tab':   1,   'key':'innovationType.id',   'type':   'singleSelect',
'optionsSource':{'concept':'InnovationType', 'valueKey':'id', 'textKey': 'description'}},
    {'order': 15, 'tab': 3, 'key':'documents', 'type': 'documents'},
    {'order': 16, 'tab': 2, 'key':'ChallengeOthers', 'required': true, 'type': 'radioButton',
'options':[{'key':'0',     'value':'DontChallenge'},{'key':1,     'value':'SendToAll'},{'key':2,
'value':'SelectCooperators'}]},
    {'order': 17, 'tab': 2, 'key':'Message', 'type': 'textarea', 'condition':{'field':
'ChallengeOthers','operator':'!=', 'condition':'0'}},
    {'order': 18, 'tab': 2, 'key':'ChallengedCooperators', 'required': true, 'type':
'multipleSelect',       'requiredCondition':{'field':       'ChallengeOthers','operator':'==',
'condition':'2'}, 'condition':{'field': 'ChallengeOthers','operator':'==', 'condition':'2'},
    'optionsSource':{'concept':'Cooperator', 'valueKey':'id', 'textKey': 'name'}},
    {'hidden': true, 'key': 'status.status.id'},
    {'hidden': true, 'key': 'status.status.id'},
    {'hidden': true, 'key': 'responsible.type'},
    {'hidden': true, 'key': 'status.deadlineStatus'}
]}
```

Esta estrutura Json tem em conta o seguinte:

1. Elemento “Tabs” - Array com os tabs a apresentar. Este elemento pode ser facultativo. Cada elemento deste array pode ter os seguintes elementos
   1.1. Elemento “order” – qual o ordinal na apresentação das tabs
   1.2. Elemento “key” – resource key que será utilizada na contextualização
   1.3. Elemento “active” – booleano com indicação se a tab está ativa por omissão.
2. Elemento “Form” – Array com os campos a apresentar. Cada elemento deste array pode ter os seguintes elementos

2.1. Elemento “order” - qual o ordinal na apresentação dos campos

2.2. Elemento “tab” – qual o ordinal da tab em que o campo deve ser apresentado

2.3. Elemento “key” - resource key que será utilizada na contextualização e que deve coincidir com o nome do atributo no conceito (separado por pontos)

2.4. Elemento “required” – booleano com indicação se o campo é obrigatório

2.5. Elemento “type” - qual a representação do campo (ex. text, date, dateToday, radioButton, multipleSelectSticky, checkbox, textarea, singleSelect, documents, multipleSelect, etc)

2.6. Elemento “requiredCondition” – Elemento composto que indica qua a condição de obrigatoriedade. Este elemento é constituído pelos seguintes elementos

  2.6.1. Elemento “field” – qual o campo do qual depende no formulário

  2.6.2. Elemento “operator” – qual o operador de comparação de valor com o campo identificado

  2.6.3. Elemento “condition” – qual o valor que torna a condição verdadeira

2.7. Elemento “condition” – Elemento composto que determina as condições de visibilidade do campo, ou seja, só é visivel se determinado campo assumir determinado valor. Este elemento é constituído pelos seguintes elementos

  2.7.1. Elemento “field” – qual o campo do qual depende no formulário

  2.7.2. Elemento “operator” – qual o operador de comparação de valor com o campo identificado

  2.7.3. Elemento “condition” – qual o valor que torna a condição verdadeira

2.8. Elemento “optionsSource” – Elemento composto que indica como deve ser alimentado as opções de select. Este elemento é é composto pelos seguintes elementos

  2.8.1. Elemento “concept” – qual o nome do conceito a ligar

  2.8.2. Elemento “valueKey” – qual o atributo do conceito a ligar que é chave

  2.8.3. Elemento “textKey” – qual o atributo do conceito a ligar que tem a representação em texto

2.9. Elemento “options” – Array com as opções possíveis para um elemento do tipo “radioButton”. Cada elemento deste array pode ter os seguintes elementos

  2.9.1. Elemento “Key” – valor/chave da opção

  2.9.2. Elemento “value” - resource key que será utilizada na contextualização

2.10. Elemento “Disabled” – booleano com indicação que o campo deve estar desabilitado (“disabled”)

2.11. Elemento “Hidden” – boolean com indicação que o campo deve estar não visível (“hidden”)

2.3. Ações UI legada e nova

(AINDA POR IMPLEMENTAR)

Configuração que mapeará as ações do Uebe.Q legado, para as quais são validados os itens de permissão, e as ações do novo sistema. O objetivo consiste na garantia de que as permissões geridas no Uebe.Q são consideradas nesta nova solução.

2.4. Resources (multilíngue)

(AINDA POR IMPLEMENTAR)

Definição das traduções, para cada idioma, de textos relativos a cada conceito. Atualmente são utilizados os ficheiros de resources de Angular (1 por idioma), localizados na pasta "src/assets/i18n" na raiz do frontend. Nesses ficheiros, as resources encontram-se agrupadas por conceito.

A.3 Exemplo de ficheiro hibernate

```
<?xml version="1.0" encoding="UTF-8"?>
<hibernate-mapping xmlns:uebeq="http://authorization.uebeq.isep.pt">
   <class name="pt.isep.uebeq.authorization.model.AppModule" table="M012_ConfModulo" uebeq:xpto="false">
      <composite-id>
      <key-property name="Module" type="string" column="Modulo" uebeq:abcd="false" />
      <key-property name="Tenant" type="int" column="CodSis" />
      </composite-id>
      <property name="DaysToNotify" column="DiasAviso" type="int"/>
      <property name="Type" column="Tipo" type="char"/>

   <many-to-one name="responsibleEmployee" class="pt.isep.uebeq.authorization.model.Employee"
    insert="false" update= "false" not-found="ignore">
    <column name="Responsavel" not-null="false" uebeq:aqfd="false"/>
        <column name="CodSis" not-null="false" />
   </many-to-one>

   <many-to-one name="responsibleGroupEmployee" class="pt.isep.uebeq.authorization.model.GroupEmployee"
    insert="false" update= "false" not-found="ignore">
    <column name="Responsavel" not-null="false"/>
        <column name="CodSis" not-null="false" />
   </many-to-one>

   <many-to-one name="responsibleRole" class="pt.isep.uebeq.authorization.model.Role"
    insert="false" update= "false" not-found="ignore">
    <column name="Responsavel" not-null="false"/>
        <column name="CodSis" not-null="false" />
   </many-to-one>

   </class>
</hibernate-mapping>
```