UNIVERSIDADE da MADEIRA

Geração Automática de Interfaces com o Utilizador: Uma Abordagem Baseada em MDA para a Plataforma PHP

Duarte Nuno Fernandes Homem Costa

(Licenciado)

Tese Submetida à Universidade da Madeira para a
Obtenção do Grau de Mestre em Engenharia Informática

Funchal – Portugal

Abril 2007

Imagem da Capa:

"Herakles and his squire Iolaos battle the nine-headed Hydra. Iolaos tends a fire ready to cauterize the neck stumps of the serpent.", Herakles and the Hydra, 525 BC, The J Paul Getty Museum, Malibu.

Orientador:

Professor Doutor Nuno Jardim Nunes

Professor Auxiliar do Departamento de Matemática e Engenharias da Universidade da Madeira

RESUMO

Os modelos e as técnicas de modelação são, hoje em dia, fundamentais na engenharia de software, devido à complexidade e sofisticação dos sistemas de informação actuais. A linguagem *Unified Modeling Language* (UML) [OMG, 2005a] [OMG, 2005b] tornou-se uma norma para modelação, na engenharia de software e em outras áreas e domínios, mas é reconhecida a sua falta de suporte para a modelação da interactividade e da interface com o utilizador [Nunes and Falcão e Cunha, 2000].

Neste trabalho, é explorada a ligação entre as áreas de engenharia de software e de interacção humano-computador, tendo, para isso, sido escolhido o processo de desenvolvimento Wisdom [Nunes and Falcão e Cunha, 2000] [Nunes, 2001]. O método Wisdom é conduzido por casos de utilização essenciais e pelo princípio da prototipificação evolutiva, focando-se no desenho das interfaces com o utilizador através da estrutura da apresentação, com a notação Protótipos Abstractos Canónicos (PAC) [Constantine and Lockwood, 1999] [Constantine, 2003], e do comportamento da interacção com a notação *ConcurTaskTrees* (CTT) [Paternò, 1999] [Mori, Paternò, et al., 2004] em UML.

É proposto, também, neste trabalho um novo passo no processo Wisdom, sendo definido um modelo específico, construído segundo os requisitos da recomendação *Model Driven Architecture* (MDA) [Soley and OMG, 2000] [OMG, 2003] elaborada pela organização *Object Managent Group* (OMG). Este modelo específico será o intermediário entre o modelo de desenho e a implementação da interface final com o utilizador. Esta proposta alinha o método Wisdom com a recomendação MDA, tornando possível que sejam gerados, de forma automática, protótipos funcionais de interfaces com o utilizador a partir dos modelos conceptuais de análise e desenho.

Foi utilizada a ferramenta de modelação e de metamodelação MetaSketch [Nóbrega, Nunes, et al., 2006] para a definição e manipulação dos modelos e elementos propostos. Foram criadas as aplicações *Model2Model* e *Model2Code* para suportar as transformações entre modelos e a geração de código a partir destes. Para a plataforma de implementação foi escolhida a *framework* Hydra, desenvolvida na linguagem PHP [PHP, 2006], que foi adaptada com alguns conceitos de modo a suportar a abordagem defendida neste trabalho.

PALAVRAS-CHAVE

Engenharia de Software

Desenho de Interfaces com o Utilizador suportado por Modelos

Modelação de Tarefas

Modelação de Apresentação

Arquitectura Conduzida por Modelos

Geração de Código

ABSTRACT

Models and modeling techniques, today, are fundamental in the software engineering area, due to the growing complexity and sophistication of information systems. The Unified Modeling Language (UML) [OMG, 2005a] [OMG, 2005b] has become the standard language for modeling in different areas and domains, but it is widely recognized that it lacks support for user interface design [Nunes and Falcão e Cunha, 2000].

This dissertation explores the feasibility of bridging between software engineering and human-computer interaction with the Wisdom development process [Nunes and Falcão e Cunha, 2000] [Nunes, 2001]. This process is driven by essential use cases and by the evolutionary prototyping principle, focusing on the user interface design with the presentation structure, through the use of the Canonical Abstract Prototype (CAP) notation [Constantine and Lockwood, 1999] [Constantine, 2003], and with the interaction behaviour using the *ConcurTaskTrees* notation (CTT) [Paternò, 1999] [Mori, Paternò, et al., 2004] in UML

In this dissertation we propose another step in the Wisdom method by defining a specific model compliant with the *Object Managent Group* (OMG) recommended Model Driven Architecture (MDA) [Soley and OMG, 2000] [OMG, 2003], which will be the intermediary between the design model and an implementation of the user interface. This proposal will align the Wisdom method with the MDA recommendation making it possible to automatically generate interface prototypes from conceptual models.

For the creation of the proposed models and elements, was used the modeling and metamodeling tool MetaSketch [Nóbrega, Nunes, et al., 2006]. The applications *Model2Model* and *Model2Code* were created for supporting the transformations between models and from models to code. The chosen implementation platform was the Hydra *framework*, developed in PHP language [PHP, 2006]. In this *framework* some concepts were added for supporting the approach of this dissertation.

KEYWORDS

Software Engineering

Model-Based User Interface Design

Task Modeling

Presentation Modeling

Model Driven Architecture

Code Generation

À minha mulher Tânia e à minha filha Leonor,

que dão sentido à minha vida.

Aos meus pais, Isidro e Merícia,

obrigado pelo vosso amor e apoio em todas as etapas da minha vida.

AGRADECIMENTOS

Quero começar por agradecer ao meu orientador Professor Nuno Nunes, em primeiro lugar, pela confiança que me depositou ao aceitar a minha proposta de dissertação. Agradeço ainda o apoio e enorme capacidade de orientação e crítica ao desenvolvimento deste trabalho, sendo inspiradora a forma como tem realizado a sua vida profissional, principalmente, quando estamos numa universidade pequena e "isolados" numa ilha.

Ao Professor Leonel Nóbrega, o meu "co-orientador", pela enorme paciência com que sempre me recebeu no seu gabinete para as perguntas de "15 minutos" que se tornaram fundamentais para a concretização deste trabalho. Foi graças ao seu trabalho, nomeadamente, à ferramenta MetaSketch, que consegui concretizar os objectivos a que me propus neste trabalho. É extremamente motivante ouvir a forma apaixonada com que fala do seu trabalho.

Aos meus colegas do Sector de Comunicações e Informática da Universidade da Madeira, pelo seu elevado nível profissional e técnico, permitindo podermos, como equipa, evoluir e querer ir sempre mais além.

Aos meus pais, Isidro e Merícia, pela forma incondicional com que sempre me apoiaram ao longo de toda a minha vida, dando-me sempre a possibilidade de fazer as minhas escolhas. Foi graças à prenda do meu Pai, no Natal de 1988, um fantástico computador ZX Spectrum, que decidi que a informática era o que eu queria fazer "quando fosse grande".

À minha mulher Tânia, a minha alma-gémea, amiga e companheira nos bons e maus momentos, que durante estes dois anos deu-me apoio incondicional, principalmente quando o cansaço começava a pesar e pensei mesmo em desistir da dissertação.

À minha filha Leonor, que tinha quatro meses quando iniciei este trabalho, que é sem dúvida o meu melhor e mais perfeito "projecto", e que deu um novo sentido à minha vida.

ÍNDICE

LISTA DE FIGURAS

ACRÓNIMOS

.Net – Microsoft .Net Framework

CORBA - Common Object Request Broker Architecture

CSS - Cascading Style Sheets

CTT - ConcurTaskTrees

CWM - Common Warehouse Metamodel

J2EE - Java 2 Enterprise Edition

MDD - Model Driven Development

MOF - Meta-Object Facility

MSSQL Server – Microsoft SQL Server

MySQL – OpenSource Database Server

OCL - Object Constraint Language

OMG – Object Management Group

PAC – Protótipos Abstractos Canónicos

PHP - PHP: Hypertext Preprocessor

QVT - Query View Transformation

UML - Unified Modeling Language

URL - Universal Resource Locator

W3C – The World Wide Web Consortium

WISDOM - Whitewater Interactive System Development with Object Models

XMI - XML Metadata Interchange

XML - Extensible Markup Language

I. INTRODUÇÃO

A utilização de modelos tem sido uma abordagem recorrente entre analistas, engenheiros, cientistas e outros profissionais que, têm de lidar com estruturas ou sistemas complexos. Com os modelos é conseguida uma representação, com um determinado nível de abstracção, do produto concreto que se pretende construir.

A aplicação de modelos não é aconselhada em todas em situações. Se o domínio do problema é bem conhecido, se a solução é relativamente simples de construir ou futuramente não existe necessidade de grandes alterações ou de novos requisitos, possivelmente, não existe a necessidade de construir modelos neste âmbito. Por outro lado, se existe alguma complexidade no sistema a construir devem ser criados e mantidos modelos ao longo do processo de desenvolvimento. Um processo conduzido por modelos permite aos intervenientes envolvidos visualizar, comunicar e documentar diferentes desenhos e soluções.

Durante anos, na engenharia de software, os modelos não foram considerados fundamentais [Cernosek and Naiburg, 2004]. O software pode ser, facilmente, criado e alterado, sem se investir em grandes recursos de equipamento, permitindo a instalação de uma cultura "faça você mesmo": idealize, construa e altere sempre que necessário. Actualmente, os sistemas de software tornaram-se mais complexos, têm de ser integrados com outros sistemas e a sua manutenção e adaptação a novos requisitos tornaram-se essenciais para a engenharia de software. Quem desenvolve tem de conhecer a área e o domínio do que está a construir, sendo os modelos fundamentais para isso. A modelação na engenharia de software é, actualmente, um paradigma fundamental e aceite, permitindo gerir a complexidade e os riscos do desenho da solução de um sistema de software.

Relativamente à forma de como modelar, a linguagem *Unified Modeling Language* (UML) veio normalizar a forma de representar os modelos de software, sendo actualmente amplamente aceite pela indústria de desenvolvimento de software. A adopção da UML foi um passo

importante para implementação da abordagem conduzida por modelos na engenharia de software. Com a UML, além de existir uma notação normalizada, é fornecida uma linguagem de modelação formal e bem definida, através de um metamodelo, facilitando a sua manipulação por diferentes intervenientes e promovendo a interoperabilidade por diferentes ferramentas.

No desenvolvimento de software, outro aspecto, que actualmente é considerado importante, é a parte interactiva do sistema com o utilizador e a sua facilidade de utilização. O foco, nos aspectos relacionados com a estrutura e comportamento da interface com o utilizador, aumenta a produtividade [Nunes, 2001] e reduz o custo dos sistemas baseados em software, embora este aspecto não seja ainda muito aplicado pela indústria de software. As áreas de engenharia de software e da interacção humano-computador estão actualmente bem estabelecidas, através de contribuições e investigação ao longo de vários anos, embora falte alguma cooperação e conjunção de esforços de modo a aproveitar soluções complementares para os problemas e desafios no desenvolvimento de software.

Na engenharia de software a linguagem UML tornou-se uma norma de modelação, mas é reconhecida a sua falta de suporte para a modelação de interfaces com o utilizador [Nunes and Falcão e Cunha, 2000]. Uma das principais razões está relacionada com o facto do desenvolvimento baseado em UML ser centrado no sistema, ou seja, foca a estrutura interna do sistema e não a arquitectura interactiva, podendo conduzir a sistemas com fraca usabilidade. Neste trabalho iremos utilizar a palavra "usabilidade" como resultado da tradução da palavra "usability" e, embora não exista no dicionário português facilita, o entendimento do conceito, no contexto de modelação e desenho de interfaces como o utilizador.

Embora a UML forneça modelos e esquemas para um processo de desenvolvimento de software, nenhum desses diagramas está, especificamente orientado, para o desenho de interfaces com o utilizador. Relativamente a estes conceitos, convém definirmos que um modelo é uma interpretação de um sistema, segundo um determinado ponto de vista, e envolve a sua especificação a um certo nível de abstracção e de detalhe. Por outro lado, um esquema é a especificação de um modelo, usando uma determinada linguagem, a qual pode ser formal, informal (e.g., linguagem natural); de texto ou gráfica. Quando a representação do esquema, é gráfica e designa-se, usualmente, por diagrama.

Para a comunidade da área de engenharia de software a UML define uma linguagem comum para especificar, visualizar e documentar sistemas de software, permitindo e promovendo a interoperabilidade ao nível semântico entre modelos. A adopção da UML pela comunidade da área de interacção humano-computador é também uma meta importante [Nunes, 2001] para a normalização e evolução do desenvolvimento de sistemas interactivos.

Em [Nóbrega, Nunes, et al., 2005b] os autores propõem uma abordagem formal, baseada em modelos e compatível com a UML, para o desenho de interfaces com o utilizador. Esta abordagem combina uma notação de apresentação, os Protótipos Abstractos Canónicos (PAC) [Constantine and Lockwood, 1999] [Constantine, 2003] e uma notação de modelação de tarefas, a *ConcurTaskTrees* (CTT) [Paternò, 1999] [Mori, Paternò, et al., 2004] adaptando-a ao processo de desenvolvimento Wisdom [Nunes and Falcão e Cunha, 2000] [Nunes, 2001], que sistematiza o desenvolvimento de sistemas interactivos baseados em software, concentrando-se, fundamentalmente, no desenvolvimento da interface com o utilizador. A notação PAC define um conjunto de componentes para construir interfaces abstractas com o utilizador, enquanto que a notação *ConcurTaskTrees* suporta uma descrição hierárquica das tarefas, a serem desempenhadas num sistema interactivo, definindo relações temporais e semânticas entre tarefas.

Neste trabalho propomos mais um passo na abordagem Wisdom, ao integrá-la na iniciativa *Model Driven Architecture* (MDA). A MDA [Soley and OMG, 2000] [OMG, 2003] é uma especificação da organização *Object Management Group* (OMG) que recomenda uma abordagem no desenvolvimento de sistemas, baseada na definição, utilização e transformação de modelos, recorrendo a normas como a UML. Esta iniciativa centra-se na manipulação de modelos, deixando os detalhes de implementação para um nível secundário, conduzindo à geração automática de código.

I.1. MOTIVAÇÃO

No Sector de Comunicações e Informática da Universidade da Madeira (SCI-UMa) foi constituída uma equipa de pequena dimensão, da qual o autor faz parte, para o desenvolvimento e evolução dos sistemas de software de suporte à actividade pedagógica da Universidade da Madeira (UMa), ou seja, num ambiente real e com alguma complexidade.

Foi tomada a decisão de que todo o sistema deveria assentar num ambiente de interacção Web com três camadas: camada de apresentação, camada de negócio e camada de dados. Foi também, adoptado o paradigma de desenvolvimento, conduzido por modelos para a análise e desenho das aplicações a desenvolver. O processo de desenvolvimento Wisdom [Nunes and Falcão e Cunha, 2000] [Nunes, 2001], foi escolhido por assentar em modelos e ter um foco na interacção com o utilizador, o que foi considerado importante, pela equipa, num contexto de desenvolvimento Web.

Posteriormente, após de alguns projectos, surgiu a necessidade de automatizar alguns fluxos no processo de desenvolvimento adoptado. Para tal, foi desenvolvida, na área de interfaces gráficas com o utilizador, uma *framework* de suporte ao desenvolvimento Web. Esta *framework* foi construída com base no método Wisdom e teve em consideração padrões de interacção, identificados durante a análise e implementação de vários projectos Web, desenvolvidos pela equipa do SCI-UMa. Os padrões de interacção descrevem soluções de desenho para problemas recorrentes que os utilizadores finais encontram ao interagir com o sistema de software, conduzindo a uma maior usabilidade e qualidade da interface.

A opção de desenvolvimento de uma *framework* de raiz, por oposição à reutilização de uma já desenvolvida como o *Struts* [Hall, 2004], deveu-se à necessidade de definirmos uma estrutura de programação, que suportasse a passagem da fase de desenho para a fase de implementação das aplicações Web. A *framework,* em questão, implementa mecanismos relativos ao comportamento da interface com o utilizador, ou seja, implementa apenas a camada de apresentação. As camadas de negócio e de dados não são abrangidas por esta *framework*, embora sejam fornecidos os mecanismos de interligação entre estas e a camada de apresentação.

Com o processo de desenvolvimento conduzido por modelos, conseguiu-se documentar e visualizar de diversas perspectivas e níveis de abstracção o sistema de software para responder às necessidades de gestão pedagógica da instituição. Por outro lado, com a *framework* de apoio à implementação da camada de apresentação e de interacção com o utilizador, conseguiu-se

melhorar a transição entre a actividade de desenho e o código de implementação. Mas, com o aumento de requisitos e da sua complexidade surgiu a necessidade de tornar mais eficiente a transição entre os modelos de desenho e a implementação, pois o processo existente não é automático. Os modelos são criados na actividade de desenho e são, posteriormente, consultados pela equipa, sendo depois necessário outro processo criativo para implementar o que foi modelado.

Uma transição total ou parcialmente automática, entre os modelos de desenho e o código da camada de apresentação, irá permitir a abstracção de detalhes de programação, permitindo que a equipa de desenvolvimento do SCI-UMa dê mais atenção ao desenho da solução e não tanto à sua codificação. A geração automática de código, total ou parcial, para a camada de apresentação, irá se traduzir num aumento de produtividade, fazendo com que a equipa tenha mais tempo para se concentrar na qualidade e usabilidade das interfaces com o utilizador.

I.2. PROBLEMA

É reconhecido que na engenharia de software a criação de um sistema tem menores custos do que a sua manutenção e evolução. Tal facto poderá ser minimizado conseguindo, através de técnicas de modelação e rastreabilidade entre modelos, reduzir a lacuna entre o desenho e o código do sistema de software. É importante levar a modelação até próximo da geração de código, permitindo, posteriormente, rastrear desde o código, passando pelos modelos de desenho e chegando à análise e aos requisitos. Por outro lado, a automação da transição entre os modelos e o código de um sistema de software, é um item cada vez mais importante, em termos de produtividade e eficiência de uma equipa de desenvolvimento.

O problema a solucionar neste trabalho é a aplicação dos paradigmas anteriores num método de desenvolvimento concreto, para um ambiente de desenvolvimento real. Pretende-se resolver a integração no método de desenvolvimento Wisdom [Nunes and Falcão e Cunha, 2000] [Nunes, 2001] de técnicas formais e padronizadas de modelação, para suportarem a transição automática dos modelos para uma *framework* de implementação Web, aumentando a produtividade e eficácia do processo de desenvolvimento, facilitando a manutenção e evolução dos sistemas de software.

A solução a encontrar, deverá integrar no método Wisdom técnicas de geração automática de código, a partir dos modelos de desenho do método, tendo em vista uma plataforma Web. Apesar do método ser centrado numa arquitectura e conduzido por modelos, existe a necessidade de expandi-lo, para além da actividade de desenho. Ou seja, permitir que a geração de código, para uma plataforma específica, seja integrada no método, suportada por modelos e executada de forma automática, através de regras de transformação.

A introdução de técnicas de geração de código deve ser assente em modelos e em tecnologias normalizadas. Existem notações [Nóbrega, Nunes, et al., 2005b] no método Wisdom, definidas sob a base formal e normalizada da UML, para a modelação de árvores de tarefas e de protótipos abstractos, utilizadas no desenho da interacção com o utilizador. Neste trabalho, devem ser expandidas essas técnicas e notação, tendo em vista a geração do código para as interfaces com o utilizador. Os modelos de desenho da parte de negócio e de dados, e a geração de código nestas áreas, estão fora do âmbito deste trabalho.

Para que sejam adicionadas ao método técnicas normalizadas e assentes em modelos, para a geração de código, deve ser analisada a possibilidade de o alinhar com a recomendação *Model*

Driven Architecture (MDA) [Soley and OMG, 2000] [OMG, 2003], introduzindo uma nova actividade de modelação específica. Na actividade de modelação específica os modelos da actividade de desenho Wisdom, nomeadamente as árvores de tarefas com a notação *ConcurTaskTrees* e os protótipos abstractos com a notação Protótipos Abstractos Canónicos, devem estar representados, de uma forma mais específica, relativamente aos componentes concretos a implementar, seguindo as recomendações da MDA e utilizando uma linguagem de modelação como a UML, de modo a fazer a interligação com a plataforma de implementação de destino.

Salientamos, ainda, que o que se pretende resolver é um problema de engenharia, sendo este trabalho uma dissertação de engenharia. Numa dissertação científica, é definido um problema e pela observação e experimentação é formulada uma hipótese, e, posteriormente, uma teoria, que explique o problema inicial. Por outro lado, numa dissertação de engenharia, são aplicados conhecimentos técnicos e científicos para resolver um problema real, através de uma solução prática e útil, sendo nesta definição que este trabalho se enquadra.

I.3. OBJECTIVOS

O que se pretende com este trabalho, de uma forma genérica, é a aproximação entre as áreas de engenharia de software e de interacção humano-computador, de modo a se conseguir dar mais passo na definição de um método de desenvolvimento, com o foco na interacção com o utilizador, onde as técnicas de análise, modelação e geração de código sejam aplicadas sob uma base formal e normalizada comum.

Conforme já foi referido, será utilizado o método de desenvolvimento Wisdom cuja base formal é definida em UML. As técnicas e notações [Nóbrega, Nunes, et al., 2005b] no método Wisdom utilizam a UML para a definição formal da modelação de árvores de tarefas e de protótipos abstractos utilizadas no desenho de interfaces com o utilizador.

O principal objectivo deste trabalho é introduzir tecnologias de engenharia de software no método Wisdom, formais e normalizadas, de modo a que seja possível a transformação dos modelos de desenho em código, de forma automática, dando origem a protótipos de interfaces com o utilizador totalmente funcionais. As tecnologias de engenharia de software a introduzir devem estar de acordo com a recomendação *Model Driven Architecture* (MDA), sendo essa a garantia da sua formalização e normalização, devendo ser analisada a possibilidade de alinhar o processo Wisdom com esta recomendação.

Posteriormente, deve ser criada uma nova actividade de modelação específica para o método, onde será definido um modelo de desenho específico para uma plataforma de implementação Web. O modelo específico fornece detalhes de implementação, que permitem a geração automática de código para uma plataforma tecnológica específica. O modelo específico, a ser proposto, deve estar de acordo com a definição *Platform-Specific Model* da MDA e será definido no âmbito de uma *framework* direccionada para a implementação de interfaces Web com o utilizador. Embora esta abordagem possa ser aplicada, em diferentes plataformas de implementação, sendo este inclusive um dos objectivos da MDA, neste trabalho iremos focar os desafios de suportar esta abordagem para uma plataforma Web, suportada por tecnologia *server-side*.

No contexto deste trabalho iremos utilizar o termo "plataforma Web" como referência ao ambiente tecnológico, do lado do servidor, de execução de aplicações Web. A expressão "tecnologia server-side" descreve o paradigma de implementação, amplamente utilizado por

diversas tecnologias Web, onde quase todo o processamento da aplicação é realizado do lado do servidor e, só depois, é enviado o resultado para o *browser* Web do cliente.

Para ser atingido o objectivo, têm de ser dados diversos passos, descritos nos seguintes pontos:

i) A primeira meta a atingir será a análise da possibilidade de alinhar o método Wisdom com a recomendação *Model Driven Architecture* (MDA), tendo em vista a geração de código a partir dos modelos da actividade de desenho.

ii) Devem ser descritos os conceitos, elementos e relações dum modelo específico, segundo os requisitos da recomendação MDA, para ser utilizado na modelação da estrutura e do comportamento das interfaces definidas na arquitectura Wisdom. Este modelo específico será o intermediário entre o desenho e a implementação do sistema para uma plataforma Web suportada por uma *framework*.

ii) O modelo específico, seguindo os requisitos da MDA de normalização e formalização, deve ser construído em UML e guardado num formato que permita uma maior interoperabilidade, por exemplo em XMI (*XML Metadata Interchange*) [OMG 2005c] [Grose, Doney, et al. 2002], possibilitando que possa ser manipulado por diferentes ferramentas de modelação.

iii) Implementar uma ferramenta que recorrendo a regras de transformação pré-configuradas, faça de forma automática a transformação entre os modelos da actividade de desenho e o modelo especifico da plataforma Web de implementação.

iv) Na *framework* de suporte à plataforma de implementação devem ser codificados conceitos que fazem parte da actividade de desenho Wisdom, nomeadamente as árvores de tarefas com a notação *ConcurTaskTrees* e o respectivo algoritmo de execução.

v) Implementar uma ferramenta que, recorrendo a regras de transformação para a plataforma de execução, suportada por uma *framework*, possa converter o modelo específico em código, de forma automática, implementando assim protótipos funcionais das interfaces com o utilizador.

II. A ABORDAGEM WISDOM

Nesta secção do documento, iremos descrever os principais conceitos, fases e notações descritas no processo de desenvolvimento de software Wisdom (*Whitewater Interactive System Development with Object Models*) [Nunes and Falcão e Cunha, 2000] [Nunes, 2001], vocacionado para o desenvolvimento de sistemas interactivos por equipas de dimensão média ou reduzida. Será detalhada a notação que suporta a arquitectura Wisdom e as notações *ConcurTaskTrees* e Protótipos Abstractos Canónicos, adoptadas, posteriormente, pela abordagem de modo a enriquecer a especificação da estrutura e do comportamento do desenho da interacção entre o sistema e o utilizador.

II.1. WISDOM: O PROCESSO

> "O processo Wisdom promove um modelo de prototipificação evolutiva rápida, adaptado aos requisitos de equipas de desenvolvimento de software de reduzida dimensão, que trabalham, tipicamente, em actos aleatórios de codificação" [Nunes, 2001]

O processo descreve as fases e passos para a aplicação do método e técnicas, defendidas pelo Wisdom para uma equipa de desenvolvimento de software. O processo assenta numa arquitectura que promove um conjunto de modelos UML de suporte ao desenvolvimento de sistemas de software centrados no utilizador e na utilização. São definidas quatro fases (incepção, elaboração, construção e transição) e, em cada fase, existem várias actividades ou fluxos de trabalho (*workflows*). Além das diversas fases e actividades, o processo Wisdom defende dois conceitos fundamentais à sua aplicação:

i) Prototipificação Evolutiva: " É um processo contínuo para a adaptar um sistema de software às rápidas mudanças de requisitos e outras restrições ambientais. Na prototipificação evolutiva podem ser desenvolvidos todos os tipos de protótipos mas os sistemas-piloto têm uma maior importância." [Nunes, 2001].

ii) Casos de Utilização Essenciais: "Correspondem a tarefas de alto nível e definem uma forma de utilização do sistema que é completa e bem definida para o utilizador [Constantine and Lockwood, 1999]. Devido a isso, os casos de utilização, essenciais, correspondem a objectivos ou tarefas externas, conforme definido pela engenharia de usabilidade, ou seja, o estado do sistema que um humano pretende atingir." [Nunes, 2001].

O conceito de evolução está sempre presente no processo defendendo a definição dos objectivos a atingir em cada actividade ou fluxo de trabalho que dá origem um artefacto (modelo) ou parte do sistema (protótipo). Entre evoluções, é realizada uma avaliação dos resultados, comparando-os com os objectivos estabelecidos no início do fluxo de trabalho. Esta avaliação deve envolver a equipa de desenvolvimento e os utilizadores finais do sistema, que está a ser construído. Os artefactos produzidos nas actividades das diversas fases implicam a construção de modelos, utilizando a notação definida pelo Wisdom, e a implementação de protótipos funcionais de partes do sistema previamente priorizadas. O processo é evolutivo, também, porque promove a criação de uma sequência de protótipos do sistema que irão dar origem, através de várias evoluções, ao produto final.

O outro conceito, importante, é o caso de utilização essencial, que difere na sua definição relativamente aos casos de utilização convencionais. Os casos de utilização, na engenharia de software, centram-se nas funcionalidades internas que o sistema deve executar e não nos objectivos que os utilizadores pretendem atingir com o sistema. Os casos de utilização essenciais correspondem a objectivos ou a um estado do sistema que o utilizador pretende atingir e para o qual é necessário desempenhar um conjunto de tarefas. Além disso os casos de utilização essenciais não devem conter nenhuma descrição de carácter técnico sobre a implementação e a tarefa que o utilizador pretende realizar, promovendo a criatividade no desenvolvimento da solução.

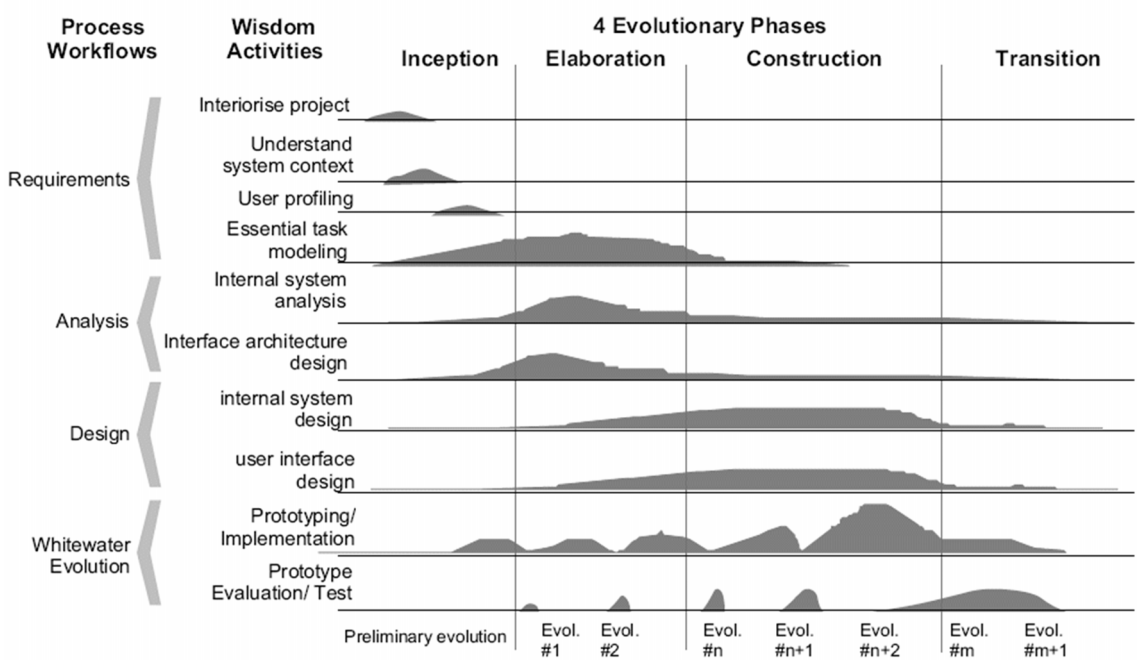

Figura II-I – O Processo Wisdom. Adaptado de [Nunes, 2001].

Conforme já referido anteriormente, o processo Wisdom define quatro fases de desenvolvimento: i) incepção, ii) elaboração, iii) construção e iv) transição. Por outro lado existem os fluxos de trabalho (*workflows*) ou actividades do processo, que descrevem as tarefas desempenhadas pelos engenheiros de software e os artefactos que são produzidos. Na Figura II-I está representado um esquema que ilustra, de forma resumida, as diversas fases e que actividades devem ser executadas, em cada uma delas.

Para as diversas fases, apesar de ocorrerem simultaneamente várias actividades que dão origem a modelos e a protótipos do sistema, é possível identificar alguns dos principais objectivos e artefactos produzidos. Na fase de **incepção** é definida a perspectiva de negócio do produto que caracteriza o objectivo do projecto. Nesta fase o risco e o custo do projecto são também estimados. Na incepção o principal artefacto produzido é um modelo de casos de utilização. Na fase de **elaboração,** os casos de utilização são detalhados e a arquitectura conceptual é definida. Neste nível, devem ser planeadas as actividades e recursos para terminar o projecto. É, também,

tomada a decisão de quais os casos de utilização mais importantes e modo a priorizar a sua implementação. Na fase de **construção** o produto é construído a partir dos modelos das actividades de análise e desenho da arquitectura conceptual. Neste nível, a meta principal é a implementação de casos de utilização para a entrega do produto de software que se pretende construir. A fase de **transição** é o período onde são executados os teste com os utilizadores do sistema e onde se pretende detectar e reportar os problemas encontrados.

Para cada uma das fases evolutivas do processo, conforme já referido anteriormente, podem ser executadas diversas actividades ou *workflows* como por exemplo: requisitos, análise, desenho e implementação/teste. De uma forma simplificada podemos definir a actividade de requisitos como a descrição do problema da perspectiva do cliente, pelo contrário a actividade de análise é a descrição do problema da perspectiva do engenheiro de software e a actividade de desenho é a especificação da solução para o sistema que deve ser implementado. O final de cada actividade deverá ter como resultado um modelo, de acordo com a arquitectura de modelos definida pelo Wisdom, ou um protótipo funcional de parte do sistema. Este deve, normalmente, corresponder a um determinado caso de utilização escolhido, priorizado pelos utilizadores e clientes, para ser analisado, desenhado e implementado.

De uma forma resumida o processo Wisdom evolui de forma iterativa até os diferentes protótipos gerados completarem os requisitos pretendidos, dando origem ao produto final. A principal meta do processo Wisdom, consiste em centrar o desenvolvimento nos utilizadores e nas suas tarefas, promovendo um desenvolvimento rápido e evolutivo da solução do sistema, permitindo responder, eficazmente, às alterações e mudanças de requisitos ao longo de um projecto.

II.2. WISDOM: O MÉTODO E A ARQUITECTURA

O processo Wisdom define as principais fases e fluxos de trabalho necessários para sistematizar o desenvolvimento de software em pequenas equipas que pretendam introduzir práticas de desenvolvimento evolutivo, centrado nos utilizadores, nas suas tarefas, e baseado em modelos. A utilização intensiva que o Wisdom preconiza para os modelos de software, descritos em UML, exige uma arquitectura de modelos que suporte o processo. A arquitectura de modelos conduz o processo desde os modelos de alto nível, construídos para capturar requisitos, até ao modelos de desenho que especificam o sistema a ser implementado. A abordagem foi definida num contexto de desenvolvimento centrado nos utilizadores e na utilização, devido a isso a arquitectura está direccionada, essencialmente, para o desenho e implementação da interface com o utilizador.

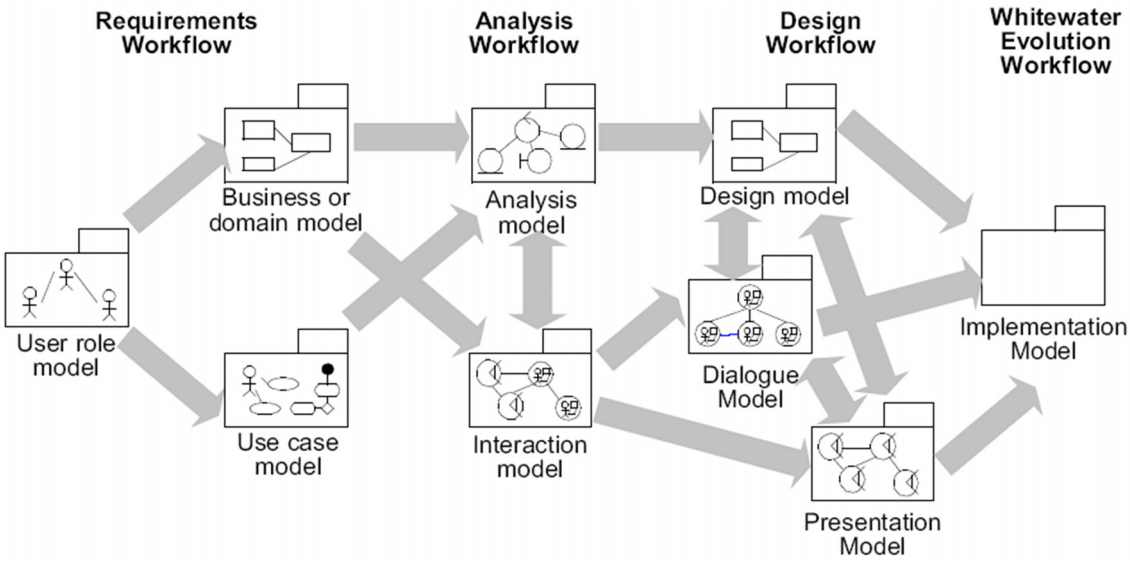

Figura II-II – A arquitectura de modelos do processo Wisdom. Adaptado de [Nunes, 2001].

O processo e arquitectura de modelos formam a base do método Wisdom que descreve como os diferentes modelos são construídos, à medida que o desenvolvimento progride ao longo das diferentes fases. Iremos descrever o método Wisdom na perspectiva das actividades do processo: requisitos, análise e desenho. Esta sequência serve, apenas, para simplificar a explicação, mas o processo deve ser sempre conduzido pelo princípio evolutivo. Isto significa que uma actividade do processo ou modelo, nunca está realmente terminada, sofrendo alterações ao longo da aplicação das actividades e fases do processo a um projecto. Na Figura II-II está esquematizado a arquitectura de modelos do processo Wisdom, descrita nas

perspectivas das actividades de requisitos (*requirements workflow*), de análise (*analysis workflow*) e de desenho (*design workflow*).

II.2.1.Requisitos: Conduzidos por Casos de Utilização Essenciais

Esta actividade tem uma ligação mais forte à fase de incepção do processo. Na primeira actividade, os requisitos dos utilizadores são descritos, através de casos de utilização essenciais [Constantine and Lockwood, 1999], onde são definidos os actores envolvidos no sistema, as suas intenções e as suas acções, dando origem ao modelo de casos de utilização (*use case model*).

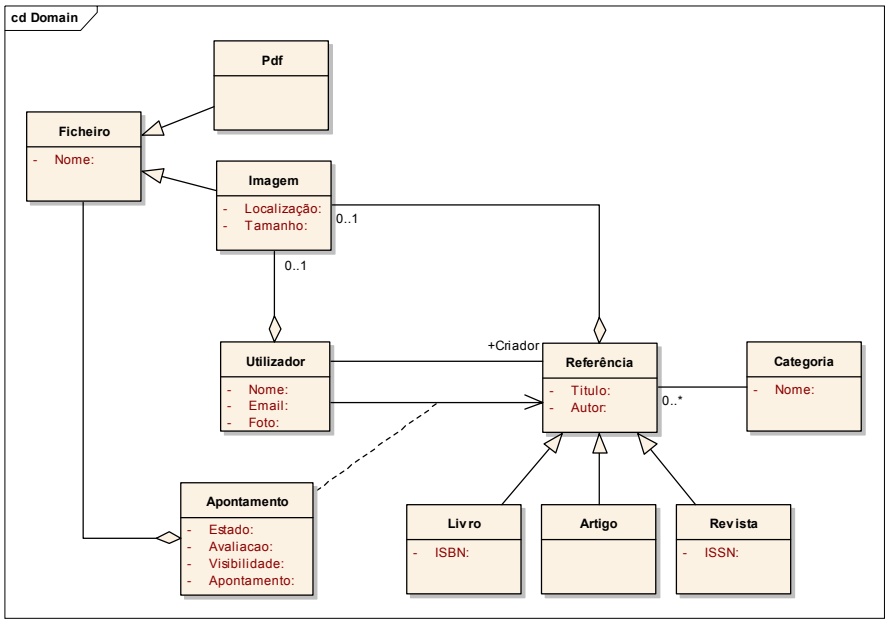

Figura II-III – Exemplo de um modelo de domínio descrito com um diagrama de classes UML.

Como resultado desta actividade, é construído o modelo de domínio, representado por um diagrama de classes UML, onde são descritas as entidades presentes no sistema a construir. Na Figura II-III, está representado um exemplo de modelo de domínio e na Figura II-IV está definido um exemplo de um diagrama de casos de utilização.

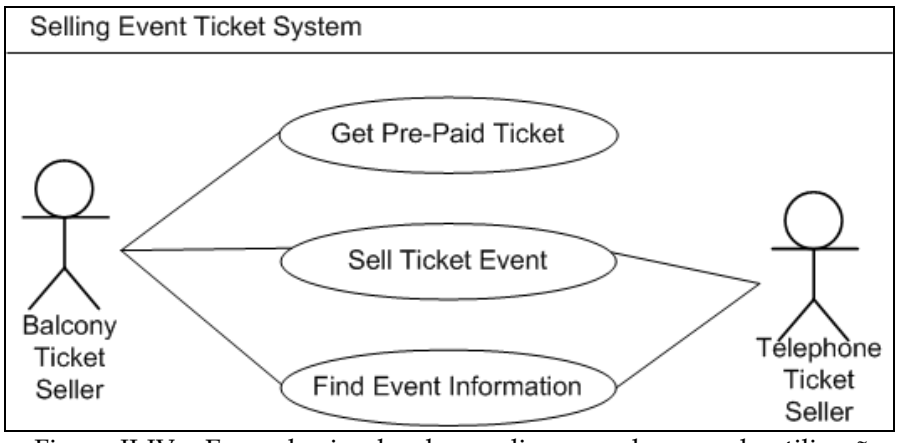

Figura II-IV – Exemplo simples de um diagrama de casos de utilização. Adaptado de [Costa and Valente, 2004].

Depois da definição dos casos de utilização essenciais, cada um destes casos de utilização é detalhado por diagramas de actividade UML com duas colunas (*swimlanes*). Na coluna da esquerda são colocadas as actividades que definem as intenções dos utilizadores, e na coluna da direita são representadas as responsabilidades do sistema perante as acções que se pretende executar. Na Figura II-V, está representado um exemplo de detalhe de um caso de utilização, através de um diagrama de actividade. Nesta actividade são produzidos os modelos de casos de utilização que correspondem essencialmente, a uma vista externa do sistema, descrita na linguagem e perspectiva do cliente e dos utilizadores.

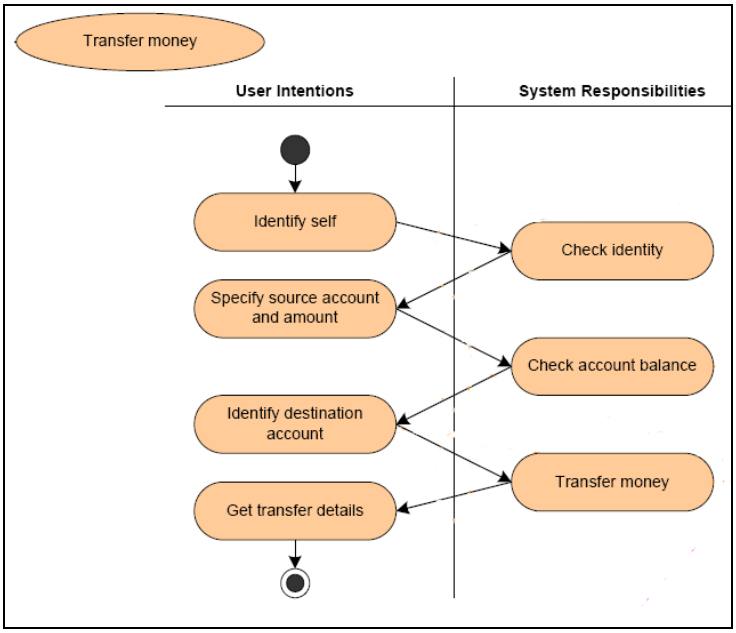

Figura II-V – Exemplo de detalhe de um caso de utilização "Transferir Dinheiro". Adaptado de [Nóbrega, Nunes, et al., 2005b].

II.2.2.Análise: A Arquitectura Conceptual

A partir dos modelos da actividade de levantamento de requisitos, são definidos os modelos de análise, que correspondem à perspectiva interna do sistema, descrita pela equipa de desenvolvimento. Os modelos nesta actividade irão estruturar o sistema, tendo em conta os casos de utilização essenciais e o modelo de domínio, em classes estereotipadas que descrevem as funcionalidades que a solução do sistema deverá ter. A estrutura do sistema, descrita por classes estereotipadas, é realizada através do modelo de arquitectura conceptual, que contém os elementos dos modelos de análise e de interacção descritos na Figura II-II. Os conceitos e elementos a utilizar no modelo de arquitectura conceptual são:

- **Espaço de Interacção** (estereótipo de classe UML **<<interaction space>>**): O espaço de interacção é usado para modelar a interacção entre o sistema e os utilizadores humanos. Uma classe espaço de interacção representa o espaço, na interface com o utilizador, de um sistema onde o utilizador humano interage com todas as funções e informação necessária

para desempenhar uma tarefa particular ou um conjunto de tarefas. As classes espaço de interacção são responsáveis pela descrição da interacção física com o utilizador, incluindo um conjunto de técnicas de interacção que definem a imagem do sistema (*output*) e a manipulação dos eventos produzidos pela interacção com o utilizador. [Campos and Nunes, 2004].

- **Tarefa** (estereótipo de classe UML: **<<task>>**) as classes tarefa são utilizadas para modelar o diálogo, ou seja, o comportamento entre o utilizador e o sistema em termos dos conjuntos de acções, completas e bem definidas, necessárias para atingir um determinado objectivo. As classes tarefa são responsáveis pela sequência dos níveis de subtarefas, pela consistência dos múltiplos elementos de apresentação e pelo mapeamento entre entidades e classes de apresentação (espaços de interacção). As classes tarefa encapsulam as complexas dependências temporais e outras restrições entre diferentes actividades necessárias para a utilização do sistema. Desta forma, estas classes isolam as mudanças na estrutura de diálogo da interface com o utilizador. [Campos and Nunes, 2004]

- **Controlo** (estereótipo de classe UML: **<<control>>**) a classe controlo representa coordenação, transacções e controlo de objectos relacionados com a lógica de negócio. Deste modo, o elemento controlo isola as mudanças aos controlos, transacções e lógica de negócio que envolve outros objectos. [Campos and Nunes, 2004].

- **Entidade** (estereótipo de classe UML: **<<entity>>**) a classe entidade é utilizada para modelar a informação persistente. As classes entidade definem a estrutura de classes do domínio (ou negócio), muitas vezes, representando a estrutura lógica de dados. Como resultado, as entidades reflectem a informação de um modo que beneficia os arquitectos de software quando estão a desenhar e a implementar o sistema, incluindo o suporte para a persistência. [Nunes, 2001].

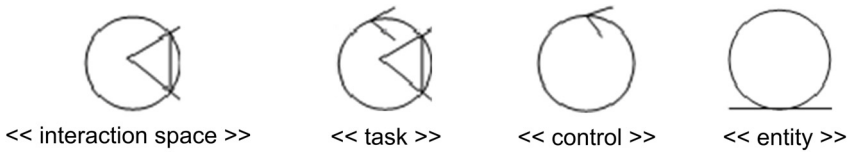

Figura II-VI – A notação gráfica para os principais elementos da arquitectura conceptual Wisdom.

Na Figura II-VI, estão representados os elementos da arquitectura conceptual com a respectiva notação gráfica. De uma forma resumida, os elementos espaço de interacção, tarefa e controlo são obtidos a partir dos diagramas de actividade utilizados para detalhar os casos de utilização. Os elementos tarefa são obtidos das actividades definidas na coluna das intenções do utilizador. Os elementos controlo são obtidos das actividades na coluna das responsabilidades do sistema. Por outro lado, os elementos espaço de interacção são o resultado da intersecção das linhas de

fluxo de controlo entre as actividades de cada coluna dos diagramas de actividade. Na Figura II-VII, está representada a técnica para obter os elementos da arquitectura conceptual, a partir do detalhe dos casos de utilização essenciais.

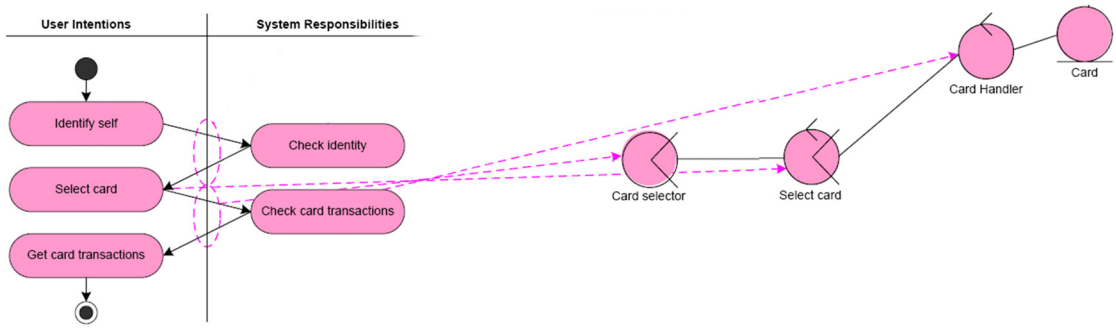

Figura II-VII – Exemplo de como obter os elementos da arquitectura conceptual a partir de casos de utilização essenciais. Adaptado de [Nunes, 2003].

Relativamente aos elementos entidade, estes são, normalmente, definidos a partir do modelo de domínio, criado na actividade de requisitos. O método para obter os elementos da arquitectura poderá não ser tão directo como descrito e os arquitectos do sistema poderão definir outros elementos que não são representados nos diagramas de actividade para descrever outros aspectos do sistema a desenvolver. Contudo, tal deverá ser efectuado, com precaução, porque todos os elementos da arquitectura deverão depender da descrição dos requisitos dos utilizadores.

Depois de representar os elementos no modelo de arquitectura Wisdom, este são interligados com associações do tipo <<*communicate*>> (comunicação) de acordo com as afinidades existentes entre os conceitos que estes elementos representam no sistema. Com a representação dos elementos e das correspondentes associações é definido o modelo de arquitectura conceptual. Este modelo não é estático, podendo ser alterado e refinado durante o decorrer do projecto, de acordo com o princípio de desenvolvimento evolutivo.

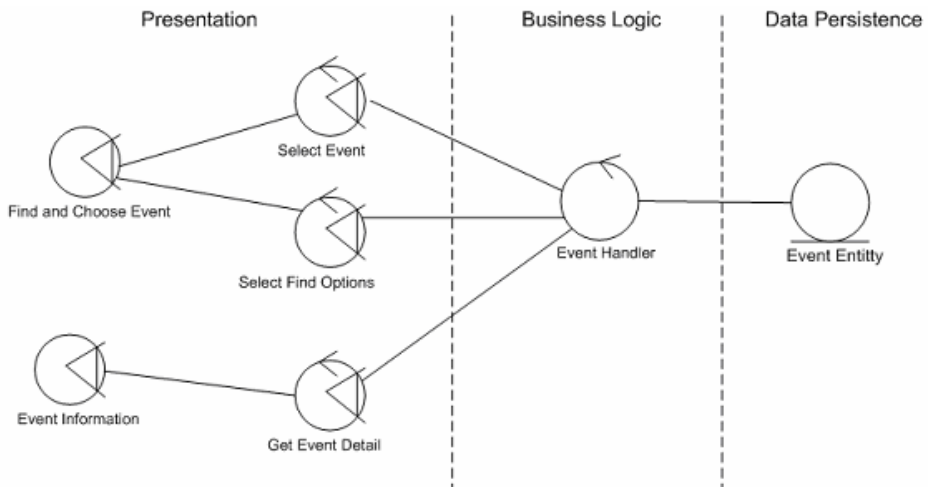

Figura II-VIII – Exemplo simplificado da uma arquitectura conceptual do método Wisdom. Adaptado de [Costa and Valente, 2004].

O modelo de arquitectura conceptual, composto pelos elementos do modelo de análise (*analysis model*) e do modelo de interacção (*interaction model*), pode ser considerado como intermediário entre a fase de requisitos e a fase de desenho da solução do sistema. Na Figura II-VIII, está representado um exemplo de um diagrama, simples, de uma arquitectura conceptual. A existência deste modelo permite organizar os elementos da arquitectura, promovendo a separação de conceitos [Puerta and Eisenstein, 1999], defendida pela área da interacção humano-computador, entre os elementos que descrevem a funcionalidade interna e os elementos que descrevem a estrutura de utilização do sistema. Este modelo, é ainda, fundamental para manter a rastreabilidade do processo entre os requisitos e o desenho do sistema final, promovendo, deste modo, um forte suporte para a adaptação e evolução do sistema às mudanças.

II.2.3. Desenho: Os modelos de Apresentação e de Diálogo

A actividade de desenho no Wisdom mantém a mesma preocupação de separação de conceitos, a um nível de abstracção mais detalhado, entre a estrutura de utilização e as funcionalidades internas. A actividade de desenho, definida no Wisdom, concentra-se essencialmente no detalhe dos elementos tarefa (<<task>>) e espaço de interacção (<<interaction space>>), recorrendo para isso, respectivamente, a dois modelos: modelo de diálogo e modelo de apresentação. A abordagem Wisdom não faz uma recomendação detalhada de como modelar a lógica de negócio ou a persistência dos dados durante a actividade de desenho, relacionadas com os elementos controlo (<<control>>) e entidade (<<entity>>). Consequentemente, a ênfase da abordagem está no desenho da interface e da dinâmica da interacção com o utilizador e não na funcionalidade interna do sistema.

A arquitectura conceptual, é o ponto de partida para definir estes dois modelos relacionados com o desenho das interfaces com o utilizador do sistema. O modelo de diálogo descreve e detalha o comportamento da interacção entre o utilizador e a aplicação a desenvolver. Este modelo centra-se nas tarefas que o sistema deve desempenhar e as relações temporais entre essas tarefas. No modelo de diálogo as tarefas, previamente identificadas na fase de análise e definidas na arquitectura conceptual, são detalhadas utilizando a notação *ConcurTaskTrees*. A notação gráfica *ConcurTaskTrees* (CTT) combina estruturas hierárquicas de tarefas concorrentes com um conjunto de operadores temporais. Esta notação representa actividades interactivas, através da decomposição de tarefas numa estrutura em árvore invertida.

O modelo de apresentação define a forma como as diferentes entidades de apresentação são estruturadas para realizar a interacção física com o utilizador, especificando um conjunto de entidades independentes da tecnologia de implementação (espaços de interacção) a serem utilizados pelo modelo de diálogo. Os elementos <<interaction space>> são responsáveis por

receber e apresentar informação aos utilizadores, de modo a que estes possam desempenhar as suas tarefas e cumprir os seus objectivos. No modelo de apresentação são definidos, além dos espaços de interacção, as seguintes extensões do UML, detalhadas em [Campos and Nunes, 2004]:

- <<***navigates***>> - estereótipo de associação entre duas classes de interacção descrevendo um utilizador a mover-se de um espaço de interacção para outro.

- <<***contains***>> - estereótipo de associação entre dois espaços de interacção descrevendo que a classe de origem contém a classe de destino.

- <<***input element***>> - estereótipo de atributo que define informação recebida do utilizador, ou seja, informação que pode ser manipulada pelo utilizador.

- <<***input collection***>> - estereótipo de atributo que define um conjunto de dados recebidos do utilizador, ou seja, um grupo ou uma lista de dados que podem ser manipulados pelo utilizador.

- <<***output element***>> - estereótipo de atributo que define informação disponibilizada ao utilizador mas que não pode ser manipulada.

- <<***output collection***>> - estereotipo de atributo que define um conjunto de dados apresentados ao utilizador, ou seja, um grupo ou uma lista de dados que pode o utilizador aceder mas não manipular.

- <<***action***>> - estereótipo de operação que define qualquer coisa que o utilizador pode fazer na interface com o utilizador e que provoca uma alteração significativa no estado interno do sistema.

Num contexto de desenvolvimento de aplicações Web, o modelo de apresentação tem um papel particularmente importante, porque, permite definir diagramas de navegação para as interfaces com o utilizador que serão implementados como páginas Web, organizando numa fase preliminar a estrutura de informação e navegação do sistema Web. Na Figura II-IX, está representado um diagrama de um modelo de apresentação Wisdom.

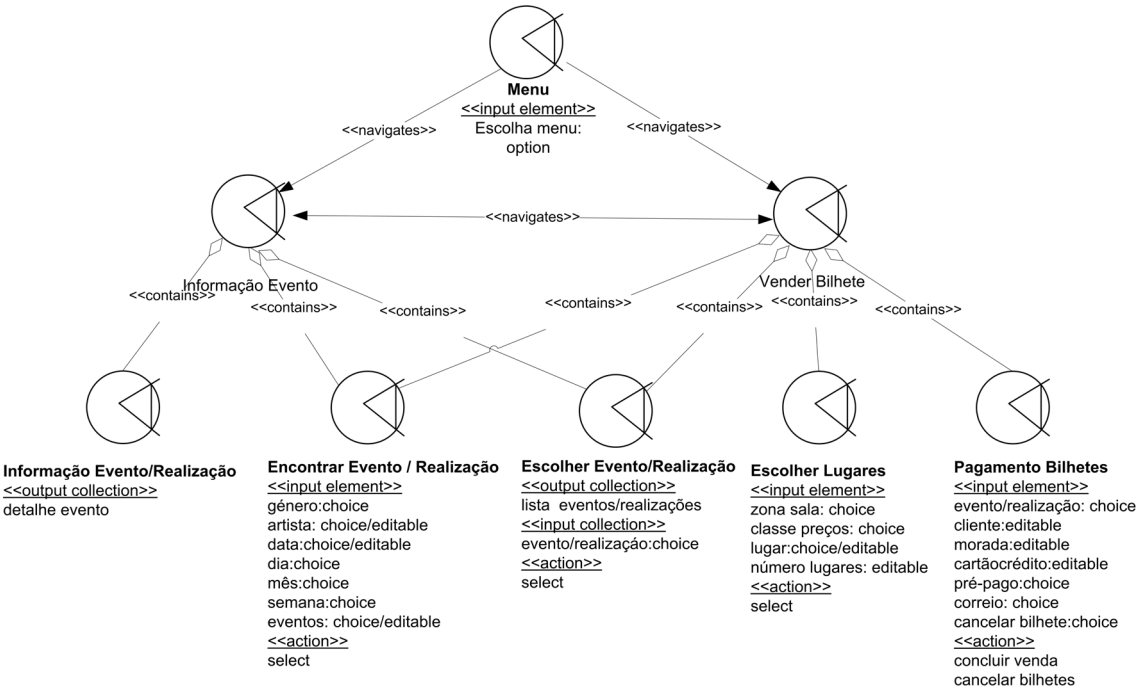

Figura II-IX – Exemplo de um modelo de apresentação para um sistema informático para venda de bilhetes ao público. Adaptado de [Costa and Valente, 2004].

Em secções posteriores neste trabalho, iremos descrever duas notações que foram, posteriormente, adoptadas [Nóbrega, Nunes, et al., 2005b] pelo método Wisdom, de modo a enriquecer os modelos da actividade de desenho, nomeadamente, os modelos de diálogo e de apresentação. Por um lado, foi adoptada e definida em UML a notação de Protótipos Abstractos Canónicos (PAC), aplicando-a ao detalhe de cada um dos espaços de interacção definidos na arquitectura conceptual. Esta extensão permite obter um nível de desenho de maior detalhe, sem entrar em pormenores técnicos ou sugestões de implementação. Foi, também, definida a notação *ConcurTaskTrees* (CTT) em termos do metamodelo da UML, aplicando-a ao detalhe dos elementos tarefa, obtendo assim uma descrição formal do comportamento da interacção, entre o utilizador e o sistema, para cada elemento deste tipo definido na arquitectura conceptual Wisdom.

II.2.4. Evolução *WhiteWater*, Implementação e Testes

O fluxo de trabalho evolução *whitewater* corresponde, essencialmente, à implementação do sistema desenhado, a sua designação resulta duma analogia [Nunes, 2001] entre a dinâmica do desenvolvimento de software evolutivo e o movimento turbulento das águas de um rio, onde existem movimentos rápidos na presença de obstáculos.

Nesta última actividade, a evolução, são utilizados os artefactos e modelos criados pelas anteriores actividades de análise e desenho de modo a produzir um protótipo funcional, normalmente de uma parte do sistema relacionada com um ou vários casos de utilização. Uma

evolução, conforme se pode verificar na Figura II-I, é definida como um protótipo com o objectivo de atingir a solução para o sistema final.

Entre cada evolução a equipa de desenvolvimento avalia e testa, com os utilizadores finais, os resultados atingidos, tendo em conta os objectivos pretendidos. Desta forma existe um maior controlo da implementação da solução, reduzindo em cada avaliação as forças de bloqueio por parte da equipa relativamente às mudanças, evitando perdas de produtividade e de motivação durante o ciclo de desenvolvimento.

II.3. MODELAÇÃO DE TAREFAS E DE PROTÓTIPOS ABSTRACTOS EM UML

Na proposta definida em [Nóbrega, Nunes, et al., 2005b] os autores especificaram notações em UML para a apresentação e para a modelação de tarefas para a actividade de desenho do método Wisdom. Nesta secção descrevemos, de forma breve, a motivação e os objectivos de promover a modelação de desenho de interfaces com o utilizador em UML, de acordo com a abordagem descrita no Wisdom.

A necessidade de suporte à modelação de tarefas em linguagens como a UML, tem vindo a ser largamente discutida como um requisito chave para aproximar e ligar as arenas da engenharia de software e da interacção humano-computador. Em [Nunes and Falcão e Cunha, 2000] foi proposta uma extensão para a UML 1.x de modo a suportar a popular notação *ConcurTaskTrees* [Paternò, 1999] [Mori, Paternò, et al., 2004]. No entanto esta abordagem tinha um fraco suporte semântico, devido aos mecanismos de extensão do UML 1.x não fornecerem a flexibilidade necessária para suportar a notação que incluía a estrutura hierárquica de tarefas e as relações temporais entre estas. Em [Nóbrega, Nunes, et al., 2005] foi proposta uma nova extensão melhorada, para suporte à notação CTT, para a versão 2.0 da UML [OMG, 2005a] [OMG, 2005b] e que tirava partido das capacidades de extensão melhoradas desta norma. Esta integração entre a notação CTT e a UML 2.0 foi conseguida por extensão do metamodelo do UML 2.0 de forma a definir novos elementos e conceitos representados num diagrama UML com uma estrutura em árvore invertida (de tarefas). Os autores demonstraram que a semântica das *ConcurTaskTrees* pode ser expressa em termos da semântica de actividades definida pela UML 2.0, permitindo a integração de conceitos de modelação de tarefas em UML.

Se por um lado com a modelação de tarefas, utilizando a notação CTT, se consegue descrever o comportamento e a dinâmica da interacção com o utilizador, por outro lado existe a necessidade de descrever a estrutura e os elementos que irão definir a apresentação, ou seja, o aspecto da interface perante o utilizador. Para a modelação da apresentação no método Wisdom foi adoptada, em [Nóbrega, Nunes, et al., 2005b], a notação Protótipos Abstractos Canónicos [Constantine and Lockwood, 1999] [Constantine, 2003].

Os Protótipos Abstractos Canónicos (PAC) permitem representar a estrutura de apresentação das interfaces com o utilizador de uma forma independente da tecnologia de implementação. Os PAC promovem a representação e discussão de desenhos intermédios entre os modelos de tarefas e os protótipos concretos de implementação. A notação proposta para os PAC propõe

um conjunto de componentes abstractos para a estrutura de apresentação da interface, sendo que cada um deles especifica uma função interactiva. Por exemplo, a introdução de dados ou uma notificação ao utilizador. Os componentes canónicos abstractos principais são materiais e ferramentas, que combinados permitem construir modelos completos das interfaces, incluindo componentes híbridos que assumem a representação das ferramentas e dos materiais. Os conceitos da notação PAC são naturalmente estruturais e foram por isso definidos, em [Nóbrega, Nunes, et al., 2005b], como uma extensão ao pacote de Classes da UML 2.0 [OMG, 2005a] [OMG, 2005b].

Devido às características descritas, as notações CTT e PAC definidas pelas referidas extensões em UML, suportam as actividades de desenho do método Wisdom. Em [Nóbrega, Nunes, et al., 2005b] os autores definiram, formalmente, um refinamento à notação Wisdom original, utilizando o metamodelo da UML 2.0, estendido com os conceitos e semântica das notações PAC e CTT, dando um passo em frente em relação à integração da especificação normalizada da UML, com conceitos para a modelação da interacção com o utilizador. A notação Protótipos Abstractos Canónicos (PAC) utilizada para o modelo de apresentação e a notação *ConcurTaskTrees* (CTT), aplicada no modelo de diálogo, foram combinados e re-definidos, estabelecendo assim uma notação e uma semântica, definida formalmente em UML, melhorando a expressividade e o detalhe do desenho destes modelos no método Wisdom. Na Figura II-X, estão representadas as alterações propostas à arquitectura de modelos Wisdom com os modelos e diagramas das duas notações referidas. Nas próximas secções são descritas com maior detalhe cada uma das notações PAC e CTT.

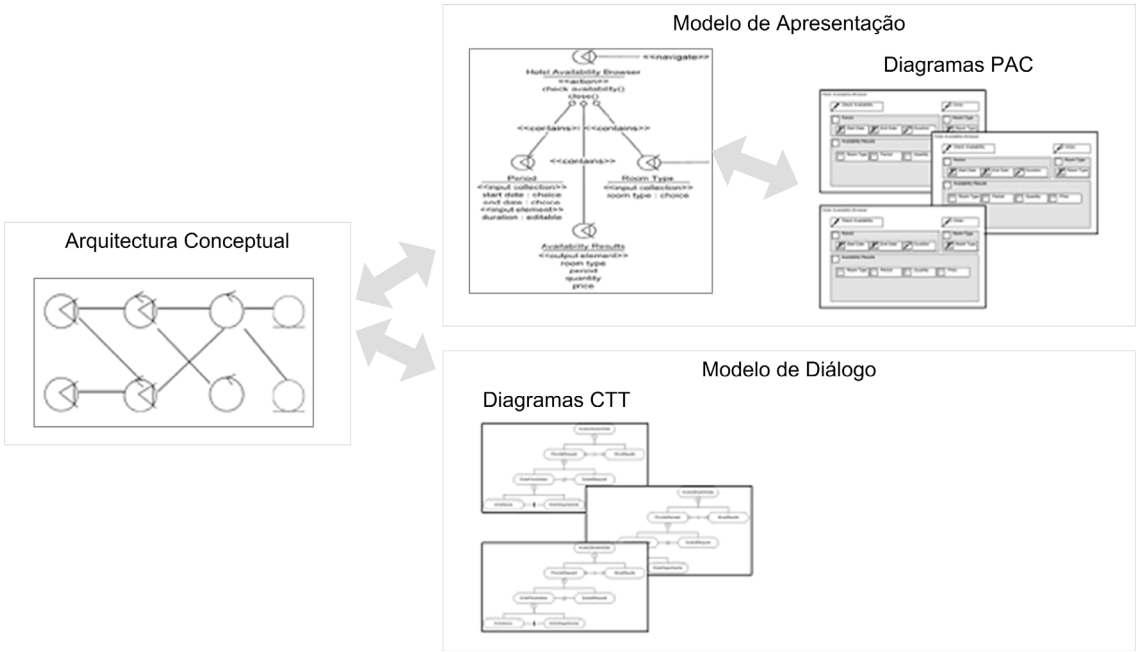

Figura II-X – Os modelos Wisdom com as novas notações propostas: Protótipos Abstractos Canónicos e *ConcurTaskTrees*.

II.4. PROTÓTIPOS ABSTRACTOS CANÓNICOS (PAC)

> "Os protótipos abstractos de interfaces com o utilizador fornecem uma forma de representação para especificar e explorar ideias de desenho visual e de interacção que são intermediárias entre os modelos de tarefas abstractas e os protótipos realísticos. Protótipos Abstractos Canónicos são uma extensão ao desenho centrado na utilização *(usage-centered design*) que fornece um vocabulário para expressar desenho visual e de interacção sem ter em conta detalhes de aparência e de comportamento. Um vocabulário de desenho, abstracto e normalizado, facilita a comparação do desenho de soluções, simplifica a descrição e o reconhecimento de padrões de desenho e assenta a base para melhores ferramentas de software." [Constantine, 2003]

Os modelos de desenho Wisdom, originais, não detalhavam em pormenor como é que os espaços de interacção deveriam ser definidos em termos da estrutura de apresentação. Com o diagrama de apresentação, definido originalmente no Wisdom, eram descritas as relações entre espaços de interacção e detalhadas as acções e os elementos de entrada e saída, mas não era especificada a estrutura de apresentação nem os elementos de interacção associados. Com a aplicação da notação Protótipos Abstractos Canónicos (PAC) ao detalhe de cada um dos espaços de interacção, definidos na arquitectura conceptual, obtém-se um nível de desenho mais refinado, sem entrar em detalhes técnicos ou sugestões de implementação. No método Wisdom os elementos PAC são definidos como extensões ao metamodelo da UML 2.0, sendo utilizados para detalhar os espaços de interacção definidos na arquitectura conceptual e no modelo de apresentação.

O desenho abstracto de interfaces facilita o processo criativo e o diálogo entre os engenheiros de software e designer. Ao evitar as preocupações com os detalhes de implementação da interface os PAC promovem a exploração de soluções centradas na apresentação, na usabilidade e na dinâmica da interacção. A notação PAC propõe um conjunto de componentes abstractos, cada um especificando um material ou uma função interactiva, como por exemplo a introdução de dados ou a disponibilização de notificações. Estes componentes podem ser agrupados ou combinados, permitindo a construção de protótipos abstractos de interface correspondentes aos principais espaços de interacção que encapsulam toda a componente de interacção de um sistema baseado em software. A notação simbólica dos PAC é construída a partir de dois símbolos genéricos, um "material genérico" ou "contentor" (*container*), representado por uma caixa quadrada, e uma "ferramenta genérica" ou "acção" (*action*), representada por uma seta.

Os "materiais genéricos" representam conteúdos, informação, dados ou outros objectos de interface com o utilizador manipulados ou apresentados ao utilizador durante a execução de uma tarefa. As "ferramentas genéricas" representam operadores, mecanismos ou controlos que podem ser utilizados para manipular ou transformar materiais. Os elementos "híbridos" ou "materiais activos" resultam da combinação das duas classes de componentes descritas anteriormente, ou seja, representam qualquer componente com características de ambos os componentes, como por exemplo uma caixa para introdução de texto. Na Figura II-XI encontram-se representados os símbolos relacionado com os três componentes básicos da notação PAC.

Figura II-XI – Os três símbolos associados à notação PAC, respectivamente, à ferramenta genérica, ao material genérico e ao elemento híbrido. Adaptado de [Constantine, 2003].

O conjunto de componentes PAC estabelecidos perfaz um total 21 elementos, sendo constituídos pelos 3 componentes genéricos básicos, 11 componentes de ferramentas abstractas especializadas, 3 contentores abstractos especializados e 7 componentes híbridos abstractos.

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	action/operation*	Print symbol table, Color selected shape
	start/go/to	Begin consistency check, Confirm purchase
	stop/end/complete	Finish inspection session, Interrupt test
	select	Group member picker, Object selector
	create	New customer, Blank slide
	delete, erase	Break connection line, Clear form
	modify	Change shipping address, Edit client details
	move	Put into address list, Move up/down
	duplicate	Copy address, Duplicate slide
	perform (& return)	Object formatting, Set print layout
	toggle	Bold on/off, Encrypted mode
	view	Show file details, Switch to summary

Figura II-XII – Componentes PAC: ferramentas genéricas. Adaptado de [Constantine, 2003].

A Figura II-XII, Figura II-XIII e Figura II-XIV, detalham o conjunto existente de componentes PAC com exemplos de aplicação. O objectivo da criação desta lista de componentes é conduzir ao equilíbrio entre a simplicidade conceptual e a possibilidade de fornecer um conjunto de elementos de trabalho para a definição de funções interactivas com utilidade no desenho de sistemas interactivos.

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	container*	Configuration holder, Employee history
	element	Customer ID, Product thumbnail image
	collection	Personal addresses, Electrical Components
	notification	Email delivery failure, Controller status

Figura II-XIII – Componentes PAC: materiais genéricos. Adaptado de [Constantine, 2003].

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	active material*	Expandable thumbnail, Resizable chart
	input/accepter	Accept search terms, User name entry
	editable element	Patient name, Next appointment date
	editable collection	Patient details, Text object properties
	selectable collection	Performance choices, Font selection
	selectable action set	Go to page, Zoom scale selection
	selectable view set	Choose patient document, Set display mode

Figura II-XIV – Componentes PAC: híbridos ou materiais activos. Adaptado de [Constantine, 2003].

A definição deste conjunto normalizado de componentes facilita a comparação de desenhos alternativos, promovendo a comunicação entre os diferentes intervenientes no desenvolvimento das interfaces com o utilizador. A notação fornece um vocabulário comum para expressar a estrutura de apresentação e de interacção, evitando preocupações com a sua implementação. Adicionalmente, preenche uma lacuna existente entre técnicas de alto nível, como espaços de interacção com classes UML, e técnicas de baixo-nível como protótipos concretos.

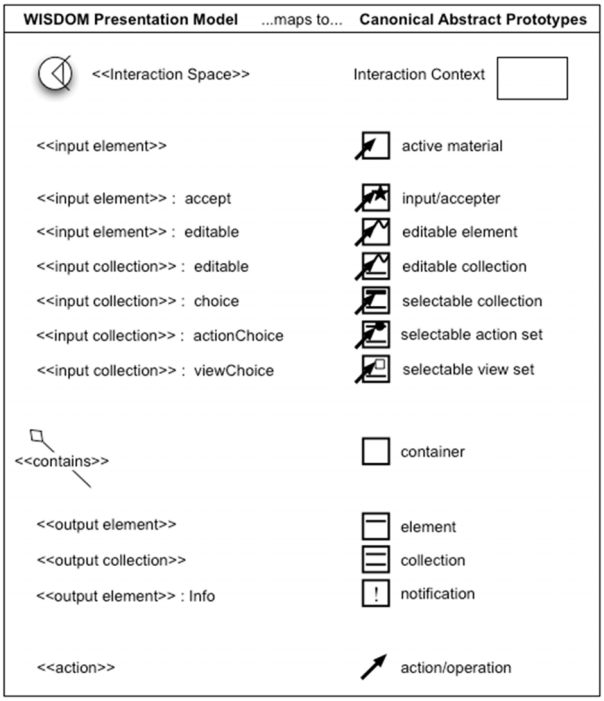

Figura II-XV – O mapeamento entre os estereótipos Wisdom e os componentes PAC. Adaptado de [Campos and Nunes, 2004].

Na Figura II-XV é definida uma correspondência entre os elementos utilizados para o modelo de apresentação Wisdom e os componentes PAC. Na Figura II-XVI podemos ver um exemplo da aplicação dessa correspondência, que demonstra a forma como a modelação da apresentação do Wisdom pode ser detalhada através de PAC sem que se perca a capacidade de abstracção.

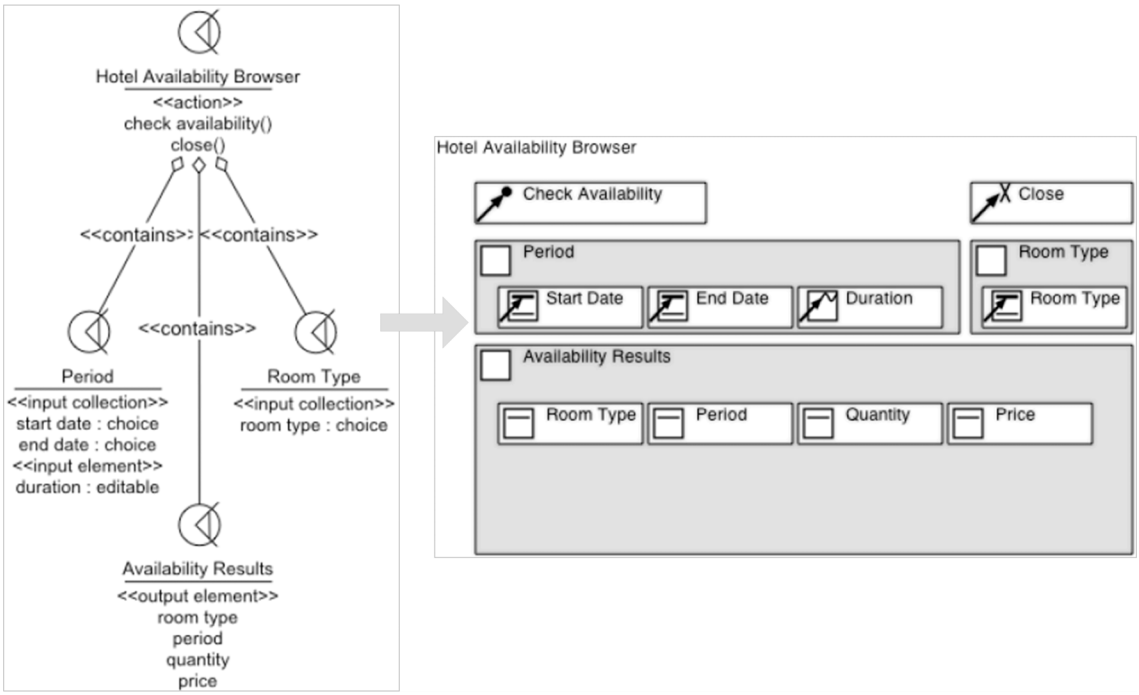

Figura II-XVI – Um modelo de apresentação Wisdom e o correspondente Protótipo Abstracto Canónico para o espaço de interacção "Hotel Availability Browser". Adaptado de [Campos and Nunes, 2004].

Embora a especificação PAC original não seja definida com rigor semântico e formal, esta lacuna foi complementada em [Nóbrega, Nunes, et al., 2005b] ao ser estendido o metamodelo da UML 2.0 com os elementos desta notação. Esta extensão permitiu o desenvolvimento de suporte de uma ferramenta de modelação à execução de diagramas e modelos PAC. O detalhe dos elementos PAC em UML e a ferramenta de suporte à elaboração de modelos de apresentação Wisdom, com esta notação, serão descritos na secção V.1 e na secção VI.1, respectivamente.

II.5. A NOTAÇÃO *CONCURTASKTREES* (CTT) EM UML

A modelação de tarefas é uma técnica generalizada na área de interacção humano-computador, mas é raramente utilizada pela engenharia de software. Os modelos de tarefas capturam o fluxo de interacção, sendo importantes para o suporte de abordagens baseadas em modelos para o desenvolvimento de sistemas interactivos. Na UML existem problemas identificados [Nóbrega, Nunes, et al., 2005] [Nunes and Falcão e Cunha, 2000] para o desenho de sistemas interactivos, pelo que a integração de modelos de tarefas na UML representa um passo em frente relativamente a estas limitações. Nesta secção, iremos descrever a notação *ConcurTaskTrees*, estabelecida e reconhecida na modelação de tarefas, e a sua adaptação ao metamodelo da UML como forma de promover o desenvolvimento de sistemas interactivos, através de uma linguagem de modelação de tarefas formalmente especificada.

As *ConcurTaskTrees* são uma notação, proposta por Fabio Paternò [Paternò, 1999] [Mori, Paternò, et al., 2004] para a modelação de tarefas, onde se definem modelos que descrevem um conjunto de tarefas e subtarefas suportadas por um sistema interactivo e as respectivas relações temporais entre subtarefas. Esta notação oferece uma sintaxe gráfica que suporta uma estrutura hierárquica de tarefas para descrever o diálogo em sistemas interactivos. Os modelos de tarefas são representados na notação CTT como estruturas de árvores invertidas. As tarefas ao mesmo nível na hierarquia da decomposição podem ser ligadas por operadores temporais. A relação temporal, entre tarefas do mesmo nível hierárquico da estrutura em árvore, é representada através de linhas descritas com diferentes operadores temporais. Os operadores temporais definidos na semântica das CTT são:

- ***Choice***: T1 [] T2. É possível escolher entre um conjunto de tarefas, quando uma escolha é feita, a tarefa escolhida pode ser desempenhada. As outras tarefas não estarão disponíveis enquanto a tarefa escolhida não for terminada.

- ***Independent concurrency***: T1 ||| T2. As acções pertencentes a duas tarefas podem ser desempenhadas por qualquer ordem sem qualquer restrição temporal específica.

- ***Disabling***: T1 [> T2. A primeira tarefa é definitivamente desactivada, assim que a primeira acção da segunda tarefa seja desempenhada.

- ***Enabling***: T1 > T2. A primeira tarefa activa a segunda tarefa, quando termina a sua execução.

- ***Suspend/Resume***: T1 |> T2. Este operador dá a possibilidade da tarefa T2 de interromper a tarefa T1 e quando T2 terminar, T1 pode ser reactivada a partir do estado onde foi interrompido pela tarefa T2.

- ***Order independence***: T1 |=| T2. Ambas as tarefas têm de ser desempenhadas mas quando uma é iniciada tem de terminar antes de ser poder ser desempenhada a segunda tarefa.

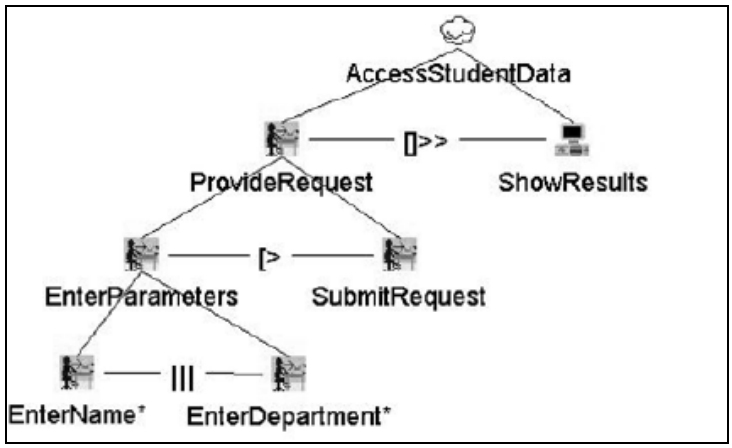

Figura II-XVII – Exemplo de uma árvore CTT. Adaptado de [Mori, Paternò, et al., 2004].

Na Figura II-XVII, é representado um exemplo de uma árvore de tarefas com a notação CTT. A integração da notação *ConcurTaskTrees* com uma linguagem de modelação normalizada como a UML foi identificada como um tópico importante e foram propostos os elementos dos diagramas de actividade da UML 2.0 [OMG, 2005a] [OMG, 2005b] para representar a semântica CTT. Nesta proposta [Nóbrega, Nunes, et al., 2005] os autores demonstraram que a semântica da *ConcurTaskTrees* pode ser definida em termos da semântica das actividades UML, permitindo uma integração de conceitos de modelos de tarefas em UML. Nesta abordagem proposta foram introduzidos conceitos necessários à modelação da notação CTT em UML:

- ***TaskTree*** como uma especialização do conceito *actividade* em UML;
- ***Task*** para modelar o conceito de *tarefa*;
- ***TaskEdge*** para modelar os *operadores temporais*;

Estes novos conceitos permitem a criação de um tipo especializado de diagramas de actividade UML para modelar árvores de tarefas, denominado de "Diagrama TaskTree". Na Figura II-XVIII, pode ser visto um exemplo de uma árvore de tarefas com a notação CTT na adaptação proposta para UML 2.0, onde se destaca também a descrição dos diversos elementos gráficos da notação.

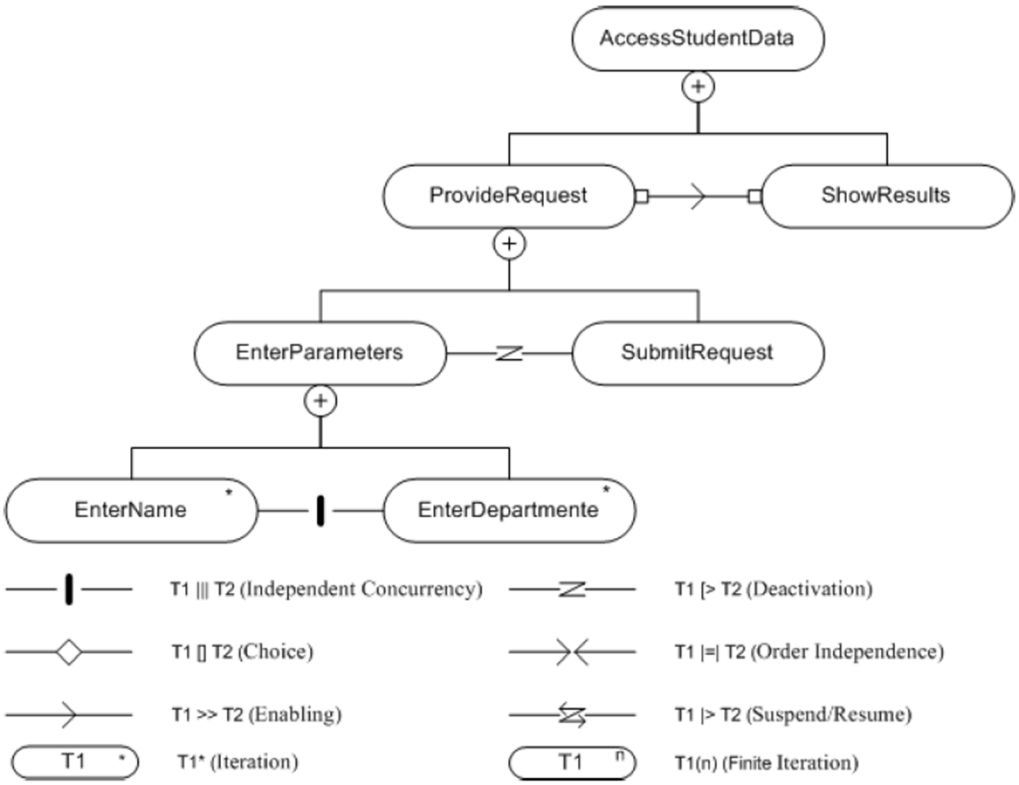

Figura II-XVIII – Exemplo da notação proposta para representar árvores CTT em UML. Adaptado de [Nóbrega, Nunes, et al., 2005].

A notação utilizada para os operadores temporais é inspirada nas notações para as actividades UML, nomeadamente os operadores *Independent Concurrency*, *Choice* e *Deactivation*. Existem semelhanças com o nó *Fork/Join*, nó *Decision* e o *Interrupting Edge*, respectivamente. As relações entre a tarefa e as suas subtarefas refinadas são inspiradas na notação para representar pacotes e os seus conteúdos no mesmo diagrama, fornecendo uma adequada representação hierárquica.

Depois da formalização da notação CTT em UML, esta foi adoptada para definir o modelo de diálogo do método Wisdom. Isto permitiu a descrição da dinâmica da interacção dos elementos <<task>>, identificados na arquitectura conceptual Wisdom, através da UML, uma linguagem de modelação formal e generalizada, aliada a uma notação de modelação de tarefas (CTT). Esta união traduziu-se numa maior expressividade para descrever o comportamento da interacção das interfaces com o utilizador. Na secção V.1 serão detalhados, com maior pormenor, os elementos e o metamodelo UML 2.0 estendido com a notação CTT.

III. A ESPECIFICAÇÃO DE ARQUITECTURA GUIADA POR MODELOS

Nesta secção é descrita a abordagem de arquitectura guiada por modelos (*Model Driven Architecture* - MDA), abordando os seus objectivos, princípios e conceitos. A abordagem descrita neste trabalho pretende alinhar o método Wisdom com os requisitos da especificação MDA de forma a conduzir à geração automática de código a partir dos modelos produzidos durante a actividade de desenho. O método Wisdom não define especificamente nenhuma técnica para a transformação dos modelos de desenho para código. Contudo ao introduzirmos os princípios e métodos defendidos na MDA, estamos a dotar o Wisdom da capacidade de ser executada a transformação de modelos para código de modo formal e aberto, recorrendo a normas estabelecidas e reconhecidas pela indústria de software como a UML [OMG, 2005a] [OMG, 2005b] e o XMI [OMG 2005c] [Grose, Doney, et al. 2002].

III.1. MISSÃO DA ABORDAGEM

A *Object Management Group* (OMG) é uma organização, sem fins lucrativos, composta por empresas, cuja principal missão é o desenvolvimento de especificações e normas para tecnologias baseadas em objectos. As especificações são desenvolvidas de forma aberta e estão disponíveis gratuitamente, promovendo a integração de aplicações, procurando garantir a reutilização de componentes, a interoperabilidade e a portabilidade do software. A OMG é responsável por normas e especificações como o *Common Object Request Broker Architecture* (CORBA), a *Unified Modeling Language* (UML), a *Common Warehouse Metamodel* (CWM), o *Meta-Object Facility* (MOF) e o *XML Metadata Interchange* (XMI).

A mais recente especificação proposta pela OMG é a *Model Driven Architecture* (MDA) [Soley and OMG, 2000] [OMG, 2003] que é baseada nas normas de modelação já definidas pela OMG, como o MOF, a UML, o XMI e o CWM. Esta especificação define uma abordagem ao desafio da constante mudança nas regras de negócio das organizações e à consequente evolução tecnológica dos sistemas de software. A MDA separa a lógica de negócio da base tecnológica de suporte, sendo criados modelos independentes da plataforma tecnológica, por exemplo, com a utilização da UML ou de outras normas abertas e formais. Estes modelos documentam as regras de negócio e de domínio, separando-as do código que implementa o sistema, permitindo, desta forma, que os aspectos de negócio e técnicos do sistema possam evoluir de forma independente.

As técnicas e especificações de modelação têm evoluído muito nos últimos anos e organizações, como a OMG e a W3C, têm produzido tecnologias como o UML [OMG, 2005a] [OMG, 2005b], XMI [OMG, 2005c], MOF, XML, XML *Schema* [Vlist, 2002], etc. Quando se trata de desenvolver e instalar aplicações para várias arquitecturas de software, estas tecnologias podem ser utilizadas para integrar, de uma forma mais completa, os diferentes sistemas de software, que têm de funcionar em interligação. Todavia, continuam a existir muitos problemas no desenvolvimento e manutenção de sistemas de software, como por exemplo, as aplicações e base de dados legados, onde é difícil a introdução de novos requisitos, vários sistemas de *middleware* que competem entre si, etc., fazendo com que o custo de manutenção de uma aplicação, assim como o custo de integração entre aplicações, seja superior ao custo inicial de implementação. Além disso, a necessidade actual de troca de informação e de interoperabilidade entre ferramentas e *middleware* de vários fabricantes é hoje fundamental na indústria do software. Normalmente uma empresa não se baseia, apenas, numa plataforma de *middleware,* para suportar o seu negócio. Ou seja, o problema de integração continua a existir e as empresas tem de encontrar

uma solução para preservar o investimento no desenvolvimento de novos componentes de software, de acordo com as mudanças no negócio e na tecnologia que o suportam.

Como forma de minimizar os problemas no desenvolvimento e evolução de sistemas com alguma complexidade, na engenharia de software, a utilização de modelos tem sido cada vez mais incentivada e aceite. A evidência encontra-se na adopção de diferentes normas e notações, que permitem visualizar e documentar o sistema, retirando pormenores desnecessários e permitindo o enfoque nos detalhes relevantes. A adopção generalizada da modelação deu origem ao paradigma dirigido por modelos (*Model Driven Development* - MDD) [Brown, 2004], onde se defende que os modelos são os principais artefactos de um processo de desenho de software, em detrimento do código. Assim, são os modelos que permitem a geração de outros modelos ou código, recorrendo a processos automáticos ou semi-automáticos. Os diferentes níveis de automação variam, desde a geração de esqueletos de código até a codificação completa de um programa. No desenvolvimento conduzido por modelos, um sistema pode e deve ser analisado de diversos aspectos e perspectivas, podendo ser utilizadas diversas notações e conceitos de modelação, para evidenciar uma vista particular de um sistema (através de um ou mais diagramas). Existe também muitas vezes necessidade de transformar um modelo noutro, com um diferente nível de abstracção ou então de obter um modelo semelhante, com uma representação diferente.

A abordagem MDA é, assim, a solução preconizada pelo OMG para resolver a necessidade de melhorar os processos de evolução dos sistemas e facilitar a interoperabilidade, tendo por base a utilização de normas abertas e os conceitos do desenvolvimento conduzidos por modelos. A especificação MDA define conceitos e estruturas conceptuais para construir sistemas, baseados em modelos, suportando todo o ciclo do desenvolvimento do software: análise, desenho, implementação, instalação, manutenção e evolução do sistema. A MDA define uma arquitectura de modelação, que fornece um conjunto de linhas orientadoras para estruturar especificações, que são, depois, expressas por modelos, tendo por objectivos a separação da modelação da lógica e comportamento do negócio, dos detalhes de implementação numa determinada plataforma tecnológica.

Um dos principais objectivos de uma arquitectura baseada na abordagem MDA, é a preparação para a evolução do sistema, com as mudanças que irão surgir, tanto ao nível do negócio, como ao nível tecnológico. Com esta abordagem, pretende-se desenvolver sistemas de uma forma mais eficiente e eficaz, preenchendo as necessidades dos clientes e oferecendo uma maior flexibilidade na evolução do sistema.

Depois de enquadrada a especificação MDA, no âmbito da organização que a (OMG) está a promover, e de definida a sua principal missão e objectivos, iremos nas próximas secções

detalhar esta abordagem relativamente aos seus conceitos, princípios e a sua aplicação num processo de desenvolvimento.

III.2. PRINCÍPIOS E CONCEITOS

Como já foi referido, anteriormente, o desenvolvimento conduzido por modelos é um dos conceitos base da MDA. Neste paradigma a criação de modelos e as transformações, entre eles, devem ser assentes num conjunto de metamodelos, isto porque a análise e transformação de modelos requerem uma forma que não seja ambígua de descrever a sua semântica. Neste sentido, os modelos numa abordagem deste tipo devem ser descritos por outros modelos designados na terminologia por metamodelos. Devido à importância dos metamodelos, e da sua definição formal, para a modelação, a OMG definiu uma série de níveis de metamodelação assim como uma linguagem para descrever esses modelos: a *Meta Object Facility* (MOF). Um metamodelo utiliza a MOF para, formalmente, definir a sintaxe abstracta de um conjunto de componentes de modelação.

Outro conceito chave, associado ao desenvolvimento conduzido por modelos, é o de transformação. As transformações são o processo de conversão de um modelo noutro modelo ou de passagem de um modelo para código. As transformações aplicam-se a aspectos do mesmo sistema, embora com perspectivas ou níveis de abstracção diferentes. Como um sistema é descrito por diversos aspectos, que interessam a diferentes intervenientes do processo de desenvolvimento, podem ser utilizadas diversas notações de modelação de modo a descrever as várias vistas do sistema. É, normalmente, necessário converter diferentes perspectivas do sistema num nível de abstracção equivalente. Por exemplo, uma transformação do modelo que facilita a passagem de uma vista estrutural para uma vista comportamental. Por outro lado, existe também a necessidade de converter modelos de modo a transformar uma perspectiva com um determinado nível de abstracção para outro mais detalhado. As transformações podem ser aplicadas a descrições abstractas de aspectos do sistema, de modo a adicionar detalhe, a tornar a descrição mas concreta, ou a executar uma conversão entre representações do sistema. Estas transformações podem ser realizadas de forma manual, embora com recurso a ferramentas de software, ou de forma automática.

Tendo por base os conceitos de modelação, metamodelação e transformação num processo de desenvolvimento assente em modelos, na MDA, é recomendado um método sistemático para guiar o ciclo de desenvolvimento, baseado em quatro princípios:

i) Os modelos definidos numa notação formal, onde a semântica é descrita com metamodelos, são uma peça basilar para compreender os sistemas de informação.

ii) A construção dos sistemas de software deve ser organizada em torno de um conjunto de modelos, aos quais se aplica uma série de transformações. As transformações de modelos para código, estão relacionadas com a implementação e podem ser executadas de forma automática ou não.

iii) Um suporte formal para a descrição dos modelos, através de metamodelos, facilita a integração e transformação entre modelos, sendo a base para a automação através de ferramentas de software.

iv) A adopção, generalizada, desta abordagem conduzida por modelos (*model-driven*), requer normas aceites pela industria, de modo a fomentar a interoperabilidade e soluções abertas, promovendo a concorrência entre os fornecedores e conduzindo à entrega de melhores soluções aos clientes.

Com a definição do conjunto de princípios para a utilização de modelos e para a aplicação de transformações entre eles foi definido um enquadramento conceptual para a MDA. Este enquadramento conceptual, que será descrito na próxima secção, promove um processo de desenvolvimento de software controlado e eficiente, conduzido por modelos com diferentes perspectivas e níveis de abstracção.

III.3. ENQUADRAMENTO CONCEPTUAL DA MDA

A crescente aceitação da utilização de modelos para representar as ideias chave nos domínios do problema e da solução no desenvolvimento de software, permite à MDA [Soley and OMG, 2000] [OMG, 2003] fornecer um conjunto de recomendações para a utilização de modelos e para a aplicação de transformações entre eles através de um enquadramento conceptual. Este enquadramento, conceptual, promove um processo de desenvolvimento de software conduzido por modelos com diferentes perspectivas e níveis de abstracção, assente nos quatro princípios definidos anteriormente (ver secção III.2) e num conjunto de normas para expressar os modelos, as relações entre modelos e as transformações entre eles.

A MDA define um sistema estruturado em modelos, de uma forma resumida, com os modelos independentes da plataforma (PIM - *Platform-independent Model*) e específicos da plataforma (PSM - *Platform-specific Model*). Um modelo PIM descreve o sistema, representando a estrutura e funcionalidade do negócio, sem estar comprometido com os detalhes tecnológicos. Um modelo PSM, é um modelo a um nível de abstracção mais detalhado e onde se reflectem os requisitos para a plataforma tecnológica que irá suportar a concretização do sistema. Os modelos PIM devem ser definidos numa linguagem de modelação normalizada, fornecendo uma especificação formal da estrutura e da funcionalidade do sistema, abstraindo os detalhes técnicos de implementação. O modelo PSM, por outro lado, descreve a implementação da funcionalidade descrita no PIM, numa plataforma específica, devendo ser obtido através de uma determinada transformação (*mapping*) baseada no PIM.

Abstrair a estrutura lógica e o comportamento do sistema, através do PIM, dos aspectos específicos de implementação, definidos pelos modelos PSM, trazem vários benefícios:

i) É mais fácil a verificação dos modelos sem estarem declaradas as especificidades da plataforma de implementação.

ii) É mais fácil produzir implementações, em diferentes plataformas, enquanto a base de validação da estrutura e do comportamento do sistema se mantiver no modelo PIM.

iii) A integração e a interoperabilidade entre sistemas são definidas de forma independente da plataforma.

Conforme já foi referido, anteriormente, outro dos conceitos chave de toda a abordagem é a noção de *mapping* ou transformação. Os *mappings* são um conjunto de regras e técnicas

utilizadas para modificar um modelo de modo a obter outro modelo. Estas transformações podem ser realizadas entre as seguintes combinações de modelos:

i) PIM para PIM: geralmente associadas ao refinamento de modelos deste tipo.

ii) PIM para PSM: realizada quando o modelo PIM está suficientemente detalhado para ser projectado para a infra-estrutura de implementação. Esta transformação é baseada nas características da plataforma alvo.

iii) PSM para PSM: geralmente associada ao refinamento dos modelos, dependentes da plataforma.

iv) PSM para PIM: geralmente associada à reengenharia, ou seja, à obtenção dos modelos abstractos de implementação existentes numa tecnologia em particular.

Na Figura III-I estão representados os diferentes fluxos que podem ser estabelecidos entre os modelos PIM e PSM, definidos pela especificação MDA num contexto de um processo de desenvolvimento de software.

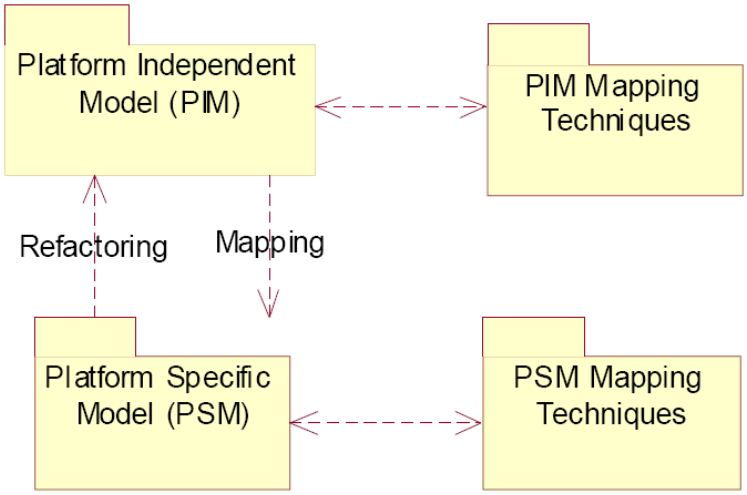

Figura III-I – Relações de mapeamento entre modelos PIM e PSM. Adaptado de [Mallia, 2005].

Os modelos e as transformações, entre eles, devem ser especificados recorrendo a normas abertas, através das quais a indústria de software pode atingir um nível de interoperabilidade, que anteriormente não era possível. A abordagem MDA não determina, especificamente, que normas ou tecnologias devem ser empregues na sua aplicação. Pelo contrário, apenas recomenda a utilização das normas de modelação, já definidos por esta organização como o MOF, a UML e o *XML Metadata Interchange* (XMI).

Devido à importância dos metamodelos (ver secção III.2), nas técnicas de modelação, é recomendada a linguagem *Meta Object Facility* (MOF), no âmbito da MDA, para definir e normalizar os modelos que descrevem outros modelos. Para a criação dos modelos PIM, no decorrer de um processo de desenvolvimento, é recomendada a utilização da UML, que tem vantagens relativamente a outros tipos de linguagens de modelação. Em particular porque, a UML pode ser expressa de forma gráfica ou de forma textual. Outra vantagem, é que um modelo UML pode incluir restrições ao comportamento, através da linguagem formal OCL (*Object Constraint Language*). A especificação formal de restrições, em vez de texto em formato livre, reduz a ambiguidade na especificação permitindo: (i) fornecer ao programador instruções mais precisas, (ii) diminuir a quantidade de trabalho necessária para conseguir diversas implementações da mesma especificação, (iii) definir testes de conformidade para as diversas implementações.

Relativamente à criação dos modelos PSM, é recomendado que sejam também realizados em UML, sendo específicos para uma determinada plataforma de *middleware* (J2EE, .Net), apesar do UML ser independente das tecnologias. A descrição destas características específicas pode ser efectuada através de um perfil UML, onde se utilizam um conjunto de extensões ao UML utilizando estereótipos e etiquetas (*tagged values*). Os conceitos de perfil UML, estereótipo e etiqueta (*tagged value)* são desenvolvidos mais à frente neste trabalho na secção V.2.2.

Relativamente à descrição formal das transformações entre modelos, em particular entre os modelos PIM e PSM, encontra-se em definição a linguagem interrogativa *Query View Transformation* (QVT) [Tata Consultancy, 2003] [QVT-Merge Group, 2005] que tem por objectivo criar uma especificação que forneça meios normalizados, como linguagens ou modelos, para expressar transformações, que possam ser manipulados por pessoas e executados por computadores. No âmbito da MDA, a QVT é neste momento uma referência como um meio de definir formalmente as transformações entre modelos.

A combinação das diferentes normas descritas anteriormente (MOF, UML, OCL, QVT) define uma base conceptual para a aplicação dos princípios e conceitos MDA, num processo de desenvolvimento, permitindo: (i) a portabilidade de aplicações, ao possibilitar que o mesmo modelo possa ser realizado em múltiplas plataformas através de normas auxiliares de transformação (*mapping*); (ii) a integração baseada em modelos entre diferentes aplicações e componentes, garantindo a interoperabilidade baseada nas relações definidas nos modelos e (iii) a utilização de várias tecnologias de *middleware* como o J2EE, .NET e XML (*Extensible Markup Language*) sem ter de ser

realizada outra análise ou desenho do modelo independente da plataforma (PIM); e finalmente (iv) a utilização de MOF e de XMI (*XML Metadata Interchange)* promovendo a interoperabilidade entre as ferramentas de modelação.

III.4. CICLO DE DESENVOLVIMENTO

Depois de descritos os objectivos e princípios base da MDA, assim como a arquitectura de modelos para a sua aplicação, nesta secção descreve-se, de forma sucinta, o ciclo de desenvolvimento recomendado pela especificação MDA, que permite aplicar os conceitos e o enquadramento dos modelos recomendados.

O primeiro passo para aplicar a MDA, será criar o modelo independente da plataforma PIM através, por exemplo, da linguagem UML. Nesta fase podem também ser representadas, num modelo com maior refinamento, restrições e condições, por exemplo, através de OCL. O modelo PIM pode ser refinado, várias vezes, até o nível de detalhe pretendido.

Com a utilização de uma ferramenta baseada em MDA, aplica-se uma transformação, ao modelo PIM, de modo a gerar um outro modelo UML especifico da plataforma – o modelo PSM. Este modelo é conseguido através de transformações do PIM para um modelo específico de tecnologia de *middleware*, recorrendo a regras pré-definidas, podendo ser utilizado, por exemplo, um perfil UML para a descrição formal e normalizada dos seus elementos. O modelo PSM, também, pode ser refinado diversas vezes, até atingir o nível de detalhe necessário.

Com uma ferramenta MDA, poderemos ainda aplicar regras normalizadas de mapeamento de modo a gerar os modelos específicos, os PSM, para diversas plataformas de *middleware* a partir do mesmo modelo PIM. Consequentemente, poderá ser gerada, a maior parte do código, a implementar numa determinada plataforma tecnológica.

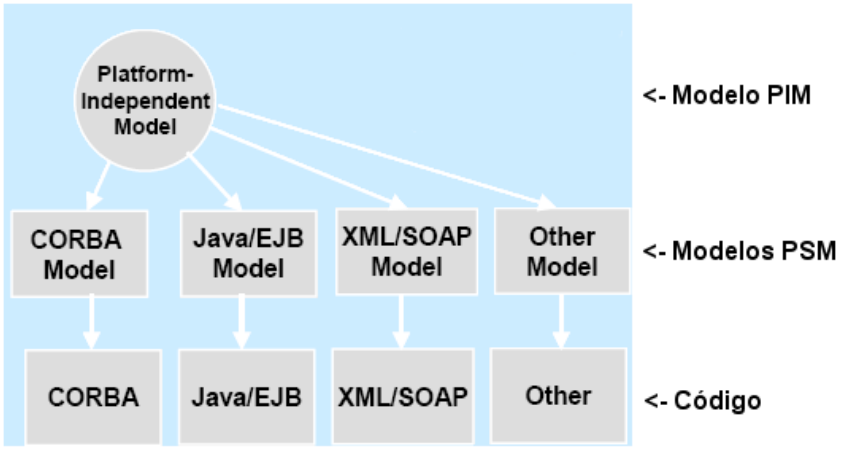

Figura III-II – As várias fases e modelos numa arquitectura MDA. Adaptado de [Soley, 2003].

Na Figura III-II estão representadas as várias fases e respectivos modelos a aplicar numa arquitectura MDA. De uma forma resumida, podemos verificar que, no ciclo de vida da MDA, o arquitecto deve concentrar-se em criar o modelo de negócio independente da plataforma de implementação. Posteriormente, o responsável pelo desenho do *middleware* aplica uma transformação do modelo PIM num modelo mais específico, definido, por exemplo, através de um perfil UML apropriado, tendo em perspectiva a plataforma em que este será implementado. Finalmente o programador utiliza o modelo PSM, para gerar o código e completar o que falta para a implementação da aplicação.

III.5. O MÉTODO WISDOM E A MDA

Sendo uma das principais missões da OMG, a resolução dos problemas de integração, fornecendo especificações abertas de interoperabilidade, a proposta MDA segue essas linhas orientadores ao promover a integração e evolução dos sistemas de uma organização, abrangendo todo o ciclo de vida de um sistema desde a modelação do negócio, desenho, desenvolvimento de componentes, a integração, a instalação, à manutenção e à evolução.

A MDA define uma arquitectura para estruturar modelos que fazem uma separação de conceitos relativamente à integração, interoperabilidade e portabilidade dos sistemas. Desta forma a MDA explora a capacidade que poderá ser descrita como de *zoom-in* e *zoom-out* de qualquer modelo de um sistema. Desta forma, é possível expor ou esconder o detalhe dos objectos e do seu comportamento, em particular porque a arquitectura separa, de uma forma uniforme, as especificações das suas realizações. A abordagem encoraja a utilização eficiente de modelos num processo de desenvolvimento de software, recomendando uma forma de organizar arquitecturas de informação, gerindo os modelos de suporte e as respectivas transformações entre eles.

Ao alinhar o método Wisdom com a MDA, iremos obter as vantagens já referidas, conduzindo à automação da geração de código parcial ou completa, facilitando a implementação dos modelos de desenho de um sistema em várias plataformas tecnológicas, promovendo a reutilização do desenho e da análise do negócio e do domínio da aplicação.

O método Wisdom, cumpre um requisito fundamental, definido pela MDA, que é o facto de ser assente e conduzido por modelos e em metamodelos bem definidos. Outra questão importante, embora não fundamental, é a linguagem de modelação utilizada no método ser a UML. As três principais perspectivas de modelos definidas no método Wisdom, arquitectura conceptual, modelo de apresentação e modelo de diálogo, irão ser enquadradas na definição de PIM, sendo descritos os metamodelos que servem de suporte à semântica das notações utilizadas. Será, depois, proposta uma notação e elementos para um modelo PSM, devidamente fundamentado por um metamodelo, seguindo os requisitos de utilização de modelos e metamodelos definidos pela MDA. Serão também abordadas as regras e técnicas de transformação a aplicar de modo a que os modelos Wisdom possam ser convertidos nos modelos PSM e, posteriormente, em código. Estes itens irão ser discutidos, em detalhe, na secção V. , no âmbito da aplicação do método Wisdom no contexto de desenvolvimento de sistemas Web, seguindo os requisitos da iniciativa MDA.

IV. A FRAMEWORK HYDRA

No Sector de Comunicações e Informática da Universidade da Madeira (SCI-UMa), para o desenvolvimento de sistemas de software de suporte à actividade pedagógica, foi tomada a decisão de que o sistema deveria assentar num ambiente de interacção Web, com três camadas (apresentação, negócio e dados), sendo adoptado o processo de desenvolvimento Wisdom [Nunes and Falcão e Cunha, 2000] [Nunes, 2001]. Posteriormente, para automatizar alguns fluxos foi desenvolvida, para a camada de apresentação com o utilizador, uma *framework* de suporte ao desenvolvimento Web. A *framework* Hydra foi construída com base no método Wisdom e em padrões de interacção, identificados durante vários projectos Web, implementando mecanismos para a interface com o utilizador. As camadas de negócio e de dados não são abrangidas por esta *framework*, embora sejam fornecidos os mecanismos de interligação. A *framework* Hydra fortalece o suporte à implementação dos modelos que definem a estrutura e comportamento da interface com o utilizador, num contexto de desenvolvimento de projectos Web.

Este trabalho na área de modelação e geração automática de código, conforme já descrito anteriormente, tem como suporte a metodologia Wisdom e a *framework* Hydra desenvolvida na linguagem PHP 4.x [PHP, 2006]. Nesta secção do documento, são descritos os principais conceitos, estruturas e serviços da Hydra.

IV.1. MOTIVAÇÃO E OBJECTIVOS

Durante o trabalho desenvolvido no Sector de Comunicações e Informática da Universidade da Madeira (SCI-UMa), surgiu a necessidade de automatizar certos passos do processo de desenvolvimento com o objectivo de melhorar a produtividade e aumentar a reutilização de software. Neste sentido, foi construída, na área de interfaces com o utilizador, uma *framework* de suporte ao desenvolvimento Web. Esta base de trabalho foi construída, tendo em consideração a metodologia Wisdom e alguns padrões de interacção, identificados durante a análise e implementação de vários projectos Web, pela equipa de desenvolvimento do SCI-UMa.

Os padrões em software descrevem soluções para problemas recorrentes no desenvolvimento de sistemas de software, sendo definido o problema, a solução e quando aplicar a solução. Um dos principais objectivos é a reutilização de soluções de desenho encontradas anteriormente. Os padrões de interacção, por outro lado, focam-se em soluções para problemas que os utilizadores finais encontram ao interagir com o sistema de software, conduzindo a uma maior usabilidade e qualidade da interface. Os padrões de interacção promovidos pela Hydra são um compromisso entre os objectivos dos padrões de software e os objectivos dos padrões de interacção com o utilizador, tentando conciliar, na maior parte dos casos, os dois conceitos.

A opção de desenvolvimento de uma *framework* de raiz, em vez de ser utilizada uma já desenvolvida, como por exemplo a *Struts* [Hall, 2004], teve como principal razão a necessidade de haver uma estrutura de implementação, que desse suporte à metodologia de desenvolvimento adoptada pela equipa do SCI-UMa (Wisdom). Em particular, na passagem da actividade de desenho para a actividade de implementação de aplicações Web, promovendo a rastreabilidade entre os modelos e o código, sendo este um requisito importante num processo de desenvolvimento baseado em modelos. A *framework Struts* contém a separação de conceitos defendida pelo Wisdom, sendo centrada numa classe que gere as acções despoletadas pelo utilizador, mas outros elementos e conceitos poderiam ser complexos de adaptar e de rastrear entre os modelos de desenho (modelo de apresentação, modelo de dialogo) e os artefactos de código produzidos pela *Struts*.

Assim, optou-se por implementar uma *framework* que além de permitir definir um conjunto de mecanismos básicos para a implementação de aplicações Web, consegue manter uma rastreabilidade entre a arquitectura conceptual, os modelos de desenho e a implementação, trazendo as naturais vantagens de um desenvolvimento centrado numa arquitectura e em modelos. A *framework* em questão implementa mecanismos relativos ao comportamento e

estrutura da interface com o utilizador, ou seja, implementa apenas a camada de apresentação. A camada de dados e de lógica de negócio não é abrangida pela *framework*, embora sejam fornecidos os mecanismos de interligação entre a camada de apresentação e estas camadas.

A framework foi projectada e construída no início de 2004, pelo autor deste trabalho, com uma primeira versão constituída por classes e componentes básicos de suporte a interfaces Web. Posteriormente, no final de 2004, foram integrados no projecto da Hydra o Eng.º Gonçalo Teixeira e Eng.º Jorge Freitas, possibilitando um maior desenvolvimento de novas funcionalidades e mecanismos da *framework*. Actualmente, à altura da elaboração deste trabalho, a Hydra dá suporte aos sistemas Web, desenvolvidos e mantidos pelo SCI-UMa, nomeadamente o Portal da Universidade (http://www.uma.pt), o Serviço de Informação dos Alunos (https://infoalunos.uma.pt) e o Serviço de Informação dos Docentes (https://sidoc.uma.pt).

A Hydra facilita, assim, a passagem do desenho para a implementação, ao concretizar alguns dos conceitos e elementos utilizados no método Wisdom, relativamente à parte de desenvolvimento da interface com o utilizador.

IV.2. MECANISMOS E SERVIÇOS

A *framework* Hydra implementa mecanismos de suporte ao desenvolvimento, apenas, relativos à parte de apresentação, ou seja, interfaces com o utilizador, em particular de aplicações Web. A Hydra foi desenvolvida de modo a permitir uma forte ligação e rastreabilidade com a fase de desenho do método de desenvolvimento utilizado, mais, concretamente, aos modelos de estrutura e de comportamento da interface com o utilizador, definidos no método Wisdom.

Na Hydra, estão definidos padrões de comportamento da interface, que foram identificados ao longo de vários projectos. Os padrões de comportamento implementados na *framework* foram identificados como resposta a problemas comuns e recorrentes encontrados em várias situações e ao longo de vários projectos. Desta forma a Hydra implementa mecanismos de interacção que facilitam o desenvolvimento de aplicações ao ter já soluções para problemas como:

- Operações CRUD *(Create, Read, Update e Delete*);
- Estruturas de apresentação: Listas, Colecções de Dados, *Combos*, Grupos de Campos, Tabuladores;
- Navegação entre módulos, espaço de agregação e espaços de interacção;
- Troca de dados e interligação entre espaços de interacção;

Uma vez que o método Wisdom utiliza a notação CTT em UML, no modelo de diálogo da actividade de desenho, foi implementado na Hydra, no âmbito deste trabalho, o algoritmo de execução das árvores CTT associadas a cada espaço de interacção. Aqui define-se a semântica do comportamento da interface em termos de elementos de interacção presentes na interface, que são colocados no estado activo ou bloqueado. Este algoritmo, que será descrito em detalhe na secção V.3, permite a utilização das definições e semântica dos modelos de desenho, para executar e determinar o comportamento dos elementos da interface.

O comportamento de uma interface, para além de ser expresso na notação CTT em UML, é ainda pré-definido nos diferentes padrões de interacção implementados. Se existir alguma especificidade, ou então se os padrões de interacção implementados não dão resposta aos requisitos da interface, existe a possibilidade de implementar esse comportamento ao estender alguns métodos das classes dos elementos de espaços de interacção.

As regras de comportamento, numa aplicação suportada pela Hydra, são definidas em artefactos de código, ou seja, ficheiros em formato XML, associados à classe do elemento do

modelo de desenho. A estrutura dos ficheiros XML, ou seja, os elementos e atributos que fornecem a configuração às classes de código, têm um esquema próprio definido internamente pelo SCI-UMa mas que é formalizado e validado através de um conjunto de S*chemas* XML [Vlist, 2002]. Um *Schema* XML é uma linguagem para definição de regras de validação para XML, fornecendo um meio de definir como deve ser a estrutura, conteúdo e semântica de documentos em formato XML.

Relativamente à interligação da Hydra com as outras camadas da aplicação, nomeadamente lógica de negócio e dados, a *framework* foi projectada, permitindo que possa, posteriormente, ser utilizada numa arquitectura de implementação em duas ou três camadas. Numa arquitectura em duas camadas, onde existem a camada de apresentação e a de dados, as classes de interfaces gerados pela Hydra podem ligar-se a bases de dados, por exemplo *MSSQL Server* e *MySQL*. Numa arquitectura em três camadas, onde existem as camadas de apresentação, de negócio e a de dados, as classes de interface da Hydra utilizam a tecnologia *Webservices* como forma de diálogo com a camada de negócio, não interagindo directamente com os dados.

Relativamente à parte estrutural da apresentação a gerar para uma interface, cujos elementos são retirados directamente da perspectiva do modelo de apresentação definido pelo Wisdom, a Hydra utiliza um sistema de *templates* HTML, que são interpretadas pelas classes dos elementos da aplicação e que interliga com a configuração do comportamento e dos dados. Em cada módulo da aplicação são utilizadas as *templates* comuns ao projecto de modo a gerar uma interface uniforme em toda a aplicação diferenciando-se no conteúdo do "corpo" a apresentar. Seguidamente, em cada espaço de interacção são colocados os elementos de interacção que irão permitir o diálogo com o utilizador. Este sistema de *templates*, para a parte estrutural da apresentação, obedece à separação de conceitos entre a apresentação e o comportamento das interfaces com o utilizador.

Depois de serem descritos os serviços fundamentais da *framework*, relativamente à metodologia e à separação de conceitos, defendida entre a estrutura da apresentação e a estrutura do comportamento, iremos descrever de forma, sucinta, alguns dos mecanismos suportados pela *framework*.

Na *Hydra*, os mecanismos e serviços de suporte base são os seguintes:

- Classes de gestão de ligações a base de dados;
- Classes de ligações a *Webservices*;
- Classes de gestão de sessão e de autenticação;
- Classes de gestão de idiomas;
- Classe de gestão de menus da aplicação;

- Classe de gestão de *logs* da aplicação;
- Gestão de regras de validação de campos;
- Gestão de regras de bloqueio de Campos;
- Classes de gestão de excepções e mensagens;
- Classes de ligação a ferramentas externas de *reporting* (*CrystalReports* e *OpenReports*);
- Classes de gestão de *templates* com as estruturas de apresentação;
- Classes de gestão manipulação de ficheiros (*upload/download*);
- Classes de execução de árvores CTT em UML;

As regras de bloqueio e validação, podem-se aplicar a botões de acção ou campos de introdução de dados. Nos ficheiros de configuração de cada espaço de interacção, são definidos os parâmetros para a validação e bloqueio de botões e campos associados aos elementos de interacção identificados no desenho. O bloqueio de campos e botões, além de serem definidos pela árvore CTT, definida no modelo de diálogo, associada ao espaço de interacção, pode ser definido com regras mais específicas através de funções e processos pré-definidos.

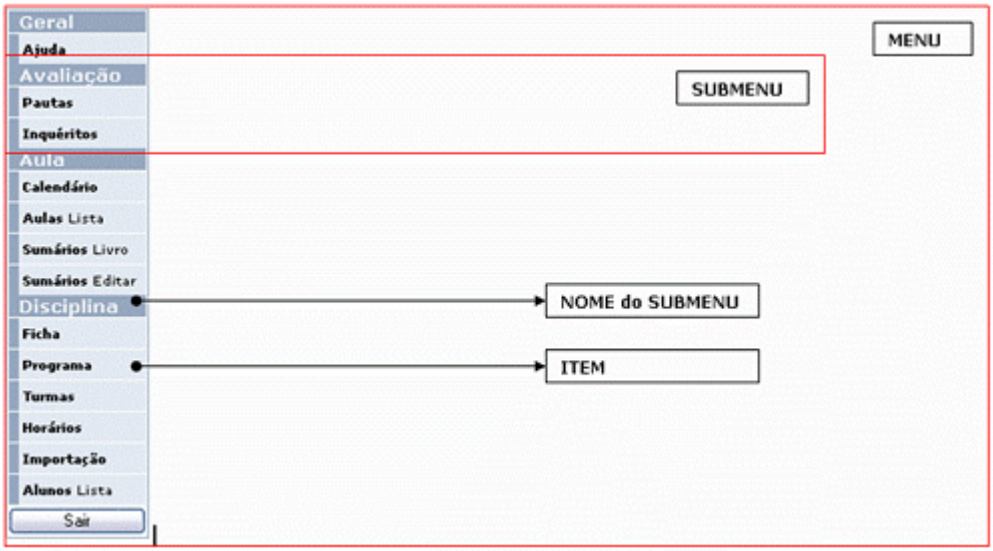

Figura IV-I – Exemplo de um menu gerado pela *framework* Hydra com indicação de algumas das suas estruturas: Menu, submenu, nome do submenu e item.

Outro item importante, que é retirado directamente da estrutura de elementos definidos para a aplicação, é a definição dos menus da aplicação para permitir o acesso às várias funcionalidades da aplicação. A configuração de um menu é executada, através de ficheiros em formato XML, contendo também *templates* HTML associados para que se possam definir vários aspectos de apresentação, sem afectar o conteúdo do menu. A configuração dos menus, a partir dos elementos da aplicação, permite, ainda a criação de uma barra de navegação *breadcrumb* que

permite identificar o utilizador em que nível hierárquico da aplicação se encontra. Na Figura IV-I está representado um menu gerado pela Hydra.

Ao nível das *templates* de apresentação dos elementos da aplicação, podem também ser definidos de forma automática tabuladores. Os tabuladores são importantes para tentar separar interfaces complexas ou com muita informação. Na Figura IV-II, é dado um exemplo de uma interface Web, com tabuladores, suportada pela *framework* Hydra.

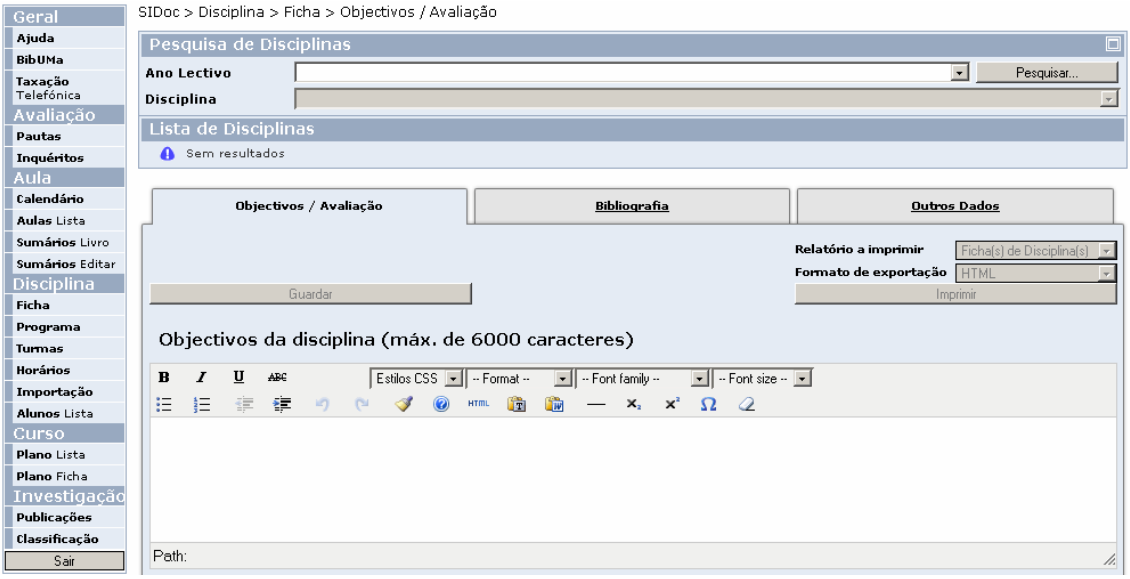

Figura IV-II. Exemplo de uma interface Web com tabuladores e outros componentes suportados pela *framework* Hydra.

Na *framework* Hydra existem ainda outros serviços de baixo nível mas que facilitam o desenvolvimento das aplicações, como a manipulação de carregamento (*upload*) e descarregamento (*download*) ficheiros e a criação de relatórios complexos de dados. A *Hydra* permite a interligação com ferramentas de relatórios como o *CrystalReports* e o *OpenReports*, podendo associar a ligação a relatórios remotos destas ferramentas, recebendo ou não parâmetros de filtragem de dados.

Estes mecanismos e padrões de comportamento estão implementados em diversos pacotes de código que constituem a *framework Hydra*. Os ficheiros dos diversos pacotes da *Hydra* são, depois utilizados e instanciados, a partir de uma única classe: *clsGestaoDados*. A classe *clsGestaoDados* é, deste modo, o ponto comum entre os mecanismos disponibilizados pela *framework* e as classes de código da aplicação que irão depois utilizar estes mecanismos. As classes de código de uma aplicação e o *workflow*, resumido, de execução da *framework* serão descritos nas próximas secções.

IV.3. ELEMENTOS DE CÓDIGO

Na secção anterior foram descritos os serviços e mecanismos disponibilizados pela *framework* Hydra, implementados em vários pacotes de código que definem a base de suporte às aplicações Web. Nesta secção descrevem-se os elementos e organização das estruturas de código a serem utilizadas nas interfaces de uma aplicação Web suportada pela Hydra.

A unidade de código básica da *framework*, como consequência da metodologia de desenvolvimento Wisdom, é o espaço de interacção. Este conceito, introduzido pelo Wisdom, é implementado ao nível do código como uma classe a partir de um ponto de extensão da *framework*. Outro elemento, importante, é o espaço de agregação que foi criado devido à necessidade de reutilização dos espaços de interacção, definidos a partir da arquitectura conceptual Wisdom. Desta forma os espaços de interacção, codificados, através de classes, são agregados noutras classes, definidas pelos espaços de agregação. Assim, um espaço de interacção pode ser reutilizado em vários espaços de agregação e em vários módulos da aplicação. Com a finalidade de organizar os artefactos e o código gerado foi definido o elemento módulo, que tem uma relação com os casos de utilização definidos na fase de análise, tendo, consequentemente, o objectivo de agrupar o código com funcionalidades ou objectivos similares.

Os espaços de agregação, teoricamente, podem ser considerados espaços de interacção, que têm uma relação de agregação com outros espaços de interacção. Quando se passou à implementação desse conceito na Hydra, foi decidido dar o nome de espaço de agregação aos espaços que têm uma relação de agregação com outros, pois na prática, ou seja, na implementação do conceito ele não iria conter elementos de interacção, sendo estes exclusivos da configuração dos espaços de interacção. Logo, a distinção dos conceitos deveu-se a uma decisão de implementação, embora teoricamente possam ser considerados semelhantes.

Estes três elementos de código, o módulo, o espaço de agregação e o espaço de interacção, constituem as principais classes que definem uma aplicação suportada pela *framework* Hydra, dando origem a que a estrutura de código que preenche os requisitos da *framework* tenha as seguintes classes de código: *phpProject*, *phpModule*, *phpAggregationSpace* e *phpInteractionSpace*. A classe *phpModule*, implementa o conceito de módulo, logo é utilizada para organizar o código de implementação, enquanto que o elemento *phpAggregationSpace* ao implementar o conceito de espaço de agregação irá dar origem ás páginas Web, compostas por elementos

phpInteractionSpace. A classe *phpInteractionSpace* implementa o espaço de interacção como unidade básica da *framework* e define a apresentação e interacção com o utilizador.

Os elementos *phpModule*, *phpAggregationSpace* e *phpInteractionSpace* são codificados como classes PHP 4.x [PHP, 2006] e cada um deles tem associado outros dois artefactos de código: (i) Um artefacto de configuração e (ii) um artefacto com uma *template* de apresentação. O artefacto de configuração é definido utilizando XML e contém as regras de comportamento e outros parâmetros de configuração como as opções para ligação à fonte de dados, regras de validação, campos de dados e a semântica das árvores CTT em UML associadas. O artefacto de *template* de apresentação é definido utilizando HTML e é responsável pela apresentação de cada classe associada (*phpModule*, *phpAggregationSpace* ou *phpInteractionSpace*).

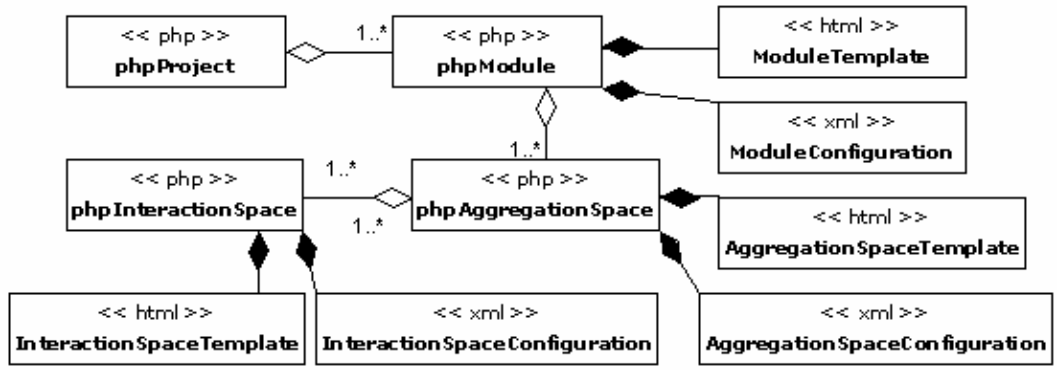

Figura IV-III – Diagrama de classe com os elementos de codificação de uma aplicação baseada na *framework* Hydra.

Na Figura IV-III são descritos os principais elementos de código para a plataforma de implementação. No esquema da Figura IV-III, por razões de simplificação, não foi utilizado nenhum perfil UML existente para a Web ou para XML, em vez disso definimos os seguintes estereótipos: <<php>> para uma classe PHP; <<html>> para uma template HTML; <<xml>> para um artefacto XML. Para mais informações acerca de perfis UML para a modelação Web e XML podem ser consultadas as referências [Carlson, 2001] e [Conallen, 2002].

Relativamente à estrutura de código de um projecto, para uma aplicação baseada na Hydra, a estrutura de implementação das interfaces com o utilizador, ou seja, da camada de apresentação é definida pelos seguintes directórios:

- "/**classes**": contém classes PHP de utilização comum ao projecto.
    - "/**conf**": contém ficheiros PHP e XML com as configurações comuns ao projecto.
- "/**css**": contém os ficheiros de *Cascading Styles Sheets* (CSS) comuns ao projecto.
- "/**imagens**": contém as imagens que são comuns ao projecto.
- "/**javascript**": contém código *javascript* a ser utilizado pelas *templates* HTML .
- "/**modulos**": contém os sub-directórios dos módulos da apresentação.

- "/**templates**": contém as *templates* genéricas HTML, comuns ao projecto.

Para cada um dos módulos, ou seja, para cada elemento módulo definido na directoria "/modulos" da estrutura principal, da camada de apresentação de uma aplicação, existe a seguinte estrutura de directórios:

- "/**classes**": directório onde estão as classes dos espaços de interacção do módulo.
- "/**conf**": directório com a configuração das interfaces e comportamento das classes de espaços de interacção.
- "/**css**": ficheiros *Cascading Styles Sheets* (CSS) utilizados pelas *templates* HTML do módulo.
- "/**html**": ficheiros auxiliares com HTML estático.
- "/**imagens**": imagens relativas apenas ao módulo actual.
- "/**templates**": Ficheiros com *templates* HTML com variáveis dinâmicas. Estas *templates* serão processadas pelas classes, recorrendo à configuração da directoria "/conf".

Cada directoria do módulo contém um ficheiro "**index.php**", que serve para aceder, verificar e processar o espaço de interacção pretendido, sendo o único ponto de entrada e saída de cada módulo da aplicação Web. Toda a configuração associada aos espaços de interacção, módulos, etc., que permite definir a estrutura e comportamento do projecto, está implementada em formato XML. Desta forma pretende-se normalizar e facilitar o processo de configuração dos espaços de interacção e de agregação, criados a partir dos modelos de desenho.

Conforme já foi descrito anteriormente, cada espaço de interacção tem associado uma *template* HTML que define a apresentação e um ficheiro XML onde é configurado o seu comportamento. O comportamento está dependente dos padrões simples de interacção definidos na *framework* e também da árvore de tarefas associada, definida com a notação CTT em UML. Um espaço de interacção é implementado através de uma classe PHP, com o comportamento configurado em XML e a parte de apresentação caracterizada por uma *template* HTML, ficando inserido dentro da directoria de módulo, com a seguinte estrutura de código:

- Ficheiro de **classe PHP**, que fica na directoria "/classes".
- Ficheiro de **configuração XML**, que fica na directoria "/conf", com a estrutura de comportamento.
- Ficheiro com ***template* HTML**, que fica dentro da directoria "/templates", com a estrutura da apresentação.

Esta estrutura de distribuição de código, com a separação dos conceitos de apresentação e de comportamento, aplica-se também às classes dos elementos de espaços de agregação e dos elementos de módulo. Na Figura IV-IV está representada a estrutura de directórios de uma aplicação Web suportada pela Hydra

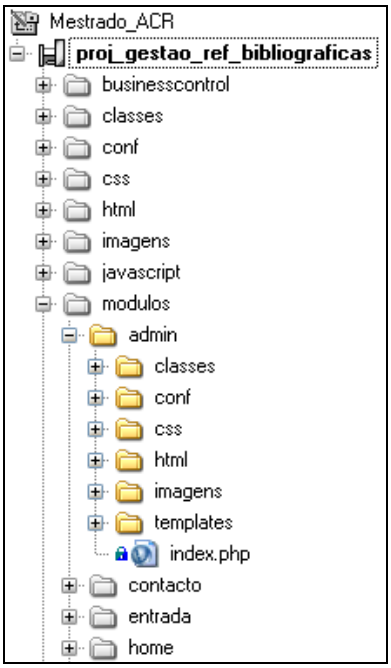

Figura IV-IV – Exemplo da estrutura de código da camada de aplicação de uma aplicação Web suportada pela Hydra. Adaptado de [Costa, Teixeira, et al., 2004].

Com a estrutura dos artefactos de código a reflectir os elementos da metodologia e dos modelos de desenho, obtém-se uma rastreabilidade entre o código produzido e o caso de utilização que conduziu o desenho, permitindo que a evolução do sistema possa ser feita, de forma controlada, documentada e consistente.

IV.4. *WORKFLOW* DE EXECUÇÃO

Na secção anterior foram descritos os elementos e estruturas de código a serem utilizadas numa aplicação Web suportada pela *Hydra*. Nesta secção descreve-se, de forma resumida, o processo de execução de uma aplicação baseada na *framework* em questão.

Na estrutura de código de uma aplicação Web baseada na *Hydra*, as classes de espaço de interacção são instanciadas a partir da classe "*clsGestaoDados*", ficando assim disponíveis os serviços e mecanismos da *framework,* a partir desta relação de especialização. Por outro lado, as classes dos elementos de módulo e de espaços de agregação utilizam os serviços e mecanismos da *framework* por instanciação directa das classes.

O ponto de entrada de execução de um evento, despoletado pelo utilizador ou não, para uma aplicação faz-se através do ficheiro "**index.php**" do módulo, invocado através de um *Universal Resource Locator* (URL) Web. A partir deste ponto de entrada é depois a classe de módulo instanciada que escolhe qual o espaço de agregação que deve ser, por sua vez, instanciado para que o pedido seja satisfeito.

A classe de espaço de agregação, por outro lado, tem como missão o processamento de todas as classes de espaço de interacção que lhe estão agregadas. A escolha de qual o espaço de interacção, de qual o elemento de interacção e qual o evento a executar é efectuada depois dentro da respectiva classe do espaço de interacção. Desta forma o processamento da interacção com o utilizador faz-se nas classes de espaço de interacção instanciadas, que constituem um espaço de agregação. Os espaços de agregação instanciados dão origem, na prática, às páginas Web da aplicação. A diferença entre um espaço de agregação e um espaço de interacção foi definida na secção IV.3.

Quando ocorre um evento, despoletado pelo utilizador, é sempre identificado em que módulo, espaço de agregação e espaço de interacção deve ser processado esse pedido. As outras classes de espaços de interacção, cujos elementos não interagiram com o utilizador, ou seja, que não estão a executar o evento despoletado, encontram-se à escuta e verificam qual o identificador do espaço de interacção que está a ser executado, decidindo se o evento em execução irá implicar, ou não, alguma acção da sua parte. Este mecanismo está relacionado com a associação de <<*subscribe*>> definida no método Wisdom entre elementos de espaço de interacção.

As classes de espaços de interacção não executadas têm configurado por defeito um evento que é despoletado sempre que não seja a classe a responder directamente a um evento. Normalmente este evento é definido como um carregamento de dados. A classe de espaço de interacção que deve responder ao evento despoletado pelo utilizador executa as acções de acordo com esse evento e que normalmente seguem esta ordem:

- Carregar dados (se aplicável);
- Aplicar regras de validação de dados;
- Aplicar regras específicas de bloqueio de campos;
- Aplicar semântica da árvore CTT associada;
- Aplicar evento de alteração de dados (se aplicável);
- Gerar *template* associada;
- Devolver *template* final com dados;

É na classe de espaço de interacção em execução que são aplicadas as regras de comportamento e de apresentação associadas aos eventos. É nesta classe que também são invocados, se aplicáveis, os serviços de suporte como, por exemplo, a ligação a dados ou ao negócio. No final do processamento, que poderá variar da lista anterior de acordo com o evento em execução, é devolvida a *template* final do espaço de interacção a ser apresentada ao utilizador. Relativamente à aplicação da semântica da árvore de tarefas, definida com a notação CTT, o algoritmo é descrito em detalhe na secção V.3 deste trabalho e foi implementado na *Hydra,* no âmbito deste trabalho, resultando numa estrutura de suporte para executar o comportamento da interface com a utilização da semântica das árvores CTT em UML.

O resultado da execução do evento, em formato HTML, é depois devolvido à classe do espaço de agregação instanciado, que tem como objectivo, principal, agrupar todos os resultados de *templates* finais das classes de espaços de interacção associadas e aplicar este conjunto à *template* com a estrutura de apresentação definida para o respectivo espaço de agregação. Na *template* do espaço de agregação fica definida a estrutura de apresentação para os diversos espaços de interacção que o compõem. No final deste processamento o espaço de agregação devolve, para a classe de módulo, a *template* final com os campos e dados requisitados.

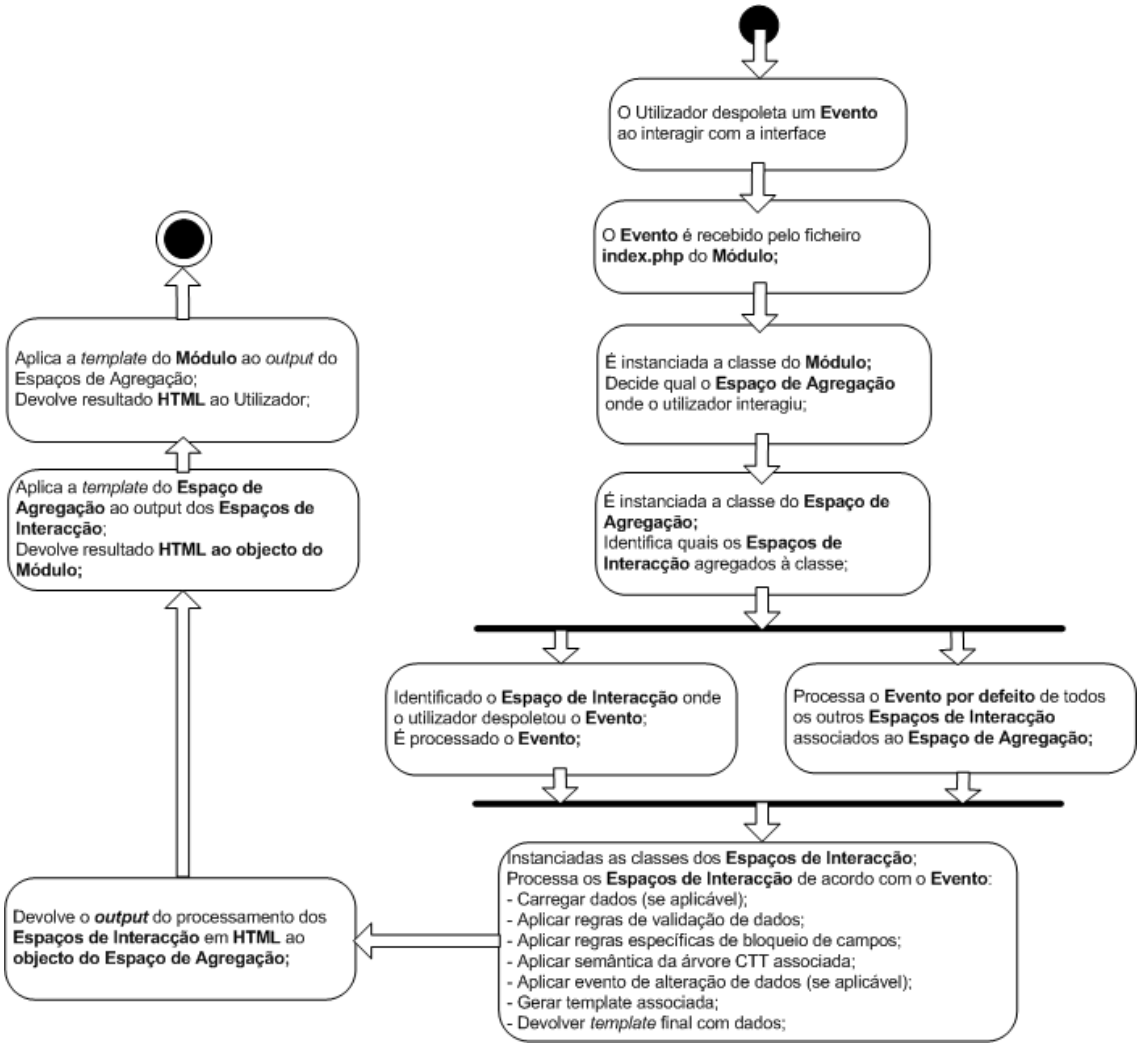

Figura IV-V. Esquema que resume o processo de execução interno de uma aplicação Web suportada pela Hydra.

A classe de módulo, por sua vez aplica a *template* comum a toda a aplicação, gera o menu da aplicação, gera o cabeçalho e rodapé e apresenta a página final ao utilizador. De uma forma resumida são estes passos que são aplicados ao processamento de um evento para uma aplicação baseada na Hydra, mas que poderão ser mais simples ou mais complexos, de acordo com o evento em questão. Na Figura IV-V está representada um esquema com os passos de execução de evento na Hydra. Para um evento em que seja apenas de carregamento de dados e apresentação numa página Web sem campos de introdução de texto, ou seja sem um formulário HTML, a sequência de acções é mais simples. Por outro lado se for um evento para introdução, alteração ou eliminação de dados as funções a aplicar são um pouco mais complexas. Mas sejam mais simples ou mais complexas, as funções a aplicar centram-se sempre na classe de espaço de interacção, devolvendo o resultado às outras classes de espaços de agregação e de módulo que têm por missão quase apenas a organização e apresentação de resultados HTML das *templates* associadas.

V. MODELOS E METAMODELOS PROPOSTOS

Foram descritos, anteriormente, o processo e o método Wisdom de desenvolvimento de software, no qual este trabalho assenta, seguidamente descreveu-se de forma resumida os objectivos, conceitos e princípios da iniciativa MDA [Soley and OMG, 2000] [OMG, 2003], indicando as vantagens de alinhar o método Wisdom nas recomendações desta especificação. Neste capítulo, iremos detalhar a proposta, que é o cerne deste trabalho, e que permite alinhar o método Wisdom com a abordagem MDA. As três principais perspectivas de modelos definidas no método Wisdom, arquitectura conceptual, modelo de apresentação e modelo de diálogo, serão enquadradas na definição de PIM (*Platform-independent Model*). Será ainda proposta uma notação e elementos para um modelo PSM (*Platform-specific Model*), fundamentado por um metamodelo, que, posteriormente, levará à geração automática de código a partir dos modelos Wisdom. Finalmente, outro aspecto importante que será focado é o algoritmo, que foi implementado na *framework* Hydra, de modo a permitir a execução da semântica da notação *ConcurTaskTrees* nos componentes da plataforma de implementação.

A proposta deste documento promove a utilização de tecnologias recomendadas pela OMG para a modelação e implementação de interfaces com o utilizador, sendo este um assunto que vem sendo discutido, há algum tempo, na comunidade de desenvolvimento de interfaces com o utilizador baseado em modelos.

V.1. OS MODELOS INDEPENDENTES DA PLATAFORMA NO DESENHO DE INTERFACES COM O UTILIZADOR

O método Wisdom concentra-se no desenvolvimento das interfaces de sistema com o utilizador, os aspectos de negócio e de domínio não são detalhados nem são propostas técnicas ou notação para os descrever em detalhe. Consequentemente, este trabalho foca-se nos modelos Wisdom das actividades de análise e de desenho para o detalhe da estrutura e comportamento da interacção com o utilizador, logo, o enquadramento do conceito PIM (*Platform-Independent Model*), descrito na secção III.3, será aplicado, apenas, a estes modelos.

Um PIM descreve a funcionalidade do sistema, sem considerar os detalhes de implementação, definindo um elevado nível de independência entre a descrição da funcionalidade e os detalhes de implementação para uma plataforma alvo. Numa abordagem de desenho de interfaces com o utilizador (DIU), o PIM deve reflectir a estrutura e o comportamento da interface com o utilizador, com um alto grau de abstracção e não conter nenhuma referência aos seus componentes concretos de implementação.

Desta forma o PIM segue a separação de conceitos, sugerida para o DIU, entre a interacção e a estrutura de apresentação numa interface. A abordagem descrita em [Nóbrega, Nunes, et al., 2005b], onde foram adicionados notações e vistas ao método Wisdom [Nunes, 2001], define três perspectivas de modelos para a descrição da interface com o utilizador e o respectivo comportamento da interacção. Estas três perspectivas são i) a arquitectura conceptual, ii) o modelo de apresentação, iii) e o modelo de diálogo.

A arquitectura conceptual, conforme foi já descrito na secção II.2, define uma vista de alto nível da arquitectura do sistema, em termos de classes conceptuais, estendidas por estereótipos. Os estereótipos de classe definidos para esta perspectiva foram descritos na secção II.2.2, e são: <<entity>>, <<control>>, <<task>> e <<interaction space>>. Resumidamente, os elementos que descrevem a estrutura e o comportamento da interface com o utilizador são as classes UML estereotipadas <<interaction space>> e <<task>>. As classes <<task>> são utilizadas para modelar o diálogo entre o sistema e os actores humanos. Cada elemento <<interaction space>> da arquitectura conceptual é detalhado no modelo de apresentação, utilizando a notação de Protótipos Abstractos Canónicos (PAC) adaptada ao UML. Por outro lado, cada elemento <<task>> é detalhado pelo modelo de diálogo, utilizando a notação *ConcurTaskTrees* (CTT) em UML.

Em termos de extensões ao metamodelo do UML os elementos PAC necessários ao modelo de apresentação são: i) *InteractionElement* ; ii) *Material* ; iii) *Tool* ; iv) *Hybrid*. De igual forma as seguintes extensões ao metamodelo do modelo de diálogo são necessárias pela notação CTT a ser utilizada: i) *TaskTree* ; ii) *TaskNode* ; iii) *TaskEdge*. Os elementos *Material*, *Tool*, *Hybrid* e *TaskEdge* especializam outros elementos descritos mais à frente, na secção V.2 e esquematizados na Figura V-I e na Figura V-II.

Defendemos que a descrição anterior das três perspectivas de modelos na abordagem Wisdom proposta em [Nóbrega, Nunes, et al., 2005b], já descrita e detalhada na secção II.4 e II.5, devido à sua independência tecnológica, encaixa no conceito PIM definido pela iniciativa *Model Driven Architecture* (MDA).

Ao ser definido o PIM, através dos modelos das actividades de análise e de desenho do método Wisdom, na próxima secção será proposto um modelo específico para a plataforma Web, baseada na *framework* Hydra, de modo a permitir a geração automática de código da aplicação modelada.

V.2. O *PLATFORM-SPECIFIC MODEL* (PSM) PROPOSTO

Nesta secção, iremos descrever os elementos PIM que devem ser passados para o modelo específico. Propomos um modelo específico da plataforma (PSM) intermédio porque, conforme definido na recomendação MDA, o PIM deve ser uma representação da solução independente da tecnologia e a solução tecnológica deve ser reflectida no PSM. O PSM proposto contém as mesmas perspectivas Wisdom definidas para o modelo PIM: arquitectura conceptual, modelo de apresentação e modelo de diálogo.

V.2.1.Os elementos PIM reflectidos no PSM

Para a arquitectura conceptual do PIM, os elementos que irão ser transformados e representados no PSM, são os estereótipos de classe <<interaction space>> e <<task>>. Para a parte estrutural, definida na vista de modelo de apresentação do PIM, devem ser reflectidos no PSM elementos que representam a semântica e notação dos Protótipos Abstractos Canónicos (PAC): o elemento de interacção (*interaction element*) e as correspondentes especializações *Material*, *Tool* e *Hybrid*. Um diagrama com o metamodelo dos elementos PAC definidos como extensões da UML, é representado na Figura V-I.

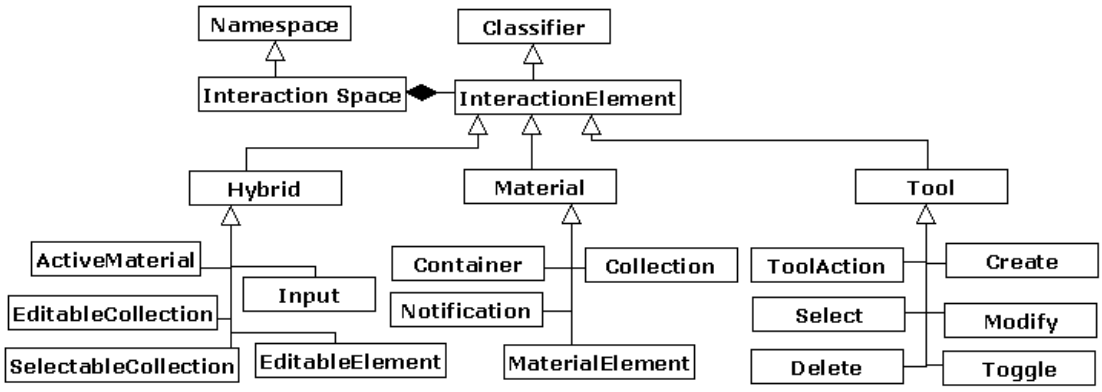

Figura V-I – Metamodelo UML 2.0 [OMG, 2005a] [OMG, 2005b] estendido com os elementos PAC. Adaptado de [Nóbrega, Nunes, et al., 2005b].

No diagrama, da Figura V-I, pode-se verificar que o elemento *Material* contém as especializações *Container*, *Element*, *Collection* e *Notification*. Por outro lado, o elemento *Tool* tem as seguintes especializações: *ToolAction*, *Create*, *Delete*, *Modify*, *Select* e *Toggle*. O elemento *Hybrid* inclui as especializações: *ActiveMaterial*, *EditableCollection*, *EditableElement*, *Input* e *SelectableCollection*. A notação original PAC, definida em [Constantine and Lockwood, 1999]

contém 21 elementos. Nesta parte do trabalho, iremos apenas descrever os mais significativos, de modo a construir modelos num contexto de desenvolvimento para a Web. O conjunto total de extensões para o metamodelo da UML foi definido em [Nóbrega, Nunes, et al., 2005b].

Depois de definidos os elementos da parte estrutural a representar no PSM, são descritos agora os elementos a incluir relativamente à parte do comportamento da interacção. Os componentes de comportamento associados à interface com o utilizador de destino, representados na vista do modelo de diálogo do PIM, envolvem os elementos da notação *ConcurTaskTrees* (CTT) em UML que serão reflectidos no PSM. Os elementos do PIM estendidos a partir dos metaelementos *Action*, *Activity* e *ActivityEdge* da UML 2.0, são, respectivamente, *Task*, *TaskTree* e *TaskEdge*. Na Figura V-II está representado um diagrama de metamodelo com os elementos CTT que estendem o UML 2.0. No diagrama pode-se verificar que o elemento *TaskEdge* tem os seguintes elementos especializados, relacionados com os operadores temporais da notação CTT: *ChoiceEdge*, *IndependentConcurrencyEdge*, *DeactivationEdge*, *EnablingEdge*, *OrderIndependenceEdge*, e *SuspendEdge*. A notação e semântica originais CTT foram definidas em [Paternò, 1999] [Mori, Paternò, et al., 2004] enquanto que o metamodelo UML foi estendido com estes elementos em [Nóbrega, Nunes, et al., 2005].

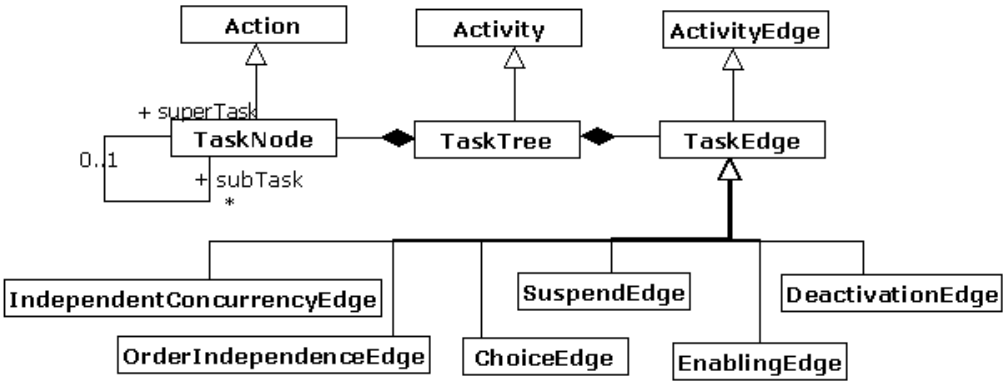

Figura V-II – Metamodelo estendido UML 2.0 [OMG, 2005a] [OMG, 2005b] como os elementos CTT. Adaptado de [Nóbrega, Nunes, et al., 2005b].

Para descrever a interligação entre os elementos PAC e os elementos CTT foi definida em [Nóbrega, Nunes, et al., 2005b] uma relação de associação entre o elemento *TaskNode* e um elemento *InteractionElement*. Esta associação permite interligar um elemento de interacção PAC a um nó de tarefa da árvore CTT, relacionando a semântica de comportamento do espaço de interacção com os elementos estruturais, que são apresentados ao utilizador. Na Figura V-III está descrita esta relação de interligação entre os elementos estruturais e os elementos comportamentais da modelação da interface.

Figura V-III – Representação da parte do metamodelo Wisdom que define a relação de associação entre os elementos PAC e os elementos CTT.

Os elementos descritos nesta secção das três perspectivas do PIM são reflectidos directamente em elementos específicos do PSM proposto. Estes elementos específicos serão detalhados mais à frente neste trabalho na secção V.2.3.

V.2.2.O Perfil UML: Estereótipos e Etiquetas

A UML permite a definição de modelos para domínios específicos através de um conjunto de mecanismos de extensão (estereótipos, etiquetas - *tagged values* e restrições - *constraints*), especializando os seus elementos para um domínio de aplicação particular. Esses conjuntos de extensões UML são agrupados em perfis UML (*UML Profiles*). Os perfis UML têm um papel importante na recomendação MDA e nas técnicas para geração de código [Fuentes-Fernández and Vallecillo-Moreno, 2004]. Na especificação MDA, os perfis UML são, particularmente, importantes para descrever os modelos específicos da plataforma, garantindo que os modelos derivados irão ser consistentes com a norma UML, recomendada pela MDA. Consequentemente, um perfil UML é uma técnica recomendada para a definição do PSM. Além dos estereótipos, que especializam e estendem os conceitos do PIM, os mecanismos de extensão UML incluem também etiquetas, que permitem adicionar características a um elemento, que não poderiam ser adicionadas através de outros conceitos UML. Sendo os estereótipos e as etiquetas recomendados para a definição do modelo específico, iremos descrever estes elementos de extensão num perfil UML, associado à proposta do PSM.

V.2.3.Os Elementos do PSM

Depois de descrever os elementos PIM, é necessário apresentar uma proposta para o PSM de modo a reflectir estes elementos numa plataforma de destino específica. Nesta secção iremos descrever um modelo para uma plataforma Web, que, além de especializar elementos do PIM, irá conter, também, outras propriedades e elementos relacionados com detalhes de implementação.

O perfil UML, que será utilizado no PSM, define estereótipos UML divididos em: i) elementos estruturais, relacionados com a arquitectura conceptual e os espaços de interacção do modelo de apresentação, onde se utiliza a notação PAC, e ii) elementos de comportamento para a

adaptação UML da notação CTT. O PSM irá reflectir as perspectivas de modelo, definidas no método Wisdom para o PIM, relacionadas com o desenho de interfaces com o utilizador.

Os estereótipos propostos para o PSM, tendo em conta a perspectiva da arquitectura conceptual são:

- **<<WebModule>>**; **<<WebAggregationSpace>>**: Mapeado dos casos de utilização do PIM.
- **<<WebInteractionSpace>>**: Mapeado do elemento espaço de interacção do PIM. Representa o espaço na interface com o utilizador onde o utilizador irá interagir com o sistema. Este elemento define também a ligação entre a arquitectura conceptual e o modelo de apresentação.
- **<<WebTask>>**: Mapeado do elemento tarefa do PIM. Define a estrutura do diálogo entre o utilizador e o sistema, estabelecendo o comportamento da interface com o utilizador. Este elemento estabelece também a ligação entre a arquitectura conceptual e o modelo de diálogo.

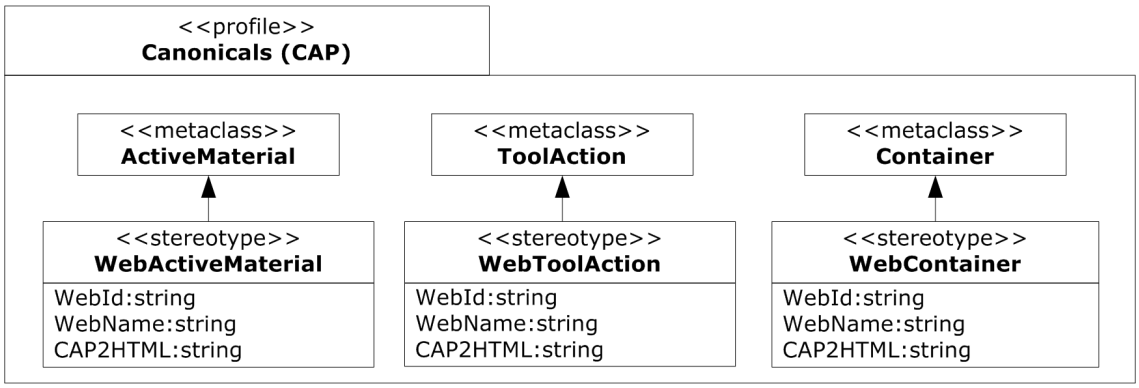

Figura V-IV – Perfil UML com os elementos estereótipos para o modelo de apresentação. Para a simplificação do diagrama foi só representada uma metaclasse para cada sub-conjunto de *Material*, *Tool* e *Hybrid*. Adaptado de [Costa, Nóbrega, et al., 2007].

Os estereótipos, propostos para a vista do modelo de apresentação no PSM, que irão definir uma especialização da notação PAC, são:

- **<<WebContainer>>**; **<<WebMaterialElement>>**; **<<WebCollection>>**; **<<WebNotification>>**: Mapeados do sub-conjunto que especializa os elementos do elemento *Material,* definido no PIM. Um componente *Material* representa conteúdo, informação, dados ou outros objectos da interface com o utilizador manipulados ou apresentados ao utilizador durante o decurso de uma tarefa.
- **<<WebToolAction>>**; **<<WebCreate>>**; **<<WebDelete>>**; **<<WebModify>>**; **<<WebSelect>>**; **<<WebToggle>>**: Mapeados do sub-conjunto de especialização do

elemento *Tool*. Um componente *Tool* representa operadores, mecanismos ou controlos que podem ser utilizador para manipular ou transformar materiais (elemento *Material*).

- **<<WebActiveMaterial>>**; **<<WebEditableCollection>>**; **<<WebEditableElement>>**; **<<WebInput>>**; **<<WebSelectableCollection>>**: Mapeados do sub-conjunto de elementos especializados do elemento *Hybrid* do PIM. Um *Hybrid* ou *Active Material* é o resultado da combinação das classes de componentes *Material* e *Tool* e representa qualquer componente com características de ambos os conjuntos de elementos, por exemplo uma caixa de entrada de texto.

.A Figura V-IV contém um diagrama de pacotes para o perfil UML, que inclui os estereótipos e etiquetas da notação PAC para o PSM. Para os estereótipos, descritos anteriormente, devem ser definidas etiquetas, de modo a descrever a posição relativa dos elementos na interface final com o utilizador. Outras etiquetas sugerem o elemento concreto para a interface com o utilizador, neste caso, em particular, para o âmbito deste trabalho, será um elemento HTML. As etiquetas, mais significativas definidas para os estereótipos, do PSM, relacionados com o modelo de apresentação são:

- "**CAP2HTML**": Define um valor para identificar o componente real de interface associado com o estereótipo que será implementado no protótipo de interface a gerar. Este valor apesar de ser, inicialmente, sugerido de forma automática, pode ser modificado mais tarde.

- "**WebId**": Identificação do componente de interacção relacionada com a plataforma de implementação e a *framework* Hydra.

- "**WebName**": Descrição do componente de interacção relacionada com a plataforma de implementação e a *framework* Hydra.

Em [Campos and Nunes, 2004] [Campos and Nunes, 2006] foram propostas correspondências, específicas, entre os conceitos PAC e elementos concretos de interface com o utilizador para a plataforma Web. Desta forma, conseguimos uma transformação inicial do PIM para o PSM, onde existirá uma correspondência com um elemento concreto Web a implementar em HTML, definido através da etiqueta "CAP2HTML". Uma tabela, com as correspondências entre elementos PAC e componentes HTML concretos, está representada na Figura V-V.

CAP Elements	HTML Element
Materials	
Container	Div, Table, Form
Element	TextPlain
Collection	PlainText, Table
Notification	PlainText, Popup Window
Hybrids	
Active Material	Button, Link, FileBrowser
Editable Element	InputText, TextArea
Selectable Collection	ComboBox, RadioButton, FileBrowser

CAP Elements	HTML Element
Tools	
Action	Button, Link
Select	Link, CheckBox, FileBrowser, ComboBox, RadioButton
Create	Button
Delete	Button
Modify	Button
Toggle	CheckBox, Button

Figura V-V – Correspondências entre elementos PAC e componentes HTML concretos. Adaptado de [Costa, Nóbrega, et al., 2007].

Para os elementos do PSM, relacionados com o comportamento da interface com o utilizador, ou seja, que são representados no modelo de diálogo, propomos os seguintes estereótipos que são extensões aos elementos da notação CTT em UML:

- **<<WebTaskTree>>**: Mapeado do elemento *TaskTree* definido no metamodelo do PIM. Este elemento define a raiz da árvore CTT em UML, a qual irá detalhar o comportamento da interacção. Este elemento será ligado com o elemento *tarefa,* definido na vista arquitectura conceptual do sistema.

- **<<WebTaskNode>>**: Mapeado do elemento *TaskNode* do metamodelo do PIM. Este elemento define uma tarefa ou acção que irá ser desempenhada na interface. Este estereótipo incluirá etiquetas UML para definir as condições de início e de fim da execução de uma tarefa. As condições de início e de fim estão relacionadas com a semântica da notação CTT e são apenas aplicadas para tarefas básicas (nós folha da árvore) isto porque é este tipo de tarefas que reflectem uma interacção directa com o utilizador na interface. Isto também quer dizer que apenas as tarefas básicas (folhas) irão ser ligadas a um elemento PAC no modelo de apresentação (ver Figura V-III).

- **<<WebIndependentConcurrencyEdge>>**;**<<WebChoiceEdge >>**;**<<WebEnablingEdge>>** ;**<<WebOrderIndependenceEdge>>**;**<<WebDeactivationEdge>>**;**<<WebSuspendEdge>>**: Mapeados dos diversos elementos de operadores temporais do PIM, estes elementos representam os estereótipos para cada tipo de operador temporal, descrito na notação CTT em UML.

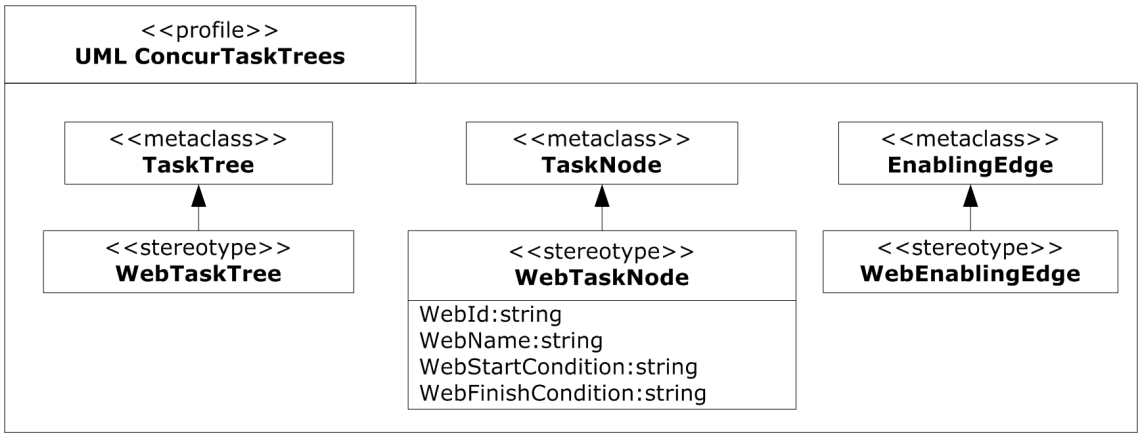

Figura V-VI – Perfil UML com os elementos estereótipos para o modelo de diálogo. Para a simplificação do diagrama foi só representada uma metaclasse para o sub-conjunto de *TaskEdge*. Adaptado de [Costa, Nóbrega, et al., 2007].

Para os estereótipos da notação CTT em UML, são também definidas etiquetas de forma a permitir a geração automática de código. As etiquetas, mais significativas relacionadas com os estereótipos de comportamento, são:

- "**WebStartCondition**": Define o valor para a condição de execução que verifica se o estado *isStarted* de um nó de tarefa deve ser colocado a *true* ou *false*.
- "**WebFinishCondition**": Define o valor para a condição de execução que verifica se o estado *isFinished* de um nó de tarefa deve ser colocado a *true* ou *false.*
- "**WebId**": Identificação relacionada com a plataforma de implementação e a *framework* Hydra.
- "**WebName**": Descrição relacionada com a plataforma de implementação e a *framework* Hydra.

Devido ao contexto Web, para onde foi direccionada a abordagem, as condições de início e de fim, de um nó de tarefa CTT, foram simplificadas e ocorrem, em simultâneo, durante uma interacção, ficando os atributos de início e de fim, respectivamente "WebStartCondition" e "WebFinishCondition", com o mesma identificação da condição a ser verificada. Uma condição já definida, tem o identificador "WITH_VALUE" e significa que, só quando o servidor Web receber um valor não nulo, definirá a tarefa como terminada, processando em seguida o seu estado para activo ou desactivo de acordo com os operadores temporais associados ao nó de tarefa. Na Figura V-VI está representado um diagrama de pacotes de um perfil UML como os estereótipos e etiquetas (*tagged value*) para a notação CTT em UML.

A representação gráfica dos elementos no PSM é a mesma que está definida em cada uma das notações PAC e CTT em UML. A principal diferença gráfica é que o texto "<<*stereotype*>>" é colocado em cada um dos elementos do PSM.

V.2.4.Transformações PIM para PSM

O processo de transformação entre o PIM e o PSM é executado, automaticamente, e resultará na criação dos elementos do PSM com estereótipos e etiquetas. Existe uma relação directa entre os elementos do PIM e do PSM, conforme sugerido pelo perfil UML proposto, anteriormente, por exemplo, um elemento *Container* é mapeado para o estereotipo <<*WebContainer*>>. No processo de transformação é colocada informação, adicional, através das etiquetas, como a identificação do elemento HTML sugerido para o PAC específico e as condições de início e de fim para os elementos *TaskNode*.

Para simplificação do trabalho, as regras de transformação são definidas de uma maneira directa e simples através de XML. A norma *Query View Transformation* (QVT) [Tata Consultancy, 2003] [QVT-Merge Group, 2005] está ainda numa fase inicial da especificação mas poderá ser uma abordagem, interessante, para um futuro método formal e normalizado de definição das regras de correspondência entre PIM e PSM.

O PIM e o PSM são serializados num formato XMI [OMG 2005c] [Grose, Doney, et al. 2002], seguindo a recomendação MDA, de modo a promover uma melhor integração e transformação entre modelos do PSM para a plataforma de implementação. As regras de transformação e o software desenvolvido, para executar as conversões, são discutidos em pormenor na secção VI.3.

V.3. O ALGORITMO PARA EXECUTAR UMA ÁRVORE DE TAREFAS (CTT)

Uma importante consequência do método de desenho de interfaces com o utilizador, definido em [Nóbrega, Nunes, et al., 2005b] e que estabelece o Wisdom com as notações PAC e CTT em UML, é a possibilidade de simular os modelos de diálogo, com a semântica CTT, utilizando um algoritmo especifico de execução. O algoritmo para a árvore de tarefas, permite activar e desactivar nós de tarefas definidos na estrutura em árvore e relacionados através de operadores temporais, de acordo com a especificação original CTT.

O objectivo do algoritmo é a identificação das tarefas da árvore necessárias para definir o estado activo ou desactivo, tendo em consideração o operador temporal associado e as tarefas executadas pelo utilizador. Os nós de tarefa, que podem ser executados, são os nós-folha da árvore e representam uma interacção directa com o utilizador.

Nesta secção, descrevemos alguns dos aspectos do algoritmo, definido em [Nóbrega, Nunes, et al., 2005b] comparando-o com o algoritmo definido para a notação CTT original, descrita em [Paternò, 1999] [Mori, Paternò, et al., 2004]. Este algoritmo, para processar árvores de tarefas em UML, é fundamental para a geração e execução dos protótipos de interface na plataforma de implementação. Combinando as notações de apresentação e de tarefas em UML, podemos simular protótipos abstractos de uma forma que promove uma avaliação preliminar da usabilidade das interfaces e também a geração automática de código.

V.3.1.A Execution Task Tree (ETT)

O primeiro passo no algoritmo, para as árvores CTT em UML, é a criação de uma árvore binária que pode ser percorrida por um método recursivo. Denominamos esta árvore, intermediária, de árvore de execução de tarefas (*Execution Task Tree* - ETT) e construímo-la, a partir da estrutura da árvore CTT em UML. O segundo passo verifica quais os nós de tarefa que devem ficar no estado activo ou desactivo. Um conceito fundamental, aplicado no algoritmo de execução, é a capacidade de recursivamente percorrer a árvore. A precedência dos operadores temporais, entre tarefas, é outro aspecto importante, conforme se encontra definido na especificação original da notação CTT em [Paternò, 1999] [Mori, Paternò, et al., 2004].

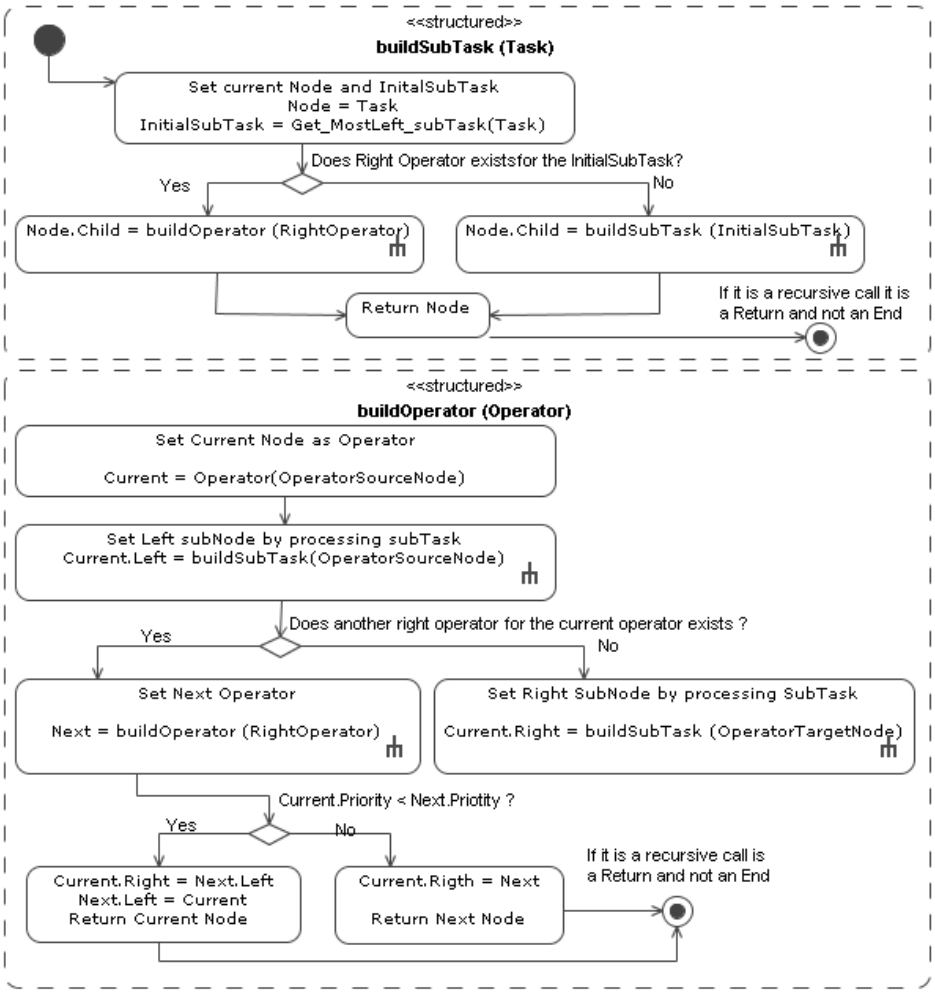

Figura V-VII – Diagrama de Actividades que representa a parte mais importante do algoritmo [Nóbrega, Nunes, et al., 2005b] que cria a árvore de execução de tarefas (ETT). Adaptado de [Costa, Nóbrega, et al., 2007].

O método, para obter a árvore binária, começa por percorrer a árvore CTT em UML, desde o nó raiz e pode ser dividido em duas grandes operações: O método *buildSubTask* e o método *buildOperator*. Primeiro é aplicado o método *buildSubTask* que define a tarefa actual como um nó, na nova árvore binária, e procura a subtarefa mais à esquerda, denominada *InitialSubtask*. Depois, se a *InitialSubTask* tem um operador à direita, é chamado o método *buildOperator* e é retornado no fim um nó de tarefa para a nova árvore binária. Caso contrário, é chamado o método *buildSubTask,* recursivamente, atravessando a árvore CTT, até encontrar uma condição para poder chamar o *buildOperator*, conforme descrito anteriormente.

O método *buildOperator* recebe um operador da árvore CTT, que é definido como o nó actual na árvore binária. Depois, o nó filho esquerdo do operador actual é definido ao ser invocado o método *buildSubTask*, tendo por parâmetro a tarefa de origem do operador actual da árvore CTT. O próximo passo, para o método *buildOperator,* é verificar se o operador actual tem um outro operador à direita. Se o operador à direita existir, o método define de forma recursiva o

nó filho à direita para o operador actual, tendo em consideração a prioridade do operador actual em relação ao próximo.

As regras de prioridade, definidas na notação CTT original [Paternò, 1999] [Mori, Paternò, et al., 2004], decidem que operador deve ser o nó pai e quais devem ser os nós filhos. Se um operador, à direita do operador actual não existir, o nó filho, à direita, é definido invocando o método *buildSubTask*, tendo por parâmetro a tarefa destino do operador na árvore CTT. O método *buildOperator* retorna um nó com o operador para a árvore binária, com os correspondentes nós filho definidos. No final do processamento, os métodos *buildSubTask* e *buildOperator* retornam os nós e respectivos filhos que, em conjunto, definem a árvore binária.

A árvore resultante, é denominada de árvore de execução de tarefas (*Execution Task Tree* - ETT) [Nóbrega, Nunes, et al., 2005b], que é a consequência da aplicação do método descrito, em que, todos os nós, representam uma tarefa ou um operador. Um diagrama de actividade, contendo a parte do algoritmo que constrói a ETT, é representado Figura V-VII. Uma comparação entre a árvore CTT em UML e a correspondente árvore *Execution Task Tree* está representada na Figura V-VIII.

V.3.2.Determinar Estado dos Nós: Activo ou Desactivo.

A secção do algoritmo, que verifica que nós de tarefa foram executados, ou seja, se foram iniciados e terminados, é semelhante ao algoritmo definido para a especificação original da notação CTT. A árvore binária é criada e percorrida de modo a identificar qual o nó de tarefa que foi executado pelo utilizador. Depois, as regras de precedência são aplicadas a partir do nó executado, de modo a definir o estado, activo ou desactivo, para cada nó de tarefa na árvore binária.

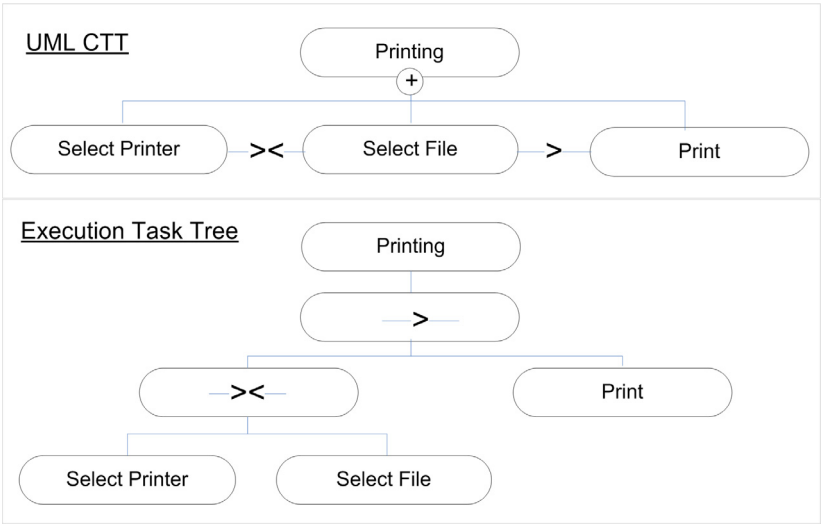

Figura V-VIII – Comparação entre a estrutura de uma árvore CTT em UML e a sua correspondente representação numa árvore *Execution Task Tree*. Adaptado de [Nóbrega, Nunes, et al., 2005b].

A secção do algoritmo, que define o estado dos nós de tarefa para a árvore binária ETT começa com um método que define o seu estado inicial, marcando-os com um estado activo ou desactivo, considerando as regras de precedência dos operadores temporais associados aos nós. O método é do tipo *top-down* e começa pelo nó raiz da árvore. Este processo continua, a partir da tarefa base, ou seja da folha da árvore, executada através da interacção com o utilizador. A tarefa base executada está associada a um elemento de interacção na interface com o utilizador. O algoritmo, para definir o estado de cada nó de tarefa, executa dois métodos *bottom-up*: *OnChildStarted* e *OnChildFinished*. Estes métodos estão relacionados com as propriedades internas *IsStarted* e *IsFinished*, dos nós de tarefas que são importantes para processar a semântica dos operadores temporais. Inicialmente, o processo corre o método *OnChildStarted* a partir do nó pai da tarefa base executada, continuando a subir na hierarquia da árvore até atingir o nó raiz ou uma condição num operador temporal parar o ciclo. Se o nó pai é uma tarefa, continua para cima na árvore para o próximo nó pai. Se o nó pai é um operador temporal, os sub-nós deste são processados e o seu estado é modificado, de acordo com a semântica definida para os operadores temporais associados. Neste método, o estado interno para a propriedade *IsStarted* é definida como *true* tendo em conta as regras de precedência dos operadores temporais associados.

Quando o método *OnChildStarted* termina, o método *OnChildFinished* é executado de forma semelhante, percorrendo um ciclo *bottom-up,* a partir da tarefa básica executada, a diferença está na propriedade *IsFinished* que é definida como verdadeira de acordo com a influência da semântica dos operadores temporais. Com estes dois métodos *bottom-up* o estado final para cada nó de tarefa é definida como activo ou desactivo, tendo em consideração as propriedades *IsStarted* e *IsFinished* e os nós que representam operadores temporais. Embora este processo seja tipicamente *bottom-up*, o nó que é processado é o nó pai (operador temporal) que, por sua vez, define o estado activo ou desactivo para os nós filhos (tarefas).

V.3.3.Algoritmo CTT vs Algoritmo ETT

Comparando os aspectos principais do algoritmo, definido para a notação CTT original [Paternò, 1999] [Mori, Paternò, et al., 2004] com o que foi proposto para a notação CTT em UML [Nóbrega, Nunes, et al., 2005b], em ambos os métodos a estrutura CTT deve ser transformada numa árvore binária, antes de serem percorridos os nós de tarefa e processado o seu estado para activo ou desactivo.

No método proposto para a notação CTT em UML, a precedência dos operadores é também fundamental, conforme é igualmente definido na notação CTT original. Uma das principais diferenças entre os dois algoritmos está relacionado com a transformação inicial da estrutura CTT para uma árvore binária. Enquanto que na árvore binária da CTT original, os nós tarefa são

criado e ligados por operadores, na proposta de árvore de execução, as tarefas e operadores são definidos como nós, por isso todos os nós numa ETT representam uma tarefa ou um operador temporal.

Outra diferença é encontrada na parte do algoritmo que processa o estado, activo ou desactivo, dos nós da árvore. Todos os nós da árvore binária, na notação CTT original, são inicialmente processados de modo a identificar as tarefas básicas (folhas da árvore), depois são aplicados métodos *bottom-up* a partir destes nós de modo a alterar o estado dos nós. Na proposta para a ETT, não é necessário percorrer todos os nós da árvore binária. A tarefa básica, que foi executada é identificada, sendo deste nó folha que os métodos *bottom-up* são aplicados de modo a activar ou desactivar os nós de tarefa.

O algoritmo descrito nesta secção foi implementado no âmbito deste trabalho, na plataforma Web de destino (ver secção IV.2), resultando numa estrutura de suporte para executar o comportamento da interface conforme definido no PSM com a utilização da semântica da notação CTT.

VI. PROCESSO DE DESENVOLVIMENTO E DE GERAÇÃO DE CÓDIGO

Com a abordagem Wisdom, definiu-se um método para o desenvolvimento de interfaces com o utilizador, guiado por casos de utilização e baseado em modelos, rastreável ao longo das actividades do processo. Por outro lado, com uma ferramenta de modelação, como o editor MetaSketch que será descrito nesta secção, consegue-se obter um suporte gráfico para as actividades de análise e desenho do método. Contudo, depois de serem construídos todos os artefactos para o desenho do sistema e da interface com o utilizador, como o modelo de diálogo e o modelo de apresentação, não está definido, no método Wisdom, uma técnica que possa conduzir o programador directamente para a fase de código. Os modelos são consultados pela equipa de implementação, existindo outro processo criativo para gerar o código necessário à implementação do sistema descrito nos modelos.

Com a abordagem defendida neste trabalho, propomos um alinhamento do método Wisdom com a recomendação MDA, permitindo a geração automática de código a partir dos modelos de desenho. O código será gerado para um contexto de desenvolvimento Web, para uma plataforma suportada pela linguagem PHP [PHP, 2006] e pela *framework* Hydra. Pretende-se atingir um nível onde o desenvolvimento da camada de apresentação de uma aplicação Web, baseada no método Wisdom e nos respectivos modelos, suporte a geração completa de código a partir de regras de transformação e de um modelo específico. Uma das principais vantagens é que a equipa de desenvolvimento poderá concentrar-se apenas nos requisitos que o sistema deverá suportar e não na forma como os mesmos serão implementados, melhorando a qualidade do sistema e a eficiência do desenvolvimento.

Nesta secção, são descritos os principais aspectos do processo de desenvolvimento e de geração automática de código proposto nesta dissertação. Começamos por descrever a ferramenta de modelação utilizada, depois são detalhados alguns aspectos das regras de transformação entre

o modelo PIM e o modelo PSM, designadas por *model-to-model*, e do modelo PSM para o código, designadas por transformações *model-to-code*. Serão também descritas as ferramentas criadas no âmbito deste trabalho para as transformações *model-to-model* e *model-to-code*, permitindo a partir dos modelos Wisdom gerar de forma automática e completa a aplicação Web desenhada. Por fim será apresentado de forma resumida o ciclo de desenvolvimento para a aplicação do método proposto, desde a criação dos modelos de análise e desenho até à geração de código.

VI.1. A FERRAMENTA DE MODELAÇÃO METASKETCH

O editor MetaSketch [Nóbrega, Nunes, et al., 2006] é uma ferramenta de metamodelação capaz de editar modelos e metamodelos. A ferramenta suporta a mais recente tecnologia da OMG, incluindo as normas *XML Metadata Interchange* (XMI) 2.1 [OMG 2005c] e *Meta-Object Facility* (MOF) 2.0, permitindo assim a criação de modelos numa notação capaz de ser expressa segundo a estrutura MOF.

O editor MetaSketch foi inicialmente planeado para suportar a definição de extensões ao UML, como forma de satisfazer um conjunto de necessidades em termos de linguagens de modelação por parte dos autores da ferramenta. A decisão de iniciar o desenvolvimento do editor resultou da incapacidade que a maioria das ferramentas existentes manifestam em relação à definição de extensões à linguagem, e a forma como essas extensões podem ser realizadas e usadas.

Apesar de ser possível utilizar ferramentas de modelação UML, para criar metamodelos, porque o MOF 2.0 e a UML 2.0 [OMG, 2005a] [OMG, 2005b] partilham um conjunto base de elementos da especificação *UML 2.0 Infrastructure* [OMG, 2005a], estas ferramentas não estão desenhadas para suportar a metamodelação. O metamodelo não é o objectivo final mas sim um meio para produzir modelos, logo não é viável criar uma nova ferramenta de cada vez que for necessário um novo metamodelo. Com a ferramenta MetaSketch é possível que o metamodelo seja apenas um novo parâmetro que pode ser adicionado à aplicação, permitindo o suporte flexível à actividade de metamodelação ultrapassando as limitações da maior parte das ferramentas actuais.

O MetaSketch permite editar modelos compatíveis com a norma MOF 2.0, e consequentemente com a UML 2.0 porque esta é uma instância do MOF. Assim, o MetaSketch foi escolhido no âmbito deste trabalho para a criação dos modelos e metamodelos de suporte às actividades Wisdom. Suporta, ainda, a geração de código ao fornecer uma serialização dos modelos no formato XMI. O facto desta ferramenta estar assente em tecnologias OMG, é importante no alinhamento da abordagem deste trabalho com a recomendação MDA.

VI.1.1.A Arquitectura

A arquitectura do editor foi criada de forma a reflectir o suporte à metamodelação referido anteriormente. Assim, em vez de desenvolver um editor alinhado com a última versão da linguagem UML, os autores optaram por arquitectar um editor que fosse capaz de se ajustar a

uma qualquer linguagem de modelação cujo metamodelo fosse descrito através da linguagem subjacente ao MOF.

Toda a arquitectura do MetaSketch Editor é concebida de forma a potenciar as características de metamodelação do MOF, ou seja, na arquitectura do editor o metamodelo (e consequentemente a linguagem de modelação) não está fixo, ao contrário do que vulgarmente acontece com a maioria dos editores de modelação. O que se encontra fixo na arquitectura do MetaSketch é o meta-metamodelo MOF.

Utilizando a arquitectura proposta pelo OMG para as linguagens de modelação, baseado num padrão arquitectural de quatro camadas, a ferramenta contém um nível M3 fixo, através da implementação do MOF, ficando os restantes níveis M2 e M1 determinados durante a execução. A escolha da linguagem e do metamodelo de suporte à linguagem ficam assim diferidas para o tempo de execução. Na Figura VI-I estão representados os quatro níveis definidos pela OMG, por outro lado, na Figura VI-II é descrito o posicionamento da ferramenta MetaSketch em relação a esses níveis.

A hierarquia de quatro camadas da OMG, representada na Figura VI-I, permite diferentes níveis de abstracção relativamente à definição das linguagens de modelação. O nível mas elevado é designado por camada de meta-metamodelação, sendo o objectivo, neste nível, a definição da linguagem para especificar um metamodelo. Este nível é, normalmente, referenciado como M3. O MOF é um exemplo de um meta-metamodelo.

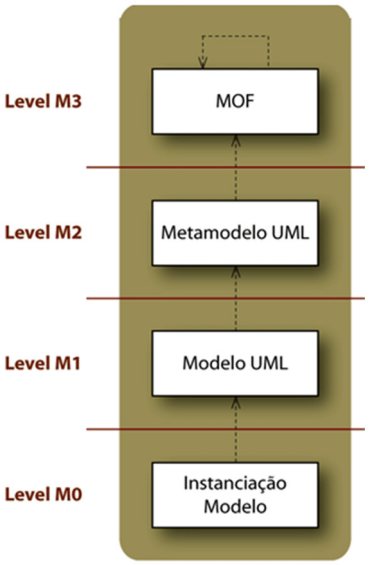

Figura VI-I – A arquitectura de quatro camadas proposta pela OMG para a definição das linguagens de modelação. Adaptado de [Nóbrega, 2007].

A camada de metamodelação, designada por M2, tem como objectivo a definição de um linguagem para especificar modelos. Um metamodelo é uma instância de um meta-

metamodelo, ou seja, cada elemento do metamodelo é uma instância de um elemento do meta-metamodelo. A linguagem UML é um exemplo de um metamodelo definido neste nível. Relativamente à camada de modelo, designada por M1, tem como responsabilidade a definição de linguagens que permitam aos utilizadores a possibilidade de modelar diversos problemas em diferentes domínios. Nesta camada um modelo é uma instância de um metamodelo. Finalmente, a camada M0 contém as instâncias de elementos do modelo definidos no nível M1.

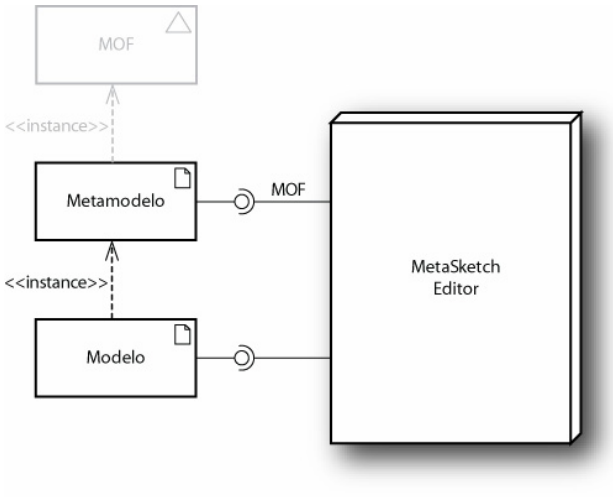

Figura VI-II – Identificação da ferramenta MetaSketch em relação aos quatro níveis definidos pela OMG para a definição das linguagens de modelação. Adaptado de [Nóbrega, 2007].

VI.1.2.Notação e Diagramas

Um dos aspectos importantes, numa linguagem de modelação, é a sua notação. A notação permite que os modelos possam comunicar, transmitir conceitos e ser uma base de discussão entre os intervenientes humanos num processo de desenvolvimento.

A consequência imediata de não se ter uma linguagem de modelação fixa é que também a notação usada na especificação dos modelos para essa linguagem, não pode estar fixa. Para suportar a representação gráfica dos elementos da linguagem a utilizar foi definido, na ferramenta MetaSketch, um formato XML onde através da composição de formas gráficas simples, tendo por base um conjunto pré-definidos e elementos gráficos, se conseguem criar as notações associadas aos elementos da linguagem definidos no metamodelo. Além das formas geométricas é também possível definir propriedades aos componentes gráficos da notação de uma forma dinâmica e flexível.

A criação de um modelo é sempre concretizada num contexto de um diagrama, sendo este, da perspectiva do editor, uma área de desenho onde se podem compor os elementos da linguagem disponíveis de acordo com o contexto do diagrama (estrutural, comportamental, etc.). É necessário, relativamente aos diagramas, especificar cada tipo de diagrama que pode ser

utilizado numa linguagem. Na Figura VI-III é apresentada a ferramenta MetaSketch durante a criação de um diagrama CTT em UML.

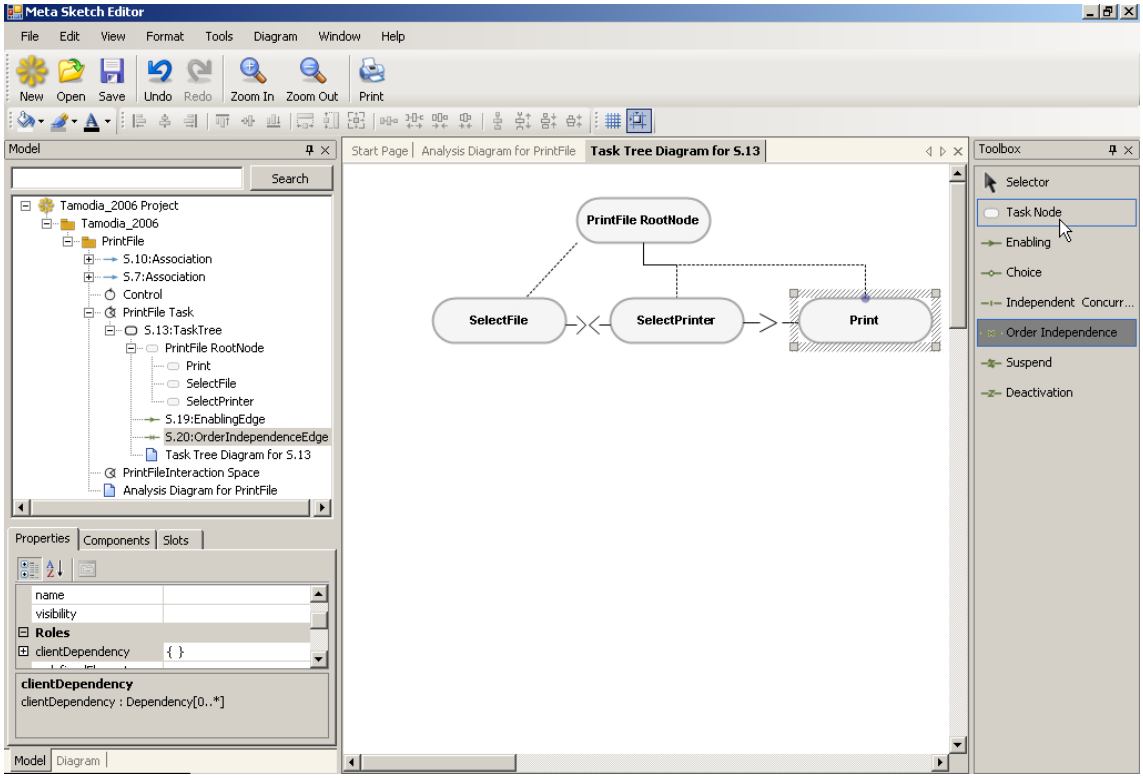

Figura VI-III – Aspecto da ferramenta MetaSketch onde se pode notar a criação de um diagrama com a notação CTT em UML.

VI.2. CRIAÇÃO DOS METAMODELOS E MODELOS PIM E PSM

A ferramenta Metasketch tem já definida a especificação MOF que é a base da capacidade de criação de diversos metamodelos. Adicionalmente, existe um projecto pré-definido com os metamodelos UML 2.0 [OMG, 2005a] [OMG, 2005b] e Wisdom (com as notações PAC e CTT). Nesta secção iremos descrever a criação do metamodelo para suportar a criação dos elementos do PSM (ver secção V.2), sendo também descrita a utilização desse mesmo metamodelo para a criação de modelos PSM. O metamodelo PSM define, na prática, uma linguagem que é um subconjunto da UML e dos elementos Wisdom, a qual se designou por *WebWisdom*.

VI.2.1.PSM: Perfil UML vs Metamodelo

Na secção V.2, foram definidos, conceptualmente, os elementos propostos para o PSM, derivados dos elementos Wisdom com as notações PAC e CTT em UML, que colectivamente compõem um perfil UML. Este perfil UML, para ser aplicado, tem que ser introduzido na ferramenta de modelação, de modo a podermos criar modelos com esses elementos.

A utilização de perfis surge como uma solução para a necessidade de extensão e especialização dos metamodelos das linguagens, sem que as ferramentas de modelação possibilitem a alteração do metamodelo da linguagem. Mas na altura de realização deste trabalho, a versão do editor Metasketch, não suportava ainda a criação de perfis UML 2.0 formal e semânticamente correctos, pelo que tivemos de optar por outra solução. Assim, em vez de criar um perfil UML 2.0, optou-se por estender o metamodelo já existente na ferramenta para suportar o PIM, com os elementos Wisdom, PAC e CTT em UML 2.0. Esta solução, semanticamente, encontra-se muito próxima da solução de utilização de um perfil.

Na Figura VI-IV está representada uma comparação entre a definição de um elemento num perfil UML e a definição de um elemento por extensão do metamodelo. Quando estamos a definir um elemento num perfil UML o que estamos a fazer é estender ou especializar um elemento do conjunto da linguagem UML, através de um estereótipo. Por outro lado se é criado um elemento no metamodelo da UML através da especialização de um elemento existente, semanticamente estamos a estender um elemento existente na linguagem, que é semelhante ao resultado da utilização de estereótipos um perfil UML.

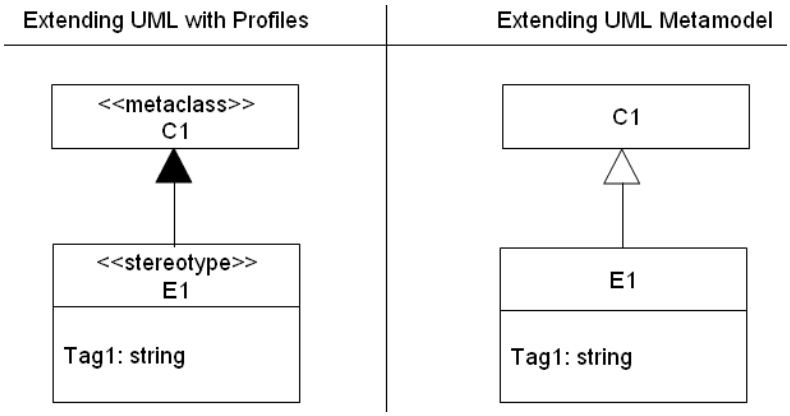

Figura VI-IV – Comparação entre a definição de um estereótipo num perfil UML e a definição de um elemento estendendo o metamodelo UML.

VI.2.2. Definição do Metamodelo do PSM

O editor MetaSketch contém já predefinidos os elementos do PIM, ou seja, os elementos do Wisdom com PAC e CTT, que estendem o metamodelo UML 2.0 para suportar a especificação abstracta de interfaces independente da plataforma. Consequentemente, só foi necessário definir o metamodelo com os elementos do PSM propostos neste trabalho, ou seja, os elementos específicos para a plataforma Web correspondentes ao *WebWisdom,* descrito na secção V.2.

Para definir as extensões do PSM à UML, no MetaSketch, o primeiro passo é abrir o ficheiro da ferramenta que contém as extensões ao UML 2.0 onde se encontram os elementos do PIM. Com este ficheiro em formato XMI, na ferramenta e graficamente, são criados elementos pacote (*package*) de modo a organizar os novos conceitos que irão estender os existentes. Foram criados os seguintes pacotes:

- ***WebConceptualArchitecture:*** Neste pacote, foram definidos os elementos específicos da arquitectura conceptual Wisdom, nomeadamente os elementos *WebModule*, *WebAggregationSpace*, *WebInteractionSpace*, *WebTask* e *WebControl*, com os respectivos atributos, que correspondem às etiquetas definidas para o perfil UML proposto;

- ***WebCanonicals:*** Neste pacote, foram definidos os elementos específicos que estendem os elementos PAC, agrupados de acordo com os tipos *Material*, *Tool* e *Hybrid* e com os respectivos atributos;

- ***WebTaskTrees***: Contém os elementos específicos para a notação CTT em UML, nomeadamente *WebTaskTree*, *WebTaskNode* e os diversos operadores temporais que são um subconjunto do elemento *WebEdge*;

- ***WebPresentations:*** Este pacote foi definido de modo a podermos organizar e descrever a relação entre os elementos específicos PAC e os elementos específicos CTT. Esta é uma

relação, já definida na integração das notações PAC e CTT no método Wisdom [Nóbrega, Nunes, et al., 2005b], que permite associar um elemento de interacção PAC a um nó de tarefa da árvore CTT, definindo a semântica de comportamento do espaço de interacção;

- ***WebWisdom***: Este pacote conjuga todos os outros pacotes de modo a definir e organizar o metamodelo PSM final. Além dos pacotes específicos, *WebConceptualArchitecture*, *WebCanonicals*, *WebTaskTrees* e *WebPresentations*, são colocados os pacotes pré-definidos na ferramenta para a UML 2.0 com o pacote L1 e para o método Wisdom, PAC e CTT em UML com os pacotes *Dialog*, *TaskTrees*, *Analysis*, *Presentations* e *Canonicals*. O pacote L1 está relacionado com os níveis de compatibilidade (*compliance levels*) [UML, 2005b] definido na especificação da super-estrutura da UML 2.0. O nível de compatibilidade L0 contém uma única unidade de linguagem para modelar estruturas de classes, com o nível L1 são adicionadas unidades de linguagem para casos de utilização (*use cases*), interacções (*interactions*), acções (*actions*) e actividade (*activities*). Para mais informações, sobre os níveis L0 e L1, pode ser consultado o capítulo 2.2 em [UML, 2005b].

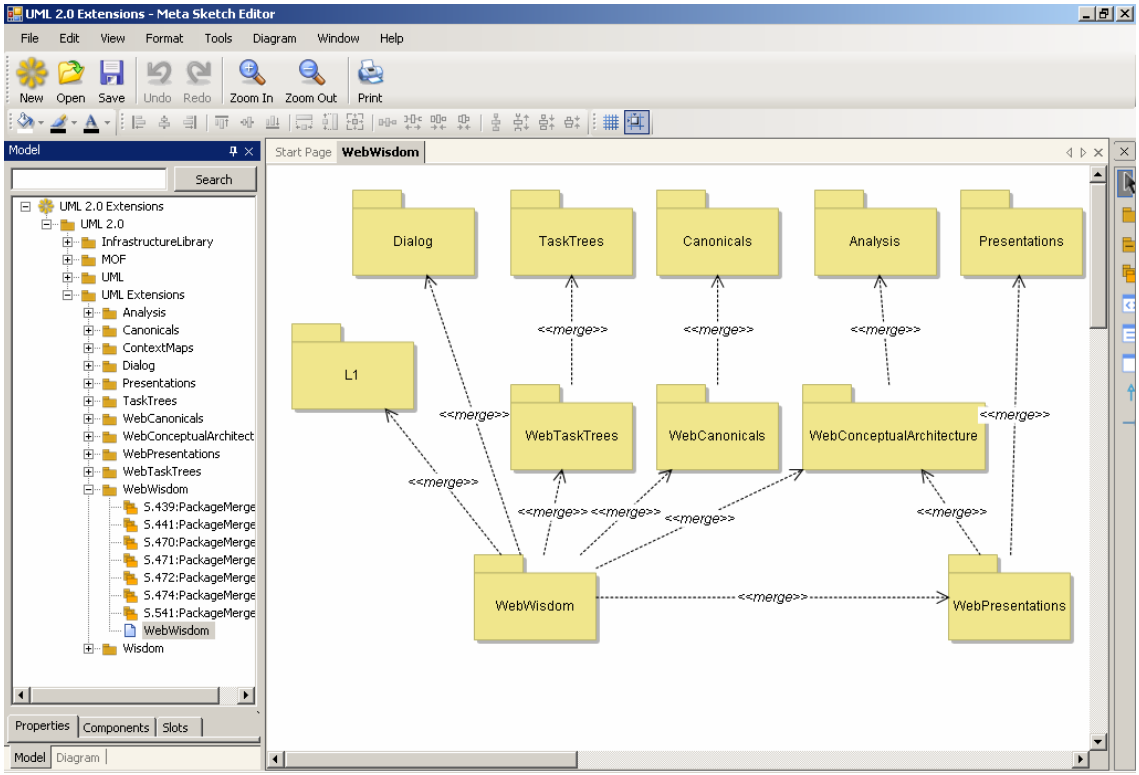

Figura VI-V – O editor MetaSketch com a vista do pacote *WebWisdom* para o metamodelo do PSM.

Na Figura VI-V está representado o pacote *WebWisdom*, enquanto que na Figura VI-VI é detalhado uma perspectiva do conteúdo do pacote *WebTaskTrees*. Depois de definidos e validados os elementos do metamodelo é utilizada a operação da ferramenta para gerar a linguagem definida no metamodelo PSM. Neste caso será gerada uma linguagem que contém a

UML 2.0, os elementos Wisdom, PAC e CTT e os novos elementos específicos, definidos para o PSM proposto. Na operação da ferramenta "Sketch Model Generator", é seleccionado o ficheiro onde foram definidos os pacotes, os diagramas e os elementos da nova linguagem. Em seguida é executada a operação "generate", que cria a estrutura da linguagem definida pelo metamodelo e finalmente é executada a operação "export", que coloca a estrutura final num ficheiro XMI [OMG 2005c] [Grose, Doney, et al. 2002] na pasta "\metamodels" da ferramenta. No âmbito deste trabalho foi gerado o ficheiro "WebWisdom 2.0.xmi" com o metamodelo do PSM, proposto na secção V.2.

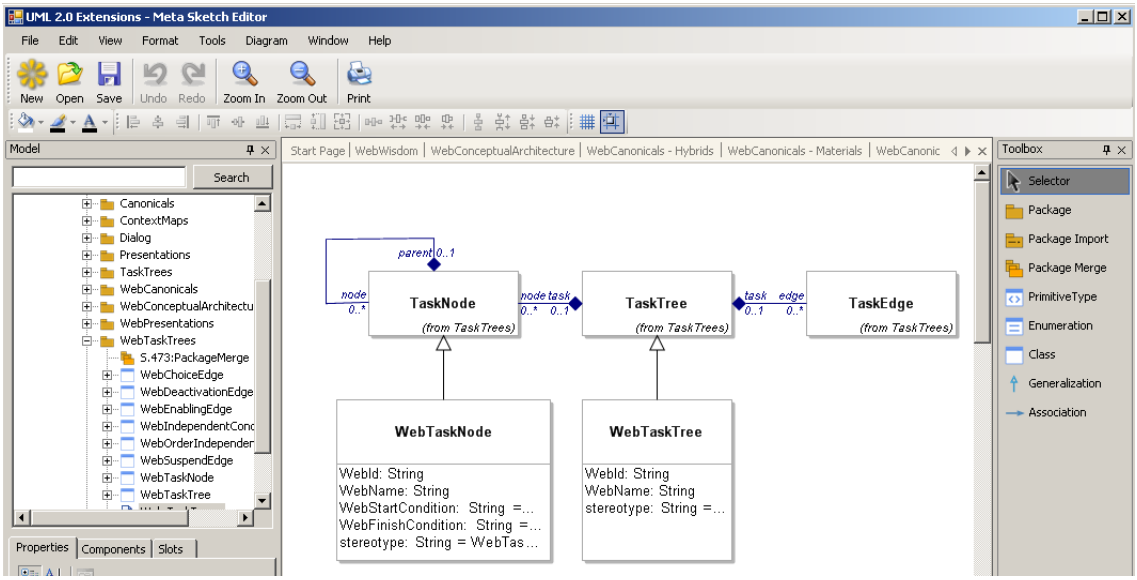

Figura VI-VI – O editor MetaSketch com a vista do pacote *WebTaskTrees* para o metamodelo do PSM, proposto na secção V.2.

VI.2.3.Definição da Notação e dos Diagramas

Depois de criado o ficheiro, que contém o metamodelo PSM validado, devem ser criados os diagramas que definem as áreas de desenho para a manipulação dos novos elementos. Esta definição é executada através de um ficheiro com um formato XML exemplificado na Figura VI-VII.

Figura VI-VII – Estrutura para definição dos diagramas para a utilização da linguagem UML/*WebWisdom 2.0* na ferramenta MetaSketch

Além dos elementos do metamodelo é também necessário definir a notação gráfica associada a cada um dos novos elementos. Foram reutilizados os ficheiros em formato XML com a notação Wisdom, PAC e CTT, uma vez que a notação dos novos elementos do PSM iria ser muito semelhante - as diferenças encontram-se na caixa que identifica o elemento específico através do texto <<*stereotype*>>. Embora estes elementos não sejam estereótipos, pois não foram definidos com recurso a um perfil UML, são sim, novos elementos que estendem o metamodelo da UML 2.0. Assim a designação <<*stereotype*>> foi colocada na notação para manter uma diferenciação entre o PIM e o PSM, pois partilham a representação gráfica dos elementos. A notação dos elementos do PSM foi definida através de três ficheiros:

- "**WebWisdom 2.0 - Analysis.xml**": Definição da representação gráfica dos elementos Wisdom.
- "**WebWisdom 2.0 - Canonicals.xml**": Definição da representação gráfica dos elementos PAC.
- "**WebWisdom 2.0 - TaskTrees.xml**": Definição da representação gráfica dos elementos CTT em UML.

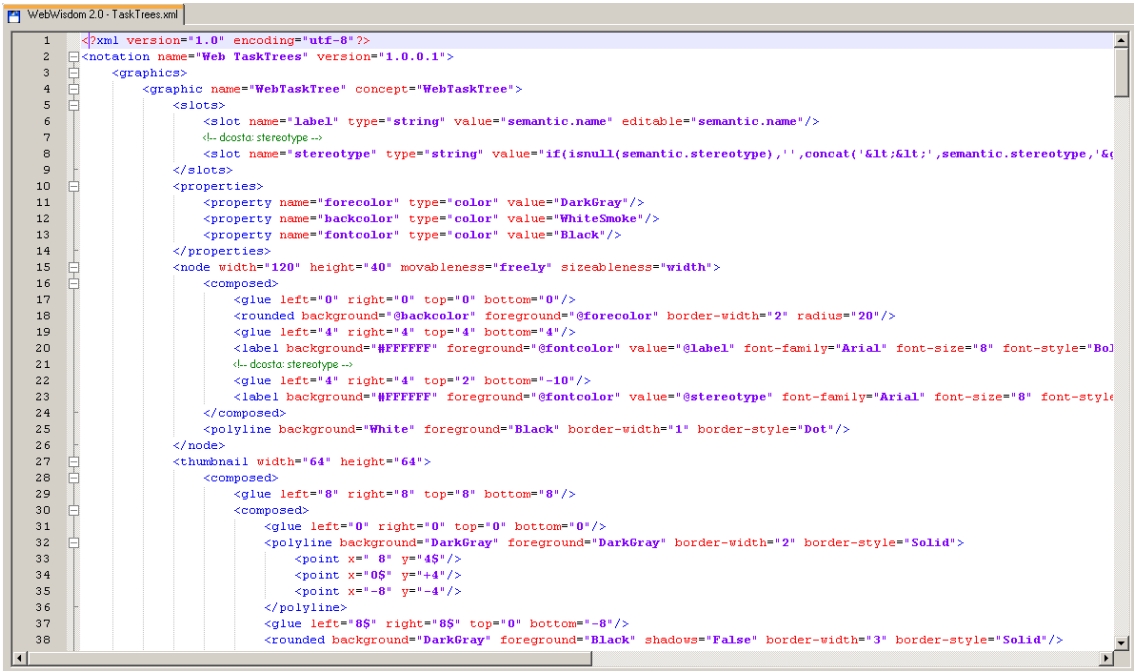

Figura VI-VIII – Estrutura XML, da ferramenta MetaSketch, para definir a notação gráfica do elemento *WebTaskTree* do metamodelo.

Um exemplo da estrutura XML, para a definição da representação gráfica associada aos elementos do metamodelo, está representado na Figura VI-VIII para o elemento *WebTaskTree*.

VI.2.4.Criação de Modelos PIM e PSM

Depois da definição dos elementos do metamodelo PSM, obteve-se a linguagem *WebWisdom 2.0* que não é mais do que a UML 2.0 estendida com os elementos específicos para a Wisdom, PAC e CTT.

Depois desta fase, a utilização do MetaSketch deixa de estar no âmbito da metamodelação passando a ser utilizado o novo metamodelo definido para criar modelos, ou seja, instâncias desse metamodelo. Para a criação de modelos PIM, na ferramenta MetaSketch, é seleccionada a linguagem *Wisdom 2.0*, que contém a UML 2.0 estendida com os elementos Wisdom, PAC e CTT. Para a criação de modelos PSM deve ser seleccionada, previamente, a linguagem *WebWisdom 2.0* que contém a UML 2.0 estendida com os elementos específicos. Conforme será descrito na próxima secção deste documento, o PSM não é criado de raiz pelo editor, apesar de tal ser possível, sendo obtido através de uma transformação do PIM.

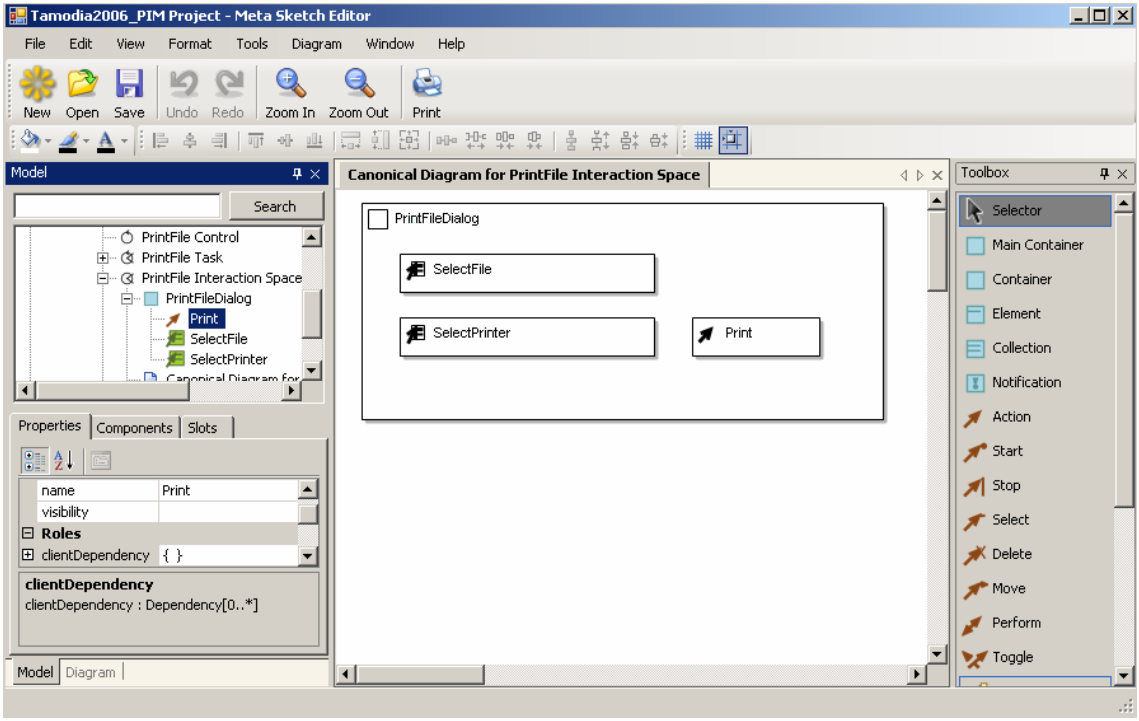

Figura VI-IX – Criação de um diagrama, no MetaSketch, com a notação PAC em UML.

O MetaSketch ao ser utilizado no âmbito de modelação, e não de metamodelação, guarda os modelos criados em ficheiros com formato XMI, permitindo a interoperabilidade entre outras ferramentas de modelação, se tal for necessário. Na Figura VI-IX está um exemplo de criação de um diagrama PIM para o modelo de apresentação, no MetaSketch, com a notação PAC em UML.

VI.3. TRANSFORMAÇÕES MODELO PARA MODELO E MODELO PARA CÓDIGO

Nas secções anteriores descrevemos a forma como os metamodelos utilizados neste trabalho foram definidos através da ferramenta MetaSketch. Nesta secção descrevemos de forma mais detalhada as transformações a aplicar na abordagem de modo a obter o PSM e o código do sistema definido pelos modelos através da extensão *WebWisdom*.

VI.3.1. *PIM-para-PSM*

A transformação do PIM para o PSM é executada utilizando a aplicação *Model2Model*, desenvolvida no âmbito deste trabalho, onde as regras de transformação estão configuradas. O ficheiro, com o PIM, é lido pela aplicação, as regras de transformação são aplicadas e o processo termina com a criação do ficheiro XMI [OMG 2005c] [Grose, Doney, et al. 2002] com os elementos PSM obtidos. A aplicação *Model2Model* foi desenvolvida em PHP [PHP, 2006], pelo que funciona totalmente em ambiente Web e pode ser visualizada na Figura VI-XI. As regras de transformação, nesta fase do trabalho, são definidas como um conjunto de correspondências directas. Como trabalho futuro poderá ser necessário utilizar uma norma como o QVT [Tata Consultancy, 2003] [QVT-Merge Group, 2005] para formalizar as regras de transformação entre o PIM e o PSM. As regras de transformação *PIM-para-PSM*, mais importantes, são descritas na tabela da Figura VI-X.

PIM element >>> is mapped to >>> PSM element	
UseCase	WebModule and WebAggregationSpace
InteractionSpace	WebInteractionSpace
Task	WebTask
TaskTree	WebTaskTree
TaskNode	WebTaskNode. The tagged values *StartCondition* and *FinishCondition* are set with default values;
TaskEdge	WebTaskEdge;
Canonicals (CAP) [*InteractionElementName*]	Web[*InteractionElementName*]. The tagged value *CAP2HTML* is set following the mappings in Figure V.4;

Figura VI-X – Tabelas com as regras de transformação mais importantes de *PIM-para-PSM* . Adaptado de [Costa, Nóbrega, et al., 2007].

O PSM resultante e os respectivos elementos podem ser editados e modificados utilizando a ferramenta MetaSketch, promovendo um nível de independência entre o PIM e o PSM gerado. Quando o PSM está terminado, para uma determinada evolução do sistema em

desenvolvimento de acordo com o método Wisdom, o correspondente ficheiro XMI deve ser carregado para a aplicação *Model2Code*, que transformará o modelo em código.

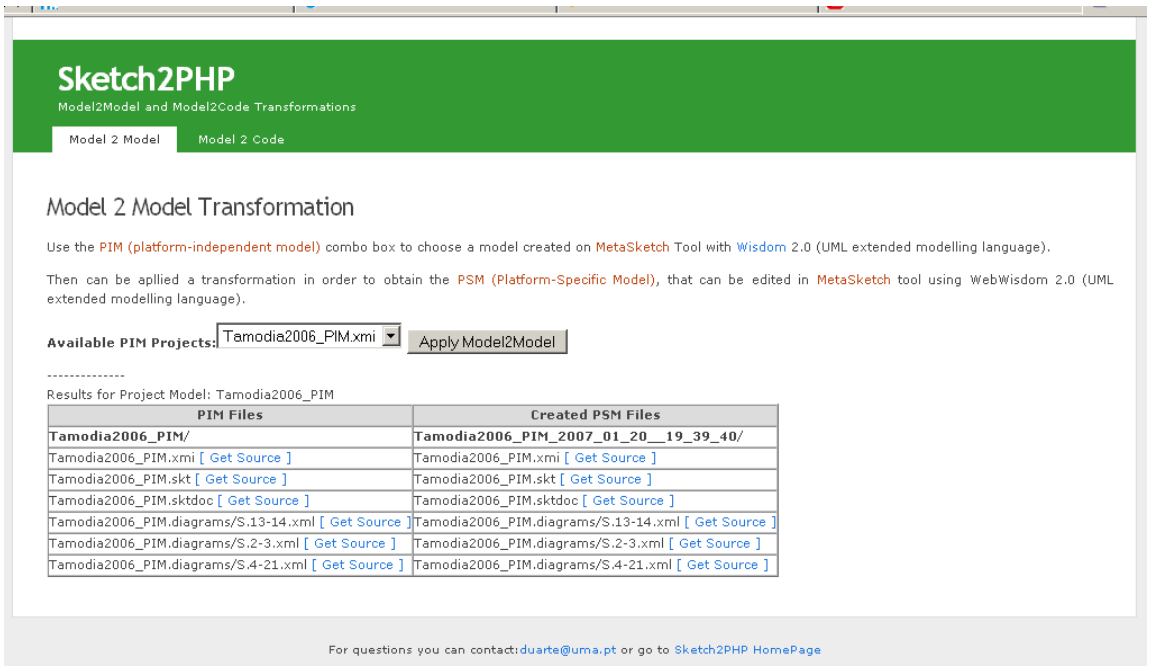

Figura VI-XI – A aplicação Web *Model2Model*, desenvolvida em PHP 4.x no âmbito deste trabalho.

VI.3.2. *PSM-para-Código*

Neste passo, a aplicação *Model2Code* percorre a estrutura interna com o modelo XMI, aplicando a cada elemento um *template* de código pré-configurado, que é preenchido com a informação sobre os elementos do modelo previamente carregados. A aplicação *Model2Codel* foi também desenvolvida no âmbito deste trabalho, na linguagem PHP, funcionando em ambiente Web, partilhando juntamente com a aplicação *Model2Model* a responsabilidade de todas as transformações entre os modelos e o código. As *templates* de código, pré-configuradas na aplicação *Model2Code*, são associadas a um elemento ou propriedade do modelo. Os resultados da aplicação destas *templates* compõem a estrutura do código do projecto. As principais regras de transformação para obter os artefactos de código, a partir dos elementos do PSM, são descritas na tabela da Figura VI-XII.

PSM element >>> is mapped to >>>	Code Artefact
WebModule	PHP Class, a XML file and a HTML template file
WebAggregationSpace	PHP Class, a XML file and a HTML template file
WebInteractionSpace	PHP Class, a XML file and a HTML template file
UML's CTT Profile: WebTask,WebTaskTree,WebTaskNode and WebTaskEdge	UML's CTT Profile semantics are placed in a XML File (the same associated with the *WebInteractionSpace*)
CAP's Profile: Web[*InteractionElementName*]	Parameters in the HTML Templates and in XML file (the same associated with the *WebInteractionSpace*)

Figura VI-XII – Tabelas com as regras de transformação mais importantes de *PSM-para-Código*. Adaptado de [Costa, Nóbrega, et al., 2007].

A estrutura de código resultante, conforme descrito nas tabelas da Figura VI-XIII, tem um directório raiz do projecto que contém as classes base para a *framework* Hydra e as configurações gerais ao projecto, onde se definem parâmetros como a ligação às fontes de dados (Base de Dados ou *Webservices*), etc. Outro item importante é o directório "/modulos" que contém os artefactos de código gerados a partir dos elementos do PSM. Neste directório são criados subdirectórios para cada elemento *WebModule* definido no modelo. Cada directório *WebModule* contém por sua vez estruturas de código para os elementos *WebAggregationSpaces* e *WebInteractionSpaces*, compostas por classes PHP, XML e ficheiros de *templates* HTML. A semântica para os elementos *WebTask* e *WebTaskTree*, contendo as regras de comportamento da interface com o utilizador, são colocadas no mesmo ficheiro XML associado ao elemento *WebInteractionSpace*. Na secção VII.3, na Figura VII-VIII está descrita, em maior pormenor, uma estrutura de código real, gerada no âmbito deste trabalho.

(1) Main Directories	Contains:
/classes	Hydra framework classes
/conf	Main configuration files
/css	Framework definitions
/html	Framework definitions
/imagens	Framework definitions
/javascript	Framework definitions
/modulos	WebModule's Sub-Directories
/templates	Global presentation structure

(2) WebModule's Sub-Directories	Contains:
/classes	PHP Classes for WebInteractionSpaces
/classes/eagr	PHP Classes for WebAggregationSpaces
/classes/modulo	PHP Class for WebModule
/conf	XMl Files for WebInteractionSpaces
/conf/eagr	XMl Files for WebAggregationSpaces
/conf/modulo	XMl File for WebModule
/templates	HTML Templates for WebInteractionSpaces
/templates/eagr	HTML Templates for WebAggregationSpaces
/templates/modulo	HTML Template for WebModule

Figura VI-XIII – Tabelas com a estrutura de código principal (1) e com a estrutura detalhada para elemento *WebModule* (2). Adap. de [Costa, Nóbrega, et al., 2007].

A estrutura de código da *framework* Hydra é flexível, permitindo que as classes, as *templates* e o código de configuração possam ser modificados depois da geração automática de código

permitindo que possa ser refinado algum aspecto da apresentação ou comportamento da interface Web com o utilizador. A aplicação *Model2Code* pode ser visualizada na Figura VI-XIV.

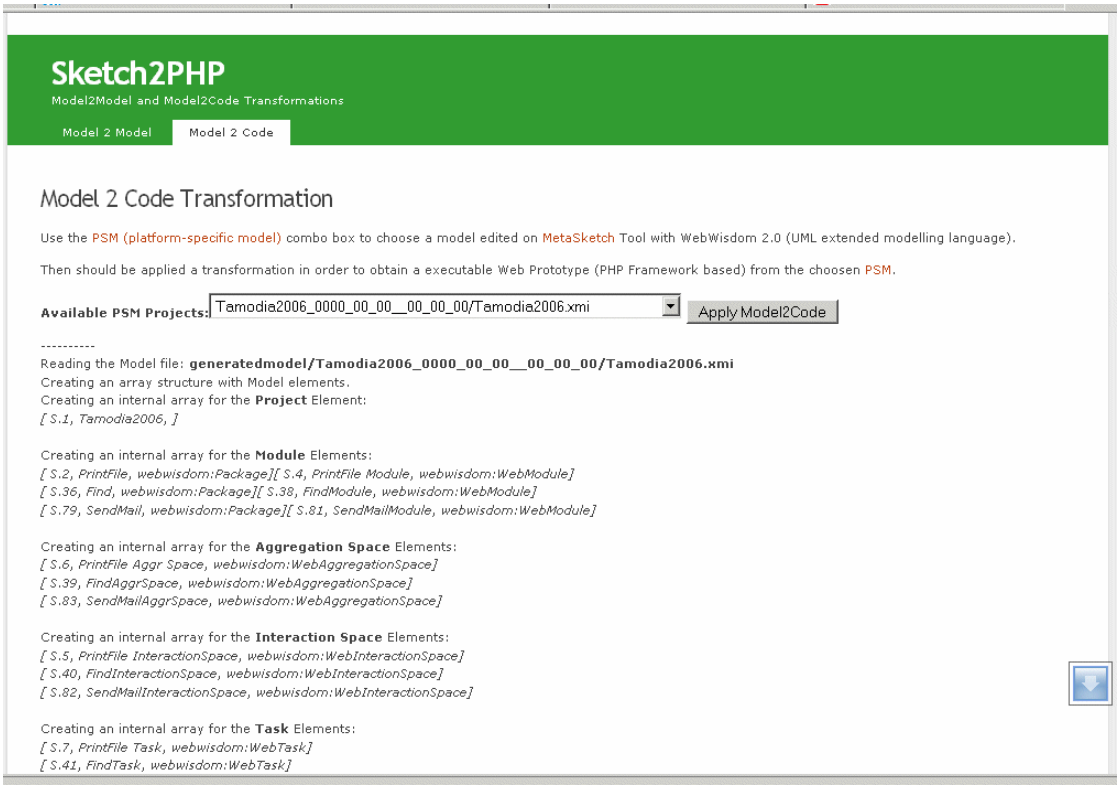

Figura VI-XIV – A aplicação Web *Model2Code*, desenvolvida em PHP 4.x no âmbito deste trabalho.

VI.4. CICLO DE MODELAÇÃO E GERAÇÃO DE CÓDIGO

Nesta secção resumimos o ciclo de desenvolvimento proposto pela abordagem definida neste trabalho, de modo a conduzir à geração de código a partir de modelos Wisdom. Apresentamos o ciclo de desenvolvimento como uma sequência de passos apresentados de forma sequencial para simplificar a descrição do método. Como o método Wisdom preconiza a prototipificação evolutiva, os passos ilustrados não são lineares ou sequenciais, existe sempre o conceito de evolução em qualquer um das fases e actividades do método, o que implica que uma actividade como o desenho pode ser executada diversas vezes durante o projecto.

Um ciclo de desenvolvimento típico deverá envolver uma sequência de acções semelhante ao descrito em seguida:

(i) Numa primeira fase, deve ser criado o modelo PIM, com o editor Metasketch, utilizando o metamodelo *Wisdom 2.0*, definido previamente;

(ii) Num segundo passo, são aplicadas ao modelo PIM, serializado num ficheiro em formato XMI [OMG 2005c], as regras de transformação através da aplicação Web *Model2Model* onde estão configuradas as regras de mapeamento e de transformação de elementos e de associações;

(iii) Através da transformação *model-to-model* obtém-se o modelo PSM, em formato XMI, que deve ser editado através da ferramenta MetaSketch e utilizando a linguagem *WebWisdom 2.0*, de modo a introduzir os detalhes necessários para a plataforma de implementação;

(iv) O passo final será utilizar a aplicação *Model2Code* de modo a gerar o código para a plataforma Web, neste caso baseada na *framework* Hydra, permitindo o teste e execução do sistema desenhado.

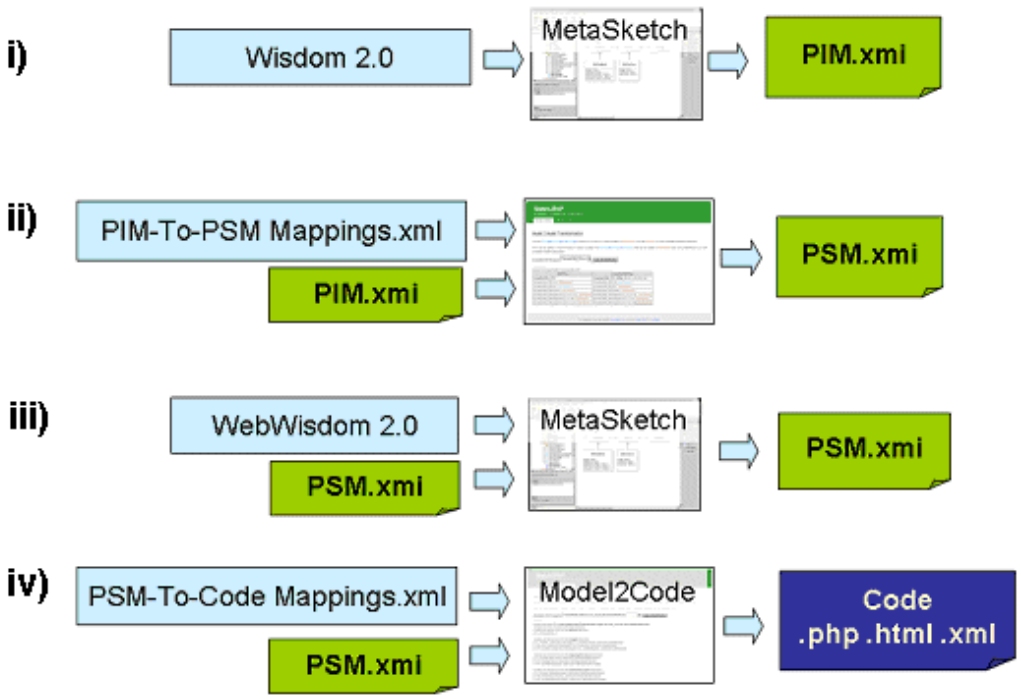

Figura VI-XV – Ciclo de Desenvolvimento para a abordagem proposta composto por quatro passos: (i)PIM, (ii) *PIM-para-PSM*, (iii) PSM e (iv) *PSM-para-Código.* Adaptado de [Costa, Nóbrega, et al., 2007].

Na Figura VI-XV, estão representados os quatro passos do ciclo de desenvolvimento proposto para a abordagem Wisdom, alinhada com a recomendação MDA.

VII. EXEMPLO DE APLICAÇÃO: "ENVIO DE MAIL"

Neste capítulo, apresentamos um exemplo de aplicação do método proposto, para o desenho e geração de interfaces com o utilizador, que engloba a definição do PIM (*Platform-independent Model*), a transformação e alteração do PSM (*Platform-specific Model*) e a geração de código, a partir dos elementos do modelo específico.

Neste exemplo, são descritos os modelos das actividades de análise e de desenho (ver secção II.2) que foram construídos, utilizando a ferramenta MetaSketch (ver secção VI.1). São ainda descritas também as transformações aplicadas para obter o modelo específico. Os protótipos funcionais gerados são executados numa plataforma Web na linguagem PHP 4.x, com a *framework* Hydra (ver secção IV.) a suportar os mecanismos da interface.

No editor MetaSketch, utilizando o metamodelo com a linguagem *Wisdom 2.0* que estende o metamodelo da UML 2.0, definiram-se os elementos PIM. É, também, utilizado o metamodelo da *WebWisdom 2.0*, que contém a UML 2.0 estendida com os elementos PSM. Estes metamodelos foram descritos na secção V. deste documento e contêm as notações e a semântica para o método Wisdom, refinado em [Nóbrega, Nunes, et al., 2005b] com as notações PAC e CTT, e com os novos elementos específicos do PSM.

A plataforma de implementação Web, utilizada para execução dos protótipos de interface e suportada pela *framework* Hydra, inclui os mecanismos e conceitos definidos pelo método de desenho de interfaces com o utilizador. A organização, interna, dos ficheiros e classes reflectem os objectos das perspectivas Wisdom da arquitectura conceptual e do modelo de apresentação, contendo ainda outras classes para a semântica comportamental, definida pelo modelo de diálogo. A plataforma inclui o algoritmo para execução da notação CTT em UML, definido em [Nóbrega, Nunes, et al., 2005b], utilizada para definir a semântica do modelo de diálogo. A

framework tem, como objectivo, o suporte da estrutura e do comportamento da camada de apresentação, de uma aplicação Web, fornecendo mecanismos de interligação com as camadas de dados e de negócio.

VII.1. MODELO PIM

Para ilustrar a abordagem, proposta neste trabalho, é apresentado um caso prático para uma aplicação Web simples que tem por objectivo o envio de e-mail. O PIM deste exemplo segue as diversas vistas e modelos do método Wisdom, onde se descrevem elementos estruturais, definidos na arquitectura conceptual e na perspectiva do modelo de apresentação com a notação PAC, e elementos que descrevem o comportamento e interacção no modelo de diálogo com a notação CTT em UML. O modelo PIM foi criado através da ferramenta MetaSketch, a partir do metamodelo *Wisdom 2.0*. Para simplificar a aplicação do método proposto e como só está a ser definido um elemento de espaço de interacção, não é representada a perspectiva do modelo de apresentação com as relações, por exemplo de navegação, entre espaços de interacção. Na Figura VII-I, está representada a sequência de transformações e as perspectivas dos modelos a aplicar no método proposto.

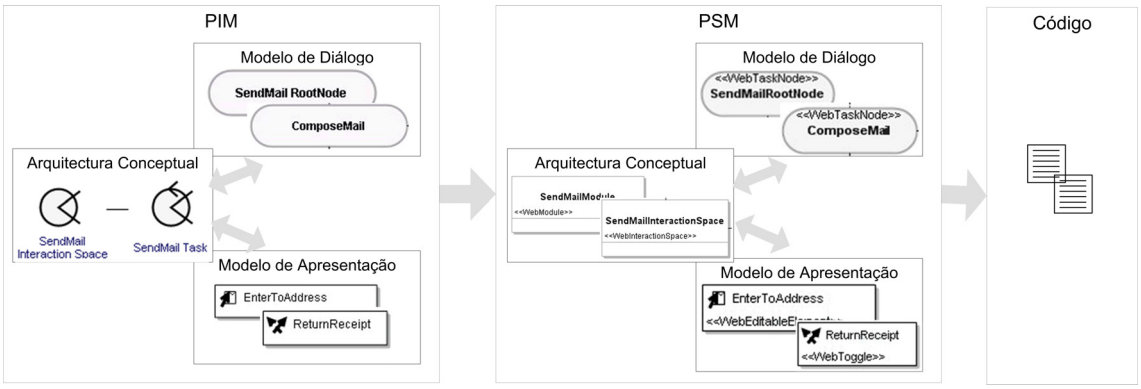

Figura VII-I – Esquema ilustrativo das várias perspectivas dos modelos para a aplicação do método proposto, com a definição do PIM, a transformação para o PSM e a geração de código.

Os elementos definidos na arquitectura conceptual do PIM, para o caso em análise, são os seguintes:

- "**SendMail Interaction Space**": Classe de espaço de interacção que representa a perspectiva da estrutura de apresentação da interface;
- "**SendMail Task**": Classe de tarefa que representa a perspectiva de comportamento ou de diálogo da interface;
- "**SendMail Control**": Classe de controlo que representa as acções internas do sistema para concretizar o envio da mensagem de e-mail;

Nenhum elemento do tipo entidade é definido na arquitectura conceptual porque neste caso prático não foi tido em consideração a necessidade de haver persistência de dados. Na Figura VII-II, está representada a arquitectura conceptual do PIM para o caso em análise.

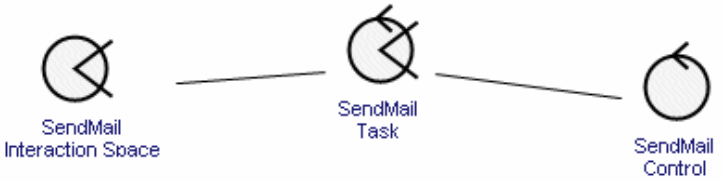

Figura VII-II – A arquitectura conceptual do PIM para o caso em análise.

Os elementos da vista do modelo de apresentação do PIM, definidos com a notação PAC e que estão associados e detalham o elemento "SendMail Interaction Space", são os seguintes:

- "**SendMail Dialog**": Elemento *Material Container* que contém todos elementos da estrutura da apresentação;
- "**EnterToAddress**", "**EnterSubject**" e "**EnterBodyMessage**": componentes *EditableElement*;
- "**SelectFile**: componente *SelectableCollection*;
- "**AttachFile**", "**Cancel**" and "**SendMessage**": componentes *ToolAction*;
- "**ReturnReceipt**": componente *Toggle*;

A Figura VII-III representa o diagrama com a vista do modelo de apresentação do PIM, com os elementos utilizados no exemplo.

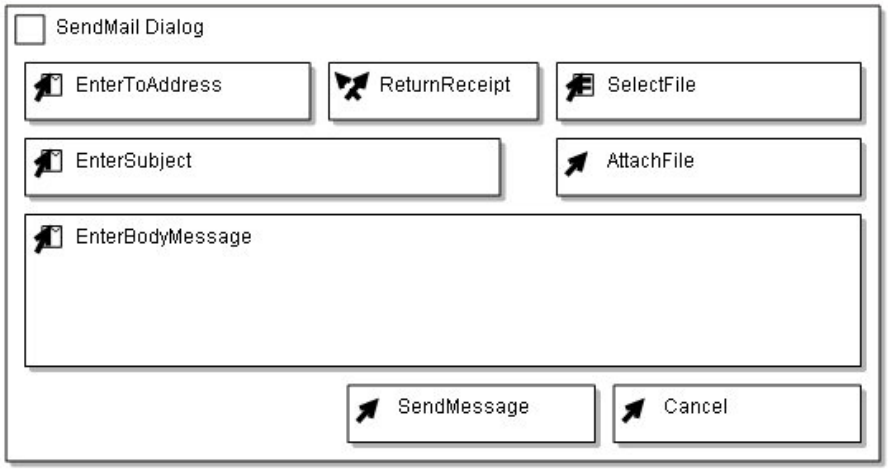

Figura VII-III – A vista do modelo de apresentação do PIM para o caso prático "Envio de Mail".

Os elementos do modelo de diálogo do PIM, definidos com a notação CTT em UML, que estão associados e detalham o elemento "SendMail Task", são os seguintes:

- "**SendMail RootNode**": elemento *TaskTree* que representa o nó de raiz;
- "**EnterSubject**", "**ReturnReceipt**", "**AddAttach**" e "**EnterBodyMessage**": nós de tarefa opcionais, o que significa que a propriedade *Optional* dos elementos está definida como *true*.
- "**Z**": operador temporal *Deactivation*;
- "**|**": operador temporal *Independent Concurrency*;
- "**>**": operador temporal *Enabling*;

Todos os outros elementos do modelo de diálogo ("ComposeMail", "Cancel", etc.) são nós de tarefa, ou seja, são instâncias *TaskNode* e estão representados na Figura VII-IV. Os nós-folha da árvore, que representam as tarefas básicas, estão directamente associados aos elementos PAC da apresentação e têm, inclusivamente, os mesmos nomes. Por exemplo, o nó de tarefa "Cancel", do modelo de diálogo, está associado com o componente *ToolAction* "Cancel" do modelo de apresentação.

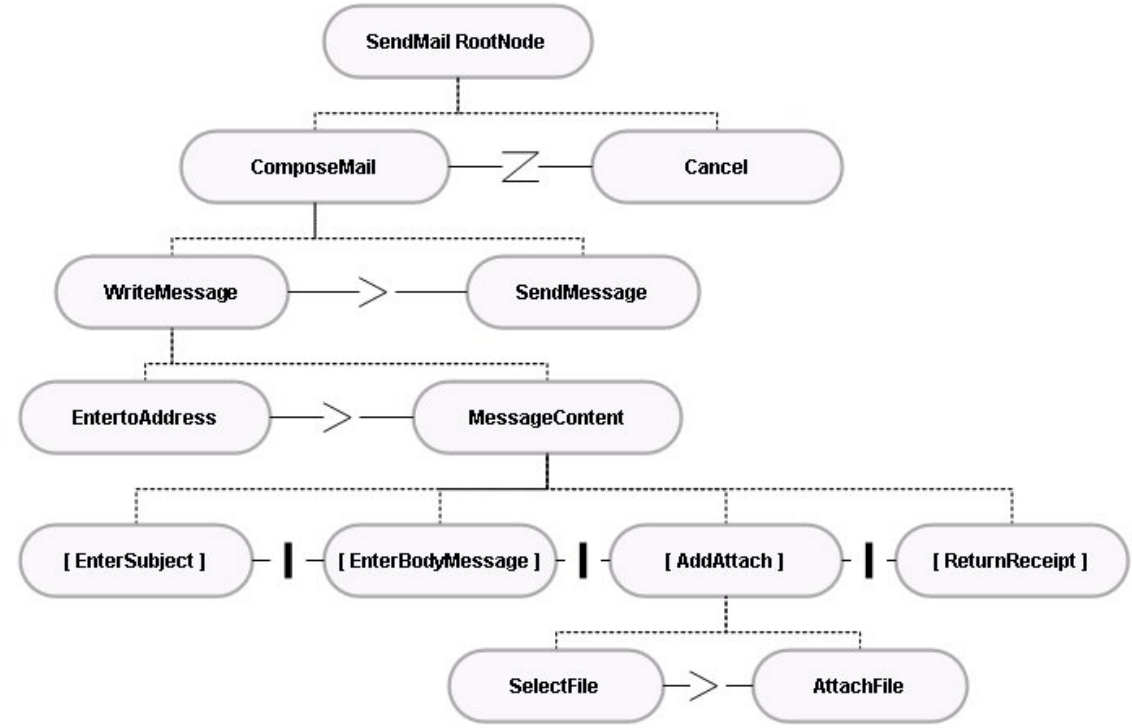

Figura VII-IV – A vista do modelo de diálogo para o PIM do caso prático "Envio de Mail".

Analisando o diagrama do modelo de diálogo no PIM (ver a Figura VII-IV), onde se encontra definida a semântica da interacção, podemos verificar que, no estado inicial, todas as tarefas básicas estão desactivadas, com excepção dos nós "EnterToAddress" e "Cancel". Se o elemento "Cancel" é seleccionado todos os outros elementos serão desactivados, terminando, assim, a interacção. Quando o elemento "EnterToAddress" é introduzido, as tarefas básicas "EnterSubject", "EnterBodyMessage", "ReturnReceipt", "SelectFile" e "SendMessage" são activadas e os seus elementos PAC associados também. Depois deste passo, é possível enviar a mensagem de e-mail ou editar outras opções antes do envio. Se for escolhido o elemento "SelectFile", este ficará bloqueado e o "AttachFile" passará para o estado activo.

VII.2. MODELO PSM

Com os elementos anteriores, que definem o PIM para o caso prático em análise, e com as regras de transformação, configuradas na aplicação *Model2Model* (ver secção VI.3), é criado o PSM, com os elementos e atributos para a sua implementação, em componentes Web concretos. Depois da obtenção, automática, do PSM, é adicionada ou alterada informação, com a ferramenta MetaSketch, relacionada, por exemplo, com as regras de transformação entre os elementos PAC e os componentes Web a implementar, sendo esta informação colocada no atributo "CAP2HTML" (ver secção V.2.3). Tanto o PIM como o PSM, manipulados com a ferramenta MetaSketch, são guardados num formato de acordo com a especificação XMI.

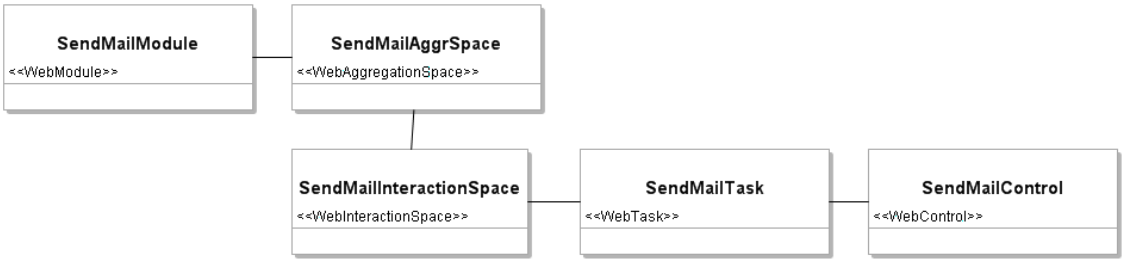

Figura VII-V – A vista da arquitectura conceptual do PSM para o caso em análise "Envio de Mail".

O PSM tem as mesmas perspectivas de modelação definidas no PIM com a arquitectura conceptual, o modelo de apresentação e o modelo de diálogo. Como já foi referido, só foi definido um espaço de interacção na arquitectura, não sendo, por isso, representada a perspectiva do modelo de apresentação, com as relações entre espaços de interacção. Desta forma os novos elementos do PSM, obtidos a partir dos elementos do PIM e das regras de transformação da secção VI.3.1, são descritos, em seguida, nas três perspectivas do modelo. Na Figura VII-V estão representados os elementos da arquitectura conceptual do PSM para o caso em análise.

Para a arquitectura conceptual, com a transformação, foram obtidos os seguintes elementos do PSM:

- "**SendMailInteractionSpace**", manteve o nome e foi transformado num elemento *WebInteractionSpace*.
- "**SendMailTask**", manteve o mesmo nome e foi transformado num elemento *WebTask*.

- "**SendMailControl**", manteve o mesmo nome e foi transformado num elemento *WebControl*.

Foram ainda criados novos elementos na arquitectura conceptual, necessários para a plataforma de implementação:

- "**SendMailModule**", elemento *WebModule*.

- "**SendMailAggrSpace**", elemento *WebAggregationSpace*.

Relativamente ao modelo de apresentação do PSM foram, obtidos os seguintes elementos:

- "**SendMailForm**": obtido do elemento "SendMail Dialog", foi transformado num elemento *WebContainer*. No novo atributo "CAP2HTML", foi colocado o valor *Form,* indicando que o componente concreto, a implementar, será um formulário HTML.

- "**EnterToAddress**", "**EnterSubject**" e "**EnterBodyMessage**": obtidos dos elementos PIM com o mesmo nome foram convertidos em componentes *WebEditableElement*. No novo atributo "CAP2HTML", foi colocado o valor *InputText,* indicando que o componente concreto, a implementar, será um campo de introdução de texto HTML.

- "**SelectFile**": obtido do elemento PIM com o mesmo nome foi convertido num componente *WebSelectableCollection*. No novo atributo "CAP2HTML", foi colocado o valor *FileBrowse,* indicando que o componente concreto, a implementar, será um campo de escolha de ficheiros em HTML.

- "**AttachFile**", "**Cancel**" e "**SendMessage**": obtidos dos elementos PIM com o mesmo nome foram convertidos em componentes *WebToolAction*. No novo atributo "CAP2HTML", foi colocado o valor *SubmitButton,* indicando que os componentes concretos, a implementar, serão botões de submissão do formulário em HTML.

- "**ReturnReceipt**": obtido do elemento PIM com o mesmo nome foi convertido num componente *WebToggle*. No novo atributo "CAP2HTML", foi colocado o valor *CheckBox,* indicando que o componente concreto, a implementar, será uma *checkbox* em HTML.

- "**SendValues**": novo elemento do tipo *WebToolAction,* foi colocado devido à necessidade de existir um componente para submeter os dados do formulário, sem interferir com os outros botões de submissão, associados ao comportamento descrito pela árvore CTT.

A informação sobre as posições dos elementos nos protótipos abstractos, definidas no diagrama PAC, através da ferramenta e na notação gráfica, é também guardada e reflecte a posição relativa, que o elemento deverá ter na interface final. Neste caso de aplicação do método,

seguindo as correspondências definidas na Figura V-V da secção V.2.3: i) mapeou-se o componente PAC *SelectableCollection* para um componente concreto HTML do tipo *FileBrowse*; ii) o componente *ToolAction* é definido como um componente HTML do tipo *SubmitButton*; e iii) o elemento *Material Container* é mapeado, neste caso, para uma estrutura HTML do tipo *Form* porque contém o elemento *ToolAction,* que implica uma submissão de dados por parte do utilizador.

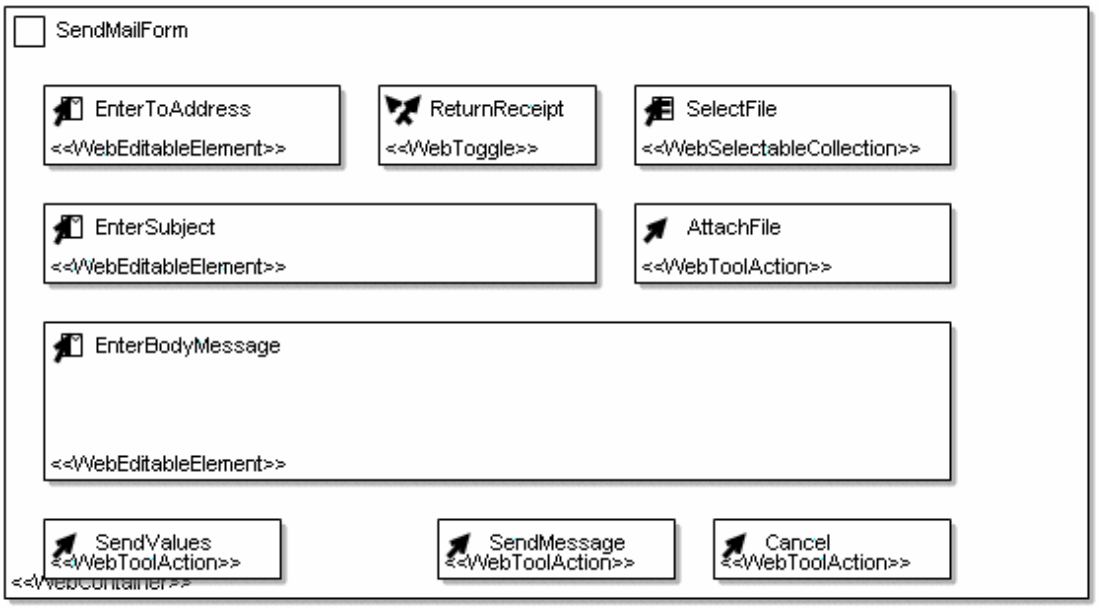

Figura VII-VI – A vista do modelo de apresentação do PSM para o exemplo "Envio de Mail".

Na Figura VII-VI, estão representados os elementos do modelo de apresentação do PSM para o caso em análise. Relativamente ao modelo de diálogo, com a árvore CTT em UML, foram definidos os seguintes elementos:

- "**SendMailRootNode**": Obtido do elemento PIM com o mesmo nome, foi convertido num componente *WebTaskNode*.
- "**EnterSubject**", "**ReturnReceipt**", "**AddAttach**" e "**EnterBodyMessage**": Obtidos dos elementos PIM, com o mesmo nome, foram convertidos em componentes *WebTaskNode,* mantendo o atributo opcional com o valor *true*.
- "**Z**": Operador temporal convertido em *WebDeactivation*, mantendo a sua semântica.
- "**|**": Operador temporal convertido em *WebIndependentConcurrency,* mantendo a sua semântica.
- "**>**": Operador temporal convertido em *WebEnabling* e mantendo a sua semântica.

Os restantes elementos PSM do modelo de diálogo, que no PIM eram elementos *TaskNode,* foram convertidos em componentes *WebTaskNode*. Na Figura VII-VII, estão representados os elementos do modelo de diálogo do PSM para o caso em análise.

Conforme descrito na secção V.2.1, com a Figura V-III, e na secção VI.2.2 com o pacote *WebPresentations*, existe uma relação de associação entre os elementos PAC e os elementos dos nós de tarefas básicas das árvores CTT em UML. Cada nó de tarefa básico no modelo de diálogo, poderá ter uma relação de "um-para-um" com um elemento PAC do modelo de apresentação. Esta relação, definida inicialmente no PIM, é relevante para o PSM de modo a detalhar as condições de início e de fim da interacção dos nós da árvore CTT, sendo também fundamental para determinar o estado activo ou desactivo dos respectivos nós. Neste trabalho, devido ao contexto de interacção Web, onde está a ser desenvolvida a abordagem, as condições de início e de fim foram simplificadas e ocorrem, em simultâneo, durante uma interacção. Por exemplo, a tarefa "EnterToAddress" está relacionada com o elemento PAC *EditableElement* e está mapeada para um componente HTML do tipo *InputText*. Os atributos para as condições de início e de fim, respectivamente, *WebStartCondition* e *WebFinishCondition*, são definidos com o valor "WITH_VALUE", o que significa que só quando o servidor Web recebe o campo *InputText* com um valor, é que irá definir a tarefa como terminada, processando, em seguida, o seu estado para activo ou desactivo, tendo em consideração os operadores temporais associados.

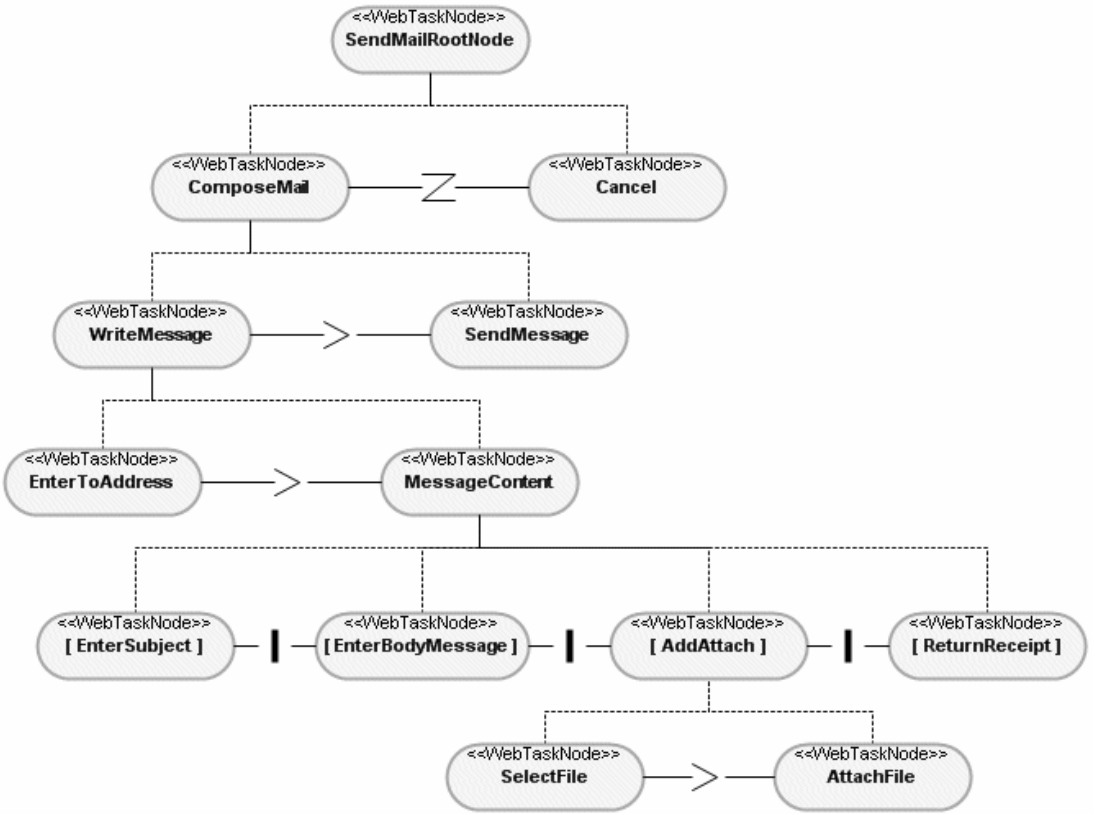

Figura VII-VII – A vista do modelo de diálogo para o PSM do caso prático "Envio de Mail".

VII.3. CÓDIGO GERADO

A partir da representação XMI [OMG 2005c] [Grose, Doney, et al. 2002] do PSM, o código de implementação é gerado para a plataforma Web, sendo para isso utilizada a aplicação *Model2Code* com as regras de transformação pré-definidas entre os elementos do modelo e os artefactos de código. Relativamente aos principais aspectos do código gerado, a plataforma de implementação suporta a separação entre os componentes de apresentação e os componentes de comportamento, conduzindo a uma relação próxima com os elementos do PSM. Os artefactos de código da apresentação são organizados num grupo de classes e *templates* e podem ser modificados, posteriormente, de modo a refinar o código gerado automaticamente. Os artefactos de comportamento são compostos por um grupo de classes e ficheiros de configuração XML que contêm a estrutura e regras semânticas da interacção, definidas com a notação CTT em UML.

A estrutura de código da aplicação é gerada, a partir dos modelos PSM, de acordo com as regras descritas na secção VI.3.2. Os elementos de código da raiz incluem as directorias comuns a toda a aplicação e a directoria "/modulos", que contém o código relacionado com os elementos do desenho. Para o caso, em análise, foi gerado o directório "/modulos/SendMailModule", que por sua vez, contém sub-directórios e um ficheiro na raiz, o "index.php", que é o único ponto de acesso para o processamento do módulo "SendMail", gerado no âmbito do caso em estudo. A estrutura do directório raiz '/' é a seguinte:

- "/**classes**": contém os ficheiros da *framework* Hydra.
- "/**conf**" : contém as configurações comuns a todo o projecto, por exemplo, ligações a base de dados ou a *webservices*.
- "/**css**" ; "/**javascript**" ; "/**html**" ; "/**images**": contém ficheiros comuns a todos os módulos do projecto, relativamente à formatação HTML.
- "/**modulos**": contém os módulos gerados para o projecto, onde existe o código criado a partir dos elementos dos modelos.
- "/**templates**": contém ficheiros de *templates* HTML comuns a todo o projecto, definindo um aspecto homogéneo a toda a aplicação.

Na directoria "/modulos", conforme já foi referido, foi gerada a sub-directoria "/SendMailModule" que contém o código específico dos elementos do caso em análise. Por sua vez, na directoria "/SendMailModule", foram geradas subdirectorias que contêm os ficheiros com a estrutura e a semântica definida nos modelos de desenho do PSM, referentes à modelação da apresentação e da interacção da interface com o utilizador. Estes ficheiros reflectem os elementos definidos na arquitectura conceptual do PSM, nomeadamente os elementos "SendMailModule", "SendMailAggrSpace", "SendMailInteractionSpace" e "SendMailTask". As estruturas de código, contidas na directoria "/SendMailModule", são:

- "**/SendMailModule/classes**": Esta directoria contém as classes PHP dos elementos PSM, nomeadamente a classe "/modulo/cls_modulo_SendMailModule.inc.php", relacionada com o elemento do tipo *WebModule*, a classe "/eagr/cls_eagr_SendMailAggrSpace.inc.php", relacionada com o elemento do tipo *WebAggregationSpace* e a classe "cls_SendMailInteractionspace.php" relacionada com o elemento do tipo *WebInteracionSpace*.

- "**/SendMailModule/conf**": Esta directoria contém os ficheiros com as configurações da semântica do comportamento da interface. Os artefactos criados são "conf_modulo_SendMailModule.inc.php" e "conf_modulo_SendMailModule.xml" para o elemento do tipo *WebModule*, "conf_eagr_SendMailAggrSpace.inc.php" e "conf_eagr_SendMailAggrSpace.xml" para o elemento do tipo *WebAggregationSpace* e por último "conf_SendMailInteractionSpace.inc.php" e "conf_SendMailInteractionSpace.xml" para o elemento do tipo *WebInteractionSpace*.

- "**/SendMailModule/templates**": Nesta directoria, são criados os ficheiros com as *templates* HTML que definem a estrutura da apresentação das interfaces.

- "/**css**"; "/**html**"; "/**imagens**": Nestas directorias, podem ser adicionados ficheiros específicos relativos à formatação HTML.

Na Figura VII-VIII está representada a estrutura de código da directoria "/modulos/SendMailModule", que contém todos os artefactos de código, gerados para o caso em análise "Envio de Mail".

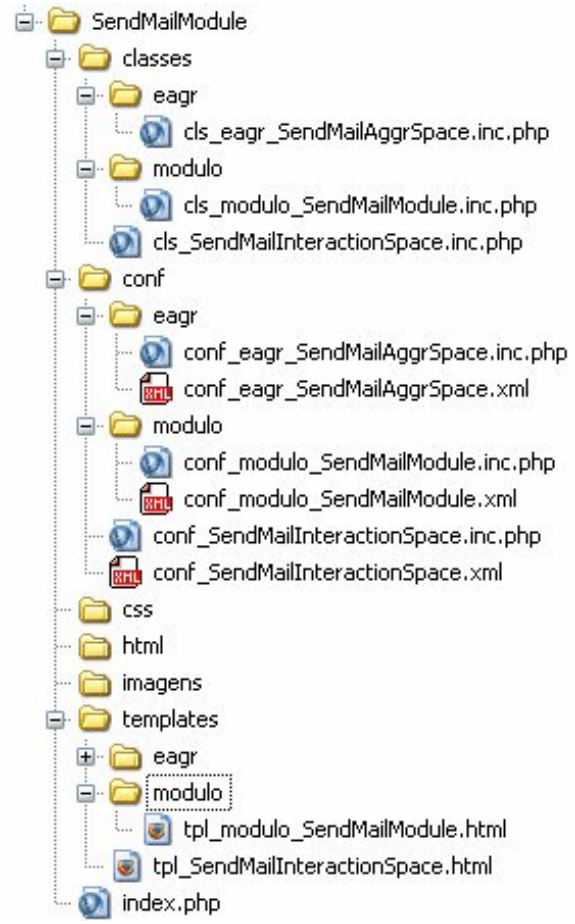

Figura VII-VIII – Estrutura de código gerada na directoria "/modulos/SendMailModule/" que contém os ficheiros que definem a estrutura e a semântica dos modelos, para o caso em análise "Envio de Mail".

Conforme já foi referido (ver secção V.1), o método Wisdom está mais vocacionado para o desenvolvimento das interfaces com um utilizador, enquanto que, os aspectos de negócio e de domínio não são detalhados, nem são propostas técnicas ou notações para os descrever em detalhe. Consequentemente, para os elementos do tipo "Controlo" e "Entidade", definidos na arquitectura conceptual não são propostos modelos para os detalhar em pormenor. Para o caso em estudo de "Envio de Mail", o elemento <<*control*>>, (ver secção II.2.2) é implementado com um mecanismo existente na *framework* Hydra, não sendo, por isso, necessário ligar a camada de apresentação, cujo código é gerado, automaticamente, à camada de dados ou de negócio do sistema. Por esta razão este elemento não é detalhado ou descrito em pormenor. Na Figura VII-IX está representado a interface final em HTML para o caso prático em análise de um formulário para envio de e-mail, gerado, automaticamente, a partir do PSM e de acordo com o que foi definido para as perspectivas de diálogo e de apresentação.

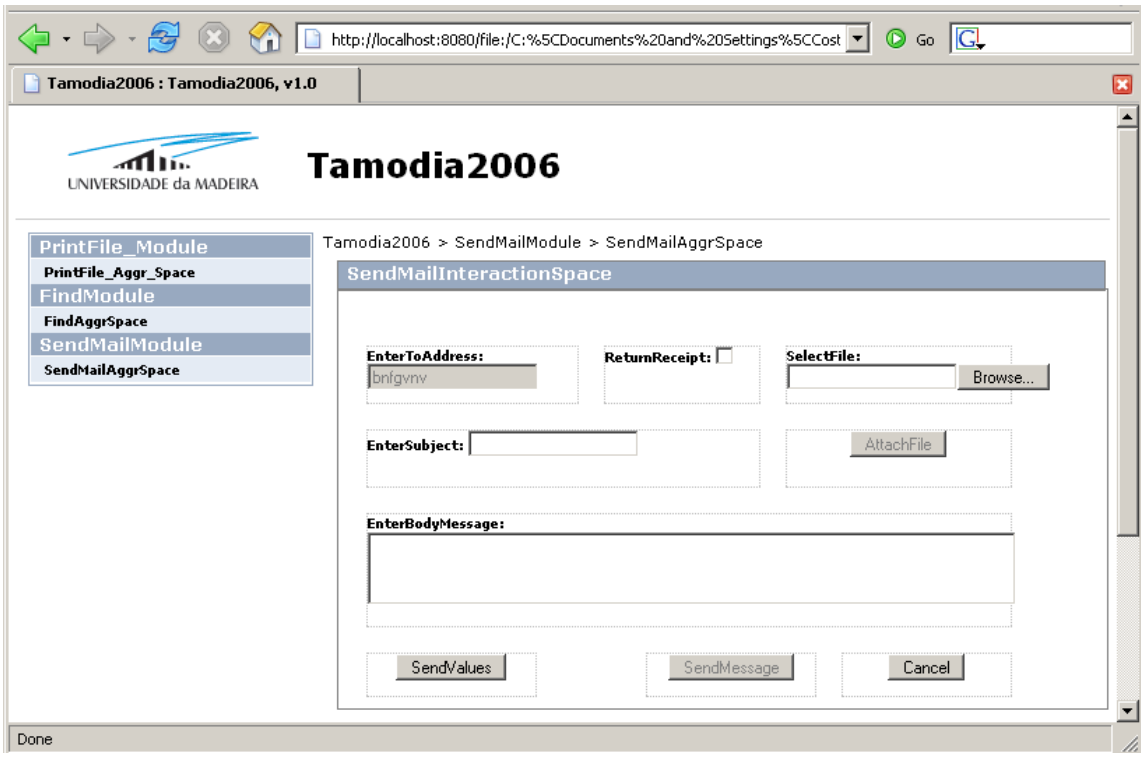

Figura VII-IX – O protótipo da interface com o utilizador gerado a partir do PSM para o caso "Envio de Mail".

Na Figura VII-X, apresentamos um resumo das várias fases e modelos na aplicação do método para o caso. São esquematizados os modelos PIM, o PSM, o código e a aplicação Web gerada.

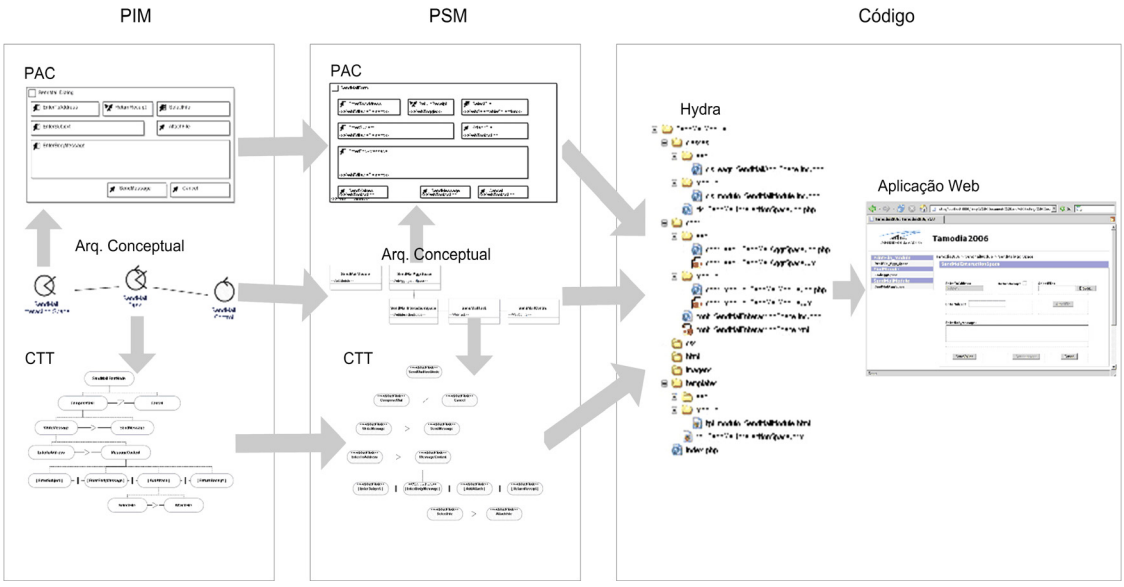

Figura VII-X – Resumo dos modelos para o caso de aplicação do método "Envio de Mail".

Com este caso prático de aplicação do método, demonstramos que os elementos do PSM são transformados em componentes de código, gerados, automaticamente, permitindo desta forma obter um protótipo concreto e executável para a interface com o utilizador. Este método permite

completar todos os passos necessários num processo de desenvolvimento de uma interface com o utilizador, desde a análise e desenho até à implementação do código numa plataforma tecnológica Web.

VIII. CONCLUSÕES

Este trabalho estende a abordagem Wisdom de desenho de interfaces com o utilizador, referida em [Nóbrega, Nunes, et al., 2005b], de modo a corresponder aos requisitos da especificação *Model Driven Architecture* (MDA). Esta solução integra modelação abstracta, para o desenho de interfaces com o utilizador, com a recomendação MDA, permitindo a geração automática de código a partir dos modelos de desenho.

A partir dos modelos de apresentação e de diálogo, que se complementam, utilizando Protótipos Abstractos Canónicos e *ConcurTaskTrees* em UML, respectivamente, foi proposto um modelo *Platform-Specific Model* (PSM) e um perfil UML (*UML profile*) de modo a detalhar os requisitos para a geração automática de interfaces Web. Com esta solução, que está de acordo com os requisitos da MDA e é sustentada na UML, podemos obter dos modelos de desenho um protótipo de interface, completamente funcional, permitindo simular numa fase inicial ou entregar uma interface final. O PSM e o perfil UML foram definidos para uma plataforma Web de implementação, tendo por base a *framework* Hydra, conduzindo à organização e rastreabilidade dos componentes de código, mantendo-se a relação com os elementos do nível de desenho do sistema. Foram ainda criadas, no âmbito deste trabalho, as aplicações *Model2Model* e *Model2Code* de modo a permitir as transformações automáticas entre modelos e de modelos para código.

Pensamos que a nossa proposta produz resultados satisfatórios no que diz respeito às interfaces Web geradas. Está de acordo com os requisitos definidos pelas tecnologias da OMG e suporta a prática de separação de conceitos, defendida pela área de interacção humano-computador. Acreditamos que a abordagem, descrita neste trabalho, prova a exequibilidade de ligar as áreas de engenharia de software e de interacção humano-computador sob a base comum da UML e da MDA.

Relativamente à aplicação prática da abordagem proposta neste trabalho, podemos referir que a *framework* Hydra e o método Wisdom estão a funcionar num ambiente real de desenvolvimento no Sector de Comunicações e Informática da Universidade da Madeira (SCI-UMa). A introdução de transformações, automáticas, entre os modelos Wisdom e a Hydra, produzindo interfaces funcionais foi apenas testado no âmbito deste trabalho. Outro aspecto, que ainda não está a ser utilizado em ambiente de produção, é a notação CTT e o respectivo algoritmo, implementados na *framework* no âmbito deste trabalho, para detalhar o comportamento da interface e dos seus elementos de interacção concretos. Esta notação é utilizada, actualmente, no SCI-UMa, apenas para descrever o comportamento da interacção a um nível mais alto de abstracção.

A aplicação da abordagem proposta, com a actual fase de implementação em que se encontra, será útil durante a criação de novos projectos ou novos módulos para projectos existentes, sendo mais eficiente e vantajosa do que a criação manual das classes, *templates* HTML e ficheiros de configuração XML. Por outro lado, uma limitação poderá ser, durante a manutenção e evolução das interfaces, não ser possível o *reverse-engineering*, ou seja, obter a partir do código o modelo PSM correspondente. Outro aspecto limitativo é que as interfaces finais, utilizando a abordagem e a *framework*, acabam por ficar com algumas restrições devido aos mecanismos e funcionalidades disponibilizadas pela *framework* Hydra. Com a *framework* foi definida uma estrutura base igual para todos os projectos, deixando de ser necessário ter de perceber o código desenvolvido por um outro elemento da equipa, apenas tem de ser entendida a estrutura e processo de execução da Hydra. Ao ganharmos essa uniformização da estrutura do código perdeu-se alguma flexibilidade e usabilidade, pois, embora possamos sempre estender as funcionalidades da *framework*, acabamos por ter certas restrições aos mecanismos disponibilizados pela Hydra. Esta abordagem, com o PSM e o perfil UML direccionados para a *framework* Hydra, acabam por ser também afectada com alguns problemas de flexibilidade e usabilidade das interfaces, mas, ganhando por outro lado, eficiência e facilidade de manutenção.

Resumindo as vantagens e desvantagens da abordagem proposta pensamos que no contexto de desenvolvimento com uma pequena equipa, num ambiente complexo e de alguma dimensão, para o qual a abordagem deste trabalho foi projectada, tem maiores vantagens relativamente à implementação manual. Possivelmente, noutro contexto mais específico, para o desenvolvimento de aplicações para um determinado fim mais controlado e específico, esta abordagem não terá mais valias, relativamente à implementação manual ou a outro método de geração de código.

Relativamente às maiores dificuldades na elaboração deste trabalho, podemos começar por referir a definição das transformações defendidas pela MDA, através de uma forma

normalizada. Foi analisada a especificação QVT [Tata Consultancy, 2003] [QVT-Merge Group, 2005], mas além de estar numa fase preliminar da especificação revelou-se complexo, quando as transformações que se pretendia implementar foram desenhadas para serem simples e quase directas. Outro ponto que, no início parecia ser simples, por todas as ferramentas de modelação o permitirem, era a definição da estrutura XMI dos modelos PIM e PSM a construir e a posterior leitura desse XMI pelas aplicações de transformação *Model2Model* e *Model2Code*. Antes de ser utilizado o editor MetaSketch, foram analisadas, de forma superficial, diversas ferramentas de modelação e a estrutura XMI que era gerada, o que se concluiu foi que nenhuma estrutura estava, totalmente, de acordo com a especificação XMI 2.1 [OMG, 2005c]. Ou seja, se fosse utilizada uma ferramenta para construir o PIM ou o PSM, tinha de ser construído um carregamento específico do XMI para as aplicações de transformação. A solução foi utilizar o MetaSketch para definir o metamodelo e criar os modelos, pois a sua estrutura XMI está de acordo com a especificação [OMG, 2005c]. Ainda outro aspecto, que apresentou dificuldades, foi a análise de outros projectos de aplicação da MDA a um processo de desenvolvimento. Foram analisadas de forma, superficial, diversas ferramentas [Costa, Teixeira, et al., 2004b] como o OptimalJ, AndroMDA, ArcStyler, etc., o que se verificou foi que cada ferramenta, tem a sua forma de implementar a recomendação e, em muitos aspectos, não são normalizados, implicando que temos de nos adaptar ao método proposto pela ferramenta MDA e não ao contrário.

Ao nível de trabalho futuro, existem vários aspectos que podem melhorar a abordagem. Em primeiro lugar, o modelo de apresentação, com a relação entre os espaços de interacção e os elementos de entrada e saída, deve ser mais explorado. Neste trabalho, foi dado mais ênfase ao detalhe de cada espaço de interacção, da sua estrutura e comportamento individual, não sendo tratadas as relações, por exemplo de navegação, existentes entre os espaços de interacção da arquitectura. Outro aspecto, importante, para que a abordagem possa ser utilizada num ambiente de produção, é a integração das aplicações de transformação *Model2Model* e *Model2Code* no editor MetaSketch, pois sem a integração será pouco eficiente a utilização do método de geração de código. Ao nível das regras de transformação, poderá ser útil pensar numa forma normalizada de as definir, por exemplo com a especificação QVT, se a complexidade das regras aumentar.

REFERÊNCIAS

Achour, M., B. Friedhelm, et al. (2006). PHP Manual. PHP Documentation Group from www.php.net

AndroMDA (2005). An Open Source MDA Generator. AndroMDA Project.

Brown, A. (2004). An Introduction to Model-Driven Architecture. IBM Rational developers Works.

Campos, P. and N. J. Nunes (2004). Canonsketch: A user-centered tool for canonical abstract prototyping. In Proceedings of DSV-IS'2004, 11th International Workshop on Design, Specification and Verification of Interactive Systems, Springer-Verlag.

Campos, P. and N. J. Nunes (2006). Tools of the Trade: The Practitioners' Tools and Workstyles. IEEE Software accepted for publication.

Carlson, D. (2001). Modeling XML Applications with UML: Practical e-Business Applications. Addison-Wesley Professional.

Castagnetto, J., H. Rawat, et al. (2000). Professional PHP Programming. Wrox Press.

Celic, B. (2005): What's New in UML 2.0. Rational Software.

Cernosek, G., Naiburg, E.(2004). The Value of Modeling. Rational Software.

Conallen , J. (2002). Building Web Applications with UML. Addison-Wesley Professional.

Constantine, L. (2003). Canonical Abstract Prototypes for abstract visual and interaction design. In: Jorge, J., Nunes, N. and Falcão e Cunha, J. (eds.): Proceedings of DSVIS' 2003, 10th International Conference on Design, Specification and Verification of Interactive Systems. Lecture Notes in Computer Science, Vol. 2844. Springer-Verlag, Berlin Heidelberg New York.

Constantine, L. and L. A. D. Lockwood (1999). Software for use: a practical guide to the models and methods of usage-centered design. Addison Wesley, Reading, Mass.

Costa, D., L. Nóbrega, et al.(2007). An MDA Approach for Generating Web Interfaces with UML ConcurTaskTrees and Canonical Abstract Prototypes. Proc. of 5th International Workshop on TAsk MOdels and DIAgrams for user interface design (TAMODIA'2006), Hasselt, Belgium, Springer.

Costa, D., Valente, P. (2004). Desenho Centrado no Utilizador – Sistema de Venda de Bilhetes. Trabalho no âmbito da Pós-Graduação em Engenharia de Software, Universidade da Madeira.

Costa, D., G. Teixeira, et al. (2004). Aplicações Centradas em Redes – Sistema de Gestão de Referências Bibliográficas. Trabalho no âmbito da Pós-Graduação em Engenharia de Software, Universidade da Madeira.

Costa, D., G. Teixeira, et al. (2004b). Métodos e Ferramentas de Desenvolvimento de Software – Arquitectura MDA: Análise de uma Ferramenta e a Implementação de um Caso de Estudo. Trabalho no âmbito da Pós-Graduação em Engenharia de Software, Universidade da Madeira.

Fuentes-Fernández, L. and A.Vallecillo-Moreno (2004). An Introduction to UML Profiles. The European Journal for the Informatics Professional, Vol. V, No. 1.

Hall, M. (2004). Jakarta Struts: An MVC Framework. Sun Microsystems Press

Grose, T., G. Doney, et al. (2002). Mastering XMI. John Wiley & Sons.

Mallia, T. (2005). MDA Reality/Implementation: OMG Model Driven Architecture in the Application Life Cycle. CIBER Inc.

MIA (2005). MIA-Generation – User Guide. Mia-Software.

Mori, G., F. Paternò, et al. (2004). Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. IEEE Transactions on Software Engineering, Vol. 30, No. 8, IEEE Press.

Nóbrega, L. (2007). O Editor MetaSketch: Construção reflexiva de linguagens de modelação. PhD Thesis, Universidade da Madeira, Portugal.

Nóbrega, L., N. J. Nunes, et al. (2005). Mapping ConcurTaskTrees into UML 2.0. 12th International Workshop on Design, Specification, and Verification of Interactive System (DSV-IS'2005).

Nóbrega, L., N. J. Nunes, et al. (2005b). DialogSketch: Dynamics of the Canonical Prototypes. 4th International Workshop on TAsk MOdels and DIAgrams for user interface design: For Work and Beyond (TAMODIA'2005), Gdansk, Poland, September 26-27, ACM Press.

Nóbrega, L., N. J. Nunes, et al. (2006). The Meta Sketch Editor, A reflexive modeling editor. In Proceedings of CADUI'2006.

Nunes, N. J. (2001). Object Modeling for User-Centered Development and User Interface Design: the Wisdom Approach. PhD Thesis, University of Madeira, Funchal, Portugal.

Nunes, N. J. (2003). What drives software development: issues integrating software engineering and human-computer interaction. In Proc. of Interact´2003 Conference.

Nunes, N. J. and J. Falcão e Cunha (2000). Towards a UML profile for interaction design: the Wisdom approach. In Proc. of UML´2000 Conference, Kent – UK, A. Evans (Ed.), Springer Verlag LNCS, New York (2000) 50-58.

OMG (2003). MDA Guide Version 1.0.1. Object Management Group/03-06-01.

OMG (2005a): Unified Modeling Language: Infrastructure version 2.0 (formal/05-07-05). Object Management Group.

OMG (2005b). UML 2.0 - UML Superstructure Specification, Version 2.0 (formal/05-07-04). Object Management Group.

OMG (2005c). XMI 2.1 - MOF 2.0 / XMI Mapping Specification, Version 2.1 (formal/05-09-01). Object Management Group.

Paternò, F. (1999). Model-Based Design and Evaluation of Interactive Applications. Springer Verlag.

PHP (2006). PHP Web Site from www.php.net

Puerta, A. R., and J. Eisenstein (1999). Towards a General Computational Framework for Model-Based InterfaceDevelopment Systems. In Proc. of the 4th International Conference on Intelligent User Interfaces. ACM Press, New York, NY, 171-178.

QVT-Merge Group (2005). Revised submission for MOF 2.0 Query/View/Transformation RFP. QVT-Merge Group, Object Management Group.

Schlossnagle, G. (2004). Advanced PHP Programming. Developer's Library.

Soley R. (2003). Model Driven Architecture: An Introduction. Object Management Group.

Soley R. and OMG (2000). Model Driven Architecture. Object Management Group, White Paper.

Tata Consultancy Services (2003). Revised submission for MOF 2.0 Query / Views / Transformations RFP. QVT-Partners.

Vanderdonckt, J. (2005). A MDA-Compliant Environment for Developing User Interfaces of Information Systems. Proc. of Conference on Advanced Information Systems Engineering (CAiSE'2005), Porto.

Vlist, E.(2002). XML Schema. O'Reilly & Associate