Beatriz Franco Martins Souza

Evolução do Método FrameWeb para o Projeto de Sistemas de Informação Web Utilizando uma Abordagem Dirigida a Modelos

Vitória, ES

2016

Beatriz Franco Martins Souza

Evolução do Método FrameWeb para o Projeto de Sistemas de Informação Web Utilizando uma Abordagem Dirigida a Modelos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Vítor E. Silva Souza

Vitória, ES

2016

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Setorial Tecnológica,
Universidade Federal do Espírito Santo, ES, Brasil)

Souza, Beatriz Franco Martins, 1973-

S729e Evolução do método FrameWeb para o Projeto de Sistemas de Informação Web utilizando uma abordagem dirigida a modelos / Beatriz Franco Martins Souza. – 2016.

151 f. : il.

Orientador: Vítor Estevão Silva Souza.

Dissertação (Mestrado em Informática) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Framework (Programa de computador). 2. Software – Desenvolvimento. 3. Sistemas de recuperação da informação. 4. Engenharia Web. 5. FrameWeb. 6. Desenvolvimento Orientado a Modelos. I. Souza, Vítor Estevão Silva. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título

CDU: 004

Evolução do Método FrameWeb para o Projeto de Sistemas de Informação Web Utilizando uma Abordagem Dirigida a Modelos

Beatriz Franco Martins Souza

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 18 de abril de 2016 por:

Prof. Dr. Vítor Estêvão Silva Souza - PPGI/UFES

Prof. Dr. João Paulo Andrade Almeida - PPGI/UFES

p/ Profª. Drª. Fernanda Lima - UNB/DF

O julgamento dessa dissertação foi realizado com a participação por meio de videoconferência do **membro externo** a Profª. Drª. Fernanda Lima seguindo as normas prescritas na portaria normativa no. 1/2016. Desse modo, as assinaturas do membro externo são representadas neste documento pelas respectivas assinaturas do presidente da comissão julgadora, o Prof. Dr. Vítor Estêvão Silva Souza.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória-ES, 18 de abril de 2016.

Ao meu filho, para que ele tenha em mim um exemplo de esforço, respeito e dedicação na busca do conhecimento e crescimento pessoal.

Agradecimentos

Chegar até aqui não foi uma tarefa fácil, tampouco simples, que começou quando ingressei no ensino pré-escolar, passou por todas as etapas de minha formação, incluindo também o processo seletivo deste Mestrado e as etapas de pesquisa necessárias para completá-lo. Neste processo Deus foi guia de minhas decisões e ações, a Ele agradeço acima de todos pois que sua vontade é maior.

Durante o Mestrado e até antes, meu grande e maior companheiro foi meu esposo, que muitas vezes demonstrou seu amor abdicando de si para que minhas atividades pudessem ser concluídas. Devo a ele, não só o compromisso de esposa assumido no passado, mas também boa parte do sucesso por ter atingido esta meta.

Me apoiaram nesta etapa da vida, meus pais, assim como fizeram desde que nasci. Espero, que a conclusão desta etapa de minha formação seja para eles orgulho, não de mim como sua filha, mas deles próprios por terem em mim pessoa de caráter e responsabilidade. A meus pais meus agradecimentos, devoção e homenagem.

Agradeço o apoio de todos os meus familiares diretos e por parte de meu esposo, bem como meus amigos, que souberam reconhecer que minha ausência em diversos momentos era para o cumprimento de uma etapa importante de minha formação. Obrigada a todos, que de perto ou de longe me apoiaram em algum momento ou de alguma forma.

Também e de forma especial agradeço a todos os meus professores que nesta estrada da busca pelo conhecimento me ensinaram, guiaram e orientaram.

"Do mesmo modo que o campo, por mais fértil que seja,
sem cultivo não pode dar frutos, assim é o espírito sem estudo."
(Cícero)

Resumo

Em um contexto de mercado cada vez mais competitivo, o desenvolvimento de Sistemas de Informação baseados na Web (WIS - *Web Information Systems*) necessita ser cada vez mais dinâmico e eficiente. A Engenharia Web (WebE - *Web Engineering*) ao longo dos anos vem propondo diversos métodos para análise, projeto e desenvolvimento de WIS. Em 2007 foi proposto o método de projeto WIS ***FrameWeb***, que define uma arquitetura básica para desenvolvimento de WIS baseado em *frameworks*. A proposta de arquitetura do método ***FrameWeb*** teve foco em um conjunto específico de *frameworks* disponíveis à época, definindo alguns procedimentos e um perfil UML para criar uma infraestrutura próxima ao estado-da-prática, cuja função é proporcionar ganho de produtividade, principalmente na transição entre as fases de projeto e de desenvolvimento, aproveitando as vantagens do uso de *frameworks* e de sua ampla utilização, bem como o amplo conhecimento de UML por parte dos modeladores e desenvolvedores. Entretanto, com o surgimento de novas técnicas, especificações e implementações de *frameworks* se fez necessário revisitar o método e avaliar as evoluções necessárias para sua continuidade. Além disso, a partir de sua aplicação, observou-se que de ferramentas CASE baseadas em UML deixam a cargo do modelador e de sua experiência a responsabilidade quanto às atividades a serem desempenhadas e ao uso dos construtos propostos, dando margem a falhas. Observando estas questões, este trabalho propõe a redefinição e formalização da linguagem do método ***FrameWeb*** por meio de um metamodelo, denominado ***FW-15***, por meio do uso das técnicas de Desenvolvimento Dirigido a Modelos (MDD - *Model-Driven Development*). O metamodelo ***FW-15*** tem a finalidade de garantir não apenas a semântica da linguagem ***FrameWeb*** no que diz respeito ao seu aspecto independente de *framework*, mas também permitir a evolução do método, pois define uma metodologia para a criação de **Definições de Frameworks**, contendo os construtos e regras necessárias para a criação e aplicação de **Perfis FrameWeb** responsáveis pelos aspectos dependente de *framework*. Assim, esta proposta pretende manter total compatibilidade com a versão original do método e ao mesmo tempo adicionar os recursos necessários para que novas técnicas e *frameworks* possam ser agregados formalmente ao método. Como contribuição adicional este trabalho apresenta um protótipo de ferramenta simplificada para modelagem com o método ***FrameWeb*** e determina as diretrizes para a evolução deste protótipo de forma a permitir o desenvolvimento de uma ferramenta gráfica apropriada ao método.

Palavras-chaves: Engenharia Web. FrameWeb. Frameworks. Desenvolvimento Orientado a Modelos.

Abstract

In a context of increasingly competitive market, the development of Web-based Information Systems (WIS) needs to be more and more dynamic and efficient. Over the years, various Web Engineering (WebE) methods for analysis, design and development of WIS have been proposed. In 2007 the ***FrameWeb*** design method was proposed, which defines a basic architecture for the development of WIS based on frameworks. The ***FrameWeb*** proposed architecture had focused on a specific set of frameworks available at that time, setting some procedures and a UML profile to create a infrastructure close to the state-of-practice, with the aim of providing productivity gains, especially in the transition between design and development phases, taking advantage of these frameworks, as well as extensive knowledge of UML by modelers and developers. However, from the rise of new techniques, specifications and frameworks implementations it was necessary to revisit the method and evaluate the necessary changes to its continuity. Moreover, from its application, it was noted that the UML-based CASE tools leave with the modeler and his experience the responsibility for the activities to be performed and the correct use of the proposed constructs, giving rise to failures. Observing these issues, this work proposes the redefinition and formalization of the ***FrameWeb*** method language through a metamodel called ***FW-15*** lead by the use of Model-Driven Development (MDD) techniques. The metamodel ***FW-15*** aims to ensure not only the semantics of ***FrameWeb*** language with regard to their framework-independent aspect, but also enable the evolution of the method, as it defines a methodology for creating **Frameworks Settings** containing the constructs and rules necessary for the creation and application of **FrameWeb profiles** responsible for its framework-dependent aspects. Thus, this proposal aims to maintain full compatibility with the original method version and at the same time add the necessary resources to new techniques and frameworks that can now be formally added to the method. As an additional contribution, this work presents a simplified prototype tool for designing ***FrameWeb*** diagrams and determines the guidelines to evolve this prototype, in order to allow the development of a suitable CASE tool for the method.

Keywords: Web Engineering, FrameWeb. Frameworks. Model-Driven Development

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AJAX	Asynchronous Javascript and XML
CASE	Computer-Aided Software Engineering
CDI	Contexts and Dependency Injection
CSS	Cascading Style Sheets
CWM	Common Warehouse Metamodel
DI	Dependency Injection
DAO	Data Access Object
DOM	Document Object Model
DSL	Domain Specific Language
DSML	Domain Specific Modeling Language
EMF	Eclipse Modeling Framework
GWT	Google Web Toolkit
HQL	Hibernate Query Language
HTML	HyperText Markup Language
HTML4	HyperText Markup Language 4
HTML5	HyperText Markup Language 5
Java EE	Java Enterprise Edition
JEE	Java Enterprise Edition
JET	Java Emitter Template
JPA	Java Persistence API
JPQL	JPA Query Language
JSF	JavaServer Faces
JSP	JavaServer Pages

JSTL	JSP Standard Tag Library
MDD	Model-Driven Development
MDWE	Model-Driven Web-Engineering
MOF	Meta Object Facility
OCL	Object Constraint Language
ORM	Object/Relational Mapping
PDM	Platform Dependent Model
PIM	Platform Independent Model
QVT	Query/Views/Transformations
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UWE	UML-based Web Engineering
WebE	Web Engineering
WIS	Web Information System
XHTML	eXtensible Hypertext Markup Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSTL	eXtensible Stylesheet Language for Transformation

Sumário

1 Introdução

Este capítulo apresenta as motivações para o desenvolvimento da pesquisa apresentada neste trabalho e expõe as contribuições alcançadas, finalizando com a organização e estrutura desta dissertação.

1.1 Contextualização

Desde o nascimento da Engenharia Web (WebE), muitos métodos que suportam a construção de aplicações para a plataforma *web* foram propostos. Alguns são muito conhecidos como por exemplo OOWS (*Object Oriented Web Solution*) proposta por Embley, Liddle e Pastor (2011), UWE (*UML-based Web Engineering*) proposto por Koch, Kraus e Hennicker (2001), MODFM (Mockup-Driven Fast-Prototyping Methodology) de Zhang e Chung (2003a), Zhang e Chung (2003b), OOHDM (*Object Oriented Hypermedia Design Method*) proposto por Schwabe e Rossi (1998), dentre outros. O ***FrameWeb*** (SOUZA, 2007; SOUZA; FALBO; GUIZZARDI, 2009), proposto em 2007, é um destes métodos e mira no desenvolvimento de Sistemas de Informação Web (*Web Information Systems* - WIS).

Devido à popularidade dos *frameworks* no desenvolvimento *web*, o método ***FrameWeb*** propõe a utilização de modelos específicos direcionados à arquitetura baseada em *containers*, que durante a fase de projeto auxiliam o modelador na tarefa de lidar com a complexidade por trás da aplicação Sistemas de Informação Web.

Porém, não só novos *frameworks*, tecnologias e plataformas surgiram, como continuarão surgindo e evoluindo. Por isso, é importante que os métodos também evoluam não só no que diz respeito a novas versões, mas também para permitir que as novidades possam ser incorporadas de forma simples e efetiva. Este trabalho pretende lidar com estas questões no contexto no método ***FrameWeb***.

1.2 Motivação

Originalmente, o ***FrameWeb*** apropriou-se das características e benefícios da UML, para desenvolver um perfil capaz de facilitar a criação de modelos para refletir os construtos da plataforma escolhida. Em especial, se beneficiou do fato da UML ser amplamente conhecida tanto pelos modeladores quanto por parte dos desenvolvedores.

Entretanto, como até o presente momento este método vem sendo utilizado com o apoio de ferramentas para modelagem UML de uso geral e além disso novas tecnologias não

cobertas na sua definição inicial vem sendo incorporadas aos projetos, alguns inconvenientes tem sido considerados:

1. Os modelos são específicos para a plataforma escolhida, significando que não se adequam a diferentes instâncias de *frameworks* daquelas escolhidas na versão original;

2. O perfil UML proposto não é rigoroso, ou seja, não previne os modeladores de incluírem elementos que não pertencem ao modelo;

3. Apesar das ferramentas de modelagem UML de modo geral serem bastante versáteis e permitem a criação dos modelos ***FrameWeb*** adequadamente, não facilitam o trabalho do modelador, pois exigem que elementos específicos do método sejam adicionados manualmente. Além disso, não executam nenhum tipo de verificação quanto às restrições impostas pelo método;

4. A abordagem não possui nenhuma ferramenta específica de apoio para que os desenvolvedores possam criar modelos válidos, gerar código, etc.

No primeiro caso, não só os *frameworks* envolvidos na definição inicial do método foram atualizados, como novos *frameworks* com diferentes abordagens foram desenvolvidos. Além disso, nos dois casos novas tecnologias como o `AJAX` (*Asynchronous JavaScript and XML*)[1], por exemplo, não só foram incorporadas mas são amplamente utilizadas (estado-da-prática).

No que diz respeito à adição de novas tecnologias a projetos ***FrameWeb***, os modeladores acabam por fazer adaptações e adições aos modelos do método de forma *ad-hoc*, baseadas em seu conhecimento tácito destas tecnologias, já que estas ainda não cobertas pelo método. Neste contexto, estas adaptações não são rigorosas, mas por outro apresentam-se como uma boa base para a pesquisa das soluções propostas neste trabalho.

Na sequência é importante considerar que apesar das ferramentas para modelagem UML de uso geral em sua maioria permitirem a adição de perfis, o desenvolvimento destes não é trivial, especialmente porque deve ser feito especifica e individualmente para cada ferramenta. Isso agrava-se, na proporção em que cada uma destas ferramentas apresenta recursos e limitações diferentes no que diz respeito a este tipo de artefato.

E, finalmente, ainda que todos os princípios definidos pelo método fossem seguidos, não há garantias de uma gramática comum dentre todas as propostas. Por causa disso, a formalização de uma linguagem para o método é de suma importância, de forma a fornecer os subsídios necessários para a definição de perfis ou criação de ferramentas específicas para atender ao método.

[1] <http://www.w3schools.com/ajax/default.asp>

Os métodos WebE tais como ***FrameWeb*** devem empenhar-se em promover um desenvolvimento rápido e produtivo, usando técnicas de modelagem e ferramentas para suporte. Neste contexto, o Desenvolvimento Dirigido a Modelos (*Model-Driven Development* - MDD) (EMBLEY; LIDDLE; PASTOR, 2011) pode proporcionar uma forma de aumentar a produtividade durante o desenvolvimento da aplicação, já que atua diretamente na definição de DSLs e na modelagem das funcionalidades de aplicações produzindo um resultado final por meio de transforações sucessivas aplicadas em diferentes níveis de abstração.

1.3 Contribuições

A partir da avaliação e estudo do método ***FrameWeb*** e de projetos desenvolvidos com o mesmo, propomos as contribuições deste trabalho:

(i) Formalizar a linguagem para o método ***FrameWeb*** e, assim, permitir o desenvolvimento de ferramentas que auxiliem desenvolvedores no uso do método;

(ii) Atualizar a proposta original do método de forma a permitir o uso de diversos tipos e combinações de *frameworks*, deixando a critério do modelador a escolha das combinações mais apropriadas de acordo com o projeto do seu Sistema de Informação Web;

(iii) Definir uma metodologia para a adição de novos *frameworks* e tecnologias ao método, dando-lhe a capacidade de evoluir ao longo do tempo na medida em que novas abordagens tecnológicas forem sendo desenvolvidas;

(iv) Desenvolver uma ferramenta simples, capaz de validar projetos desenvolvidos com o método e dar os subsídios necessários para uma posterior criação de uma ferramenta gráfica mais elaborada.

1.4 Abordagem

O método ***FrameWeb*** foi formulado baseado em três pilares:

(i) O uso de *frameworks* ou arquitetura baseada em *containers*;

(ii) A não existência de métodos focados nos *frameworks* realmente utilizados na construção de Sistemas de Informação Web, dentro da WebE;

(iii) O uso de métodos que se adéquam diretamente à arquitetura de software adotada promove uma agilidade maior ao processo, característica que é bastante desejada nos projeto Web (PRESSMAN, 2005).

O método assume que dentro do Projeto de um Sistemas de Informação Web, são usados certos *frameworks* para a sua construção, visto que o uso destes já está consagrado no estado-da-prática, dentro do desenvolvimento de tais aplicações.

Na definição original do método ***FrameWeb*** foram adotadas certas categorias de *frameworks* para sua infraestrutura, como controladores frontais, mapeamento objeto/relacional, injeção de dependência. Nestas categorias, os *frameworks* utilizados foram o `Struts`[2], `Hibernate` e `Spring Framework` respectivamente.

Neste cenário e considerando os aspectos motivacionais apresentados na Seção 1.2, a abordagem adotada neste trabalho visa não apenas atualizar o método adicionando outros *frameworks* e formalizar sua linguagem, mas definir uma infraestrutura para permitir sua evolução continuada.

Esta abordagem requer o cumprimento de algumas etapas importantes. Primeiramente é necessário avaliar os métodos existentes e suas diferentes abordagens de forma a verificar as semelhanças e diferenças em comparação com o método ***FrameWeb***. Posteriormente o estudo se direciona às diferentes categorias, *frameworks* e tecnologias a contemplarem o escopo desta abordagem.

A partir dos estudos e do escopo definido, a linguagem pode ser definida e por meio das técnicas MDD o metamodelo da linguagem é criado. Na criação do metamodelo, é importante definir as partes da linguagem que permitirão extensão a fim de proporcionar a adição e uso de diferentes combinações de *frameworks* ao projeto sempre que forem necessárias.

Fica claro que, na medida em que o metamodelo permitirá extensões, um novo papel deve ser considerado como participante no processo de construção de Sistemas de Informação Web com o método ***FrameWeb***. O papel desempenhado pelo indivíduo responsável pela formalização das definições de *frameworks* para a linguagem.

Este e os outros papeis desempenhados pelas partes interessadas no desenvolvimento de Sistemas de Informação Web com o método ***FrameWeb***, juntamente com o processo para a definição de *frameworks* participantes da linguagem faz parte da atualização do método e é responsável por proporcionar parte dos recursos para a evolução continuada do método.

A outra parte da evolução continuada que se espera para o método, está na sua capacidade de adequação quanto a adição de novas categorias de *frameworks*. Neste caso o processo de evolução do método depende da extensibilidade do metamodelo, sobre o qual novas possíveis categorias devem ser adicionadas em conformidade como as já existentes.

Evidentemente neste caso as modificações são mais profundas e requerem o uso das técnicas de MDD para sua correta aplicação. A metodologia usada no desenvolvimento do metamodelo da linguagem ***FrameWeb*** é, então, o alicerce para a adição de novas

categorias de *frameworks* a este metamodelo, fornecendo-lhe a extensibilidade desejada e necessária.

Por fim, o trabalho é avaliado por meio de provas de conceito a partir de projetos efetuados por alunos de graduação com o método ***FrameWeb***, aplicando-se os modelos por eles produzidos no protótipo de ferramenta produzido neste trabalho.

1.5 Organização

Esta dissertação é composta por seis capítulos. No Capítulo 2 é apresentada uma fundamentação teórica para introduzir os principais conceitos que são utilizados neste trabalho e servem como base para o seu entendimento. O Capítulo 3 detalha o metamodelo proposto, incluindo tanto os aspectos independente de *framework* quanto dependente de *framework*. O Capítulo 4 perpassa pelas principais questões da tomada de decisão quanto à abordagem adotada. O Capítulo 5 apresenta a implementação do metamodelo criado, o protótipo de uma ferramenta simples para edição de modelos ***FrameWeb***, incluindo sua avaliação por meio de modelos testados, além das diretrizes para o desenvolvimento de uma ferramenta gráfica mais completa para o método. Por fim, no Capítulo 6 apresenta-se a conclusão e os trabalhos a serem realizados no futuro.

2 Revisão da Literatura

Neste capítulo, apresentamos os resultados da revisão da literatura feita ao longo do curso deste trabalho. A Seção 2.1 introduz o método ***FrameWeb***, utilizado como base para nossas propostas. A Seção 2.2 apresenta os conceitos básicos do Desenvolvimento Dirigido a Modelos (MDD). Finalmente, a Seção 2.3 descreve os trabalhos relacionados à nossa proposta, e.g., métodos que fazem uso de MDD para o desenvolvimento de Sistemas de Informação Web.

2.1 Introdução ao método FrameWeb

Ao longo dos últimos anos surgiram novas tecnologias no processo de software, em especial na Engenharia Web. O ***FrameWeb*** (SOUZA, 2007; SOUZA; FALBO, 2007; SOUZA; FALBO; GUIZZARDI, 2009) é um método direcionado à fase de projeto e voltado para aplicações que fazem uso de *frameworks*.

A arquitetura proposta para o método ***FrameWeb*** foi baseada no padrão *Service Layer* (Camada de Serviço) (FOWLER, 2002), como mostra a Figura 1.

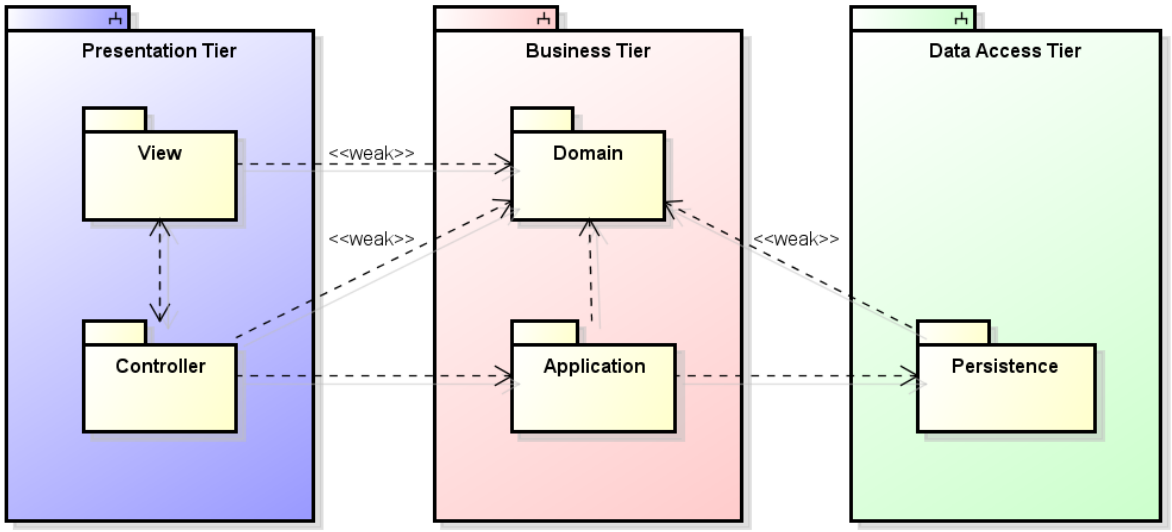


Figura 1 – Proposta de Arquitetura ***FrameWeb*** (SOUZA, 2007; SOUZA; FALBO; GUIZZARDI, 2009).

É possível observar que na arquitetura proposta há três camadas:

- A **Camada de Negócio** (**Business Tier**), responsável pelas funcionalidades relacionadas às regras de negócio da aplicação;

- A **Camada da Apresentação** (**Presentation Tier**), onde encontram-se basicamente as funcionalidades de interface com o usuário;
- E a **Camada de Acesso a Dados** (**Data Access Tier**), onde estão localizadas as funcionalidades para persistência.

Dentro desta arquitetura, o acesso feito pela camada de apresentação às funcionalidades do sistema segue o padrão arquitetural MVC (GAMMA et al., 1994). Os pacotes a serem definidos nos modelos propostos pelo método para o projeto de um Sistemas de Informação Web, seguindo o padrão MVC, se dividem em:

Model: Domínio (**Domain**) onde estão definidos os elementos do domínio do problema no âmbito do Sistema de Informação Web, a **Aplicação** (**Application**), onde se faz toda a parte dos casos de uso, por fim, **Persistência** (**Persistence**) responsável pelo armazenamento e recuperação dos dados;

View: Visão (**View**) onde estão os elementos de comunicação e estímulo ao usuário;

Controller: Controle (**Controller**) que é responsável por executar as requisições e fornecer respostas à visão.

É importante deixar claro que o método ***FrameWeb*** está focado na fase de projeto de Sistemas de Informação Web, portanto quando o autor se refere ao conceito de *domínio* ou *modelo de domínio*, está tratando do domínio da aplicação após a fase de Análise de Requisitos. Neste sentido o domínio em questão não está relacionado a outros aspectos que não os do Sistema de Informação Web modelado.

O método ***FrameWeb*** define uma linguagem de modelagem baseada em extensões leves (lightweight extensions) ao metamodelo da UML[1], com a finalidade de criar um perfil UML para ser utilizado pelo método.

A linguagem ***FrameWeb*** está alicerçada em quatro tipos de Diagramas de Classe UML, que são utilizados para representar os componentes típicos da plataforma *web* e dos *frameworks* em questão, são eles:

- **Modelo de Domínio** (**Domain Model**), referente ao pacote de Domínio da aplicação, para representar o domínio do problema da aplicação *web* e o mapeamento desses objetos para a persistência;
- **Modelo de Persistência** (**Persistence Model**), para representar os objetos persistentes utilizando o padrão de projeto DAO (ALUR; MALKS; CRUPI, 2013), referente aos elementos contidos no pacote de Persistência;

[1] <https://eclipse.org/modeling/mdt/downloads/?project=uml2>

- **Modelo de Navegação** (**Navigation Model**), para representar os diversos componentes contidos no pacote de Visão e Controle, que formam a interface com o usuário, a partir da qual é possível interagir com a lógica de negócio;

- **Modelo de Aplicação** (**Application Model**), para representar as estruturas de serviço pertencentes ao pacote de Aplicação, que são responsáveis pela implementação dos casos de uso, bem como as dependências de/para componentes de outras camadas (Controle e Persistência).

Esta abordagem permite a definição de uma linguagem e sua formalização, para posteriormente executar diversas outras atividades importantes, como, por exemplo: o desenvolvimento de ferramentas de modelagem para o método, a geração automática de código, a adição de funcionalidades para verificação dos modelos produzidos e muitas outras.

Este trabalho propõe expandir o método ***FrameWeb*** de um perfil UML leve para uma extensão UML completa, definindo uma linguagem para o método, adequando-o no sentido de não apenas fazer-lhe uma atualização, mas em facilitar suas futuras atualizações no contexto das tecnologias de projeto futuras e baseadas no padrão MVC. Este processo foi separado em várias etapas de estudo e desenvolvimento, que são abordadas ao longo do texto.

2.2 Desenvolvimento Dirigido a Modelos

O Desenvolvimento Dirigido a Modelos (*Model-Driven Development* - MDD) (EMBLEY; LIDDLE; PASTOR, 2011), preconiza que modeladores concentrem seus esforços na construção de modelos conceituais para representação de todas as características do sistema.

Desta forma, MDD propõe uma abordagem nova em comparação com a visão tradicional, na qual modelos conceituais são usados apenas para descrever as características de um software com base na elicitação de requisitos e que, posteriormente, serão usados na construção desse software utilizando linguagens de programação, separadamente dos artefatos do projeto.

Em MDD, por outro lado, os modelos conceituais são usados para “diagramar” ou modelar as funcionalidades e recursos desejados do sistema e, posteriormente, por meio de transformações entre modelos dos diversos níveis de abstração obtém-se o software final.

As principais características de MDD são:

- O modelo **é** o projeto do sistema;
- O modelo é ampliado, evolui e atualiza-se;
- Há um fluxo contínuo entre as transformações efetuadas nos modelos;
- A implementação é diretamente derivada do modelo.

Segundo (SELIC, 2003), MDD propunha ser o primeiro e verdadeiro salto geracional em desenvolvimento de software desde a introdução do o compilador. A chave está na resolução de questões pragmáticas relacionadas com os artefatos e a cultura das gerações anteriores de tecnologias de software.

Diversos autores apresentaram as vantagens do MDD (KLEPPE; WARMER; BAST, 2003; VAN DEURSEN; KLINT, 1998; TRASK; ROMAN, 2006; MERNIK; HEERING; SLOANE, 2005). A seguir estão apresentadas as mais relevantes para o processo de Sistemas de Informação Web:

Produtividade: porque ocorre um aproveitamento melhor do tempo, já que o tempo é gasto na produção dos modelos de alto nível, deixando as tarefas repetidas serem implementadas por meio de transformações;

Portabilidade: os modelos de alto nível podem ser transformados em códigos diferentes dependendo da plataforma escolhida;

Interoperabilidade: as partes dos modelos podem ser transformados para uso em plataformas diferentes, aproveitando o melhor de cada uma e definindo um ambiente heterogêneo, porém sua funcionalidade global é mantida;

Manutenção: como os modelos são parte do software as alterações são realizadas diretamente neles, por isso o código mantém-se consistente facilitando as tarefas de manutenção;

Comunicação: as partes interessadas tendem a comunicar-se melhor mesmo tendo visões diferentes do domínio em questão, já que os modelos geralmente são mais abstratos do que códigos e mais precisos do que a linguagem textual;

Otimização: os modelos fornecem recursos que permitem aos geradores prover implementações mais eficientes e com menor incidência de erros;

Corretude: geradores de código não produzem erros acidentais e além disso permitem que a identificação de erros conceituais seja facilitada, porque ocorrem em um nível de abstração mais alto.

Alguns autores citam desvantagens no uso de MDD (THOMAS, 2004; AMBLER, 2003), as principais são:

Rigidez: como a maior parte da geração de código não sofre atuação por parte dos desenvolvedores, os softwares produzidos são mais rígidos;

Complexidade: o uso de ferramentas de modelagem, transformação e geradores de código aumentam a complexidade do processo, porque em si já são artefatos de maior complexidade tanto para construção quanto para manutenção;

Desempenho: mesmo com o uso de otimizações em alto nível de abstração, os geradores podem produzir bastante código desnecessário, que eventualmente pode apresentar desempenho inferior à codificação manual;

Aprendizado: as técnicas MDD requerem profissionais com habilidades na construção de linguagens e com experiência nas ferramentas utilizadas, o que requer treinamento dedicado e maior tempo de aprendizado;

Investimento: é necessário pessoal especializado e um maior custo inicial para suprir a necessidade de uma infraestrutura própria, mas os ganhos a longo prazo são significativos.

De modo geral os autores concordam que as técnicas Desenvolvimento Dirigido a Modelos representam um avanço e no geral são mais vantajosas do que desvantajosas. Além disso há consenso de que a maioria das desvantagens pode ser superada no decorrer dos projetos e ao longo do tempo em termos de pesquisa.

Há, entretanto, alguns outros conceitos muito importantes que devem estudados para lidar com os diversos aspectos técnicos de MDD, pois a qualidade dos artefatos obtidos com estas técnicas estão diretamente relacionadas ao seu entendimento e aplicação. Veremos estes conceitos na sequência desta seção.

2.2.1 Termos relacionados

Há diversos termos derivados do paradigma MDD, podendo haver diferenças significativas dentro do paradigma MDD quando são adotados, assim devem ser reconhecidos e clarificados.

Em 2000, a OMG[2], inspirada na importância dos modelos em processos de desenvolvimento de software e na sua missão de prover soluções para o problema da interoperabilidade entre os sistemas, criou o *framework* MDA (Model- Driven Architecture)[3], totalmente dirigido a modelos.

[2] <http://www.omg.org>
[3] <http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf>

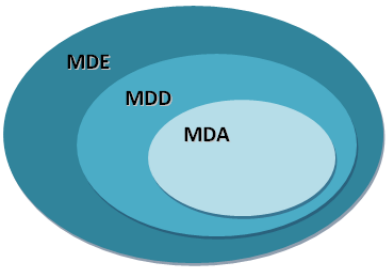


Figura 2 – Terminologia dirigida a modelos (AMELLER, 2009).

O MDA utiliza modelos formais, ou seja, que possam ser entendidos e interpretados por computadores. Este *framework* define e especifica alguns modelos, como eles devem ser usados e o relacionamento entre eles durante todas as fases do ciclo de vida de um sistema. Segundo (MELLOR, 2004), o MDA se baseia em 3 padrões criados pela OMG: UML (*Unified Modeling Language*[4]), MOF (*Model Object Facility*[5]) e CWM (*Common Warehouse Metamodel*[6]).

O paradigma MDD tem sua principal motivação no aumento da produtividade no desenvolvimento de sistemas, entretanto esta questão depende muito de diversos fatores que podem influenciar o uso das técnicas de MDD.

O uso das técnicas MDD para projeto e desenvolvimento de sistemas deve levar em consideração aspectos não só relacionados à fase de projeto, desenvolvimento e manutenção, mas a todas as fases de projeto. Não basta apenas usar MDD para obter garantia do resultado desejado, é necessário aplicar as melhores técnicas e práticas de Engenharia de Software em conjunto com MDD.

Por exemplo, a modelagem deve ser versionada corretamente de forma a manter as informações adequadas à evolução. Assim como nas de técnicas tradicionais de projeto, em MDD, rastreabilidade e documentação são de suma importância para o sucesso do projeto. Ou seja, acreditar simplesmente que pelo fato do modelo ser o projeto há garantia de documentação e rastreabilidade é um engano, que pode gerar grandes problemas.

A Engenharia Orientada a Modelos (*Model-Driven Engineering* - MDE) (SCHMIDT, 2006), lida com todo o contexto do paradigma MDD, indo além das atividades de desenvolvimento. Inclui também todos os aspectos relacionados ao processo de Engenharia de Software no contexto orientado a modelos.

MDD, MDA e MDE relacionam-se conforme a Figura 2. Tem-se então que o MDA, representa a visão particular de MDD preconizada pela OMG, sendo portanto seu subconjunto. E o MDE representa toda a estrutura que trata das questões relacionadas ao paradigma MDD, neste caso MDD é um subconjunto do MDE.

[4] <http://www.omg.org/spec/UML/2.5/PDF/>
[5] <http://www.omg.org/mof/>
[6] <http://www.omg.org/spec/CWM/1.1/PDF/>

2.2.2 Modelos, Metamodelos e Linguagens

De acordo com o guia MDA[7] da OMG, um modelo de sistema é uma descrição ou especificação de sua funcionalidade. Um modelo é geralmente composto por uma combinação de diagramas e textos. Estes podem ser escritos em uma linguagem de modelagem gráfica ou textual.

Os modelos de projeto de um Sistemas de Informação Web definem quais elementos podem existir nesse sistema, e uma linguagem gráfica pode ser usada para definir quais elementos podem existir nestes modelos. Utilizando este mesmo princípio, esta linguagem (abstrata) também pode ser descrita por um modelo. Assim, o modelo de uma linguagem descreve os elementos que podem ser usados, e a esta modelo dá-se o nome de metamodelo.

Como um metamodelo deve ser equivalente à linguagem definida por ele, é desnecessário fazer uma distinção entre o metamodelo e a sua respectiva linguagem, assim como da linguagem definida por ele (KLEPPE; WARMER; BAST, 2003). Baseado neste conceito, fica claro que para formalizar uma linguagem se faz necessário definir seu metamodelo (sintaxe abstrata), bem como sua semântica. O que, consequentemente, permite que também se crie um mecanismo para que esta linguagem possa ser utilizada de forma correta.

No nível do paradigma do MDD, pode-se trabalhar com linguagens para criação de modelos ou metamodelos, execução de transformações e geração de código. Porém as linguagens podem e devem ser usadas para formalizar domínios. Nesse sentido o MDD pode ser usado na prática para o desenvolvimento de Linguagens Específicas de Domínio (*Domain Specific Languages* - DSL), mas para isso é necessário que atenda alguns requisitos, devendo possuir:

- Uma linguagem (sintaxe) abstrata;
- Uma linguagem (sintaxe) concreta; e
- Uma semântica.

Uma sintaxe abstrata é na verdade um metamodelo, que descreve os conceitos e as regras de modelagem de uma linguagem. Neste contexto, a sintaxe abstrata desta linguagem passa a ser representada neste metamodelo.

A partir da definição de um metamodelo pode-se criar e verificar um modelo, uma instância do metamodelo, capaz de expressar certa conceituação pelo uso da notação gráfica dessa linguagem, uma sintaxe concreta.

[7] <http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf>

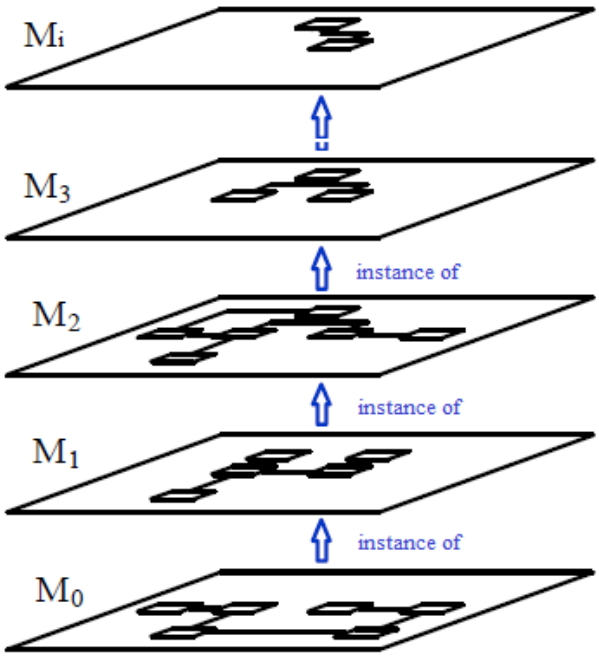


Figura 3 – Níveis de instanciação propostos em MDD.

No entanto, em si, uma sintaxe formal não é suficiente como definição de uma linguagem, por isso é necessário que se forneça significado para cada elemento da linguagem, uma semântica. Além de tudo isso, se faz necessária a construção de uma ferramenta CASE, em geral a partir de um framework de modelagem, para que a definição de uma DSL esteja completa.

O processo de metamodelagem está separado por níveis de abstração que determinam uma hierarquia de instanciações, conforme será explicado a seguir.

2.2.3 Níveis de Instanciação

A infraestrutura MDD consiste em uma hierarquia composta de modelos em níveis, os quais representam instâncias dos níveis imediatamente acima.

A Figura 3 mostra esta hierarquia, que é composta de níveis denominados "M". O nível mais baixo é o nível zero (M_0), o qual contém os elementos instanciados do nível imediatamente acima, M_1, este por sua vez representa os elementos instanciados do nível acima, M_2, e assim sucessivamente até o nível de instanciação mais alto, que normalmente corresponde ao nível M_3. Porém, são infinitas as possibilidades de maior nível, o que, na Figura 3, está representado pelo nível M_i para o mais alto possível.

A definição dos níveis de instanciação a serem tratados é de suma importância, devendo haver clareza em sua representação bem como continuidade entre seu fluxo. Porque o processo de instanciação deve ocorrer na medida certa, ou seja; se houverem poucos

níveis de instanciação há risco da perda de informações e, por outro lado, muitos níveis de instanciação causam prejuízo de clareza e bem como de performance.

Entretanto, também é fundamental que cada nível de instanciação seja representado por artefatos bem definidos. Por exemplo, pode-se definir uma representação de níveis para uma versão da linguagem gráfica usada no mapeamento de linhas de metrô, muito comum em diversas cidades mundo afora.

Desta forma, os níveis de instanciação para este exemplo apresentam-se conforme a seguir:

$\mathtt{M}_0$ Nível de elementos (instancias em tempo de execução): é o conteúdo da representação em si, os próprios dados do mundo real que se deseja representar, por exemplo o metrô da cidade de São Paulo propriamente dito. O que, na realidade, pode não representar exatamente instâncias diretas do nível de modelo, mas o resultado do que foi por ele definido;

$\mathtt{M}_1$ Nível de modelo (modelo do usuário): os diversos possíveis diagramas a serem definidos e criados (sintaxe concreta) pelo modelador (usuário), que são instâncias produzidas sob o nível de metamodelo. No exemplo exposto pode representar o diagrama do metrô da cidade de São Paulo[8];

$\mathtt{M}_2$ Nível de metamodelo (especificação da linguagem): a especificação da linguagem de modelagem para os diagramas (sintaxe abstrata), que se deseja definir (formalizar), a qual, no exemplo fornecido, representa a gramática da linguagem de modelagem definida para expressar os modelos dos diagramas do metrô de qualquer cidade;

$\mathtt{M}_3$ Nível de metametamodelo (linguagem para especificação de linguagens): representa o nível de meta objetos (*Meta Object Facility* - MOF), o qual representa uma linguagem para especificação de linguagens.

A Figura 4 mostra a definição de uma DSL para diagramas de linhas de metrô, dando continuidade ao exemplo acima exposto. Na figura, é possível observar que os meta-elementos do MOF (`Association` e `Class`) possuem uma representação gráfica no metamodelo (são instanciados em $\mathtt{M}_2$). Da mesma forma os elementos do metamodelo da DSL (`Line`, `Route` e `Station`), desenvolvida para os diagramas de metrô, possuem uma representação gráfica no modelo do usuário (são instanciados em $\mathtt{M}_1$). Por fim, o modelo do usuário (*User Model*) pode ser usado para representar as linhas de metrô de São Paulo.

[8] <http://www.metro.sp.gov.br/pdf/mapa-da-rede-metro.pdf>

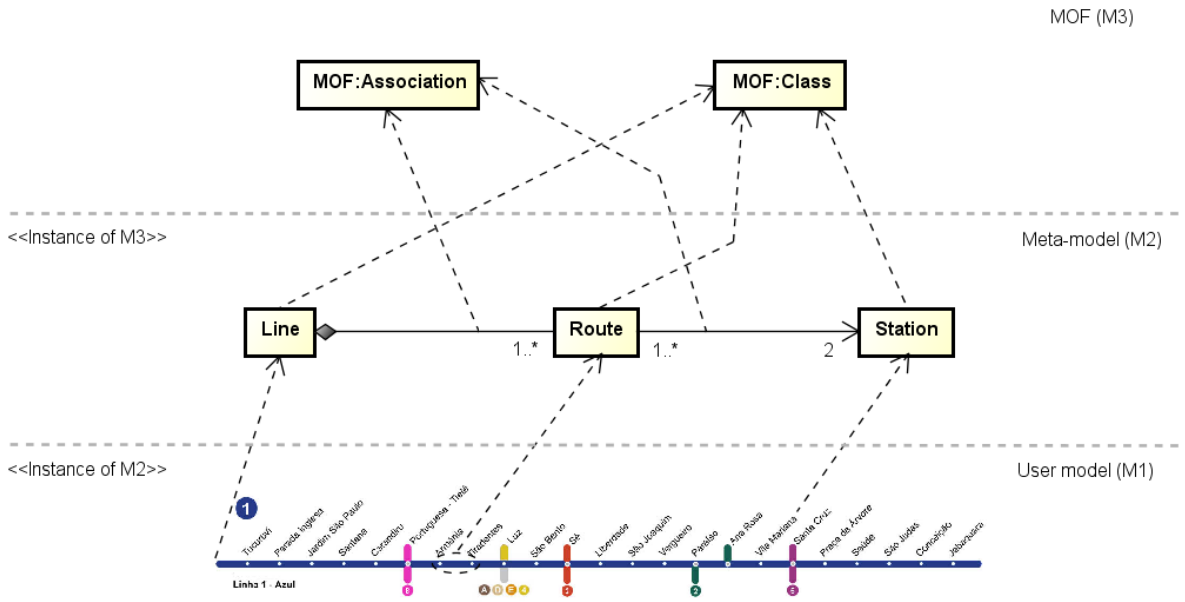


Figura 4 – Níveis de abstração propostos na definição de uma DSL simplificada para Diagramas de Metrô.

Já o nível $\mathtt{M}_0$ não está representado na Figura 4 como "*«Instance of* $\mathtt{M}_1$*»*" (instancias de $\mathtt{M}_1$). Porque, não se trata exatamente uma instanciação, já que não são as linhas propriamente ditas com todos os seus detalhes, além disso os modelos em $\mathtt{M}_1$ podem representar propostas de futuras linhas ainda nem existentes no mundo real.

2.2.4 PIM, PSM e Transformações

No paradigma MDD há uma sequência de etapas que devem ser seguidas, na definição dos níveis de abstração.

Primeiramente os modelos que conceituam elementos mais abstratos (com o nível de abstração mais alto) devem ser independentes de plataforma (*Platform Independent Model* - PIM). A seguir ocorrem uma ou mais transformações até que se alcance as características dependentes de plataforma, criando então modelos específicos para a plataforma escolhida (*Platform Specific Model* - PSM). Finalmente, é possível transformar o PSM em código, já que este está muito mais próximo de sua tecnologia.

Um PIM define um modelo conceitual de alto nível que representa conceitos e requisitos centrais de um negócio, de forma que possa ser adequado a uso em plataformas distintas. Em MDD pode ser necessária a definição de diversas camadas PIM até resultar em um modelo arquitetural de alto nível.

Já um PSM, representa uma visão mais próxima da plataforma desejada, portanto assume como recurso várias de suas características, ficando, por esse motivo, dependente.

O PSM combina a especificação PIM com as especificidades da plataforma que será usada na implementação. Também é possível haver a definição de diversos níveis para PSM, o que irá depender das características da plataforma e da abordagem que se quer dar ao modelo.

Além disso, as definições de PIM e PSM possuem em comum o conceito de plataforma, sendo necessário primeiramente defini-lo, como defende Almeida et al. (2005). Assim, tem-se que a OMG define, na especificação MDA[9], a noção de Modelo de Plataforma ("*Platform Model*") como um conjunto de conceitos técnicos, que representam diferentes tipos de partes para compor a plataforma e os serviços por ela providos.

A transição entre modelos especializando-se cada vez mais os níveis de abstração e tendo como ponto de partida um PIM até alcançar o ponto final, um PSM, é feita por meio de sucessivas transformações sob modelos.

Uma transformação representa o processo no qual um modelo é gerado a partir de outro. Portanto, dado um modelo fonte, pode-se, por meio deste processo, criar um modelo destino que constitui uma nova abstração, mas que está em conformidade com o modelo fonte.

As transformações podem ser feitas usando diferentes técnicas, desde processos manuais, semi-automáticos ou automáticos. Este processo deve ser executado por meio do uso de ferramentas especializadas, que estão baseadas nas diferentes abordagens utilizadas em MDD.

Por exemplo, em MDA há quatro diferentes abordagens de transformação descritas: (i) manual; (ii) de PIM usando perfil; (iii) usando padrões e marcações; e (iv) automática. Neste caso, a relação entre dados e metadados, a partir do MOF, é feita por meio de modelos feitos no formato XMI (*XML Metadata Interchange*)[10], adotado pela OMG a partir de 2006. Desta forma uma ferramenta pode interpretar modelos, identificar e tratar dados e metadados.

Outro exemplo é o padrão QVT (Query/Views/Transformations)[11], no qual uma notação e uma linguagem textual é usada para efetuar as transformações.

Além das propostas da OMG, há abordagens adotadas na indústria a partir da plataforma Java e com apoio de ferramentas (IDEs) como NetBeans[12] e Eclipse[13] por meio de *plug-ins*.

[9] <http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf>
[10] <http://www.omg.org/spec/XMI/2.5.1/>
[11] <http://www.omg.org/spec/QVT/1.2/>
[12] <https://netbeans.org/>
[13] <https://eclipse.org/>

2.2.5 Ferramentas MDD

As ferramentas de modelagem, surgiram para facilitar a criação dos modelos. As mais avançadas permitem a geração de uma parte do código fonte a partir de seus modelos e até mesmo a geração do banco de dados do sistema.

Existem atualmente várias ferramentas que podem ser utilizadas para realizar modelagem, sendo que cada uma oferece alguns recursos ou propriedades mais ou menos complexas que as outras. Dentre essas ferramentas podemos citar algumas: o `Umbrello UML`[14], o `Rational Rose`[15], o `Together`[16] o `Astah`[17] (antigo Jude), o `Enterprise Architect` (EA)[18], e o `Eclipse Modeling Framework` (EMF)[19].

De acordo com (GRONBACK, 2009), o EMF é um *framework* de modelagem e geração de código para construção de ferramentas e outras aplicações com base em modelos estruturados de dados. O EMF, juntamente com o alguns de seus "*plug-ins*", é capaz não só de criar modelos, mas também de executar outras tarefas muito importantes, como transformações de Modelo para Texto (*model to text* - M2T), transformações de Modelo para Modelo (*model to model* - M2M), transformações de Texto para Texto, comumente conhecidas por compilação, (*text to text* - T2T), Engenharia Reversa (*text to model* - T2M) e até mesmo os recursos para definir linguagens específicas de domínio (DSLs).

Os "*plug-ins*" são artefatos computacionais que, quando adicionados às ferramentas para os quais foram criados, permitem somar novas funcionalidades a estas ferramentas. Na lista de "*plug-ins*" para o EMF estão: o `Acelleo`[20] (M2T), o `Sirius`[21] (M2M), `XText`[22] (DSLs), além de ferramentas de "*runtime*" como o `OCL Tools`[23], que permite a interpretação de expressões da linguagem OCL[24], a linguagem `ATL`[25] para transformações, e o `OBEO Designer`[26] que, assim como sua versão aberta `Sirius`, produz ferramentas gráficas.

14 <https://umbrello.kde.org/>
15 <http://www-03.ibm.com/software/products/en/ratirosefami>
16 <http://www.borland.com/en-GB/Products/Requirements-Management/Together>
17 <http://astah.net/>
18 <http://www.sparxsystems.com.au/products/ea/>
19 <https://eclipse.org/modeling/emf/>
20 <https://eclipse.org/acceleo/>
21 <https://eclipse.org/sirius/overview.html>
22 <https://eclipse.org/Xtext/documentation/>
23 <https://wiki.eclipse.org/MDT-OCLTools>
24 <http://www.omg.org/spec/OCL/2.4/PDF/>
25 <http://www.eclipse.org/atl/documentation/>
26 <http://www.obeodesigner.com/>

O EMF, faz uso de uma linguagem própria, também definida com conceitos MDD, denominada `ECore`[27], que é editada por meio do *plug-in* `ECoreTools`[28]. A ECore permite a criação dos níveis de abstração mais altos na definição de DSLs, isto é, permite a criação de metamodelos. Além disso estes metamodelos podem ser diretamente traduzidos para códigos Java™ por meio de transformações M2T, no próprio Eclipse.

Os modelos e metamodelos são descritos no EMF por meio de arquivos no formato XMI que, juntamente com um conjunto de classes adaptadoras, permitem a criação e edição de modelos com uma interface gráfica amigável.

Como o EMF, a maioria de seus "*plug-ins*" são ferramentas gratuitas. Devido à sua flexibilidade e, é claro, por atender a todas as necessidades desta proposta, o EMF foi escolhido como ferramenta de implementação, com a finalidade de gerar um metamodelo para a linguagem do método ***FrameWeb***.

A versão utilizada é o Eclipse EMF, juntamente com os "*plug-ins*": `Sirius`, `Acelleo` e `OCL Tools` na metamodelagem e no protótipo de ferramenta não gráfica; e `Sirius` e `OBEO Designer` no protótipo de ferramenta gráfica.

2.3 Métodos MDD na Engenharia Web

As técnicas MDD têm sido usado na Engenharia Web (*Web Engineering* ou WebE) para apoiar as diversas fases do desenvolvimento de Sistemas de Informação Web. Estes métodos fazem uso das práticas de MDD para produzirem código a partir de modelos, em diversos níveis de abstração e com diferentes resultados e aplicações.

Há entretanto um viés duplo na relação entre MDD e WebE, sendo: (i) uso das técnicas de WebE em MDD; e (ii) uso de técnicas MDD no desenvolvimento de Sistemas de Informação Web.

No primeiro caso, a questão está relacionada na melhoria das técnicas MDD com a adição das melhores práticas usadas na Engenharia Web, e consequentemente de Engenharia de Software. Neste caso MDD se beneficia dos conhecimentos já adquiridos para alcançar com mais qualidade seus objetivos.

No segundo caso, a questão está relacionada ao uso das melhores práticas de MDD e aproveitar seus benefícios, no processo de desenvolvimento de Sistemas de Informação Web. Ou seja, a WebE fazendo uso de MDD. Este é o foco deste trabalho.

[27] <http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html>

[28] <http://www.eclipse.org/ecoretools/>

Na sequência desta seção, os principais métodos que fazem uso de MDD no desenvolvimento de Sistemas de Informação Web, são apresentados com foco comparativo ao contexto de aplicação de técnicas MDD neste trabalho.

Segundo (KOCH et al., 2008a), a Engenharia Web Dirigida a Modelos (*Model-Driven Web-Engineering* - MDWE) é a disciplina da WebE que lida com o estudo, desenvolvimento, aplicação e a interoperabilidade entre as metodologias existentes para o desenvolvimento de Sistemas de Informação Web. Entretanto, ainda há muito a ser feito, devido à contínua e rápida evolução dos Sistemas de Informação Web bem como das novas plataformas baseadas em Web.

O UWE (KOCH et al., 2008b) é um método para desenvolvimento de aplicações *web* que lida com a separação das preocupações, por meio da modelagem de conteúdo, estruturas UI e navegação, bem como das regras de negócio. Por meio de transformações MDD o método parte de modelos independentes de plataforma (PIM) para dependentes (PSM) até promover a geração de código. Usando perfis UML para definição de modelo, este método assemelha-se à proposta ***FrameWeb***, mas distancia-se do estado-da-prática no desenvolvimento de Sistemas de Informação Web, já que não aproveita as funcionalidades disponíveis com o uso de *frameworks*.

O RUX-Method (CALVARY et al., 2003) utiliza técnicas MDD no desenvolvimento de aplicações multimídia interativas para execução em diferentes dispositivos *web*. Separando os modelos em interface abstrata, concreta e final, este método é bastante orientado à visão do usuário, pois considera esta visão mais importante. A partir da interface final ocorre a geração do código de forma direcionada a determinado grupo de dispositivos ou plataforma para, posteriormente, tratar as questões relacionadas à lógica do negócio por meio do uso de uma biblioteca de componentes. Este método também possui uma ferramenta implementada chamada RUX-Tool (LINAJE et al., 2009), criada para dar suporte ao método. Por outro lado, assim como UWE, distancia-se do estado-da-prática no desenvolvimento de Sistemas de Informação Web, sendo muito dependente de sua biblioteca de componentes.

O padrão IFML (BARESI; GARZOTTO; PAOLO, 2001), aceito pela OMG em 2003, é uma linguagem visual independente de plataforma baseada em uma abordagem tradicional do padrão MVC. Usando as técnicas MDD efetua a geração automática de código a partir de modelos, que estão focados principalmente no ponto de vista do usuário. Entretanto por se tratar de uma DSL, requer por parte dos modeladores um conhecimento prévio da linguagem propriamente dita. Diferentemente do método ***FrameWeb***, que é baseado em UML[29], sendo a linguagem de modelagem adotada como padrão pela OMG desde 1997.

[29] <http://www.omg.org/spec/UML/2.5/PDF/>

O MIDAS (CÁCERES et al., 2004) é uma metodologia para desenvolvimento de Sistemas de Informação Web baseado em MDA, através do uso de PIM e PSM representados em UML, mas admite o uso de algumas extensões para representação de esquemas XML, ORM e XLink. Trabalha ortogonalmente avaliando os níveis de conteúdo, hipertexto e apresentação, as diferentes fases do ciclo de vida do software e os aspectos da estrutura e comportamento. Apesar de fazer uso da UML o MIDAS necessita de recursos adicionais, como RRM (ISAKOWITZ; KAMIS; KOUFARIS, 1998) e UWE (KOCH et al., 2000), para permitir a representação de aspectos mais próximos à implementação, tornando-o dependente destes recursos.

EngenDSL (PURIFICAÇÃO; SILVA, 2013) é uma DSL, que tem como objetivo construir uma abstração declarativa para construção de Sistemas de Informação Web, a fim de evitar os construtos de programação típicos, como estruturas condicionais e laços (*loops*), e o compromisso com uma tecnologia específica. Este método usa a abordagem tradicional MVC, mas não considera as diferentes instâncias de *frameworks*, resultando em projetos que não necessariamente refletem a implementação real. Já o método ***FrameWeb*** considera as especificidades dos *frameworks*, fornecendo maior acurácia tanto no que diz respeito à transição entre o projeto e a implementação, quando na codificação em si seja ela automática ou não.

O método OOH4RIA (MELIÁ et al., 2008) é uma extensão do método OOH (GÓMEZ; CACHERO; PASTOR, 2001) utilizando uma abordagem dirigida a modelos, que executa transformações M2M de forma a chegar em um PSM a partir de um modelo plataforma-independente (PIM). Esta proposta está baseada no *framework* da Google™ (*Google Web Toolkit* - `GWT`)[30] para plataforma Java, o que sob certo ponto de vista enrijece o método, porque sua visão PIM não é de fato totalmente independente já que está intimamente relacionada a aspectos da plataforma escolhida. Apesar do ***FrameWeb*** em sua apresentação ter sido direcionado a certo conjunto de *frameworks*, sua visão de projeto não é totalmente rígida o que permite a este trabalho revisitar o método e tratar mais diretamente as questões dependentes de plataforma.

Nesta mesma linha o método OOWS (FONS et al., 2008) propõe transformações M2M, PIM para PSM, mas a implementação utiliza o *OlivaNova Transformation Engines*[31] para plataforma PHP. Ao contrário, o método ***FrameWeb*** pretende ser flexível, permitindo o uso de qualquer *framework* (dentre as categorias suportadas), mesmo outros fora da plataforma Java EE.

Panach et al. (2014) propõem um *framework* para lidar com aspectos de usabilidade com um método MDD baseado no conceito de modo de uso (*Mode of Use* - MoU). Baseado no (e parcialmente dependente do) OO-Method (PASTOR et al., 2001), esse método

[30] <http://www.gwtproject.org/overview.html>
[31] <http://www.omg.org/mda/mda_files/SOSY_OlivaNova_Overview1.pdf>

aborda deficiências de MDD relacionadas com a usabilidade (de acordo com a norma ISO 9241-11[32]). Como um método de projeto, o FrameWeb pode ser integrado a esta proposta, acrescentando os elementos necessários de metamodelagem para permitir que MoUs sejam adicionados aos modelos ***FrameWeb***. Assim, esta proposta pode ser vista apenas como complementar ao nosso trabalho.

A maior motivação da proposta de ***FrameWeb*** e de sua continuidade, está no fato de que, mesmo havendo diversos métodos definindo linguagens e ferramentas para o desenvolvimento de Sistemas de Informação Web, no que diz respeito ao nosso conhecimento, nenhum deles até o momento foca no importante papel dos diversos *frameworks* na arquitetura dos sistemas de forma flexível e adaptável.

[32] <http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=16883>

3 A concepção de *FW-15*

Em sua proposta original, o método ***FrameWeb*** foi direcionado a um conjunto específico de *frameworks*, sendo assim dependente de plataforma. Essa característica origina-se do fato do método assumir uma postura bastante pragmática no sentido de aproximar as fases de projeto e implementação de software, já que assume o uso de um conjunto de *frameworks* na definição da arquitetura de um Sistema de Informação Web.

Num primeiro momento, os *frameworks* escolhidos foram suficientes para o uso e teste do método. Agora, no entanto, se faz necessário amadurecer e atualizar o ***FrameWeb*** original para que o mesmo passe a ser independente de plataforma, tornando-se mais flexível no que diz respeito à modelagem, porque permite o uso de diversas combinações de *frameworks*, a escolha do modelador. Esta atualização é discutida detalhadamente nas seções 3.1 a 3.3.

Posteriormente é necessário dissertar a respeito das principais decisões tomadas ao longo desse período e que ajudaram a definir os rumos de nossa proposta, conforme detalhado na Seção 3.4. Finalmente, na Seção 3.5 apresentam-se as considerações gerais a respeito deste capítulo.

Para facilitar o entendimento e diferenciar o método ***FrameWeb*** original de sua atualização proposta neste trabalho, utilizaremos dois termos distintos: ***FW-07*** será usado a partir de agora para tratar do método proposto em 2007, enquanto ***FW-15*** se refere à sua nova versão proposta nesta dissertação. Ao utilizarmos o termo ***FrameWeb***, estaremos considerando aspectos gerais do método pertinentes a ambas as versões.

3.1 Revisando o método *FW-07*

Conforme apresentado na Seção 2.1, o método ***FrameWeb*** faz uso de quatro diagramas na representação dos componentes pertinentes à plataforma *web*.

Entretanto, no que diz respeito ao **Modelo de Domínio**, o termo *"Domínio"*, utilizado para referir-se ao domínio de aplicação do Sistema de Informação Web a ser construído vem gerando algumas dificuldades de interpretação e por isso necessita ser reavaliado.

Assim como na Engenharia de Software, na Engenharia Web o termo *"Domínio"* é usado para identificar certo conjunto de funcionalidades, em geral relacionadas ou agrupadas a partir de determinados critérios, no contexto de sistemas de informação.

Desta maneira, um domínio pode ser definido como um conjunto de aspectos discricionais a respeito de um ou mais problemas a serem solucionados a partir de um sistema de informação. O método ***FW-07*** está baseado nesta interpretação para a utilização do termo **Modelo de Domínio**, no contexto do projeto de Sistemas de Informação Web.

Mas o termo *"Domínio"* tem diversas outras interpretações no mundo *web*, dependendo de onde é aplicado. O caso mais conhecido refere-se ao sistema de nome de domínios (*Domain Name System* - DNS)[1], resumidamente usado para identificar conjuntos de computadores na internet.

Além disso, o termo *"Domínio"* é usado no âmbito de MDD na identificação de modelos conceituais que incorporam aspectos comportamentais e de dados, conhecidos como Modelos de Domínio (*Domain Models*). O que significa dizer, que não só o termo *"Domínio"* possui várias interpretações, mas o termo *"Modelo de Domínio"* compartilha este problema.

Esta sobrecarga associada aos termos usados na nomenclatura do ***FW-07***, pode causar gerar ambiguidade na interpretação dos aspectos relacionados a este trabalho, mas o que é mais grave, confusão na interpretação do que o **Modelo de Domínio** representa no contexto do método.

Por causa do exposto, na versão ***FW-15*** do método propõe-se que o termo **Modelo de Domínio** seja ajustado, passando a ser conhecido como **Modelo de Entidades**, entretanto nenhuma de suas funções e características originais foi alterada, isto é, apenas a nomenclatura foi ajustada.

Assim o método ***FW-15*** contém agora a seguinte denominação para os modelos ***FrameWeb***:

- **Modelo de Entidades** (**Entity Model**), em substituição ao **Modelo de Domínio** (**Domain Model**) de ***FW-07*** mantendo-se todas as outras definições originais;
- **Modelo de Persistência** (**Persistence Model**), mantido de ***FW-07***;
- **Modelo de Navegação** (**Navigation Model**), mantido de ***FW-07***;
- **Modelo de Aplicação** (**Application Model**), mantido de ***FW-07***.

Apesar do ajuste de nomenclatura, o pacote **Domínio** do ***FW-15*** continua sendo usado para definir os elementos do sistema no âmbito da aplicação exatamente como definido originalmente no ***FW-07***.

[1] <http://www.rfc-base.org/rfc-1034.html> e <http://www.rfc-base.org/rfc-1035.html>

3.2 Partes interessadas e papeis em *FW-15*

As partes interessadas em um Sistemas de Informação Web assim como em Sistemas de Informação de modo geral, são conhecidas pelo termo em inglês: *"stakeholders"*, criado para uso na área de Administração em 1963, em um memorando interno da Stanford Research Institute e posteriormente formalizado por Freeman (2010), pela primeira vez editado em 1984.

Este conceito foi adotado em outras áreas como Gestão de Projetos e Engenharia de Software, de maneira geral, estando associado a todos os envolvidos em um determinado processo, projeto ou negócio.

Apesar de estar mais próximo a processos de software orientados a objeto, o método ***FrameWeb*** não impõe o uso de nenhum processo de software em especial, apenas espera-se que a equipe de desenvolvimento trabalhe com as atividades comuns à maioria dos processos de software (e.g., levantamento de requisitos, análise, projeto, codificação, testes e implantação).

Por isso, alguns papeis importantes a devem ser desempenhados pelas partes interessadas (*"stakeholders"*) no processo de Sistemas de Informação Web com o método ***FW-07***.

A Tabela 1 apresenta as principais partes interessadas, os papeis desempenhados por elas durante as fases do processo de desenvolvimento de Sistemas de Informação Web, bem como a relação atuação destes papeis no método ***FW-07***.

Tabela 1 – Partes interessadas e seus papeis no método ***FW-07***.

Parte Interessada	Papeis em *FW-07*
Usuários (*Users*)	Usuários finais do Sistema de Informação Web
Analistas de Sistemas (*System Analysts*)	Os Analistas responsáveis por propor a solução do sistema com base nos requisitos levantados.
Projetistas de Software (*System Designers*)	Os modeladores responsáveis por projetar os modelos ***FW-07*** a partir das definições fornecidas pelo processo de análise.
Desenvolvedores (*Developers*)	Desenvolvedores do software propriamente, responsáveis por codificar parcial ou totalmente o sistema de acordo com as definições propostas no projeto por meio método ***FW-07***.

Neste caso, cabe salientar que por ser um método para a fase de projeto de Sistemas de Informação Web, as partes interessadas e papeis são restritas à atuação nesta fase. Assim há diversas outras partes interessadas e papeis envolvidos no processo de desenvolvimento de Sistemas de Informação Web que não estão incluídos na Tabela 1, pois não faz parte do escopo deste trabalho ir mais além em termos do processo de software.

Usuários de um Sistema de Informação Web podem participar do sistema de diversas maneiras diferentes, e em geral compreendem um público grande e de perfil variável. Normalmente os perfis de usuários de um Sistema de Informação Web devem ser avaliados criteriosamente nas primeiras fases do processo de software, de forma a permitir que seus anseios e expectativas sejam devidamente satisfeitos.

Analistas devem, com base nos requisitos levantados, serem capazes de prover os recursos, artefatos e informações necessários aos projetistas, que por sua vez e com base nesta análise, devem ser capazes assumir as decisões de projeto necessárias para dar prosseguimento no processo de software do Sistema de Informação Web.

Além das diversas tarefas de projeto de um Sistema de Informação Web, uma das atividades dos projetistas segundo o método ***FW-07*** é a atuar como modeladores do Sistema de Informação Web, sendo estes responsáveis por projetar, graficamente nos modelos propostos, os elementos do Sistema de Informação Web de acordo com as diretrizes estipuladas no método. Posteriormente, durante a fase de codificação, estes modelos são usados pelos desenvolvedores ou programadores para dar efetivamente produzir o software.

Entretanto, no método ***FW-07***, o sucesso do projeto e consequentemente da codificação estão diretamente relacionados a dois fatores importantes, são eles:

- `O conhecimento do método` ***FW-07*** `por parte do modelador:` a qualidade dos modelos produzidos é proporcional ao conhecimento do modelador com respeito ao método, o que significa dizer, quanto mais experiente no método maior será a qualidade dos modelos produzidos pelo modelador;

- `A escolha das tecnologias e` *frameworks* `adotados:` a opção por parte dos projetistas do uso de determinada tecnologia ou *framework* influencia diretamente na sintaxe dos modelos, pois alguns elementos da linguagem são variáveis em relação a estes recursos.

No primeiro caso, tem-se que, por não possuir uma linguagem formalmente definida em um metamodelo e nem uma ferramenta de modelagem baseada neste metamodelo, o método ***FW-07*** depende dos modeladores para que as suas diversas especificidades sejam tratadas corretamente, sendo esta uma das motivações deste trabalho, conforme apresentado na Seção 1.2.

O método ***FW-07*** foi direcionado às três principais categorias de *frameworks* (ORM, MVC e DI) usadas no projeto e desenvolvimento de Sistemas de Informação Web, onde para cada uma destas categorias um determinado *framework* foi selecionado como prova de conceito para avaliação.

Tabela 2 – Categorias de *frameworks* suportados pelo método ***FW-15***.

Categoria do Framework	Suportados no ***FW-07***	Padrão Java EE	Outros Exemplos
Controlador Frontal (MVC)	`Struts`[2]	`JSF`	`VRaptor`, `GWT`, `Spring MVC`
Mapeamento Objeto Relacional (ORM)	`Hibernate`	`JPA`	`Cayenne`, `OpenJPA`, `pBeans`
Injeção de Dependência (DI)	`Spring Framework`	`CDI`	`PicoContainer`, `Guice`, `Plexus`

Por isso, determinados aspectos destes *frameworks* foram considerados e adicionados ao modelos propostos pelo método, todavia diferem de aspectos relacionados a novas tecnologias e *frameworks* atualmente usadas e consagradas pelo estado-da-prática.

Assim, ocorre por parte do modelador uma necessidade de adaptação dos modelos do método para que se adequem a tecnologias e *frameworks* diferentes dos propostos originalmente em ***FW-07***, o que requer dele um conhecimento mais profundo destes recursos. Além disso em muitos casos gera não só problemas sintáticos mas também semânticos, podendo resultar falhas de representação e interpretação destes modelos.

A Tabela 2 apresenta as categorias de *frameworks* propostos pelo método ***FW-15*** em comparação ao originalmente suportados em ***FW-07***: controladores frontais, *frameworks* de mapeamento objeto/relacional e *frameworks* de injeção de dependências.

Para que o método ***FW-15*** possa ser suficientemente flexível permitindo o uso de diversas tecnologias e *frameworks* diferentes, é necessário que os aspectos dependente de *framework* sejam separados dos conceitos gerais do método.

Desta forma os conceitos gerais do método são tratados de forma padronizada e independentemente das tecnologias e *frameworks* a serem adicionados a um dado projeto de Sistema de Informação Web, enquanto que os aspectos associados às características e comportamentos específicos de uma dada tecnologia ou *framework* selecionado para o projeto são tratados de forma particular.

Os aspectos independente de *framework* são então tratados pelo metamodelo da linguagem ***FW-15***, enquanto os aspetos dependente de *framework* são adicionados a este metamodelo por meio de perfis criados especificamente para cada *framework* ou tecnologia, cada qual sendo armazenado sua biblioteca denominada **Definição de Framework**. Os detalhes desta separação a nível de metamodelo estão apresentados no Capítulo 4 (Seção 4.3).

Esta separação das preocupações em dependente de *framework* e independente de *framework* libera o modelador para lidar apenas com aspectos próprios do desenvolvimento do Sistema de Informação Web sem ter assumir o papel de especialista em dada tecnologia

ou *framework*. Com efeito, o papel do Especialista em *framework* deve ser desempenhado em um momento anterior à fase de projeto do Sistema de Informação Web, de forma a disponibilizar aos modeladores um conjunto de diversas **Definições de Framework** disponíveis para uso. Estas **Definições de Framework** podem ser disponibilizadas como bibliotecas da ferramenta de modelagem desenvolvida com base no metamodelo ***FW-15*** e usadas pelos modeladores quando lhes convier.

O Especialistas em *framework* tem a responsabilidade de criar uma **Definição de Framework** específica de seu *framework* de conhecimento, que quando adicionada ao metamodelo ***FW-15*** é capaz de delimitar e gerenciar os aspectos dependente de *framework*, por meio do **Perfil FrameWeb** adequado. Para desempenhar esta tarefa os seguintes conhecimentos prévios são necessários ao especialista em *framework*:

- Ter experiência no projeto de Sistemas de Informação Web com o uso de seu *framework* de conhecimento;
- Conhecer a especificação detalhadamente deste *framework*;
- Conhecer o processo de adição de um **Perfil FrameWeb** ao metamodelo.

Pode-se notar que este papel e suas atividades originalmente em ***FW-07*** eram acumuladas tanto por projetistas quanto pelos desenvolvedores, por isso além de causar-lhes sobrecarga também deixava margem a problemas de continuidade no processo de software, na medida que uma destas partes pode a qualquer momento deixar de fazer parte da equipe.

Tabela 3 – Partes interessadas e seus papeis no método ***FW-15***.

Parte Interessada	**Papeis em *FW-15***
Usuários (*User*)	Usuários finais do Sistema de Informação Web
Analistas de Sistemas (*System Analysts*)	Os Analistas responsáveis por propor a solução do sistema com base nos requisitos levantados.
Especialistas em *frameworks* (*Framework Experts*)	Especialistas, responsáveis pelo processo de desenvolvimento dos recursos ***FW-15*** para os diversos *frameworks* a serem utilizados pelo Sistema de Informação Web.
Projetistas de Software (*System Designers*)	Os modeladores responsáveis por projetar os modelos ***FW-15*** a partir das definições fornecidas pelo processo de análise.
Desenvolvedores (*Developers*)	Desenvolvedores do software propriamente, responsáveis por codificar parcial ou totalmente o sistema de acordo com as definições propostas no projeto por meio método ***FW-15***.

A Tabela 3 apresenta as principais partes interessadas, os papeis desempenhados por elas durante as fases do processo de desenvolvimento de Sistemas de Informação Web, bem como a relação atuação destes papeis no método ***FW-15***, destacando-se o Especialista em *framework*.

Esta nova visão dos papeis a serem desempenhados no método ***FW-15***, proporciona não só uma clara separação entre as preocupações dependente de *framework* e independente de *framework*, como também torna o método ***FW-15*** flexível e extensível conforme será apresentado na Seção 3.3, a seguir.

3.3 Requisitos para evolução de FrameWeb

Um modelador, ao usar o ***FW-07***, em geral usa uma ferramenta de modelagem UML para a criação dos modelos necessários, obedecendo as diretrizes do método. Assim, o modelador necessita produzir os quatro modelos que compõem o método, sob a forma de diagramas de classe UML. Porém, fica a seu cargo usar adequadamente todas as diretrizes do método tendo por base a linguagem UML, não havendo qualquer formalidade obrigatória maior.

O método ***FW-07*** não define uma linguagem propriamente dita, ao contrário, usa uma fatia da linguagem UML e sobre ela adiciona suas diretrizes, um perfil UML peso leve (ou "*UML lightweight profile*"). Por outro lado, com uma ferramenta baseada em uma DSL formal, o modelador poderia trabalhar sem que seja necessário ter maiores preocupações além das próprias da modelagem, já que os modelos deixariam de ser apenas diagramas de classe UML de um perfil e seriam agora guiados por regras de sintaxe e semântica próprias para o método.

Por isso, determinamos como requisitos para ***FW-15*** a definição de uma DSL que, apesar de ser baseada em diagramas de classe UML (decisão esta que discutimos mais adiante, na Seção 3.4), possui todas as regras necessárias para o método e, ao mesmo tempo, mantém suas características originais, que se beneficiam de diversos aspectos da UML.

Outra decisão tomada nas fases iniciais de concepção deste trabalho foi a de ***FW-15*** herdar características de ***FW-07***, por considerarmos importante manter a identidade do método e as vantagens trazidas desde sua versão original. De modo geral, as seguintes características são herdadas:

- Utilização da sintaxe dos diagramas de classe UML, fazendo-se as devidas diferenciações por meio de estereótipos;
- Exposição nos modelos, das definições e regras de domínio do Sistema de Informação Web, explicitamente por meio de do uso das restrições UML ("*constraints*");

- Validação dos modelos quanto a sua sintaxe e semântica perante as regras do método;

- A possibilidade do uso e inclusão de outros modelos UML no contexto do projeto, se esse for necessário para o desenvolvimento do Sistema de Informação Web e segundo o julgamento do modelador.

Uma das motivações principais para a evolução de FrameWeb foi a possibilidade de utilizar o método com outro conjunto de *frameworks* diferente dos utilizados na definição de ***FW-07*** (SOUZA, 2007; SOUZA; FALBO, 2007). Percebeu-se, então, a necessidade de se realizar um estudo abrangente das categorias de *frameworks* tratadas na proposta original (controladores frontais ou MVC, mapeamento objeto/relacional, injeção de dependência e decoradores) para a definição de um modelo mais inclusivo.

Durante a avaliação dos diversos *frameworks* disponíveis em cada categoria para este trabalho ficou evidente a impossibilidade da cobertura de uma gama muito grande de *frameworks* no tempo disponível. Assim, delimitamos o escopo principalmente em *frameworks* MVC sem, no entanto, deixar de dar alguma atenção às categorias ORM e DI, como apresentado no Capítulo 4.

Nas subseções a seguir, apresentamos brevemente o resultado de nossa análise dos *frameworks* MVC e uma proposta para extensibilidade do método ***FW-15***.

3.3.1 Frameworks MVC

Frameworks MVC (ou controladores frontais) podem ser baseados em ações (*Action-Based* ou *MVC Push*) ou componentes UI (*Component-Based* ou *MVC Pull*). Por causa disso e para tratar mais genericamente os casos, a análise foi baseada nas diferenças e semelhanças entre estas duas abordagens e não propriamente nas diferenças e semelhanças entre *frameworks* específicos.

O ***FW-07*** utilizou a abordagem proposta pelo `Struts`[2], um *framework* baseado em ações. Diversos outros *frameworks* seguem este modelo, por exemplo: o `Spring MVC` o `VRaptor`, o `ASP.NET MVC`, o `Rails`, etc.

A Figura 5 mostra o fluxo de processamento "*MVC Push*", onde é possível notar que, a partir de uma requisição por parte do usuário, o controlador frontal localiza e aciona a classe responsável por tratar esta requisição executando o método apropriado, que é responsável pelas regras de negócio envolvidas no processamento e por acionar a persistência se for necessário. Após a execução das regras de negócio em questão, a visão é responsável por produzir o retorno adequado ao usuário, neste caso representado por uma página. Nesta situação, os dados são disponibilizados para a visão fazer sua exibição, "são empurrados", daí o nome "*MVC Push*".

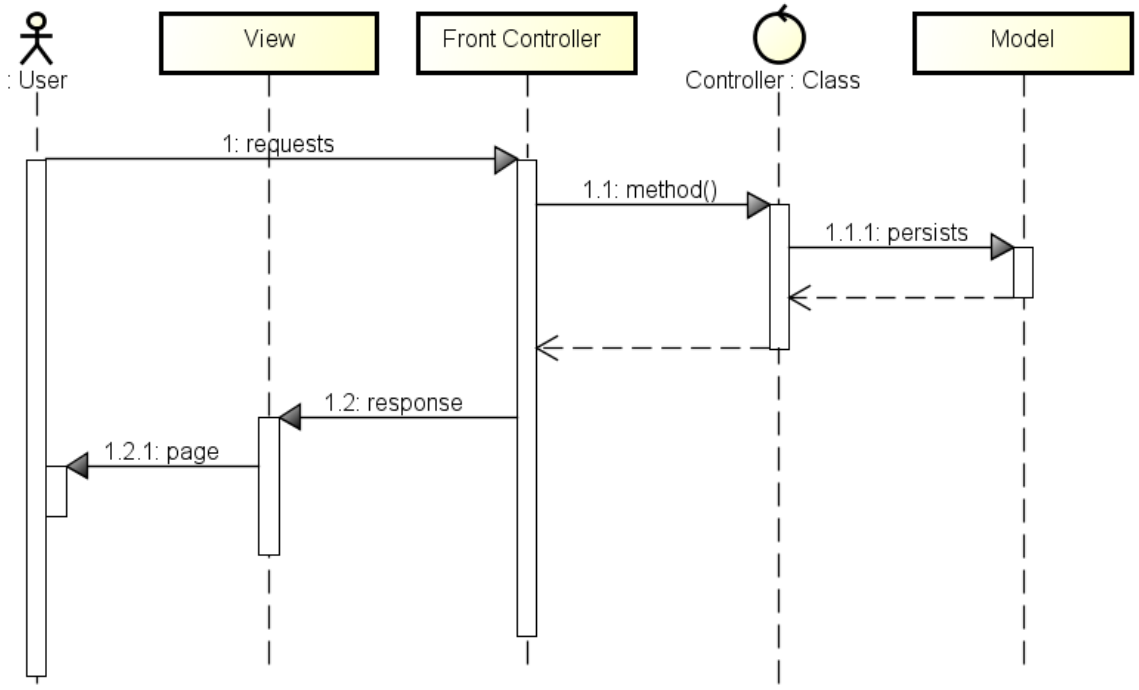


Figura 5 – *Frameworks* MVC baseados em ações (*Action Based* ou *MVC Push*).

Frameworks MVC baseados em ações são extremamente flexíveis já que os objetos visuais podem ser criados por meio de diversas tecnologias (`HTML`, `JavaScript`, `CSS`, etc.), o que permite infinitas possibilidades no desenvolvimento de Sistemas de Informação Web.

Outros *frameworks* utilizam uma abordagem baseada em componentes UI, como por exemplo: o `JSF`, `GWT`, `Wicket`, etc. A Figura 6 mostra o fluxo de processamento "*MVC Pull*", onde observa-se que, a partir da requisição feita pelo usuário, o controlador frontal processa o componente da visão associado a esta requisição, então o método da classe do controlador relacionado a este componente é acionado, sendo responsável pelas regras de negócio envolvidas no processamento e por acionar a persistência se for necessário. Após a execução das regras de negócio em questão, objetos recuperados pela classe do controlador frontal são enviados à visão que então retorna ao usuário as informações adequadas, aqui representadas por uma página. Neste caso a visão solicita os objetos necessários, "puxa os dados", daí o nome "*MVC Pull*".

Frameworks MVC baseados em componentes são capazes de produzir Sistemas de Informação Web usando uma grande variedade de componentes visuais pré-prontos, de forma a eliminar maiores preocupações no que diz respeito aos recursos usados por estes componentes (scripts, folhas de estilo, etc.) o que permite seu reuso com alta qualidade. Há diversos componentes (calendários, tabelas, formulários, objetos multimídia, etc.), porém todos eles são renderizados e exibidos via HTML.

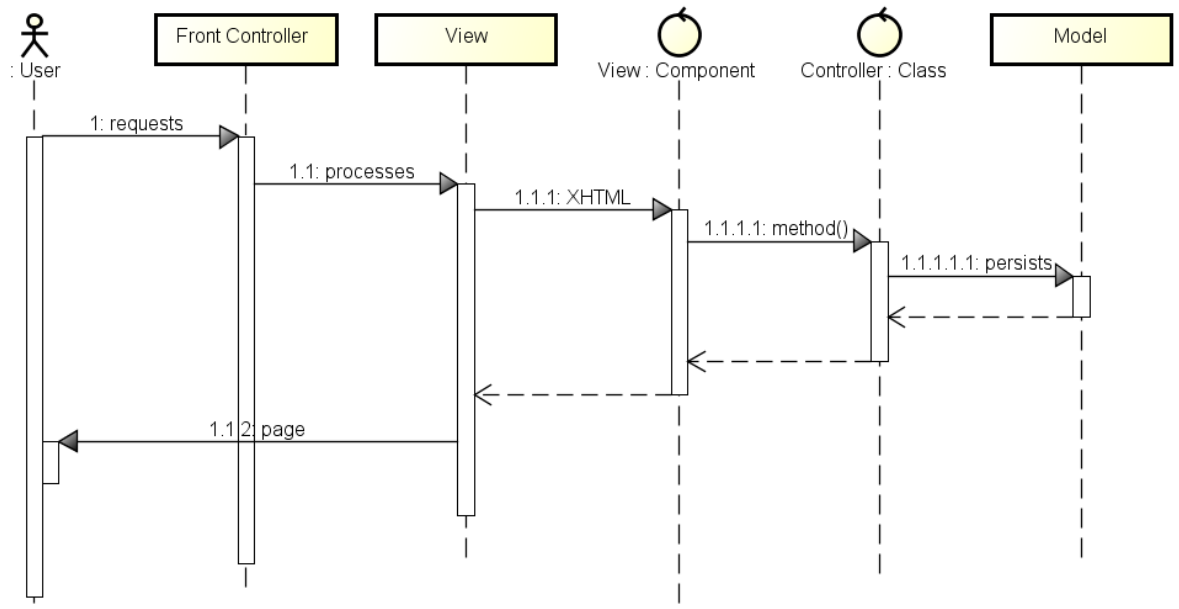


Figura 6 – *Frameworks* MVC baseados em componentes (*Component Based* ou *MVC Push*).

O ***FW-15*** além de manter-se fiel ao `Struts`[2], se propõe a ser compatível com diversos outros *frameworks* MVC baseados em componentes. Neste sentido, ao observar o metamodelo ***FW-15***, é possível notar que os elementos visuais de um Sistema de Informação Web estão sendo representados em sua totalidade como componentes UI, independentemente de serem objetos pré-concebidos ou elementos específicos como *tags* `HTML`, por exemplo. De forma semelhante, as classes de controle, sejam elas de "*actions*" ou "*beans*", também são tratadas como classes do controlador genéricas compostas de métodos também gerais.

Esta abordagem é possível porque cada **Definição de Framework** carrega consigo os estereótipos que estendem estas metaclasses de acordo com o que cada *framework* define. No que diz respeito ao fluxo de processamento diferenciado, o metamodelo ***FW-15*** não faz restrições maiores, entretanto cada **Definição de Framework** possui a capacidade de adicionar regras OCL específicas para cada situação, filtrando os fluxos permitidos ou não, já que para o metamodelo estes fluxos são representados por metaclasses que definem os relacionamentos possíveis.

Esta decisão de projeto proporciona flexibilidade ao metamodelo ***FW-15*** e é uma abordagem extensível para todas as outras categorias de *frameworks*, tendo sido usada de forma semelhante na definição dos mapeamentos para classes de domínio de *frameworks* ORM como `JPA` e `Hibernate`.

3.3.2 Novas tecnologias

Um conjunto de diversas novas tecnologias surgiram posteriormente à definição do método ***FW-07***, sendo o `AJAX` a que mais fortemente influenciou o projeto e desenvolvimento de Sistemas de Informação Web, consequentemente é muito importante que o ***FW-15*** o incorpore.

O `AJAX` é um modelo capaz de fazer rapidamente atualizações incrementais para a interface do usuário sem que seja necessário recarregar uma página inteira do navegador web. Reúne um conjunto de tecnologias, sendo as principais: `HTML` ou `XHTML`; `CSS`, `JavaScript`; `XML`; `XSTL`; `DOM` (*Document Object Model*)[2] e `XMLHttpRequest`[3].

Apesar de ser uma tecnologia específica, desde sua adoção pela Google™, conforme previu Garrett et al. (2005), o `AJAX` tem sido largamente usado em aplicações web, porque seu uso proporciona maior eficiência (SMULLEN III; SMULLEN, 2008) e melhora a usabilidade (KLUGE; KARGL; WEBER, 2007) de Sistemas de Informação Web.

Adicionalmente ao já mencionado, o `AJAX` está presente nos principais *frameworks* MVC modernos, inclusive sendo a principal abordagem de alguns deles, os quais são conhecidos como *frameworks* AJAX. Estes *frameworks* são agrupados em quatro tipos:

***Direct AJAX frameworks*:** são compostos de páginas `HTML` e o *framework* lida diretamente com os seus elementos, deixando a cargo da sua API *cross-browser* as tarefas de comunicação, manipulação `DOM`, tratamento de eventos e alterações nos elementos gráficos carregados. Um exemplo é o `ZKoss` (`ZK`)[4];

***Indirect AJAX frameworks*:** são baseados na tecnologia de compilação, na qual a codificação é feita em uma linguagem de alto nível, posteriormente traduzida para `JavaScript`. É suportado por diversos *frameworks*, como o `ATLAS`[5] e o `Outpost`[6];

***AJAX component frameworks*:** fornecem componentes pré-existentes, que lidam com seu próprio `HTML` e são criados com ajuda de `JavaScript` ou *tags* `XML` ou `HTML` em si, podendo fornecer APIs de personalização, controle de programação e extensibilidade. Nesta categoria tem-se o `Dojo Toolkit`[7], `Script.aculo.us`[8], `YUI` (*Yahoo! User Interface*)[9], `OpenRico`[10] e `TIBCO General Interface application`[11];

[2] <https://www.w3.org/DOM/>
[3] <https://www.w3.org/TR/XMLHttpRequest/>
[4] <https://www.zkoss.org/>
[5] <http://atlascode.com/technologies/atlas-framework>
[6] <http://getoutpost.org/>
[7] <https://dojotoolkit.org/>
[8] <https://script.aculo.us/>
[9] <http://yuilibrary.com/>
[10] <http://openrico.sourceforge.net/examples/index.html>
[11] <https://docs.tibco.com/pub/general_interface_enterprise_edition/3.9.1-august-2011/pdf/GI-3.

***Server-drive frameworks*:** os componentes são desenvolvidos no servidor usando uma linguagem de *scripts*, posteriormente as páginas são renderizadas por uma combinação de `HTML` parte no servidor e parte cliente, neste caso o `AJAX` é usado para comunicar as ações do usuário para o servidor, onde o *script* funciona como modelo de componente sob o qual as alterações são feitas e exibidas para o cliente automaticamente. Há diversos *frameworks* deste tipo, por exemplo: `GWT`, `ICEFaces`[12], `Microsoft AJAX`[13], e outros.

Apesar das diferentes abordagens adotadas pelos diversos *frameworks* que fazem uso de `AJAX`, todos tem em comum o uso do objeto `XMLHttpRequest`, que é uma API para recuperação de recursos, através do qual as solicitações feitas por meio do *web browser* de um cliente são processadas e respondidas, conforme apresentado na Figura 7.

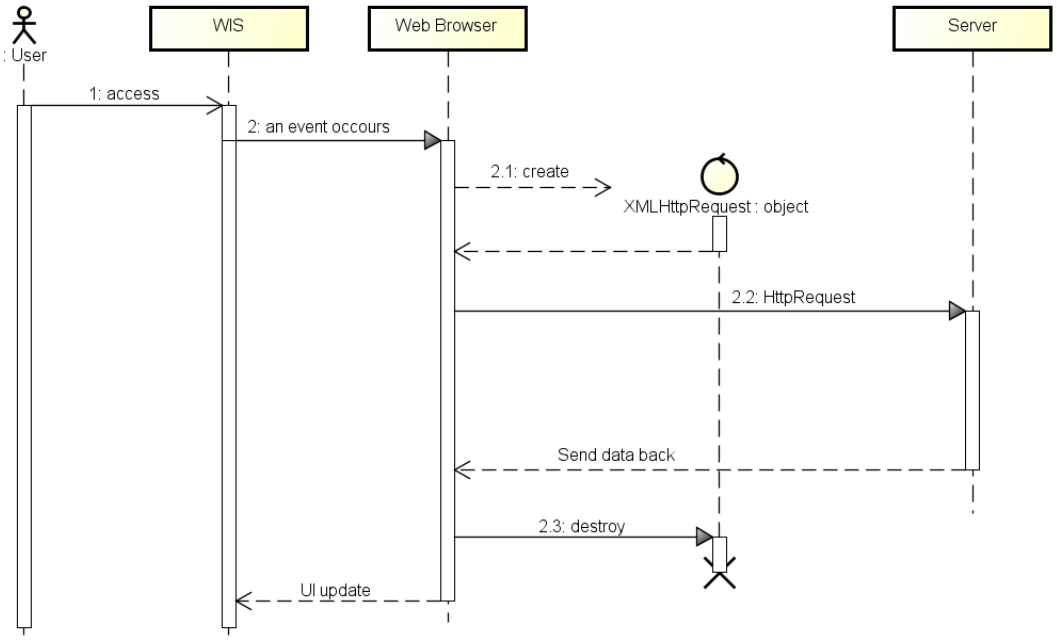


Figura 7 – Ciclo de vida de uma requisição `AJAX` – `XMLHttpRequest`.

Por meio do *web browser* o usuário do Sistema de Informação Web tem acesso e interage com sua interface, que em determinadas situações pode efetuar uma chamada `JavaScript` para o `AJAX engine` do *web browser*, fazendo um `HTTP request` ao servidor. O servidor por sua vez processa o `HTTP request`, criando e enviando a devida resposta (dados em `XML`) ao *web browser*. Ao receber os dados em `XML` como retorno do `HTTP request`, o `AJAX engine` do *web browser* monta e exibe a página (`HTML` mais dados `CSS`) como parte da interface do Sistema de Informação Web.

9-GettingStarted.pdf>

12 <http://www.icesoft.org/java/projects/ICEfaces/overview.jsf>

13 <https://msdn.microsoft.com/pt-br/library/ee341002(v=vs.100).aspx>

Este processo ocorre de forma transparente nos *frameworks*, ou seja, nem o modelador nem o desenvolvedor precisam lidar com aspectos internos relacionados ao ciclo `AJAX` apresentado na Figura 7.

Um dos objetivos dos *frameworks* AJAX é esconder totalmente a codificação `JavaScript` do desenvolvedor, fornecendo bibliotecas de classes Java que contem coleções de componentes visuais `AJAX` (*widgets*). Como estes componentes já encapsulam todos os aspectos `AJAX` necessários ao seu funcionamento não é necessário por parte do modelador fazer qualquer diferenciação especial quando um destes elementos deve ser usado em um Sistema de Informação Web, bastando apenas informar ao desenvolvedor que componente deve ser usado.

Nos *frameworks* MVC, o modelador em certas situações deverá definir os componentes UI que usarão `AJAX` e para isso alguns parâmetros devem ser informados. Os parâmetros necessários em geral são: o componente de interface em si, o uso ou não de `AJAX`, o processo envolvido e o tipo de renderização.

O ***FW-07*** foi direcionado ao `Struts`, um *framework* MVC, por conseguinte a abordagem adotada para o ***FW-15*** segue primariamente os princípios destes *frameworks*, devendo portanto atender às características desta abordagem.

Nestes casos, a identificação do uso de `AJAX` é feita pelos componentes de interface quando ocorre o retorno do processamento de algum método de controle. Por isso, os parâmetros mais comumente usados foram tratados como restrições (*constraints* UML) adicionadas às relações entre as classes de controle e a interface, conforme está explicado na Seção 4.2.7.

Com vistas a esta abordagem estes parâmetros foram incluídos na estrutura independente de *framework* do metamodelo ***FW-15*** para evitar que os especialistas em *framework* sejam obrigados a adicionar estrutura comuns a todos os *frameworks* MVC repetidamente em todo **Definição de Framework** criado.

Entretanto, como não há obrigatoriedade na adoção das restrições com respeito ao uso de `AJAX` para *frameworks* MVC e apesar desta categoria de *frameworks* não ser foco deste trabalho, a abordagem adotada pelos *frameworks* AJAX poderá ser totalmente coberta em evoluções do metamodelo, porque os componentes UI podem ser estendidos por meio das **Definições de Framework**, conforme está mostrado na Seção 4.3.1.

3.3.3 Extensibilidade

Ao estudar as diferenças e semelhanças entre estes *frameworks*, fica evidente que há a necessidade de incorporar dinamicamente partes específicas de cada um à linguagem ***FW-15***. Para possibilitar esta característica dinâmica, estas partes específicas estão organizadas em bibliotecas criadas para cada *framework* ou padrão.

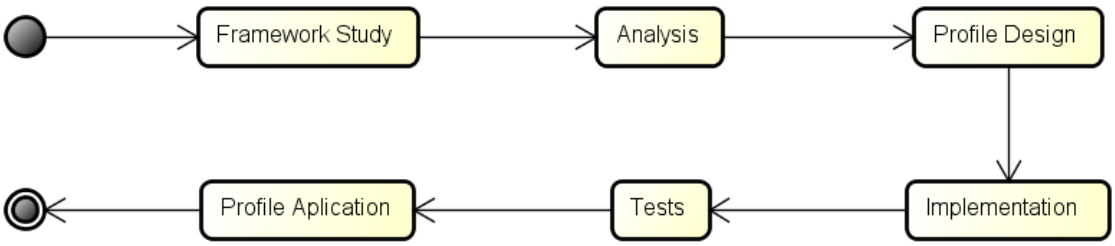


Figura 8 – Sugestão de processo para a **Definição de Framework *FW-15***.

O uso de bibliotecas, além de permitir que as características específicas de diversos *frameworks* (incluindo *frameworks* de outras categorias) possam ser incorporadas à linguagem ***FW-15***, também permite que bibliotecas criadas pelos especialistas (*framework Expert*) possam ser incorporadas à linguagem e, com elas, diversos componentes personalizados podem ser adicionados à modelagem.

Entretanto, a criação destas bibliotecas não é trivial, porque exige certo conhecimento prévio do metamodelo ***FW-15*** e suas características. Por isso é importante que cada **Definição de Framework** seja criada através de um processo sistemático e formal.

Algumas diretrizes básicas foram traçadas para orientar a criação de bibliotecas. Primeiramente sugere-se que um processo sistemático seja seguido. A Figura 8 apresenta uma sequência mínima de atividades em um processo linear a ser seguido, entretanto não há impedimento no uso de outros fluxos processuais, inclusive evoluções nesse sentido podem ser parte de trabalhos futuros.

Inicialmente é necessário estudar e avaliar as especificações do *framework* a ser aplicado ao ***FW-15***. Posteriormente, na fase de análise, é necessário avaliar o comportamento com respeito ao metamodelo ***FW-15*** de acordo com seu tipo e categoria. Durante a fase de projeto é feita a definição de quais estereótipos e regras deverão ser implementados, suas características e detalhes. Então, a implementação deverá ser feita com o apoio da própria ferramenta criada para atender ao método, finalizando-se com uma fase de verificação, validação e testes. Finalmente o **Perfil FrameWeb** integrante da **Definição de Framework** poderá ser aplicado e usado para criar modelos ***FrameWeb*** a partir do *framework* em questão.

Além destas etapas, sugere-se o uso de uma nomenclatura padrão para melhor identificar as bibliotecas pelos *frameworks* e suas categorias, como por exemplo: usando um prefixo para indicar a categoria do *framework* (`"MVC"`, `"ORM"`, `"DI"`, etc.) ou a palavra `"Standard"` quando se tratar de um padrão, seguido do caractere `"_"` e o nome do *framework* ou padrão.

Neste trabalho, diversas versões iniciais de bibliotecas para *frameworks* e padrões foram desenvolvidas, por exemplo: `MVC_Struts.frameweb`, `MVC_Struts2.frameweb`, `MVC_JSF.frameweb`, `MVC_VRaptor3.frameweb`, `MVC_VRaptor4.frameweb`, `ORM_hibernate.frameweb`, `Standard_HTML4.frameweb`, `Standard_HTML5.frameweb` e `Standard_JSTL.frameweb`. Todas as bibliotecas criadas, estão disponíveis para uso ou colaboração no repositório do projeto[14].

3.4 Decisões de projeto

Primeiramente, como cada **Definição de Framework** é uma aplicação de um **Perfil FrameWeb** a um modelo ***FrameWeb***, parece em um primeiro momento, mais razoável trabalhar diretamente com perfis UML para criar uma implementação válida, ao invés de criar uma DSL para formalizar a linguagem ***FW-15***. Entretanto esta abordagem não permitiria a extensibilidade e flexibilidade desejada, de acordo com o que foi apresentado na Seção 3.3.

Por outro lado, ao se criar uma DSL deve-se decidir entre: (i) partir do zero; ou (ii) usar uma infraestrutura já pronta como suporte inicial. Esta decisão está envolta em diversas controvérsias, havendo opiniões contrárias e favoráveis.

Definir a DSL a partir do zero, criando o metamodelo ***FrameWeb*** sem influência do metamodelo UML tem a vantagem de tornar a linguagem independente. Porém, para isso, é necessário reproduzir uma série de partes do metamodelo UML no metamodelo ***FrameWeb*** a fim de manter a compatibilidade necessária, além de ter que garantir sua correta implementação. Outra desvantagem na construção de uma ferramenta baseada neste metamodelo: perde-se (ou dificulta-se bastante) a possibilidade de evoluir o método em relação a outros diagramas pertencentes a UML e que futuramente possam ser incorporados ao método.

Partir do metamodelo UML e sob ele desenvolver o metamodelo ***FrameWeb*** tem a desvantagem de criar uma dependência da versão UML adotada, ou seja, alterações na UML podem exigir modificações no metamodelo ***FrameWeb***. Por outro lado, não seria necessário criar toda uma infraestrutura básica para a linguagem, já que o metamodelo UML já contempla esta infraestrutura. Além disso, garante-se 100% de compatibilidade com a UML sem necessidade de maiores verificações e validações. Esta abordagem também permite que uma ferramenta baseada neste metamodelo seja capaz de produzir qualquer diagrama UML, não só aqueles diretamente relacionados ao método.

É importante também avaliar a inter-relação entre o metamodelo UML e a DSL. A Figura 9 mostra o isomorfismo entre a linguagem do método ***FW-15*** e a UML. Podemos notar que a linguagem ***FW-15*** é bastante isomórfica aos diagramas de classe UML, visto

[14] <https://github.com/bfmartins/FrameWeb-Martins-2015>

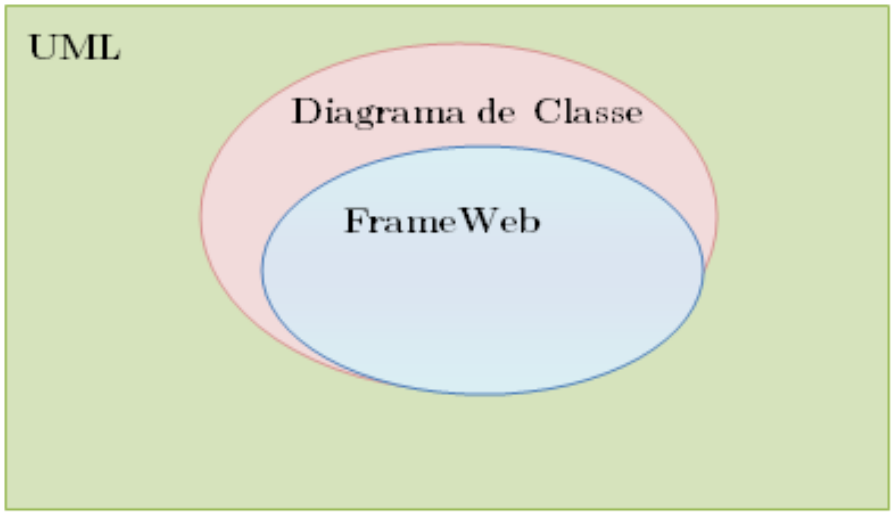


Figura 9 – Isomorfismo entre a linguagem ***FW-15*** e a UML.

que os diversos elementos de um Sistema de Informação Web são representados como classes e suas relações, havendo apenas algumas restrições tratadas em regras específicas como será mostrado no Capítulo 4.

Isto posto, os seguintes motivos foram considerados na opção adotada para este trabalho, na qual o metamodelo ***FrameWeb*** é baseado e fundamentado sob o metamodelo UML:

1. Porque ***FW-07*** foi definido com base na UML, uma linguagem consagrada e difundida entre os modeladores;
2. Para manter a compatibilidade entre as versões ***FW-07*** (original) e ***FW-15*** (proposta);
3. Devido a representação dos modelos propostos pelo método ***FrameWeb*** nos diagramas de classe UML (Figura 9);
4. Considerando que o metamodelo ***FW-15*** possui tamanho e complexidade considerável, o que poderia interferir nos prazos deste trabalho;
5. Avaliando que as alterações de versão da UML não são drásticas nem frequentes; e
6. Pela flexibilidade proporcionada com a possibilidade de definir **Perfis FrameWeb** baseados em perfis UML.

3.5 Conclusões do Capítulo

O metamodelo ***FW-15*** apresenta diversos aspectos importantes a serem analisados, em função não só de sua complexidade no que diz respeito à representação, mas também

no que tange à sua fundamentação técnica. Por conseguinte, se fez necessário aprofundar a discussão considerando os diversos aspectos de representação e técnicos que foram avaliadas durante o período deste trabalho.

Portanto, as principais decisões tomadas neste trabalho, relacionam-se aos seguintes aspectos de representação:

- A necessidade de amplo conhecimento em diversos *frameworks* no seu funcionamento, estrutura e representação;
- A representação de vários *frameworks*, mesmo quando diferenciados em categorias, representa um desafio, já que é necessário tratar o que é comum de forma semelhante e o que é distinto de forma específica;
- A definição quanto ao escopo em relação à representação do método, ou seja, representar todo ou apenas parte do método;
- A definição do escopo em relação às novas tecnologias a serem incorporadas ao método;
- A definição do escopo de abrangência do estudo, tanto para quais *frameworks*, quanto as categorias serem usadas no trabalho;
- O conhecimento de que a metodologia para extensão do método deve permitir replicação, de forma a ser utilizada tanto para adição de novos *frameworks* para as categorias já existentes, como de que o metamodelo deve permitir expansão no que diz respeito a outras categorias a serem adicionadas.

As principais decisões tomadas neste trabalho portanto, relacionam-se aos seguintes aspectos técnicos:

- Uso ou não do metamodelo UML como ponto de partida ou a definição de uma DSL a partir do zero;
- Estudo e avaliação da melhor abordagem no que diz respeito ao uso dos construtos UML;
- Escolha e estudo das ferramentas e tecnologias a serem utilizadas no desenvolvimento dos trabalhos;
- A implementação de um protótipo simples ou mais complexo, e as implicações desta escolha quanto a observância em relação ao escopo adotado.

Todas estas questões foram consideradas na implementação do metamodelo ***FW-15*** apresentado no Capítulo 4, a seguir.

4 Abordagem dirigida por modelos para o método FrameWeb

A abordagem dirigida por modelos que está apresentada neste capítulo é o resultado de um processo conduzido ao longo do curso de mestrado, envolvendo uma série de decisões relativas ao escopo do trabalho, análise dos requisitos, à implementação do metamodelo, etc. já apresentadas no Capítulo 3.

Este capítulo apresenta a formalização da linguagem ***FrameWeb*** e a evolução proposta na versão ***FW-15*** do método, utilizando uma abordagem dirigida a modelos. Na Seção 4.1 é apresentada a sintaxe abstrata da linguagem. Na Seção 4.2 encontram-se os detalhes da abordagem independente de plataforma sugerida para o método. Na linguagem proposta também estão definidos recursos para adição automática dos aspectos dependentes de plataforma, que são mostrados na Seção 4.3. Finalmente, na Seção 4.4, apresentam-se os detalhes relacionados à configuração, preparação do ambiente e implementação do metamodelo, bem como as diretrizes para o desenvolvimento de uma ferramenta gráfica para o método.

4.1 Formalização da linguagem FrameWeb

Uma sintaxe definida, em si, não é suficiente como definição formal de uma linguagem, ou seja, é necessário que se forneça significado para cada elemento dessa linguagem. Sendo assim, e tratando mais especificamente de linguagens de modelagem, dado qualquer modelo criado nesta linguagem, deve-se fornecer significado para cada representação gráfica dentro destes modelos, o que na realidade constitui a semântica da linguagem.

No desenvolvimento de uma Linguagem Específica de Domínio (*Domain Specific Language*, ou DSL), as técnicas de Desenvolvimento Orientado a Modelos (*Model-Driven Development*, ou MDD) (PASTOR et al., 2008) preconizam que sua sintaxe abstrata é na verdade um metamodelo, que descreve seus conceitos e regras de modelagem. Assim sendo, a gramática desta DSL passa a ser representada neste metamodelo. Nesta gramática está contida a sintaxe da linguagem. Nesta seção, apresentamos o metamodelo que corresponde, então, à sintaxe abstrata da linguagem ***FrameWeb***.

Para facilitar a integração entre os modelos da MDD, as linguagens são classificadas em um determinado nível de abstração, partindo do nível mais abstrato, que são metametalinguagens para descrever linguagens, que por sua vez são metalinguagens para descrever linguagens e assim sucessivamente até alcançar o nível menos abstrato.

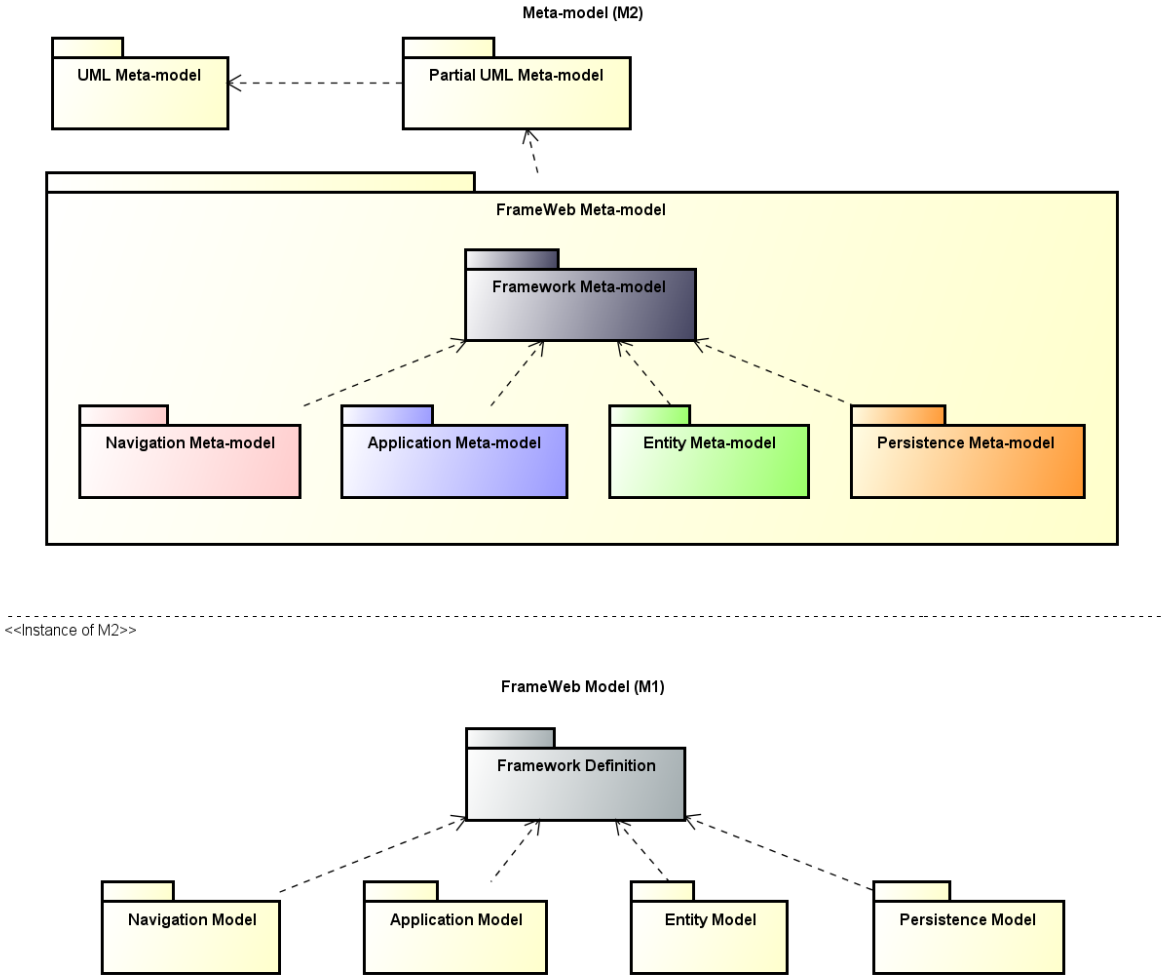


Figura 10 – Níveis de abstração M2 e M1 da linguagem ***FW-15***.

Assim, definiu-se o metamodelo ***FrameWeb*** no metanível mais alto (M_2) e dependente do metamodelo UML, como mostra a Figura 10. O metamodelo ***FrameWeb*** especifica a linguagem, que por sua vez governa a criação dos modelos do método.

Entretanto, sabendo-se que os diagramas de classe são parte isomórfica entre os metamodelos UML e ***FrameWeb***, compondo apenas uma parte menor do metamodelo UML, há um pacote intermediário chamado **Metamodelo Parcial UML** (`Partial UML Meta-model`, representado na Figura 10) para conter apenas as metaestruturas que serão usadas, e facilitar a visualização dos desenvolvedores. O metamodelo completo, que contém os elementos UML utilizados no metamodelo ***FrameWeb*** está disponível no repositório do projeto[1].

Com base no **Metamodelo Parcial UML**, e sob ele, foi criado o metamodelo ***FrameWeb*** contendo toda a sintaxe abstrata da linguagem que formaliza o método. Primeiramente este metamodelo formalizou o método ***FW-07*** original, e posteriormente,

[1] <https://github.com/bfmartins/FrameWeb-Martins-2015>

a partir desta versão, as evoluções necessárias foram implementadas para produzir o metamodelo ***FW-15***.

Este último é bastante diferente de sua versão anterior, especialmente porque a nova versão é composta de novos tipos de dados, que variam conforme o *framework* selecionado e não apenas de estereótipos e restrições (*constraints*). Esta variação motivou a separação da sintaxe abstrata do método em duas partes distintas, independente de *framework* e dependente de *framework*, detalhadas nas seções mais à frente.

A Figura 10 também exibe graficamente os pacotes do metamodelo ***FW-15***: Metamodelo de Navegação (`Navigation Meta-model`), Metamodelo de Aplicação (`Application Meta-model`), Metamodelo de Entidades (`Entity Meta-model`), Metamodelo de Persistência (`Persistence Meta-model`), todos estes dependentes do Metamodelo de Framework (`Framework Meta-model`).

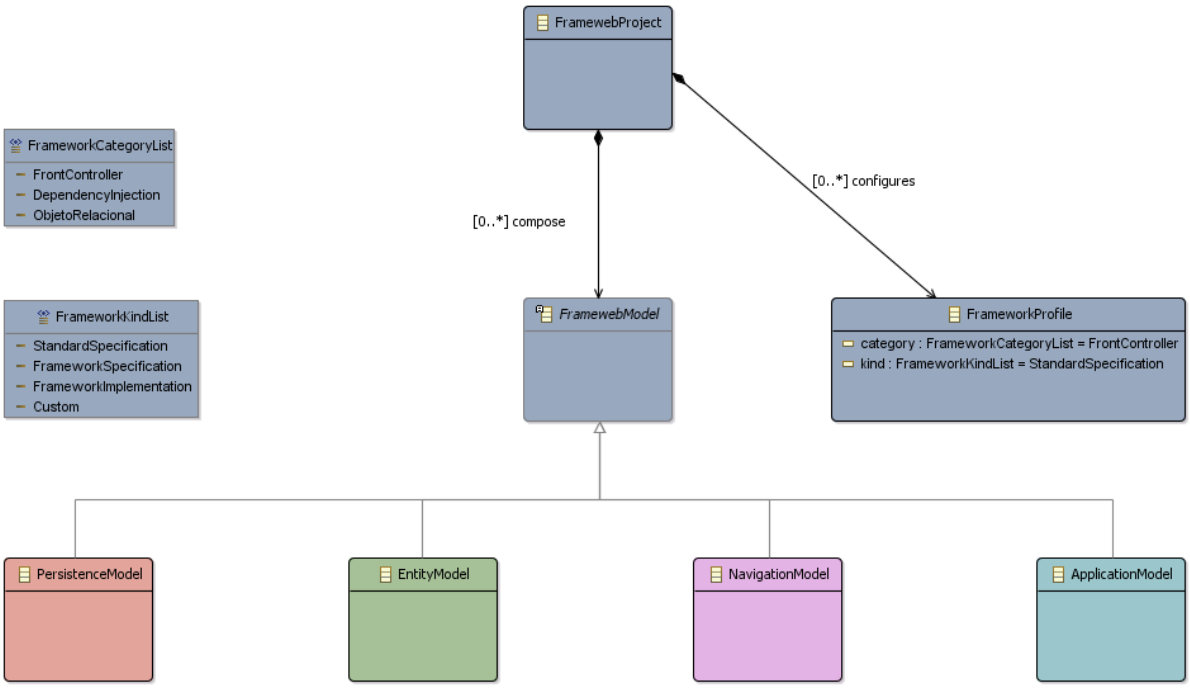


Figura 11 – *"Top-level containment"* da linguagem ***FW-15***.

Além disso os pacotes do metamodelo ***FW-15*** relacionam-se conforme mostrado no fragmento do metamodelo ***FW-15*** da Figura 11. Neste caso, podemos observar as metaclasses `NavigationModel`, `EntityModel`, `ApplicationModel` e `PersistenceModel`, como especializações de *`FramewebModel`*. Também aparece nesta figura a metaclasse `FrameworkProfile`, usada nas **Definições de Framework** e que juntamente com os modelos ***FrameWeb*** compões projetos ***FrameWeb*** (`FramewebProject`). Maiores detalhes encontram-se nas seções 4.2 e 4.3.

O nível M_1, apresentado na Figura 10, representa um ponto de vista imediatamente abaixo do nível M_2, no qual os modelos ***FrameWeb*** (definidos pela metaclasse *FramewebModel* da Figura 11) são representados como instâncias do metamodelo ***FW-15***, ou seja:

- **Definição de Framework** (Framework Definition) representa as instâncias do **Metamodelo de Framework** (Framework Meta-model);
- **Modelo de Navegação** (Navigation Model) representa as instâncias do **Metamodelo de Navegação** (Navigation Meta-model);
- **Modelo de Entidades** (Entity Model) representa as instâncias do **Metamodelo de Entidades** (Entity Meta-model);
- **Modelo de Aplicação** (Application Model) representa as instâncias do **Metamodelo de Aplicação** (Application Meta-model);
- **Modelo de Persistência** (Persistence Model) representa as instâncias do **Metamodelo de Persistência** (Persistence Meta-model).

O nível de abstração mais baixo (M_0) não está representado na Figura 10, porque não se trata exatamente de uma instanciação imediata do nível M_1. Na realidade em M_0 está uma dada aplicação modelada de acordo com os modelos definidos em M_1, um Sistema de Informação Web projetado com o método ***FrameWeb***. Isso se dá porque, apesar de um dado Sistema de Informação Web ser feito com base nos modelos gerados em M_1, este processo não ocorre por meio de uma aplicação que diretamente usa as classes dos modelos criado, instanciando-as em tempo de execução.

Por exemplo, em M_2 o metamodelo juntamente com o código produzido a partir dele permitem que, em tempo de execução, instâncias das metaclasses definidas sejam criadas para produzir os modelos em M_1. Desse processo tem-se como resultado arquivos de extensão **.frameweb** que armazenam os modelos criados (instâncias de M_2).

Porém em M_1, os modelos criados (**.frameweb**) servem apenas como base para a implementação em termos de codificação. Ou seja ao definir em um dado modelo uma classe *"Pessoa"*, por exemplo, o resultado obtido a partir de M_1 é o código fonte da classe e não propriamente uma instancia da classe *"Pessoa"* em tempo de execução.

Para facilitar o entendimento do disposto acima, apresentamos na Figura 12 um mapa conceitual proposto para o método ***FW-15***, que explica como os níveis M_2 e M_1 se relacionam com o nível M_0. Ao observar a figura, é possível notar que o metamodelo define a linguagem ***FrameWeb***, usada para dirigir a criação dos modelos ***FrameWeb***, que por sua vez são artefatos de modelagem produzidos com o apoio de uma a ferramenta de modelagem construída sob este metamodelo.

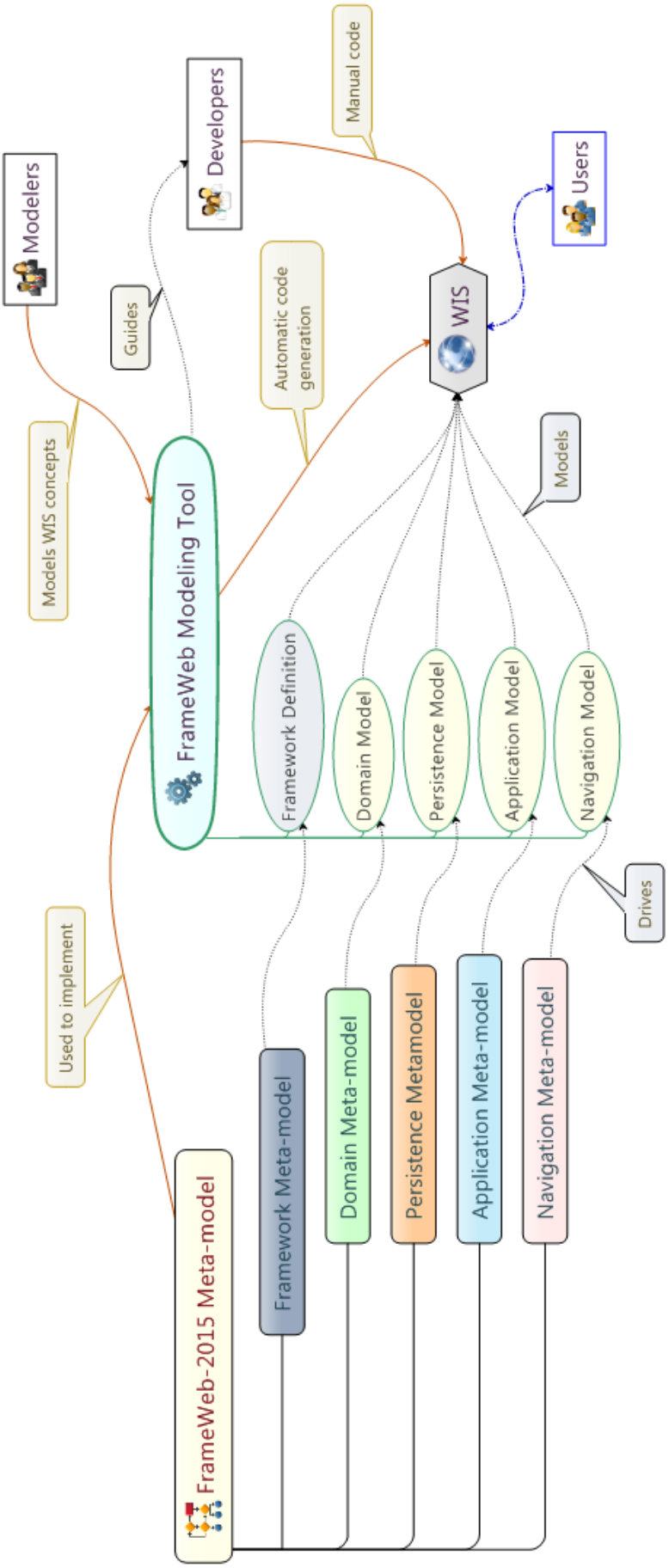


Figura 12 – Mapa conceitual do método ***FW-15***.

Os modelos ***FrameWeb*** são produzidos por modeladores durante a fase de projeto de um Sistema de Informação Web e usados por desenvolvedores na fase de desenvolvimento deste Sistema de Informação Web, também podendo ser base para funcionalidades mais avançadas capazes de efetuar geração total ou parcialmente automática de código, dependendo da capacidade da ferramenta de modelagem desenvolvida.

4.2 O Metamodelo FrameWeb

A sintaxe independente de *framework* gerencia a criação dos modelos sob um ponto de vista geral, ou seja, independentemente dos *frameworks* ou padrões instanciados que forem usados na modelagem. Isso significa dizer que todos os modelos criados no nível **M1** usando a linguagem ***FW-15*** necessitam seguir as mesmas regras, independentemente das escolhas do modelador.

Como as metaclasses do metamodelo ***FW-15*** foram especializadas a partir das metaclasses UML contidas no **Metamodelo Parcial UML**, se fez necessário um estudo mais profundo do metamodelo UML no que diz respeito aos diagramas de classe, que são usados pelo método.

Posteriormente foi possível definir as melhores práticas no desenvolvimento do metamodelo ***FW-15***, filtrando os recursos excedentes para despoluir a visão do modelador, sem, no entanto, prejudicar a compatibilidade com a UML.

As principais especializações de metaclasses filtram elementos dos diagramas de classe UML, como: `Package`, `Class`, `Dependency`, `Generalization`, `Association`, `Property`, `Operation` e outros que estejam no contexto do ***FW-15***. Desta forma, grande parte das regras definidas pelo metamodelo ***FW-15*** são semelhantes, motivo pelo qual não estão todas apresentadas na sequencia do texto. Ao contrário, está mostrado apenas um conjunto mais representativo destas regras, necessário para melhor entendimento do trabalho.

No que diz respeito aos modelos ***FrameWeb***, a maior parte dos experimentos e estudos foi feita no contexto dos **Modelos de Navegação** (`Navigation Model`), produzidos na construção de aplicações desenvolvidas por alunos de graduação. Este modelo foi escolhido por sua maior complexidade e pela enorme gama de *frameworks* existentes para a camada da arquitetura do sistema que este modelo representa.

Por esse motivo, bem como para simplificar e reduzir as explicações necessárias ao entendimento do trabalho, na sequencia desta seção, dar-se-á ênfase à apresentação do **Metamodelo de Navegação** (`Navigation Meta-model`). Porém, o metamodelo completo ***FW-15*** está implementado no repositório do projeto para consulta e cooperação.

As subseções a seguir apresentam o metamodelo ***FW-15***, cada qual focando em metaclasses específicas da UML, que foram especializadas neste trabalho.

4.2.1 Diagramas, Pacotes e Classes

O primeiro conjunto de metaclasses UML especializadas no metamodelo ***FrameWeb*** destina-se à representação dos pacotes e das classes. Como já mostrado na Seção 2.1, os **Modelos de Navegação** apresentam os aspectos da **Camada de Apresentação** e os dois pacotes contidos nela são **visão** e **controle**. Assim, as metaclasses `ViewPackage` e `ControlPackage` são especializações da metaclasse ***`NavigationPackage`***, que por sua vez é uma especialização da metaclasse UML `Package`, como mostrado na Figura 13.

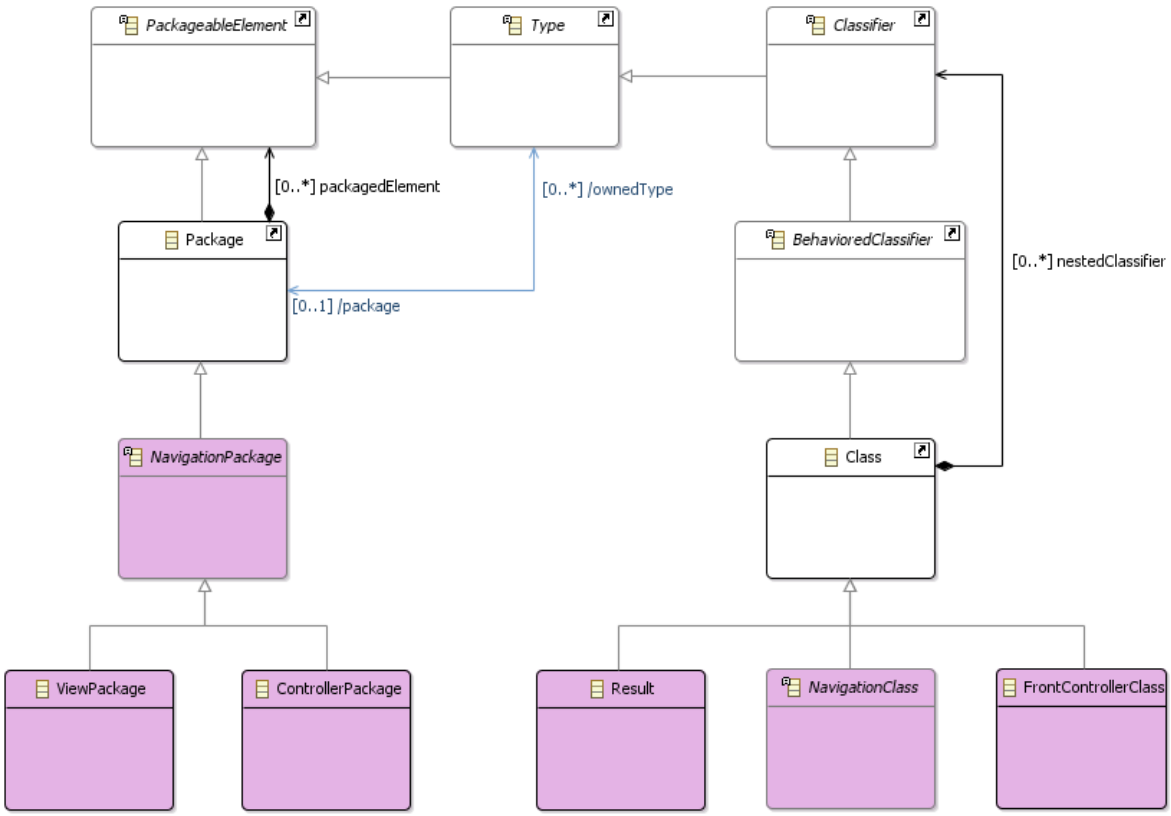


Figura 13 – Fragmento do Metamodelo de Navegação do ***FW-15***: pacotes.

Nos **Modelos de Navegação**, as classes UML são usadas na representação de elementos para navegação dos Sistemas de Informação Web, como páginas, componentes de interface com o usuário, modelos (*templates*), ou para representação de classes do controlador frontal (*framework MVC*).

Isso pode ser observado no fragmento do **Metamodelo de Navegação** da Figura 13, onde classes UML (`Class`) são especializadas nas metaclasses: ***`NavigationClass`*** para representação dos diversos elementos para navegação dos Sistemas de Informação Web; `FrontControllerClass` representando as classes do controlador frontal; e `Result`

para representação dos diversos artefatos que podem ser obtidos ou produzidos como resultado das operações efetuadas por métodos das classes do controlador frontal.

As estruturas dos modelos ***FrameWeb*** instanciáveis a partir de especializações feitas sob a metaclasse UML `Class` relacionam-se de diversas maneiras, variando conforme a representação que se deseja. Estes relacionamentos são derivados das metaclasses UML relacionais (dependências, generalizações, etc.) aplicáveis entre classes em diagramas de classe, adicionando-se a eles uma semântica adequada no contexto do método ***FrameWeb***.

Desta forma, no contexto dos relacionamentos diretos UML (*`DirectRelationship`*), o metamodelo ***FW-15*** especializa as metaclasses UML `Dependency` em suas dependências e `Generalization` em suas especializações, conforme está detalhado adiante nas seções 4.2.2 e 4.2.3.

Adicionalmente, especializações feitas sob a metaclasse UML `Class` podem ser estereotipadas para assumirem diferentes comportamentos conforme a combinação de *frameworks* selecionada pelo modelador. Maiores explicações estão detalhadas na seção 4.3 onde os aspectos dependente de *framework* são tratados.

No fragmento do **Metamodelo de Navegação** da Figura 14 é mostrado que a metaclasse das classes de navegação (*`NavigationClass`*) por sua vez, pode representar páginas (`Page`), modelos (`Template`) ou componentes de interface (`UIComponent`). Além disso também estão representadas as operações permitidas para os componentes de interface, cuja especialização representam os métodos (`FrontControllerMethod`) permitidos nas classes do controlador frontal (`FrontControllerClass`).

A princípio toda especialização de `Class` permite a inclusão de métodos (`Operation`), mas com efeito ao metamodelo ***FW-15***, apenas classes do controlador frontal (`FrontControllerClass`) devem conter métodos (`FrontControllerMethod`), enquanto classes de navegação não. Esta regra não está clara com base no fragmento do metamodelo da Figura 14, porque a linguagem gráfica não é suficiente para esta representação, como será discutido logo a seguir nesta seção.

O mesmo pode ser observado no fragmento do **Metamodelo de Entidades** da Figura 15, onde está apresentada a metaclasse das entidades do domínio do Sistema de Informação Web (`DomainClass`), que pode possuir operações (`DomainMethod`) e pertence ao pacote das entidades de domínio do Sistema de Informação Web (`DomainPackage`).

Além dos aspectos já apresentados, para que a sintaxe abstrata esteja completa também se faz necessária a criação de um conjunto de regras OCL[2], porque a linguagem gráfica usada no desenvolvimento do metamodelo não é capaz de expressar todas as restrições necessárias para a definição da linguagem ***FW-15***.

[2] <http://www.omg.org/spec/OCL/2.4/PDF/>

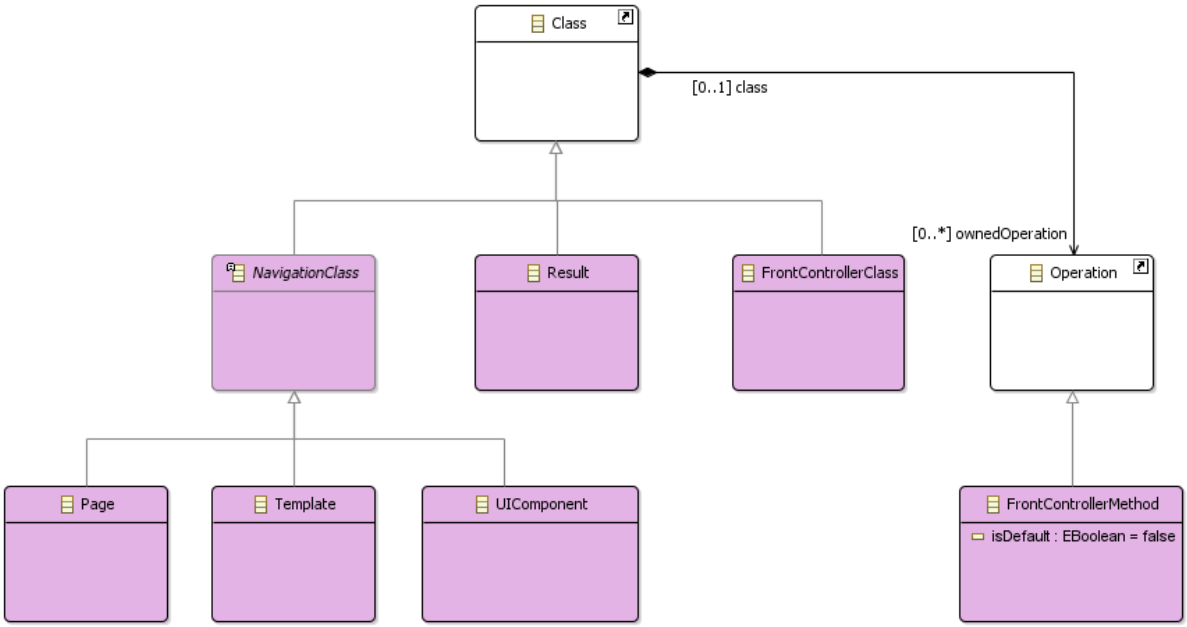


Figura 14 – Fragmento do Metamodelo de Navegação do ***FW-15***: classes.

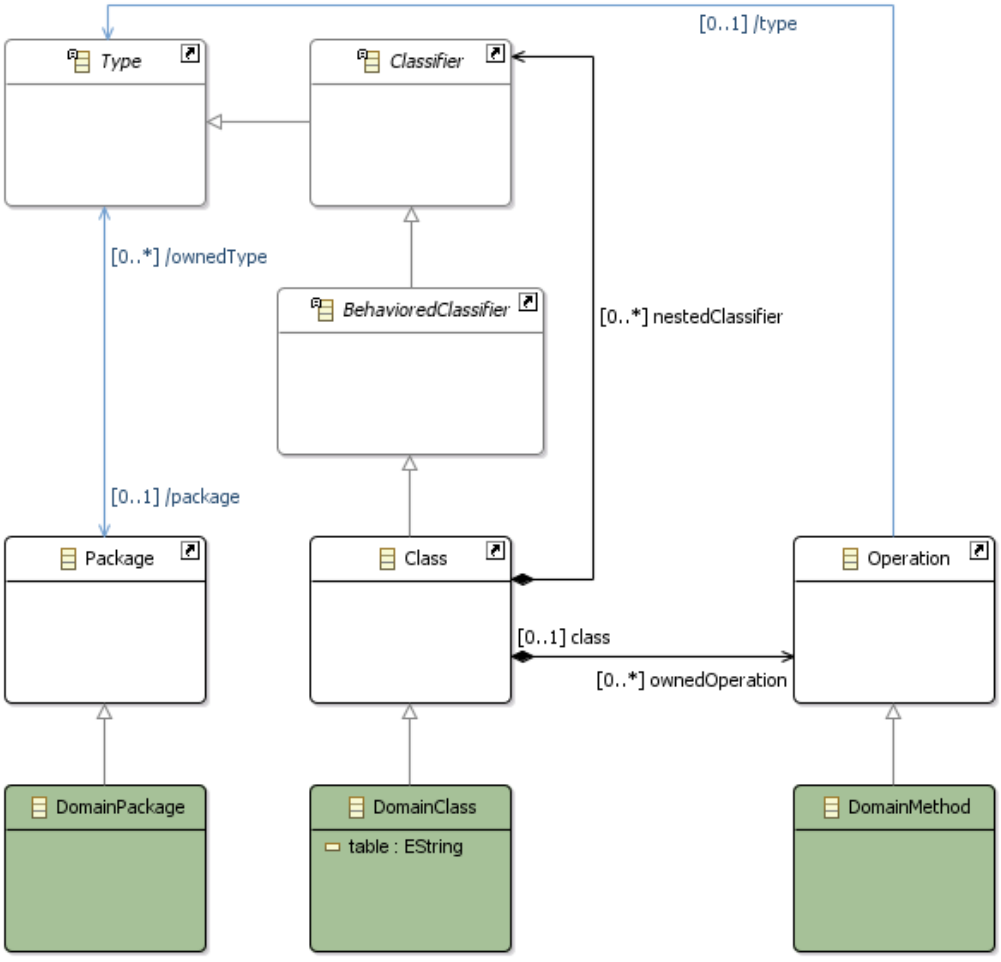


Figura 15 – Fragmento do **Metamodelo de Entidades** do ***FW-15***: classes.

A maior parte das regras OCL da sintaxe independente de *framework*, estão diretamente associadas a restrições e filtros aplicados sob o metamodelo UML para dirigir o modelador enquanto implementa os modelos do método. Desta forma, há um conjunto de regras OCL aplicadas ao metamodelo ***FW-15*** para gerenciar o comportamento da linguagem no que diz respeito às metaclasses contidas no pacote UML Parcial (`Partial UML Meta-model`). Estas regras tem aspectos gerais e por causa disso fazem parte da sintaxe independente de *framework*.

Acompanhando as figuras 13, 14 e 15, para que **Modelos de Navegação** e **Modelos de Entidades** contenham somente aquilo que lhes é permitido e nada mais, as regras OCL das listagens 4.1 e 4.2 foram incorporadas ao metamodelo ***FW-15***. Assim, podemos observar na Listagem 4.1, que as linhas 2, 9 e 19 mostram os contextos de aplicação das regras de restrição.

Listagem 4.1 – Regras OCL no contexto das figuras 13 e 14.

```
1  -- A NavigationModel must have only NavigationPackage.
2  context NavigationModel
3    inv NavigationModelContent:
4      (self.packagedElement.oclIsTypeOf(NavigationPackage)
5    inv NavitationModelMinimum:
6      (self.packagedElement->size()>=2)
7
8  -- A ViewPackage must have only NavigationClass or Result or DirectRelationship
       or Association.
9  context ViewPackage
10   inv ViewPackageContent:
11     ((self.packagedElement.oclIsTypeOf(NavigationClass)) or
12     (self.packagedElement.oclIsTypeOf(Result)) or
13     (self.packagedElement.oclIsTypeOfNavigationDependency)) or
14     (self.packagedElement.oclIsTypeOf(NavigationAssociation)))
15   inv ViewPackageMinimun:
16     (self.packagedElement->size()>=1)
17
18 -- A ControllerPackage must have only FrontControllerClass or Dependency or
       Generalization or Association.
19 context ControllerPackage
20   inv ControllerPackageContent:
21     ((self.packagedElement.oclIsTypeOf(FrontControllerClass)) or
22     (self.packagedElement.oclIsTypeOf(NavigationDependency)) or
23     (self.packagedElement.oclIsTypeOf(NavigationGeneralization)) or
24     (self.packagedElement.oclIsTypeOf(NavigationAssociation)))
25   inv ControllerPackageMinimun:
26     (self.packagedElement->size()>=1)
```

No contexto do **Modelo de Navegação** (`NavigationModel`, linha 2), são permitidos apenas **Pacotes de Navegação** como itens de pacote UML (***`PackageableElement`***). Da mesma forma, no contexto do **Pacote de Visão** (`ViewPackage`, linha 9), são permitidos apenas **Classe de Navegação** (***`NavigationClass`***), **Retorno** (`Result`), associações (`NavigationAssociation`) ou dependências (***`NavigationDependency`***). E no contexto do **Pacote de Controle** (`ControlPackage`, linha 19), são permitidos apenas **Classe do Controlador Frontal** (`FrontControllerClass`), associações (`NavigationAssociation`), dependências (***`NavigationDependency`***) ou generalizações (`NavigationGeneralization`).

Listagem 4.2 – Regras OCL no contexto da Figura 15.

```
1 -- A EntityModel must have only DomainPackage.
2 context EntityModel
3   inv EntityModelContent:
4     (self.packagedElement.oclIsTypeOf(DomainPackage)
5   inv EntityModelMinimum:
6     (self.packagedElement->size()>=1)
7
8 -- A DomainPackage must have only DomainClass or DomainGeneralization or
      DomainAssociation.
9 context DomainPackage
10  inv DomainPackageContent:
11    (self.packagedElement.oclIsTypeOf(DomainClass)) or
12    (self.packagedElement.oclIsTypeOf(DomainGeneralization)) or
13    (self.packagedElement.oclIsTypeOf(DomainGeneralizationSet)) or
14    (self.packagedElement.oclIsTypeOf(DomainAssociation)))
15  inv DomainPackageMinimun:
16    (self.packagedElement->size()>=1)
```

É importante ressaltar que as regras OCL da Listagem 4.1 além de filtrar especializações de `Class`, também filtram os relacionamentos permitidos entre estas classes nos modelos ***FrameWeb***.

Semelhante ao que foi mostrado na Listagem 4.1 e aplicado no **Metamodelo de Navegação**, a Listagem 4.2 exibe as regras de restrição no **Metamodelo de Entidades**. As mesmas considerações são extensíveis ao **Metamodelo de Aplicação** e ao **Metamodelo de Persistência** e estão disponíveis no repositório do projeto. O Apêndice A apresenta também a lista de regras OCL associadas ao metamodelo.

4.2.2 Generalizações (Especializações)

A semântica das relações de generalização da linguagem ***FW-15*** não difere da semântica UML, mas via de regra nem toda estrutura da linguagem ***FW-15*** as permite.

A Figura 16 apresenta as especializações de relacionamentos diretos UML (***Direct-Relationship***) do metamodelo ***FW-15***. Dentre elas, estão as generalizações permitidas: `NavigationGeneralization` para permitir especialização de classes do controlador frontal (`FrontControllerClass`); `DomainGeneralization` para permitir especialização de entidades do domínio (`DomainClass`); e `ServiceGeneralization` para permitir especialização de classes e interfaces de serviço.

Semelhantemente ao exposto na Seção 4.2.1, as generalizações permitidas possuem filtros específicos e além de pertencerem especificamente a certos modelos ***FrameWeb***, também só podem ocorrer entre classes específicas.

Por sua característica relacionada à interface com o usuário, o **Metamodelo de Navegação** não permite generalizações ao tratar dos recursos contidos no pacote de visão (`ViewPackage`) da **Camada de Apresentação**. Esta restrição fica evidente ao observarmos a Listagem 4.1, na qual a metaclasse `Generalization`, ou qualquer

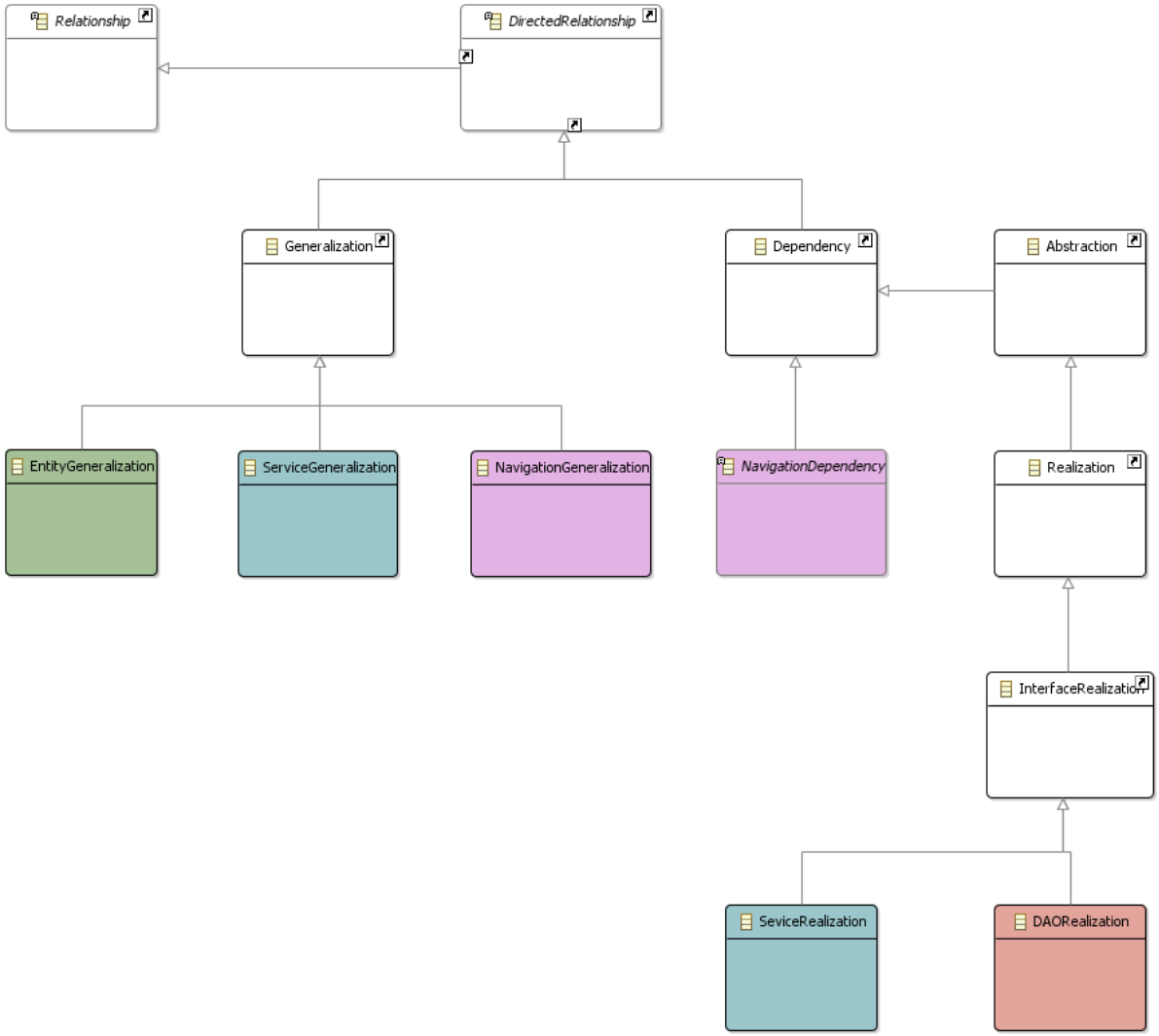


Figura 16 – Fragmento do Metamodelo de UML Parcial do ***FW-15***: relacionamentos diretos.

especialização dela, não aparece como um tipo permitido em pacotes de visão, que são especializações do (*NavigationPackage*).

De fato, as estruturas de navegação não são classes propriamente ditas, contudo como o método ***FW-07*** definiu sua representação como classes estereotipadas, estes elementos são assim tratados no metamodelo ***FW-15***. Já que, são requisitos deste trabalho manter a compatibilidade entre as versões ***FrameWeb***, bem como sua característica de representação sob o metamodelo UML.

Por outro lado, dentro do pacote de controle (`ControlPackage`), é permitido que classes do controlador frontal participem em relações de generalização entre si, conforme pode ser observado na linha 23 da Listagem 4.1 (apresentada anteriormente), desde que obedeçam às restrições impostas na Listagem 4.3, adiante.

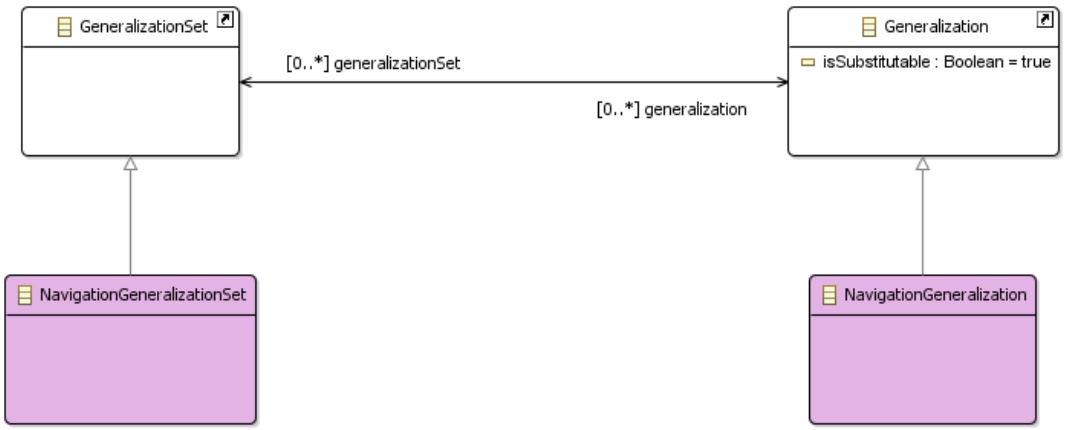


Figura 17 – Fragmento do Metamodelo de Navegação do ***FW-15***: generalizações.

A Figura 17 apresenta as generalizações permitidas no **Metamodelo de Navegação**, mais especificamente aquelas pertencentes ao `ControlPackage`, que ocorrem apenas entre as classes derivadas da metaclasse `FrontControllerClass`, incluindo igualdade entre os estereótipos que possam ser aplicados a elas. Estas generalizações devem estar associadas à metaclasse `NavigationGeneralizationSet`, que são especializações de `GeneralizationSet` da UML, conforme regras apresentadas na Listagem 4.3.

Listagem 4.3 – Regras OCL no contexto da Figura 17.

```
1 -- A NavigationGeneralization must be related only with
      NavigationGeneralizationSet
2 context NavigationGeneralizationSet
3   inv NavigationGeneralizationSetRelation
4     (self.generalization.oclIsTypeOf(NavigationGeneralization))
5
6 -- A NavigationGeneralization must have the same class kind as source and target
      (FrontControllerClass)
7 context NavigationGeneralization:
8   inv NavigationGeneralizationRelation
9     (self.generalizationSet.oclIsTypeOf(NavigationGeneralizationSet)) and
10    (self.source.oclIsTypeOf(FrontControllerClass)) and
11    (self.target.oclIsTypeOf(FrontControllerClass))
```

Também por sua característica relacionada ao domínio de Sistemas de Informação Web, o **Metamodelo de Entidades** permite generalizações com a mesma semântica UML a partir da metaclasse `DomainGeneralization`, como mostrado na Figura 18. Estas generalizações devem estar associadas à metaclasse `DomainGeneralizationSet`.

Além disso, é necessário definir o comportamento das especializações entre entidades do domínio do Sistema de Informação Web no âmbito do **Modelo de Entidades** especificando-se como deverá ocorrer o mapeamento através do atributo `mapping`, que é utilizado para definir a hierarquia da herança das entidades. Ou seja, define se os dados serão persistidos de forma conjunta ou individualmente, e neste segundo caso se terão ou não intersecção.

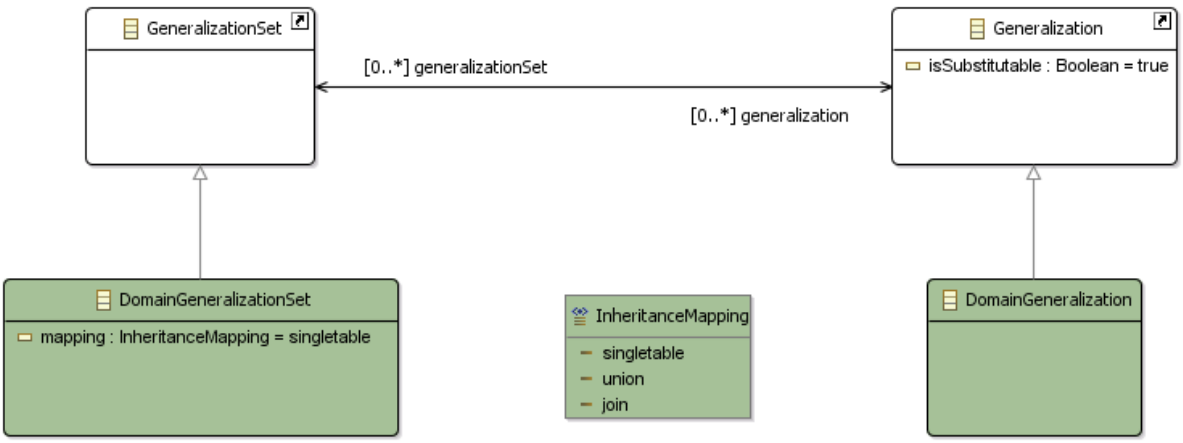


Figura 18 – Fragmento do Metamodelo de Domínio do ***FW-15***: generalizações.

Assim o atributo `mapping` pode assumir os valores segundo os valores enumerados de `InheritanceMapping`: `singletable` para mapeamento de todos os dados em uma única estrutura de representação, normalmente identificada pelo termo "*singletable*", porque em geral esta representação ocorre fisicamente em uma única tabela de um banco de dados; `union` para armazenar os dados em estruturas semelhantes e separadas para as especializações, cuja operação de união representa o conjunto dos dados da superclasse, sem que estes existam de fato em uma estrutura para a superclasse; ou *join* para armazenar os dados em estruturas separadas para as especializações e estruturalmente distintas entre si, fazendo com que seja necessário combinar as informações comuns por meio de operações de intersecção.

Listagem 4.4 – Regras OCL no contexto da Figura 18.

```
1  -- A DomainGeneralization must be related only with DomainGeneralizationSet
2  context DomainGeneralizationSet
3    inv DomainGeneralizationSetRelation
4      (self.generalization.oclIsTypeOf(DomainGeneralization))
5
6  -- A DomainGeneralization must have the same class kind as source and target
7  context DomainGeneralization:
8   inv DomainGeneralizationRelation
9    (self.generalizationSet.oclIsTypeOf(DomainGeneralizationSet)) and
10   (self.source.oclIsTypeOf(DomainClass)) and
11   (self.target.oclIsTypeOf(DomainClass))
```

O **Modelo de Entidades** permite generalizações apenas entre entidades de domínio do Sistema de Informação Web, logo as relações de generalização representadas pela metaclasse `DomainGeneralization` aplicam-se apenas entre metaclasses `DomainClass`. No entanto, não são permitidas especializações de classes diferentes entre si.

Como entidades do domínio do Sistema de Informação Web podem assumir diversos papeis, conforme a seleção do *framework* ORM escolhido pelo modelador e suas especializações devem ser feitas entre classes do mesmo tipo, é necessário adicionar uma

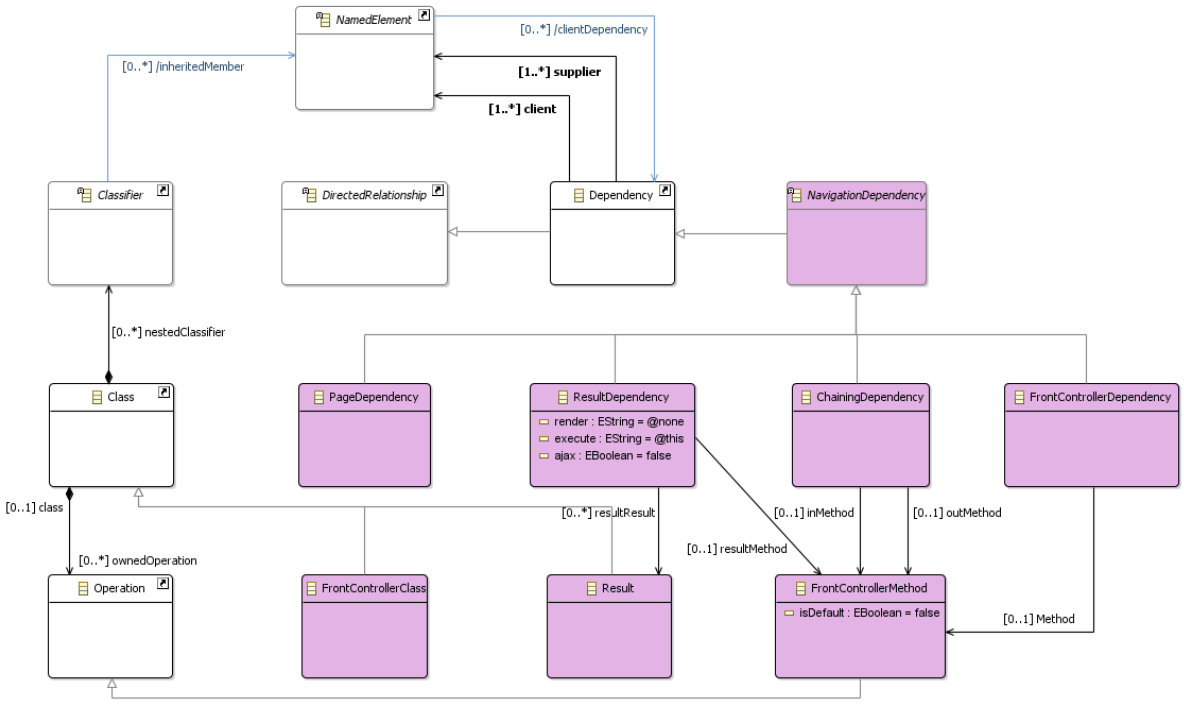


Figura 19 – Fragmento do Metamodelo de Navegação do ***FW-15***: dependências.

regra OCL específica, como mostrado nas linhas 10 e 11 da Listagem 4.4. Os detalhes de como diferentes tipos de classes de domínio são tratadas serão apresentados na Seção 4.3.2, por se tratarem de aspectos dependente de *framework*.

O **Metamodelo de Persistência** não permite generalizações por sua característica relacionada ao padrão DAO (ALUR; MALKS; CRUPI, 2013). Já o **Metamodelo de Aplicação** permite generalizações conforme apresentado na Figura 16 no início desta seção e detalhado no repositório do projeto.

4.2.3 Dependências

A semântica das dependências UML é estendida no metamodelo ***FW-15*** de forma a representar as condições especiais em que determinadas estruturas da **Camada de Apresentação** (**Presentation Tier**) de um Sistema de Informação Web dependem umas das outras.

A Figura 19 exibe um conjunto de dependências (*`NavigationDependency`*) permitidas entre as **Classes de Navegação** (*`NavigationClass`*) e a **Classe do Controlador Frontal** (`FrontControllerClass`). Estas dependências são especializações da metaclasse `Dependency` da UML, porém possuem características especiais para tratar as especificidades do método ***FW-15***.

As dependências permitidas no **Metamodelo de Navegação** possuem regras OCL detalhadas na Listagem 4.5, responsáveis por restringir as combinações de pares

de metaclasses, que podem ser a fornecedora ("*supplier*") e a cliente ("*client*") no contexto de cada dependência. Por exemplo, na linha 8 temos o contexto da dependência `FrontControllerDependency`, no qual, obrigatoriamente, a classe fornecedora deve ser uma `FrontControllerClass` e a cliente uma ***`NavigationClass`***.

Listagem 4.5 – Regras OCL no contexto da Figura 19.

```
1 -- A ResultDependency must have a FrontControllerClass as client and a
      NavigationClass as supplier.
2 context ResultDependency
3   inv ResultDependencyRelation:
4     (self.client.oclIsTypeOf(FrontControllerClass)) and
5     (self.supplier.oclIsTypeOf(NavigationClass))
6
7 -- A FrontControllerDependencyConstraint must have a NavigationClass as client
      and a FrontControllerClass as supplier.
8 context FrontControllerDependency
9   inv FrontControllerDependencyRelation:
10    (self.supplier.oclIsTypeOf(FrontControllerClass)) and
11    (self.client.oclIsTypeOf(NavigationClass))
12
13 -- A PageDependencyConstraint must have a Page as client and a Page or a Template
       as supplier.
14 context PageDependency
15   inv PageDependencyRelation:
16    (self.client.oclIsTypeOf(Page)) and
17    ((self.supplier.oclIsTypeOf(Page)) or
18    (self.supplier.oclIsTypeOf(Template)))
19
20 -- A ChainingDependencyConstraint must have a FrontControllerClass as client and
      a FrontControllerClass as supplier and these client and supplier can not be
      the same.
21 context ChainingDependency
22   inv ChainingDependencyRelation:
23    (self.supplier.oclIsTypeOf(FrontControllerClass)) and
24    (self.client.oclIsTypeOf(FrontControllerClass)) and
25    (self.supplier <> self.oclAsType(Dependency).client)
```

Observa-se que todas as relações de dependência em que a classe do controlador frontal (`FrontControllerClass`) aparece, independentemente de assumir o papel de fornecedora ou cliente, fazem referência a um método (`FrontControllerMethod`). Este, por sua vez, é uma especialização de propriedades UML (`Property`).

O **Metamodelo de Entidades** não faz uso de dependências e essa restrição está exposta na Listagem 4.2, mostrada anteriormente, onde já há uma limitação a respeito do que pertence ao **Modelo de Entidades**. O mesmo se aplica aos **Metamodelos de Persistência** e **Aplicação**, que também não fazem uso de dependências.

Desta forma e por serem aplicáveis somente no **Metamodelo de Navegação**, as relações de dependência usadas na linguagem ***FW-15***, possuem restrições que indicam uma série de informações importantes relativas à navegação do sistema. Por exemplo, nas dependências relacionadas ao `FrontControllerClass`, onde há a identificação do método que está sendo chamado, o retorno desse método e qual o tipo deste retorno são indicados.

Já as dependências de resultado (`ResultDependency`) possuem restrições para indicar qual o resultado obtido após a execução de um método do `FrontControllerClass` (`FrontControllerMethod`).

Há ainda restrições adicionais relacionadas a novas tecnologias, como por exemplo o `AJAX`, para dependências associadas à metaclasse `UIComponent`. Os detalhes destas restrições estão na Seção 4.2.7.

4.2.4 Associações

Associações (`Association`) em UML não são estruturas tão simples como mostrado anteriormente para dependências (`Dependency`) e generalizações (`Generalization`), por não serem relacionamentos diretos (*`DirectRelationship`*).

Uma associação é uma metaclasse mais complexa (especializada de `Relationship` e *`Classifier`*), assim podem conter características definidas por propriedades (`Property`). Portanto, formalizar associações para o metamodelo ***FW-15*** exige não só o uso de diversas metaclasses, mas também o uso de um conjunto maior de regras OCL, dependendo do caso.

Relações de associação podem ser subdivididas em associações, agregações e composições. A especificação UML[3] deixa em aberto várias questões a respeito do comportamento semântico destes elementos da linguagem, porque permite diversas interpretações no que diz respeito as certas representações gráficas.

Em oposição à abordagem adotada na especificação UML, para Guizzardi (2005), a semântica de uma DSML bem formada deve seguir alguns princípios: (i) *Lucidity*: **toda** entidade na especificação deve representar **no máximo uma** entidade do modelo; (ii) *Soundness*: **cada** entidade na especificação deve representar **pelo menos uma** entidade do modelo; (iii) *Laconicity*: **toda** entidade do modelo deve ser representada por **no máximo uma** entidade na especificação; e (iv) *Completeness*: cada entidade do modelo deve ser representada por **pelo menos uma** entidade na especificação.

Então, cada elemento gráfico da linguagem deve ter um, e somente um, significado semântico e essa semântica não pode ser dada a nenhum outro elemento gráfico da linguagem. Além disso, não deve haver elementos gráficos sem semântica definida, assim como não deve haver significado semântico sem representação. Em suma, não pode haver ambiguidades ou dúvidas dos conceitos representados pelos elementos gráficos da linguagem. Porém isso não ocorre em diagramas de classe UML, ao se fazer uma análise mais detalhada de sua especificação.

Isso fica evidente ao observarmos o comportamento das relações de agregações e/ou composições no que diz respeito à representação da semântica "*Todo/Parte*". Em especial

[3] <http://www.omg.org/spec/UML/2.5/PDF>

porque a UML não trata de aspectos relacionados à transitividade destas relações em sua especificação.

O mesmo ocorre em situações onde as associações representam relações de cardinalidade múltipla para os dois lados (relações "*n*" para "*n*") porque, mesmo quando estas associações são tratadas por meio classes associativas, a gramática UML é muito geral sendo incapaz de expressar as diversas variações possíveis destas relações, consequentemente tratando-as sempre de forma mais abrangente do que o desejável.

Por causa desta característica e sendo a linguagem ***FW-15*** derivada da UML, é importante neste trabalho identificar e tratar as especificidades do método ***FW-15*** com vistas a estas limitações da UML.

Contudo, apesar destas considerações, a especificação UML define algumas restrições básicas e importantes no que diz respeito a associações, que devem ser levadas em consideração ao tratarmos suas especializações na definição da linguagem ***FW-15***.

Primeiramente, em todos os casos é obrigatório que associações tenham pelo menos duas propriedades (`Property`) que assumam o papel de membros desta associação (`memberEnd`).

Além disso, para agregações e composições, só podem haver no máximo duas propriedades `memberEnd`, porque são consideradas relações "*Todo/Parte*" nas quais o "*Todo*" está representado pelo membro da associação que tenha o `AggregationKind` diferente do valor "*none*". Há também outras propriedades que vão determinar o tipo de agregação e informações pertinentes.

A Figura 20 apresenta a taxonomia adotada para associações no **Metamodelo de Navegação**. Nela é possível observar que, neste caso, são permitidas apenas composições, representadas pelas metaclasses `NavigationCompositionWhole` (representando o "*Todo*") e `NavigationCompositionPart` (representando as "*Partes*") que assumem o papel de `memberEnd`.

Só há um conjunto específico de combinações permitidas para composições no **Metamodelo de Navegação**, que estão mostradas nas linhas 4 a 6 da Listagem 4.6, expressas pelas propriedades `memberEnd` e `ownedEnd`.

A Listagem 4.6 também restringe as associações a composições no **Metamodelo de Navegação**. Assim, a propriedade `NavigationCompositionWhole`, que assume o papel de "*Todo*" para a metaclasse ***`NavigationClass`***, deve ter seu atributo `aggregation` (do tipo `AggregationKind`) carregado sempre com o valor "*composite*" e, além disso, o atributo `isComposite` deve estar marcado com o valor "*true*", ambos obrigatoriamente, como mostra a linha 13.

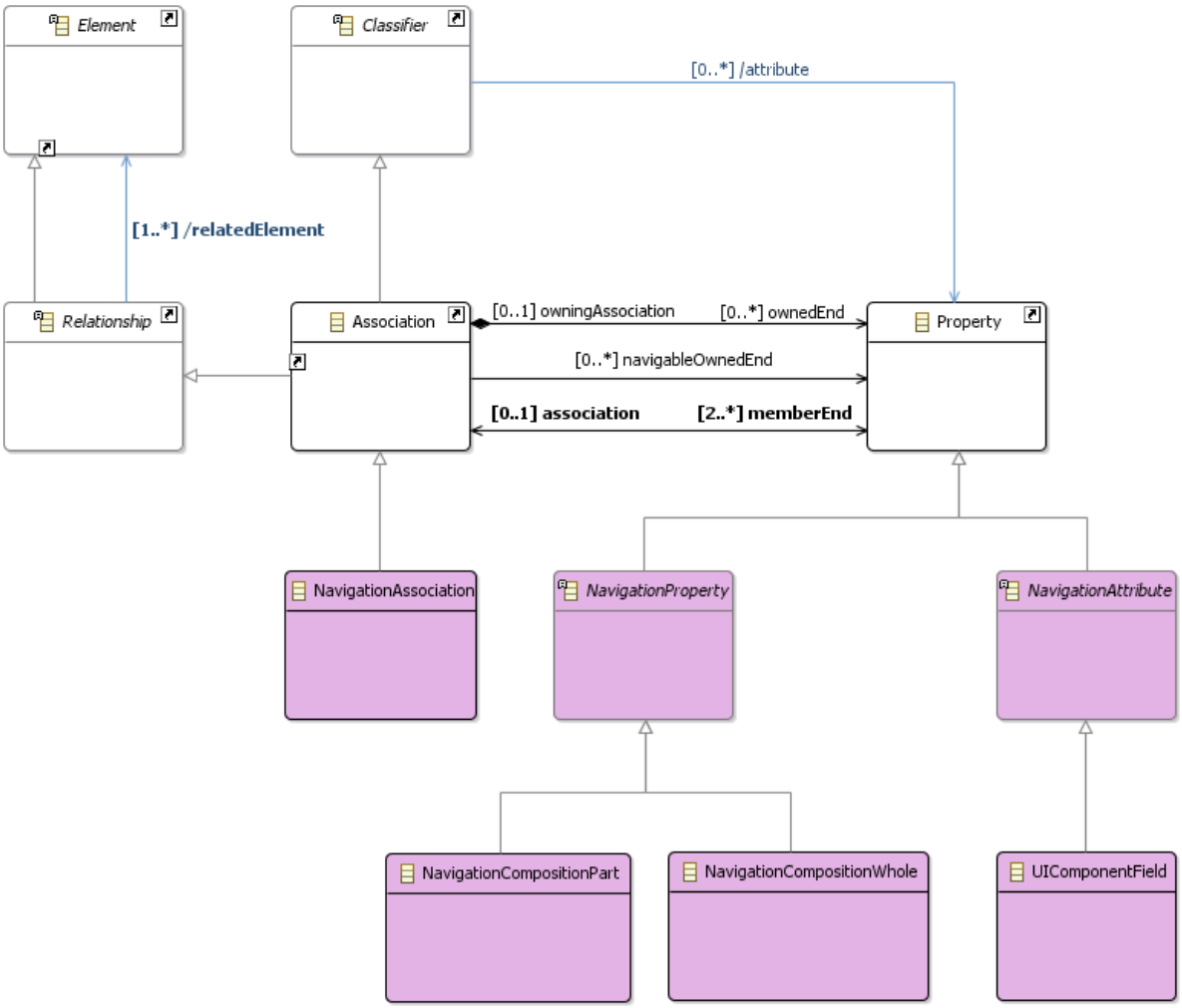
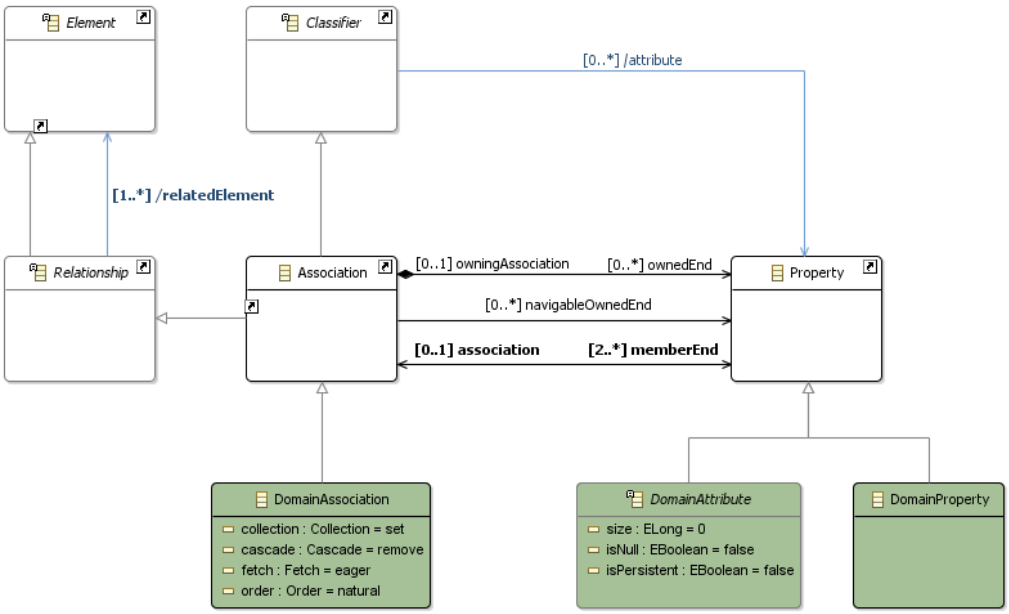


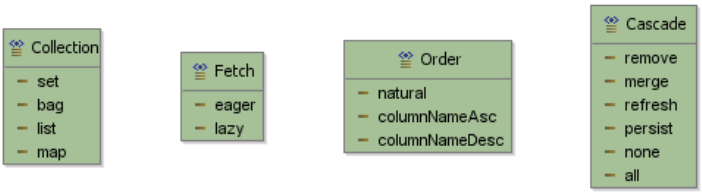
Figura 20 – Fragmento do Metamodelo de Navegação do ***FW-15***: associações.

Listagem 4.6 – Regras OCL no contexto da Figura 20.

```
1 -- A NavigationAssociation must have a UIComponent as a part and NavigationClass
      as whole.
2 context NavigationAssociation
3   inv NavigationAssociationRelation:
4     (self.ownedEnd.oclIsTypeOf(NavigationCompositionWhole)) and
5     ((self.memberEnd.oclIsTypeOf(NavigationCompositionWhole)) or
6     (self..memberEnd.oclIsTypeOf(NavigationCompositionPart)))
7
8 -- A NavigationAssociation must be a composition association and NavigationClass
      are alowed to be whole.
9 context NavigationCompositionWhole
10   inv NavigationCompositionWholeRelation
11     (self.owningAssociation.oclIsTypeOf(NavigationClass)) and
12     (self.association.oclIsTypeOf(NavigationAssociation)) and
13     (self.aggregation = 'composite') and (self.isComposite)
14
15 -- A NavigationAssociation must have only a UIComponent as a part.
16 context NavigationCompositionPartRelation
17   inv NavigationCompositionPartConstraint:
18     self.owningAssociation.oclIsTypeOf(NavigationAssociation)
19     and (self.type.oclIsTypeOf(UIComponent))
```



(a) Associações de entidade



(b) Enumerações para associações de entidade

Figura 21 – Fragmento do Metamodelo de Entidade do ***FW-15***: associações.

Por fim, nas linhas 18 e 19 define-se quem, no que diz respeito às instâncias, representa o papel de "*Parte*" na composição. Neste caso, só serão parte as instâncias da metaclasse `UIComponent`.

Isso significa dizer que uma associação de navegação será sempre uma composição, que só poderá ocorrer entre quaisquer classes de navegação ("*Todo*") e componentes UI ("*Parte*"), ou seja; páginas, modelos ou componentes UI podem conter componentes UI.

No **Metamodelo de Entidades**, as associações devem ser acrescidas de algumas informações importantes que estão mostradas na Figura 21 como atributos da metaclasse `DomainAssociation`. Estas informações são usadas nas restrições (*`DomainConstraint`*) impostas à metaclasse `DomainAssociation` para definir o comportamento da associação entre classes de domínio (`DomainClass`).

O atributo `collection` da metaclasse `DomainAssociation` identifica explicitamente a estrutura de dados a ser implementada para recuperação dos dados instanciáveis (em tempo de execução) pelas entidades associadas, podendo ser por meio de lista, lista não indexada, conjunto ou mapa ("*list*", "*bag*", "*set*" ou "*map*", respectivamente). Não devendo ser confundido com a combinação dos atributos `isOrderd` e `isUnique` da metaclasse

`MultiplicityElement` UML, que é responsável por tratar cardinalidades, por exemplo dos "*association ends*" dentre outros.

Neste caso `collection` está delimitado pelo enumerado `Collection` e será selecionado pelo modelador na definição da associação, conforme mostrado na Figura 21b. Contrariamente aos valores `isOrderd` e `isUnique` que sendo tipos booleanos identificam a característica da cardinalidade definida no "*association end*" UML, não fornecem a expressividade desejada com respeito ao metamodelo ***FW-15***.

Da mesma maneira o atributo `order` está delimitado pelo enumerado `Order` e representa explicitamente o critério de ordenação a ser usado na recuperação dos dados instanciáveis pelas entidades associadas no contexto do Sistema de Informação Web, indicando não apenas que haverá ordenação mas também como deve ser executada. Contrariamente aos valores `isOrderd` (booleano) no que diz respeito aos "*association ends*".

O atributo `fetch` define a estratégia de recuperação dos dados do Sistema de Informação Web em tempo de execução, o seja se os dados serão recuperados todos de uma vez e simultaneamente entre as entidades associadas ou parcialmente a medida que forem solicitados. Para tanto o modelador deve avaliar o volume a ser recuperado em cada caso. O atributo `fetch` está delimitado pelo enumerado `Fetch`.

E finalmente o atributo `cascade` determina a estratégia de cascateamento das associações, ou seja, é usado para definir o tipo da persistência a ser implementada no Sistema de Informação Web e diz respeito à relação de dependência no armazenamento dos dados associados em tempo de execução. O atributo `cascade` está delimitado pelo enumerado `Cascade`.

Além disso, pode-se notar que a Figura 21 muito semelhante à Figura 20, porque em ambos os casos ocorrem especializações das metaclasses UML `Association` e `Property`. A diferenciação ocorre a partir dos diferentes atributos requeridos, que são usados em conjunto com restrições ("*constraints*") impostas, e das estruturas admitidas como atributos, tratadas mais adiante, na Seção 4.2.7.

No **Metamodelo de Entidades** as associações podem ser de qualquer tipo (associações, agregações ou composições), mas possuem um conjunto obrigatório de restrições para definir certa semântica, própria dos domínios de Sistemas de Informação Web. Enquanto que no **Metamodelo de Navegação** são permitidas apenas composições e nestas não há restrições impostas com finalidade semântica.

Desta forma as regras OCL impostas no contexto da Figura 21 são mais flexíveis do que as apresentadas mais acima para o **Metamodelo de Navegação**, como pode ser visto na Listagem 4.7.

Listagem 4.7 – Regras OCL no contexto da Figura 21.

```
1 -- A DomainAssociation must be made between DomainClass.
2 context DomainAssociation
3   inv DomainAssociationRelation:
4     (self.memberEnd.oclIsTypeOf(DomainClass)) and
5     ((self.ownedEnd->size()>=1  implies
6     (self.ownedEnd.oclIsTypeOf(DomainClass)))
```

Nos **Metamodelos de Aplicação** e **Persistência** não são permitidas composições e agregações, mas apenas associações simples e estas não podem ter restrições impostas com finalidade semântica.

4.2.5 Atributos

Atributos de classes são, no metamodelo UML, especializações da metaclasse `Property`, que pertencem a classes (`Class`). Esta semântica é estendida no metamodelo ***FW-15*** para permitir conceitos especiais. No **Metamodelo de Navegação** atributos são tratados por meio da metaclasse *`NavigationAttribute`*, especializada da metaclasse UML `Property`.

Por exemplo, a Figura 22 apresenta como especializações de *`NavigationAttribute`*, os atributos pertencentes às classes de navegação (*`NavigationClass`*) definidos pela metaclasse `UIComponentField` e usados para representar as diversas estruturas de interface com o usuário. Contudo, assim como já apresentado anteriormente esta restrição é feita por meio da adição de regras OCL, não visíveis na figura.

Além disso, a definição dos tipos permitidos para as estruturas da metaclasse `UIComponentField` participam de questões dependentes do *framework* escolhido e isto encontra-se detalhado mais adiante na Seção 4.3.1.

Já a metaclasse `IOParameter` representa os atributos que podem ser usados em classes do controlador frontal (`FrontControllerClass`), os quais, quando possuem o mesmo nome dos atributos usados em classes de navegação, estão relacionados às estruturas de interface com o usuário, injetando ou permitindo a exibição das informações, como pode ser observado na Figura 22 e detalhado nas linhas 16 e 21 da Listagem 4.8, nas quais estão apresentados os dois sentidos da associação entre estas propriedades. Além disso, a Listagem 4.8 exibe também as restrições que diferenciam quais atributos são permitidos e a quais classes pertencem.

Já no **Metamodelo de Entidade**, atributos são usados na representação de propriedades dos conceitos do domínio do Sistema de Informação Web, mais precisamente representam argumentos de classes de domínio. Classes de domínio (entidades) no ***FW-15*** são usadas na representação e no mapeamento para a persistência de objetos do Sistema de Informação Web, conforme apresentado na Seção 2.1, em geral representando tabelas

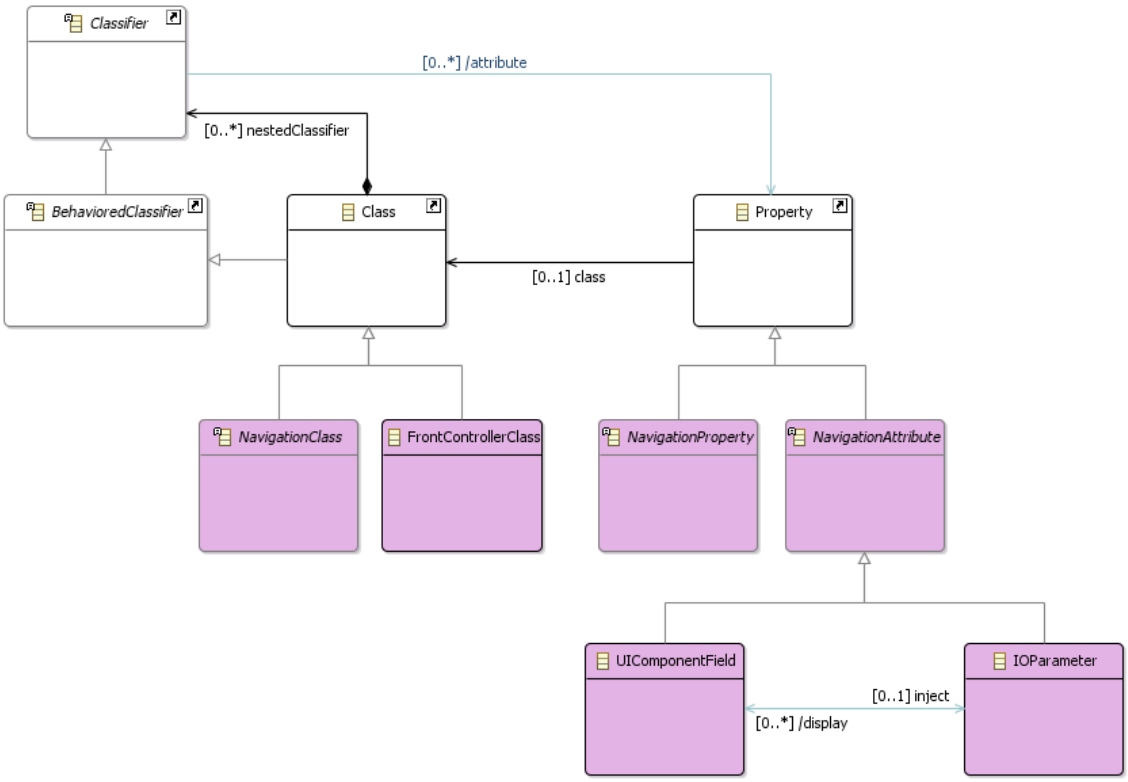


Figura 22 – Fragmento do Metamodelo de Navegação do ***FW-15***: atributos.

Listagem 4.8 – Regras OCL no contexto da Figura 22.

```
1 -- A NavigationClass only allows UIComponentFiled or NavigationProperty as
      properties.
2 context NavigationClassContent
3   inv NavigationClass:
4     (self.packagedElement.oclIsTypeOf(UIComponentFiled)) or
5     (self.packagedElement.oclIsTypeOf(NavigationCompositionWhole))
6
7 -- A FrontControllerClass only allows IOParameter as property or
      FrontControllerMethod as operation.
8 context FrontControllerClass
9   inv NavigationClassContent:
10    (self.packagedElement.oclIsTypeOf(IOParameter)) or
11    (self.packagedElement.oclIsTypeOf(FrontControllerMethod))
12
13 -- A UIComponentField displays values from IOParameter
14 context UIComponentField
15   inv UIComponentFieldRelation:
16    (self.inject->notEmpty() <-> self.name = self.inject.name)
17
18 -- A IOParameter injets values on UIComponentField
19 context IOParameter
20   inv IOParameterRelation:
21    (self.display->notEmpty() <-> self.name = self.display.name)
```

em um banco da dados. Por conseguinte, seus atributos representam as colunas destas tabelas. Desta forma o **Modelo de Entidades** permite uma associação entre os elementos e propriedades de domínio do Sistema de Informação Web com as estruturas de um banco de dados para que se promova a persistência de dados.

Assim como os atributos de navegação, atributos de entidade (`DomainAtribute`) são especializações de `Property`, porém possuem meta-atributos que são usados em restrições relacionadas às colunas das tabelas para determinar tamanho, precisão decimal e nulidade entre outros aspectos de relacionados ao domínio dos conceitos a serem modelados no Sistema de Informação Web.

O mesmo raciocínio é aplicado nos **Metamodelos de Aplicação** e **Persistência** e pode ser visto no metamodelo completo apresentado no repositório do projeto.

4.2.6 Interfaces e Realizações

Interfaces, segundo a especificação UML, são usadas para representar um tipo especial de *`Classifier`* cuja declaração é um conjunto público de características e obrigações que constituem um serviço. Em resumo, uma interface UML representa um contrato cujas cláusulas devem ser cumpridas por qualquer instância de elementos que a realizem.

Interfaces não podem ser instanciadas, ao contrário, devem ser realizadas ou implementadas por estruturas do tipo *`BehavioredClassifier`* (ou especializações dela), os quais representam uma fachada pública de acordo com o que foi especificado na interface, conforme prega o padrão *Façade* (GAMMA et al., 1994).

No método ***FW-15***, interfaces aparecem apenas nos **Metamodelos de Aplicação** e **Persistência** e especializam a mesma semântica aplicada à UML. Por exemplo, como o método ***FrameWeb*** faz uso do padrão DAO (ALUR; MALKS; CRUPI, 2013) e interfaces são parte da implementação de DAOs, o **Metamodelo de Persistência** faz uso das interfaces UML por meio da metaclasse `DAOInterface` especializada de `Interface`, como pode ser observado na Figura 23.

Na Figura 23, está também apresentada a metaclasse `DAORealization`, que é uma especialização da metaclasse UML `Realization`, usada para representar as relações permitidas entre interfaces e classes. As realizações permitidas no **Metamodelo de Persistência** são aquelas que ocorrem entre as classes DAO (`DAOClass`) e Interfaces DAO (`DAOInterface`), como pode ser observado na linha 2 da Listagem 4.9.

No **Metamodelo de Aplicação**, as interfaces e realizações seguem os mesmos princípios e estão detalhadas no repositório do projeto. .

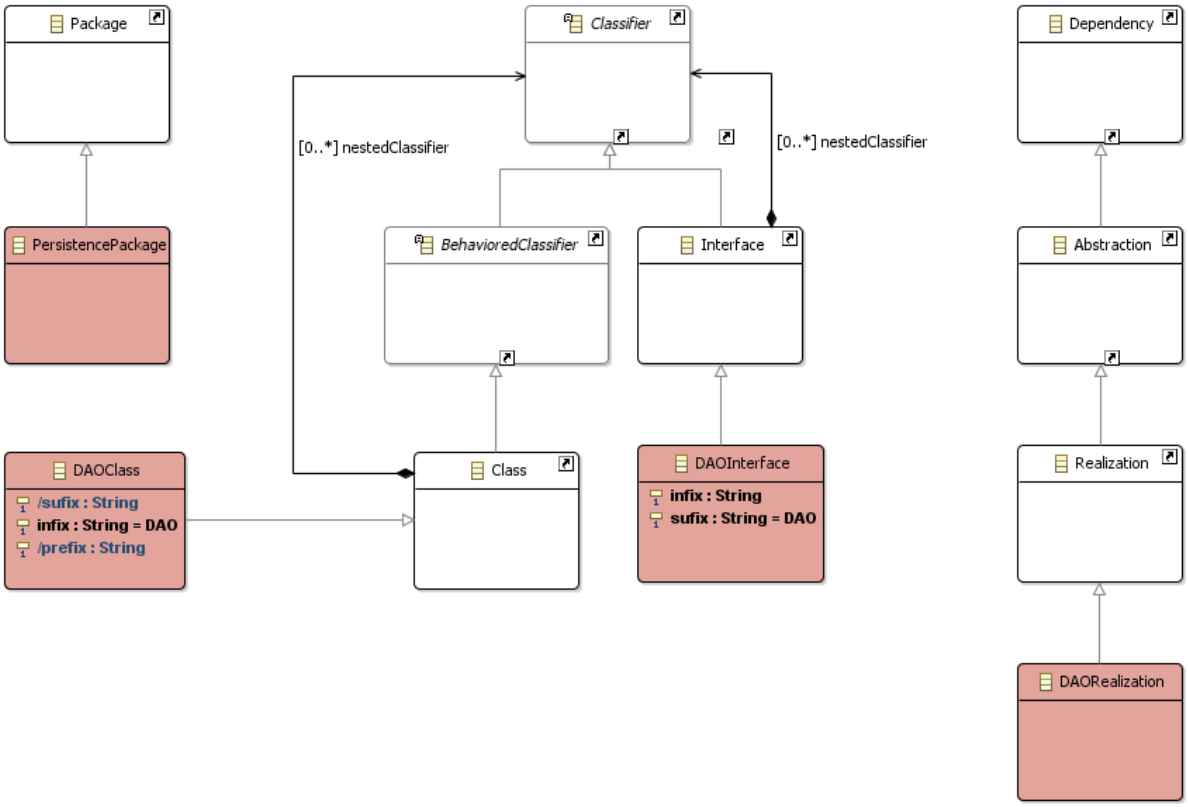


Figura 23 – Fragmento do Metamodelo de Persistência do ***FW-15***: interfaces e realizações.

Listagem 4.9 – Regras OCL no contexto da Figura 23.

```
1 -- A DAOInterface must be realized by an DAOClass.
2 context DAORealization
3   inv DAORealizationRelation:
4     (self.supplier.oclIsTypeOf(DAOInterface)) and
5     (self.client.oclIsTypeOf(DAOClass))
```

4.2.7 Restrições

No metamodelo ***FrameWeb*** as restrições são usadas para restringir o comportamento definido para as relações entre classes usadas nos modelos. Desta maneira, devem ser aplicadas para que o modelador defina comportamento e guie o desenvolvedor quanto a como e o que deve ser implementado, sob certas circunstâncias.

No contexto da UML, restrições (`Constraint`) podem ser aplicadas a quaisquer metaclasses especializadas de *`Namespace`*, permitindo, assim, que diversas estruturas como `Package`, `Class`, `Property`, `Operation` possam conter restrições, já que todas estas metaclasses são especializações diretas ou indiretas de *`Namespace`*.

Por outro lado, `Dependency` é especialização de relacionamento direto (*`DirectedRelationship`*) e não de *`Namespace`*. Isso significa dizer que não há no metamodelo UML aplicação de restrições a dependências, como mostra a Figura 24.

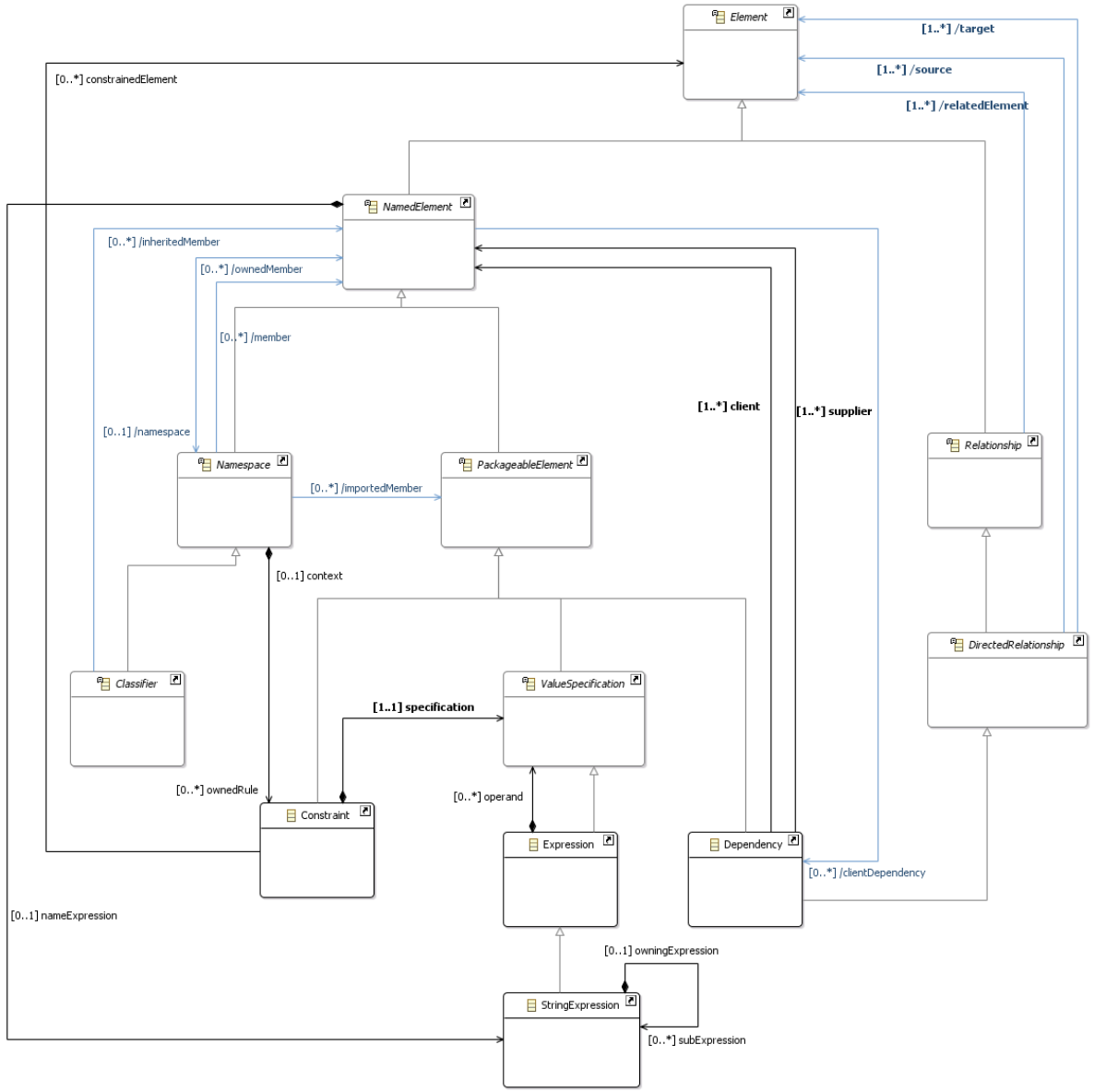


Figura 24 – Fragmento do Metamodelo da UML: restrições em dependências.

Na linguagem ***FW-07***, restrições foram usadas para dar semântica específica também para dependências, o que em termos formais gerou um problema: dependências de navegação (*NavigationDependency*) não poderiam possuir restrições (`Constraint`) em UML, já que não são especializações de *Namespace*.

Na Figura 24, está um fragmento do metamodelo UML, onde estão representadas as metaclasses `Dependency`, `Constraint` e suas relações. É possível observar que a nível de elementos empacotáveis (*PackageableElement*) não há uma relação todo-parte indicando que uma dependência possa possuir restrições. Desta forma foi necessário fazer uma reflexão a respeito do que são restrições, de como e por que não se aplicam a dependências segundo a especificação UML.

Na UML, "*constraint*" é uma condição ou restrição expressa em linguagem textual natural ou legível por máquinas, com o propósito de declarar alguma semântica de um elemento ou conjunto de elementos. Além disso, a especificação UML nos diz que uma `ValueSpecification` para "*constraint*" deve retornar um valor booleano.

E uma dependência é um relacionamento direto no qual um elemento ou conjunto de elementos requerem outros para sua especificação ou implementação. Isso ocorre, porque a semântica de uma classe que participa como cliente ("*target*") em uma dependência UML, não está completa sem a existência de uma classe que participe desta dependência como fornecedora ("*supplier*"), contudo não há maiores explicações a respeito de como a classe cliente usa a fornecedora na especificação UML.

Fica mais claro visualizar esse comportamento ao tratarmos de instâncias das dependências, porque não haverá instancias do elemento cliente que não seja dependente de algum fornecedor.

Na visão da linguagem ***FW-15***, as descrições relacionadas a uma dada dependência, correspondem a aspectos específicos dessa dependência, e não exatamente restrições, motivo pelo qual se fossem restrições seriam sempre verdadeiras.

A partir desta reflexão, tem-se que o metamodelo ***FW-15*** poderia tratar estas descrições associadas a dependências de três maneiras distintas: (i) como "*constraints*" especiais com valor especificado (`ValueSpecification`) que seja sempre verdadeiro, (ii) como expressões (`StringExpression`), ou (iii) como um tipo específico da linguagem ***FW-15***.

Adicionar "*constraints*" a *`NavigationDependency`*, especializada de `Dependency`, é de implementação simples no que diz respeito ao metamodelo ***FW-15***, mas exige que algumas regras OCL sejam adicionadas para garantir a integridade que o método necessita.

Tratar as restrições como `StringExpression` não exige nenhuma implementação a mais no metamodelo, mas deixa o modelador livre para montar as expressões seguindo apenas sua experiência pessoal, no que diz respeito ao método.

Por fim, a criação de um tipo específico para tratar esta semântica torna o metamodelo muito mais direcionado ao que se deseja semanticamente, mas também aumenta excessivamente a complexidade do metamodelo, além, é claro, de afastar o mesmo da própria UML no que diz respeito à capacidade de importar modelos UML externos.

Avaliando estas três opções a solução mais apropriada ao que se deseja alcançar neste trabalho aproxima-se mais da primeira opção, na qual a metaclasse *`NavigationConstraint`* é uma especialização de UML `Constraint` e possui uma relação de composição com a metaclasse *`NavigationDependency`*, especializada de `Dependency`, de forma a permitir que estas dependências possuam aquelas restrições, como pode ser visto na Figura 25.

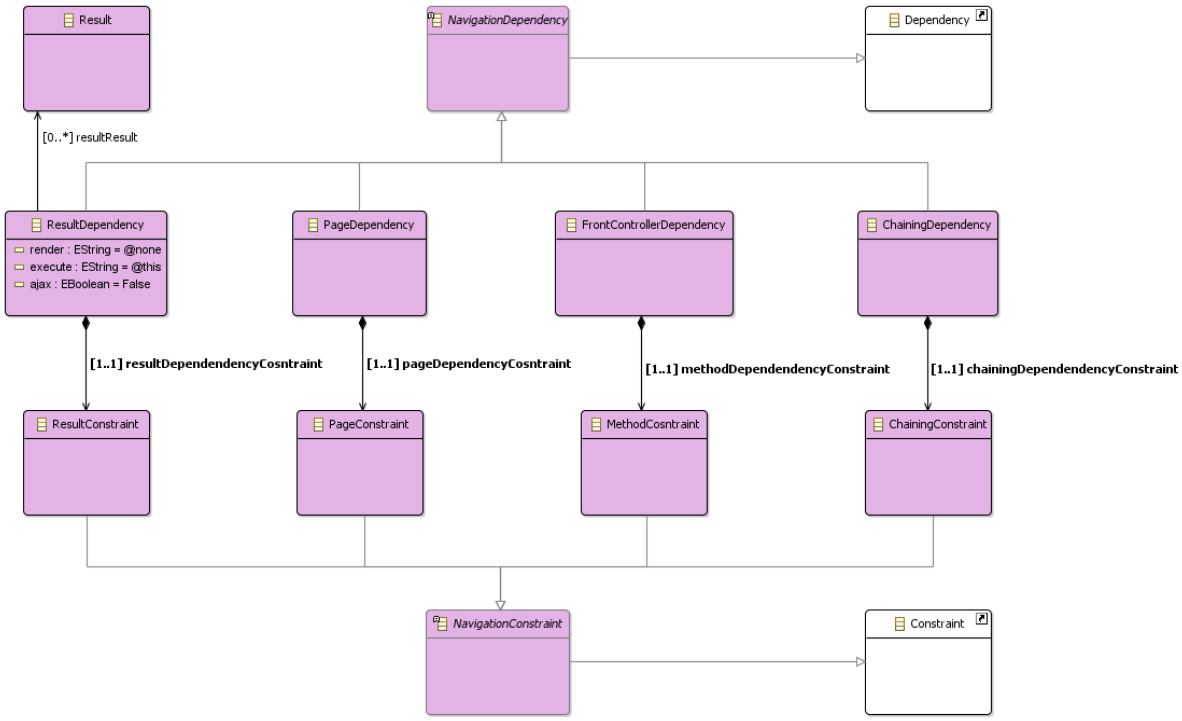


Figura 25 – Fragmento do Metamodelo de Navegação do ***FW-15***: restrições.

É importante ressaltar que há ferramentas de modelagem UML para uso geral permissivas em relação a adição de restrições em dependências, um exemplo é o Astah[4]. Isso significa dizer que o Astah assim como qualquer outra ferramenta que permite a adição deste tipo de restrição, não implementa metamodelo UML exatamente como proposto em sua especificação.

Este tipo de comportamento da sintaxe abstrata UML adotada nestas ferramentas são a causa destas restrições aparecerem também no metamodelo ***FW-07***, logo o ***FW-15*** não criou restrições em termos de ferramental que já não haviam no ***FW-07***.

Como optamos por adicionar restrições à metaclasse *NavigationDependency* e esta não é uma característica herdada do metamodelo UML, não se faz necessário adicionar regras de filtragem ao contexto da metaclasse *NavigationConstraint*.

No **Metamodelo de Navegação**, as restrições podem estar diretamente relacionadas a determinados atributos pertencentes a algumas metaclasses, para determinar o valor verdade das "*constraints*" permitidas a esta metaclasse.

Por exemplo, componentes de UI podem fazer uso de `AJAX` quando são acionados a partir de um método, conforme discutido na Seção 3.3.2. Desta forma, quando o modelador define que um dado componente UI será usado e acionará um método para executar uma dada ação, ele também deverá informar se este componente faz uso de `AJAX`, como deve ser renderizado e qual método será executado, no âmbito da dependência. Estas

4 <http://astah.net/>

definições são feitas por meio dos meta-atributos `ajax`, `render` e `execute` da metaclasse `ResultDependency`, conforme apresentado na Figura 25.

A partir das definições do modelador, no que diz respeito ao conteúdo informado para estes meta-atributos nas instâncias de `ResultDependency`, as restrições `ResultConstraint` (especializadas de *`NavigationConstraint`*) os tomarão como base de validação do seu estado *verdadeiro* ou *falso*.

Isso ocorre, no exemplo para o caso do `AJAX`, porque as instâncias da metaclasse `UIComponent` são as fornecedoras ("*supplier*") assim como instâncias de `FrontControllerClass` são clientes ("*client*") no contexto da dependência `ResultDependency`.

Em outros casos, os "*truthmakers*" das "*constraints*" podem estar relacionados a estruturas mais complexas como resultados, os quais podem representar arquivos binários, estruturas multimídia, outros sistemas que são externos ao contexto do Sistema de Informação Web a ser modelado, ou qualquer estrutura desejada pelo modelador. Nestes casos todas estas estruturas estarão representadas pela metaclasse `Result`.

Desta maneira, o modelador poderá definir que um certo conjunto de resultados é esperado a partir do processamento de um método, entretanto será necessário avaliar o resultado (`Result`) estruturalmente para determinar o estado *verdadeiro* ou *falso* das restrições `ResultConstraint`. Esta avaliação depende da definição dos resultados permitidos em cada *framework* MVC, ou seja é uma definição dependente de *framework*, que será tratada mais adiante na Seção 4.3.1.

Restrições também aparecem nos outros metamodelos ***FrameWeb***, para definir condições especiais de elementos da linguagem, porém são tratadas de maneira semelhante ao que já foi apresentado em relação ao **Metamodelo de Navegação**.

No **Metamodelo de Entidades** as restrições *`DomainConstraint`* são usadas nas associações entre conceitos do domínio do Sistema de Informação Web para fornecer uma semântica específica. Mais especificamente para tratar as relações entre os mapeamentos dos conceitos de domínio, e fornecer os recursos necessários à recuperação e persistência dos dados.

Por exemplo, no mapeamento objeto relacional (ORM) as instâncias de alguns conceitos do domínio do Sistema de Informação Web (dados em tempo de execução) podem ser obtidas por meio de diferentes estratégias de recuperação, conforme determinada necessidade das relações impostas. Em `JPA`, essa estratégia de recuperação pode ser normal (*eager*) ou preguiçosa (*lazy*) e é determinante no que diz respeito a ganho de performance, especialmente em relações `@ManyToOne`, `@OneToMany` ou `@ManyToMany` com grande volume de dados.

Em outros casos os aspectos avaliados nas restrições de associação referem-se à estrutura de dados a ser implementada no Sistema de Informação Web para definir o a

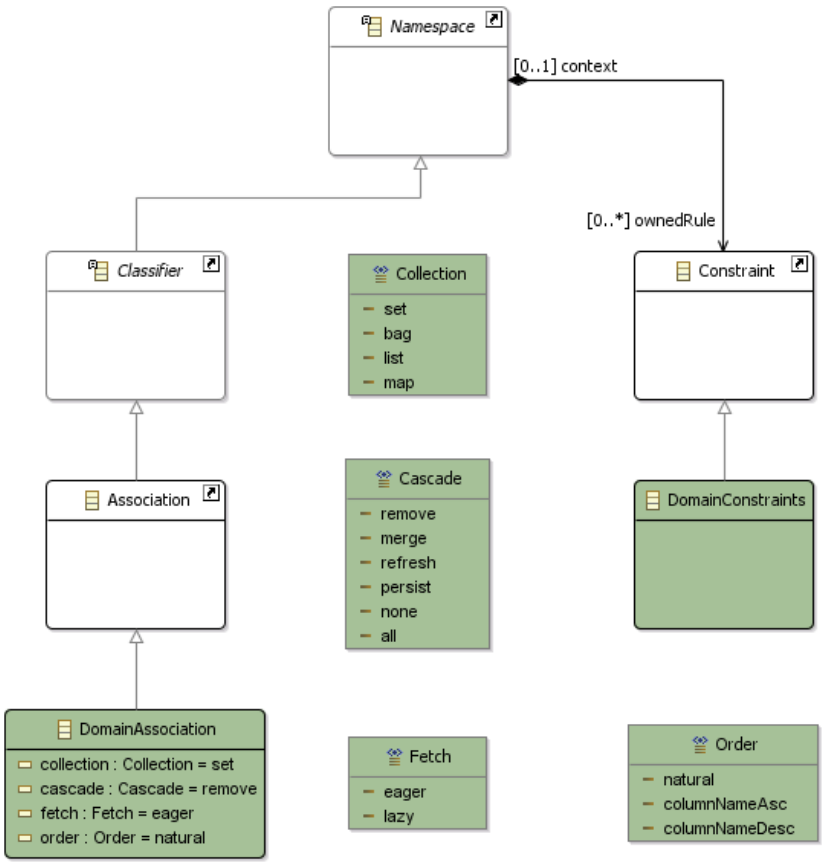


Figura 26 – Fragmento do Metamodelo de Domínio do ***FW-15***: restrições.

forma de armazenamento dos dados recuperados. Nesta situação, o modelador deve definir na restrição da associação qual estrutura será implementada, ou seja, pode tratar-se de uma lista (*list* or *bag*), conjunto (*set*) ou mapa (*map*).

A Figura 26 detalha as restrições de associações no **Metamodelo de Entidades**, incluindo o conjunto de valores permitidos aos meta-atributos de `DomainAssociation`, por meio de estruturas de enumeração (*Enumeration*).

Evidentemente, se faz necessário complementar a semântica da Figura 26 com as regras OCL necessárias, semelhantemente ao que foi apresentado anteriormente no **Metamodelo de Navegação**, exceto pelo fato de que o **Metamodelo de Entidades** é menos restritivo, pois permite que restrições também sejam impostas a `DomainAtribute`.

As restrições aplicadas a `DomainAtribute` foram implementadas no **Metamodelo de Entidades** de forma semelhante às de associação, por causa disso não estão detalhadas, mas foram implementadas. Todas estas regras estão detalhadas no Apêndice A.

Há também diversas outras restrições não mencionadas nesta seção, porém estão implementadas no metamodelo ***FW-15***. Além disso, as diversas restrições especializadas de `Constraint` e aplicáveis a *`Namespace`* possuem regras OCL associadas para que sejam devidamente filtradas segundo a semântica que se deseja impor para estruturar a

linguagem ***FW-15***. O mesmo raciocínio é aplicado nos **Metamodelos de Aplicação** e **Persistência** e pode ser visto no metamodelo completo apresentado no repositório do projeto.

4.3 Abordagem Multi-Framework

A sintaxe independente de *framework* apresentada na Seção 4.2 se aplica a todos os modelos construídos usando a linguagem ***FrameWeb***, independentemente do conjunto de *frameworks* incluídos na arquitetura do Sistema de Informação Web. Entretanto, diferentes instâncias de *frameworks* possuem características distintas que influenciam os modelos.

Por exemplo, no `Struts`[2] os campos de formulário (*form fields*) são limitados a uma única classe do controlador frontal, enquanto que em `JSF` é permitido vincular campos de um formulário a atributos em diferentes classes do controlador frontal.

Desta forma é necessário que haja uma abordagem que permita a acomodação de diferentes instâncias *frameworks* durante a modelagem, uma abordagem *multi-framework* (dependente de *framework*).

Para tratar estas questões, o metamodelo deve ser adaptável a partir das escolhas do modelador, permitindo-lhe escolher e carregar dinamicamente cada **Definição de Framework** (`Framework Definition`) desejada.

Para tratar as **Definições de Frameworks**, há o **Metamodelo de Framework** (`Framework Meta-model`), onde é definido o conjunto de meta-estruturas que determinam uma gramática específica para a definição e carga dos diferentes *frameworks*. Este metamodelo foi tratado separando-o em: (i) na estrutura dependente de *framework*, ou seja para um determinado *framework*; e (ii) na aplicação desta estrutura no metamodelo independente de *framework* ***FW-15***.

Para que um *framework* possa ser usado em conjunto com o metamodelo ***FW-15***, suas características específicas devem ser definidas de acordo com o **Metamodelo de Framework**. Isso significa dizer que uma **Definição de Framework** é, na realidade, a definição e a aplicação de um **Perfil FrameWeb** (***FrameWeb*** *profile*) em um modelo ***FW-15***.

Um **Perfil FrameWeb** representa um tipo específico de perfil UML direcionado às meta-estruturas do metamodelo ***FW-15***. Assim, permite que as modificações necessárias e dependente de *framework* sejam definidas para posteriormente serem aplicadas aos diversos modelos ***FW-15***.

Isto posto, o **Metamodelo de Framework** possui a metaclasse `Framework-Profile`, que representa a classe usada na definição de um **Perfil FrameWeb**, como pode ser observado na Figura 27. A partir de instâncias desta metaclasse, é possível definir a

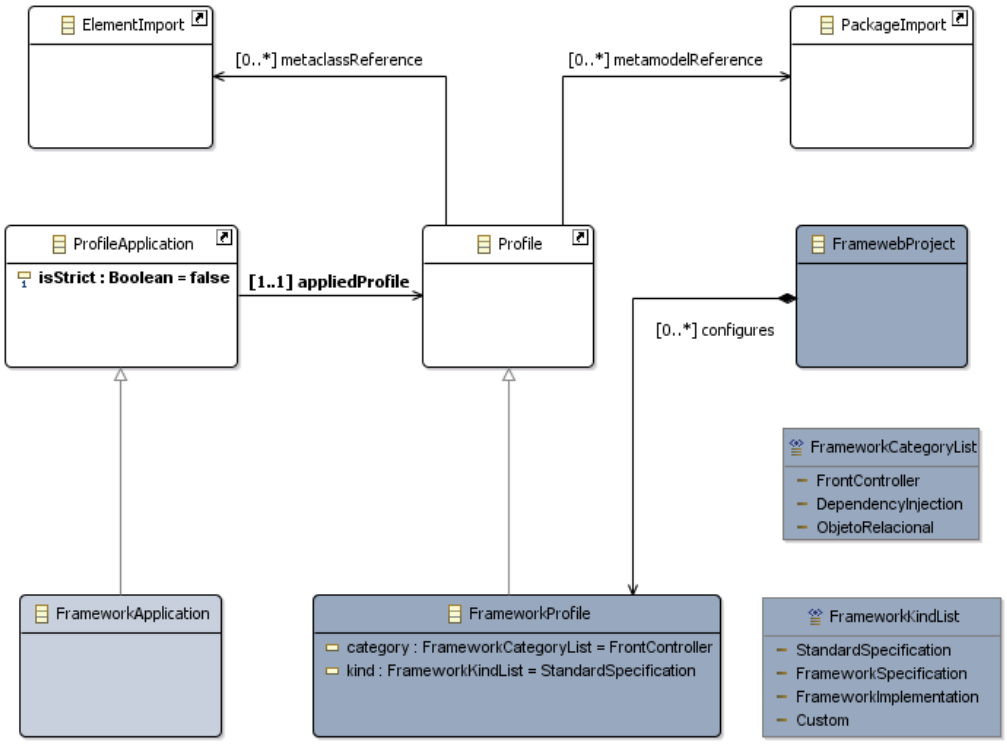


Figura 27 – Fragmento do Metamodelo de Framework do ***FW-15***: Perfil.

categoria e uma configuração específica contendo as regras e o comportamento específicos de diversos *frameworks*.

Uma **Definição de Framework** (`Framework Definition`) representa as instâncias do **Metamodelo de Framework** (`Framework Metamodel`). Estas estão agrupadas conforme a categoria do *framework* (ORM, MVC e DI), conforme pode ser observado na Figura 27 pelos valores enumerados em `FrameworkCategoryList`.

Há também uma outra classificação que determina o tipo de definição, podendo ser uma especificação de padrões (HTML4, HTML5, JSTL, etc.), uma especificação ou implementações de *framework* (`Spring MVC`, `Struts`[2], `Hibernate` e outros), ou até mesmo um tipo customizado, conforme pode ser observado na Figura 27 pelos valores enumerados em `FrameworkKindList`.

Esta classificação determina o comportamento do perfil quando aplicado ao metamodelo ***FW-15***. Isso significa dizer que a semântica da linguagem ***FW-15*** será alterada conforme os perfis carregados e em tempo de execução. Para tanto há um conjunto de regras aplicadas às meta-estruturas dos perfis já definidas no **Metamodelo de Framework**.

Cada **Definição de Framework** resulta em uma biblioteca contendo o todas as informações específicas do perfil definido. As diversas bibliotecas produzidas constituem um conjunto variado de definições disponíveis ao modelador, portanto podem ser selecionadas e carregadas no projeto de um Sistema de Informação Web, conforme necessário.

Neste sentido, um modelador pode selecionar para seu projeto de Sistema de Informação Web diversas combinações de bibliotecas, como está apresentado no Capítulo 5, que trata da avaliação deste processo.

Além da metaclasse `FrameworkProfile`, o **Metamodelo de Framework** possui a metaclasse `FrameworkApplication` especializada da metaclasse UML `ProfileApplication`, com a finalidade de determinar a aplicação de um perfil (instâncias de `FrameworkProfile`) criado para o metamodelo ***FW-15***. Assim como `ProfileApplication` descreve quais os perfis podem aplicáveis a um pacote UML, `FrameworkApplication` assume a mesma função no que diz respeito ao metamodelo ***FW-15***.

Perfis UML não removem estruturas nem restrições do metamodelo UML, apenas fornecem um mecanismo para permitir que adaptações possam ser feitas ao metamodelo, tanto filtrando meta-elementos quando adicionando novas restrições. Estes construtos são específicos a um contexto em particular.

Considerando que perfis UML não permitem a criação de novos metamodelos, é importante salientar que há diversas maneiras para definir as regras de filtragem dos perfis, que determinam quais elementos UML estarão disponíveis e/ou indisponíveis, quando o perfil for aplicado.

Por conseguinte, o todas estas considerações se aplicam ao **Perfil FrameWeb**. Logo, a metaclasse `FrameworkApplication` também tem por função participar do processo de definição das regras de filtragem do **Perfil FrameWeb**, por meio do ajuste do meta-atributo `isStrict` de `ProfileApplication`, cujo valor deve ser sempre "*verdadeiro*" ("*true*").

Posteriormente, é necessário definir que o metamodelo de referência UML a ser adotado (`metamodelReferences`) no pacote é o metamodelo ***FW-15***. Além disso, é necessário definir quais de suas metaclasses do metamodelo estão disponíveis a nível de perfil (`metaclassReferences`), como pode ser observado na Figura 27.

Assim como em outras situações, estas regras de filtragem dos perfis do **Metamodelo de Framework** foram adicionadas por meio de regras OCL que, por sua vez, consideram, além desses aspectos, a característica e o tipo do perfil a ser tratado.

Nas seções a seguir, os detalhes do **Metamodelo de Framework** são apresentados separadamente para duas categorias de *frameworks* tratadas neste trabalho: Controlador Frontal e Mapeamento Objeto/Relacional.

4.3.1 Controlador Frontal (MVC)

Cada *framework* Controlador Frontal, conhecidos também como *frameworks* MVC, possui seu próprio conjunto de regras, recursos e *tags* além daqueles oriundos dos pa-

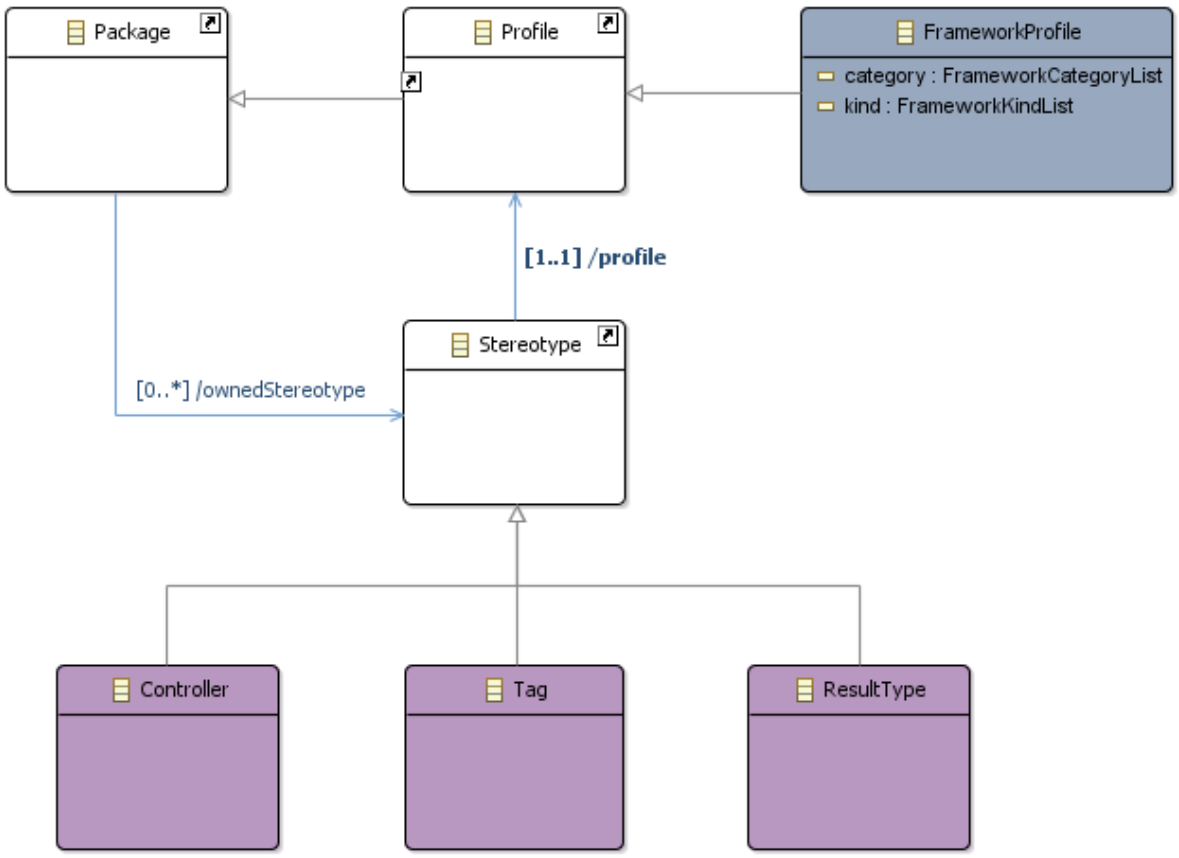


Figura 28 – Fragmento do Metamodelo de Framework do ***FW-15***: estereótipos para MVC.

drões `HTML` e `XHTML`. Por isso, o metamodelo ***FW-15***, necessita ser capaz de expressar estes diferentes recursos, de forma a prover ao modelador a possibilidade de atribuí-los corretamente, em função do *framework* selecionado em seu projeto.

A Figura 28 apresenta o fragmento do **Metamodelo de Framework** contendo os detalhes dos estereótipos usados para promover recursos MVC dependente de *framework*.

Um `FrameworkProfile` cuja categoria é `MVC`, pode possuir três estereótipos (especializações de UML `Stereotype`): `Controller`, `ResultSet` e `Tag`.

O `Controller` representa um estereótipo aplicável à `FrontControllerClass`, responsável por definir uma semântica distinta para cada classe do controlador frontal de acordo com o *framework* que se deseja definir em um **Perfil FrameWeb**. Ou seja, `Controller` estende `FrontControllerClass` por meio de uma associação de extensão, que está detalhada mais adiante nesta seção.

Desta forma, em um **Perfil FrameWeb** para `Struts`[2], é possível definir uma instância de `Controller` denominada `Action`, que automaticamente é uma extensão de `FrontControllerClass`. A partir da aplicação deste perfil em um **Modelo de Navegação**, `«Action»` aparecerá como uma estereotipo de uma `FrontControllerClass` disponível para uso.

Juntamente com a definição do estereótipo «Action» é importante restringir seu uso no que diz respeito à sua relação com campos de formulário, por meio de regras OCL adicionadas ao perfil.

Já em um **Perfil FrameWeb** para JSF, é possível definir uma instância de Controller denominada Bean para definir o estereótipo «Bean», que diferentemente da «Action» definida para Struts, não possui maiores restrições.

Ha também o estereótipo ResultType aplicável à Result para permitir que resultados obtidos a partir da execução de métodos das classes do controlador frontal possam ser representados. Os resultados podem ser tratados de formas variadas, dependendo do *frameworks* MVC.

Por exemplo, o Struts^2 possui elementos (*Result Elements*) e tipos de resultados (*Result Types*) específicos, que são determinados e estão vinculados pela "*string*" de retorno dos métodos de suas classes de ação. Esse comportamento ocorre porque estes *frameworks* são baseados em ações e não em componentes. Assim, para o Struts^2, é possível definir que uma instância de ResultType determina o seu tipo de um resultado.

Os tipos de resultado mais conhecidos do Struts^2 são: DispatcherResult para integração com outros recursos web, ChainResult para encadeamento de "*actions*" e RedirectResult para redirecionamento de URL. Há diversos outros tipos de resultados e estão detalhados na implementação da biblioteca ***FrameWeb*** para Struts^2, disponível para uso, colaboração e consulta[5].

Desta forma, em um **Perfil FrameWeb** para Struts^2, estando definido que «DispatcherResult» é uma instância de ResultType, será permitido ao modelador de um Sistema de Informação Web que faça uso deste *framework* pela aplicação desse perfil em seu **Modelo de Navegação**, definir uma instância de resultado (Result) estereotipada por «DispatcherResult» para indicar que em caso de sucesso (name = *'success'*) haverá o redirecionamento para uma determinada instância de dada página (Page).

Em conjunto com esta estratégia se faz necessário definir no **Perfil FrameWeb** para Struts^2, regras OCL para tratar as restrições de resultado (ResultConstraint) de acordo com as especificidades destes resultados, de forma a garantir o "*truthmaker*" destas restrições. O mesmo vale para ChainingConstraint. As regras OCL dependente de *framework*, são parte integrante da biblioteca deste *framework*, sendo portanto pertencentes ao **Perfil FrameWeb**.

Por fim, tem-se que Tag representa um estereótipo aplicável à UIComponent responsável por definir uma semântica distinta para cada componente UI de acordo com o *framework* que se deseja definir no **Perfil FrameWeb**.

[5] <https://github.com/bfmartins/FrameWeb-Martins-2015/tree/master/br.ufes.inf.nemo.FrameWeb_Martins2015.frameworks.MVC.Struts>

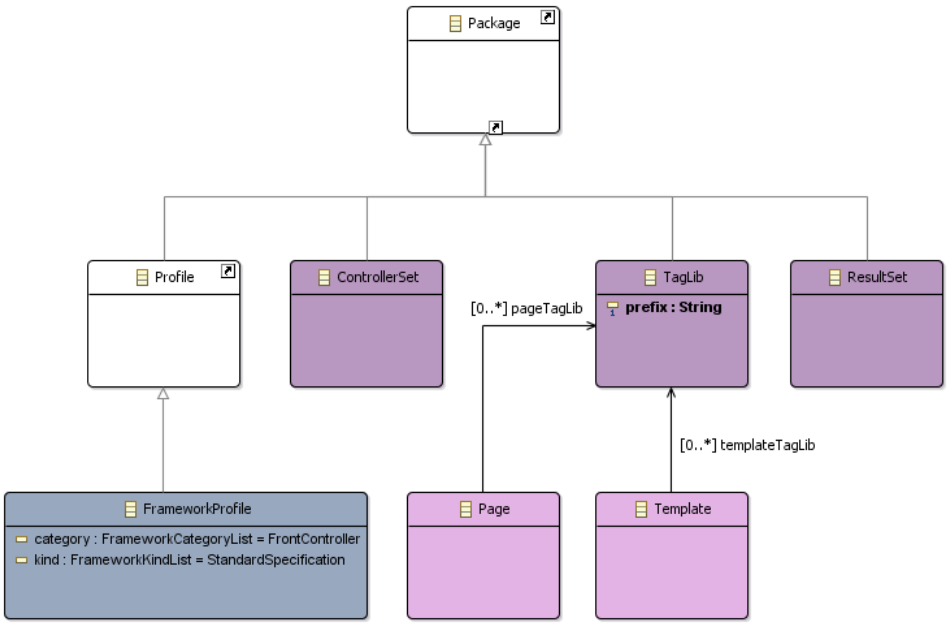


Figura 29 – Fragmento do Metamodelo de Framework do ***FW-15***: pacotes para MVC.

Desta forma em um **Perfil FrameWeb** para `JSF`, é possível definir uma instância de `Tag` denominada `Form`, que automaticamente é uma extensão de `UIComponent`. A partir da aplicação deste perfil em um **Modelo de Navegação**, `«Form»` aparecerá como uma estereotipo de um `UIComponent` disponível para uso.

Tanto para melhor organização quanto para garantir alguns aspectos semânticos importantes, os estereótipos `Controller`, `ResultSet` e `Tag` devem estar organizados em três pacotes específicos.

A Figura 29 apresenta o fragmento do **Metamodelo de Framework** contendo os detalhes dos pacotes usados para agrupar os recursos MVC dependente de *framework*. Um `FrameworkProfile` cuja categoria é `MVC`, pode possuir três pacotes principais (especializações de UML `Package`): `ControllerSet`, `TagLib` e `ResultSet`.

O pacote `ControllerSet` agrupa a diversidade de estereótipos (`Controller`) que podem ser aplicados a componentes UI e suas relações de extensão, mas não é obrigatório para todo **Perfil FrameWeb**. Este pacote também permite a definição de componentes customizados, entretanto não há maiores aspectos semânticos, porque este pacote está mais relacionado a questões de organização do perfil.

O pacote `ResultSet` também possui a função de agrupar a diversidade de estereótipos (`ResultType`) que podem ser aplicados a resultados e também não é obrigatório para todo **Perfil FrameWeb**. Para certos *frameworks* MVC, pode assumir aspectos semânticos mais específicos, entretanto em outros *frameworks* este recurso pode ser dispensável ou usado de forma mais geral, de acordo com o que for necessário.

Um pacote `TagLib` agrupa um certo conjunto de *tags*, *tag libraries* (bibliotecas de *tags*). Todo **Perfil FrameWeb** deve possuir pelo menos uma instância de `TagLib`. É possível definir *tag libraries* para padrões como `HTML`, `XHTML`, para especificações como `JSF`, para implementações de *frameworks* como `Spring MVC`, `Struts`[2], entre outras, ou até mesmo *tag libraries* customizadas pelo próprio modelador, caso ele deseje usar componentes customizáveis em suas páginas e modelos, cada qual contendo suas especificidades.

Por exemplo, em um **Perfil FrameWeb** para `JSF`, é possível definir uma instância de `TagLib` para representar a *JSF Core tag library* com prefixo (`prefix`) "*f*" e `URI` (que é meta-atributo de `Package`) <http://java.sun.com/jsf/core>. Esta *tag library* conterá as diversas *tags* do núcleo JSF disponíveis para uso neste perfil. Uma biblioteca contendo o **Perfil FrameWeb** para `JSF` está disponível para uso, colaboração e consulta[6].

A Figura 29 também mostra a relação entre a metaclasse de navegação `Page` com `TagLib`, que define o conjunto de *tag libraries* que irão compor em uma dada página web. Uma vez definido por parte do modelador, que em uma página certa dada *tag library* será usada, quaisquer de seus *tags* poderão ser usados para determinar os tipos de seus componentes `UIComponent`.

Evidentemente, é necessário acrescentar ao **Metamodelo de Framework** regras gerais, em OCL, para tratar as especificidades da relação entre os pacotes da Figura 29 e os estereótipos a eles pertencentes, semelhantemente ao que foi feito para os outros pacotes presentes no ***FW-15***. A Listagem 4.10 apresenta estas regras.

Listagem 4.10 – Regras OCL no contexto das figuras 28 e 29.

```
1 -- A ControllerLib must have only Controller or ControllerExtension
2 context ControllerLib
3   inv ControllerLibContent:
4     ((self.packagedElement.oclIsTypeOf(ControllerExtension)) or
5     (self.packagedElement.oclIsTypeOf(Controller)))
6
7 -- A tagLib must have only Tag or TagExtension
8 context ResultSet
9   inv ResultSetContent:
10    ((self.packagedElement.oclIsTypeOf(ResultExtension)) or
11    (self.packagedElement.oclIsTypeOf(ResultType)))
12  inv TagLibMinimum:
13    (self.packagedElement->size()>=1)
14
15 -- A ResultLib must have olny ResultSet or ResultExtension
16 context TagLib
17   inv TagLibContent:
18     ((self.packagedElement.oclIsTypeOf(TagExtension)) or
19     (self.packagedElement.oclIsTypeOf(Tag)))
```

Os estereótipos em UML são aplicados a classes por meio de associações de extensão (`Extension`) e são usados no **Metamodelo de Framework** para fornecer uma semântica

[6] <https://github.com/bfmartins/FrameWeb-Martins-2015/tree/master/br.ufes.inf.nemo.FrameWeb_Martins2015.frameworks.MVC.JSF>

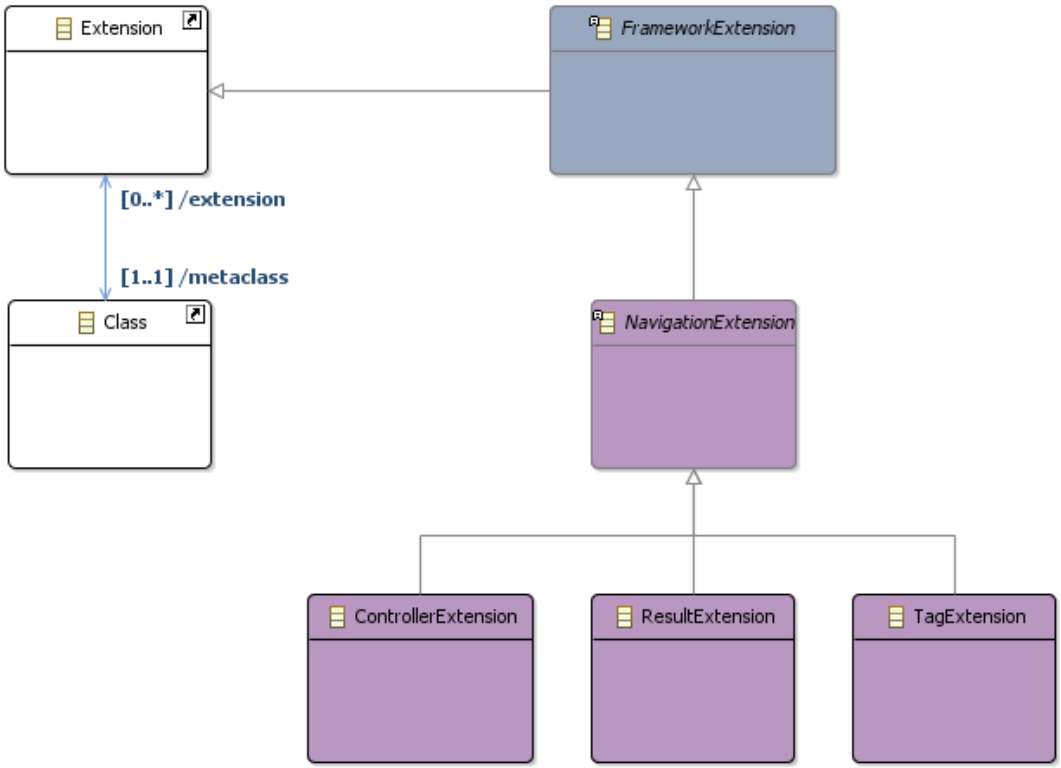


Figura 30 – Fragmento do Metamodelo de Framework do ***FW-15***: relações de extensão MVC.

específica quando aplicados. A Figura 30 apresenta o fragmento do **Metamodelo de Framework** contendo os detalhes das associações de extensão aplicáveis aos estereótipos usados para promover recursos MVC dependente de *framework*.

Um `FrameworkProfile` cuja categoria é `MVC`, pode possuir três associações de extensão (especializações de UML `Extension`): `ControllerExtension`, `ResultExtension` e `TagExtension`.

A extensão `ControllerExtension` é usada especificamente para aplicar o estereótipo `Controller` à `FrontControllerClass`, `ResultExtension` especificamente para aplicar `ResultType` à `Result` e `TagExtension` especificamente para aplicar `Tag` à `UIComponent`.

Evidentemente, é necessário acrescentar ao **Metamodelo de Framework** regras OCL para tratar as especificidades da relação de extensão (`Extension`), semelhantemente ao que foi feito para os outros relacionamentos presentes no ***FW-15***. A Listagem 4.11 apresenta as regras relacionadas às UML `Extension` na parte MVC do **Metamodelo de Framework**.

Usando essa mesma estratégia, diversas especificidades podem ser adicionadas às várias **Definições de Framework** para ajustar a semântica do metamodelo ***FW-15*** de acordo com as necessidades de modelagem.

Listagem 4.11 – Regras OCL no contexto da Figura 30.

```
1 -- A ControllerExtension must be applied only between Controller and
      FrontControllerClass
2 context ControllerExtension
3   inv ControllerExtensionRelation:
4     (self.metaclass.oclIsTypeOf(FrontControllerClass))
5
6 -- A FrontControllerClass extension must be a ControllerExtension
7 context FrontControllerClass
8   inv FrontControllerExtension:
9     (self.extension.oclIsTypeOf(ControllerExtension))
10
11 -- A ResultExtension must be applied only between ResultExtension and Result
12 context ResultExtension
13   inv ResultExtensionRelation:
14     (self.metaclass.oclIsTypeOf(Result))
15
16 -- A Result extension must be a ResultExtension
17 context Result
18   inv ResultExtension:
19     (self.extension.oclIsTypeOf(ResultExtension))
20
21 -- A TagExtension must be applied only between Tag and UIComponent
22 context TagExtension
23   inv TagExtensionRelation:
24     (self.metaclass.oclIsTypeOf(UIComponent))
25
26 -- A UIComponent extension must be a TagExtension
27 context UIComponent
28   inv UIComponentExtension:
29     (self.extension.oclIsTypeOf(TagExtension))
```

4.3.2 Mapeamento Objeto/Relacional (ORM)

O *framework* ORM (*Object/Relational Mapping* ou Mapeamento Objeto/Relacional) adotado no ***FW-07*** foi o `Hibernate`, que é, atualmente, uma implementação da especificação `JPA`. É desejável, no entanto, que além do `Hibernate` qualquer *framework* ORM (implementação `JPA` ou não) possa ser usada em **Definições de Framework** de categoria `ORM`.

Semelhantemente ao que foi exposto para a definição de *frameworks* MVC, a estratégia pode se estender para outras categorias de *frameworks*, como *frameworks* ORM. Portanto, é possível definir **Perfis FrameWeb** para categorias `ORM` semelhantemente ao que foi exposto para `MVC`. Assim, é possível criar bibliotecas para especificações como a `JPA`, ou implementações como `Hibernate`, `OpenJPA`, `EclipseLink`, dentre outras.

Por consequência, há comportamentos que são diferentes entre estas implementações e interferem no metamodelo ***FW-15***, sendo portanto necessário ou desejável (por flexibilidade) tratá-los.

O método ***FrameWeb*** trata mapeamentos de persistência, que são metadados das classes de domínio usados para permitir aos *frameworks* ORM transformar objetos de memória em linhas de tabelas em um banco de dados relacional e vice-versa. Para isso, usa restrições adicionados à representação destes mapeamentos nos **Modelo de Entidades**,

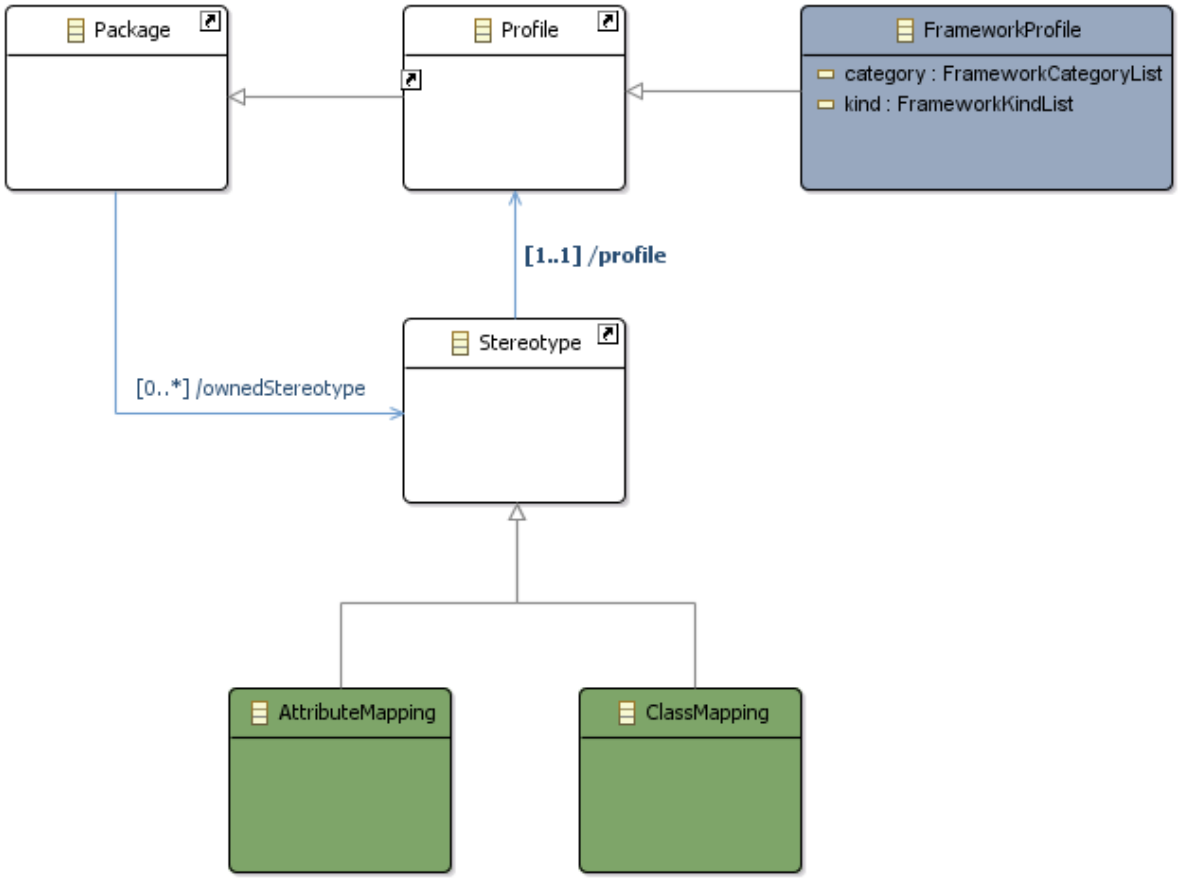


Figura 31 – Fragmento do Metamodelo de Framework do ***FW-15***: estereótipos para ORM.

como já visto na Seção 4.2.7. Porém, para tratar os aspectos dependente de *framework* são usados estereótipos específicos para os diferentes mapeamentos possíveis.

Então, semelhantemente ao que foi feito na definição de *tag libraries* para *frameworks* MVC, há um conjunto de tipos de mapeamentos possíveis e distintos quando um Sistema de Informação Web usa o `Hibernate` (sem `JPA`) e outro para `JPA` ou outra implementação `JPA`. Assim a definição destes tipos de mapeamento são feitos pelas metaclasses `ClassMapping` e `AtributeMapping`, conforme detalhado na Figura 31.

O `ClassMapping` representa um estereótipo aplicável à `DomainClass`, responsável por definir uma semântica distinta para cada classe de domínio conforme o *framework* que se deseja definir em um **Perfil FrameWeb**. Ou seja, `ClassMapping` estende `DomainClass` por meio de uma associação de extensão, que está detalhada mais adiante nesta seção.

Desta forma em um **Perfil FrameWeb** para `JPA`, é possível definir instâncias de `ClassMapping` denominadas `Persistent`, `Transient` e `Mapped`, que automaticamente são extensões de `DomainClass`. A partir da aplicação deste perfil em um **Modelo de Entidades**, `«Persistent»`, `«Transient»` e `«Mapped»` aparecerão como estereótipos de uma `DomainClass` disponível para uso.

Já para `Hibernate`, é possível definir instâncias de `ClassMapping` denominadas `Persistent`, `Transient` e `Detaiched`, que aparecerão como estereótipos de uma `DomainClass` disponível para uso como `«Persistent»`, `«Transient»` e `«Detaiched»` a partir da aplicação do perfil `Hibernate` em um **Modelo de Entidades**.

O `AtributeMapping` representa um estereótipo aplicável à `DomainAtribute`, responsável por definir uma semântica distinta para cada atributo das classes de domínio conforme o *framework* que se deseja definir em um **Perfil FrameWeb**. Ou seja, `AtributeMapping` estende `DomainAtribute` por meio de uma associação de extensão.

Então, de forma semelhante é possível definir diversas instâncias de `AtributeMapping`, que automaticamente são extensões de `DomainClass` e aparecerão como estereótipos de uma `DomainAtribute` disponível para uso, a partir da aplicação do perfil em um **Modelo de Entidades**.

Desta forma os mapeamentos definidos para um dado *framework* são tratados no metamodelo diferenciadamente, podendo ser usados para definir as diferentes classes e atributos de domínio. Veremos no Capítulo 5 exemplos do uso destas definições.

Além disso, é possível adicionar regras OCL, não só para lidar com aspectos da própria **Definição de Framework**, mas também para ajustar a semântica de um *framework* ORM específico segundo suas características, utilizando as mesmas técnicas já mencionadas anteriormente na Seção 4.3.1.

No primeiro caso, por exemplo, o método ***FrameWeb*** define que no **Modelo de Entidades**, o nome das classes (entidades) deve aparecer como padrão inicial (*default*) para tabelas, logo a coluna `table` deve ser inicializada com o valor do nome da instância da metaclasse `DomainClass`, conforme mostrado na Listagem 4.12.

Listagem 4.12 – Regras OCL no contexto do atributo `table`.

```
1 -- The default table name in a DomainClass is the class name.
2 Context DomainClass::table
3   derive:
4     self.name
```

Já para a definição de regras específicas a um dado *framework*, o método ***FrameWeb*** define no **Modelo de Persistência** por exemplo, uma nomenclatura específica para as instâncias de `DAOInterface` e `DAOClass`, onde o nome (`name`) é composto de prefixos e sufixos que variam conforme o *framework* ORM escolhido.

Assim, além de valores padrão (*default*) para nomear esses elementos é necessário incluir restrições para garantir a semântica desejada, conforme pode ser observado na Listagem 4.13.

Listagem 4.13 – Regras OCL no contexto dos nomes de `DAOClass` e `DAOInterface`.

```
1  -- The DAOInterface name must be composed by the domain interface name plus the
       sufix 'DAO'.
2  context DAOInterface
3    inv DAOInterfaceNameValue:
4      (self.name = self.infix.concat(self.sufix)) and
5      (self.infix = self.client.oclIsTypeOf(DAOClass).prefix)
6
7  -- The DAOInterface name must be composed by the domain interface name plus the
       sufix 'DAO' (only English representation).
8  Context DAOInterface::name
9    derive:
10     self.infix.concat(self.sufix)
11
12 -- The DAOInterface infix must be composed by the DAOClass prefix that realize
       its.
13 context DAOInterface::infix
14   derive:
15     self.client.oclIsTypeOf(DAOClass).prefix
16
17 -- The DAOClass name must be composed by the domain interface name plus 'DAO'
       plus the ORM framework name.
18 context DAOClass
19   inv DAOClassNameValue:
20     (self.name = self.prefix.concat(self.infix.concat(self.sufix))) and
21     (self.infix = self.supplier.oclIsTypeOf(DAOInterface).infix)
22
23 -- The DAOClass name must be composed by the domain interface name plus 'DAO'
       plus the ORM framework name (only English representation).
24 Context DAOClass::name
25   derive:
26     self.prefix.concat(self.infix.concat(self.sufix))
27
28 -- The DAOClass sufix must be composed by the the ORM framework name.
29 Context DAOClass::sufix
30   derive:
31     if Framework.category = "ORM" then Framework.name endif
32
33 -- The DAOClass prefix must be composed by the DAOInterface infix that is its
       realization.
34 Context DAOClass::prefix
35   derive:
36     self.supplier.oclIsTypeOf(DAOInterface).infix
```

Para que as derivações da Listagem 4.13 funcionem corretamente é necessário que os atributos `sufix` da `DAOInterface`, e `infix`, `sufix` da `DAOClass` não sejam passíveis de modificação (*not changeable*).

Além disso o `infix` da `DAOInterface`, e o atributo `prefix` da `DAOClass` devem estar relacionados entre si pela relação de realização `DAORealization`, porque não é possível alterar a definição original do atributo `name` dos `NamedElement`, já que a ferramenta não nos permite redefinir o atributo `name` em nas especializações de `NamedElement`.

Entretanto, há além destes exemplos várias outras regras necessárias para definir a correta semântica que se deseja, tanto no que diz respeito ao **Metamodelo de Framework**, quanto às regras específicas para tratar a questão dependente de *framework* de cada caso. Exemplos são apresentados no Capítulo 5.

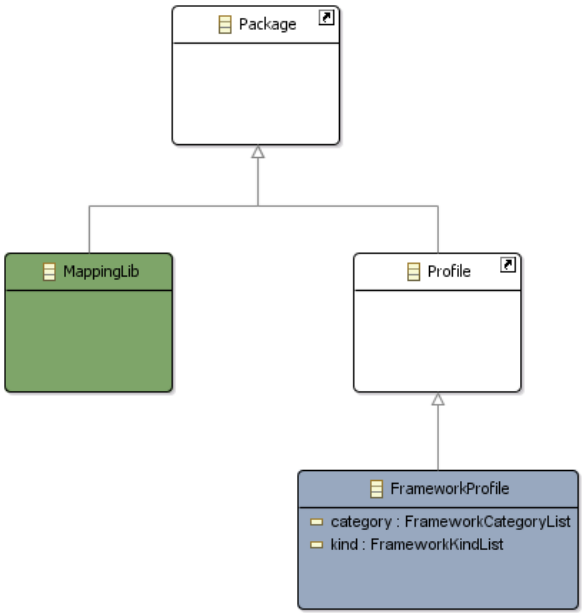


Figura 32 – Fragmento do Metamodelo de Framework do ***FW-15***: pacotes para ORM.

Diversos mapeamentos podem ser usados em *frameworks* ORM, e todos eles estão agrupados em um pacote representado pela metaclasse `MappingLib`, conforme pode ser observado na Figura 32, agrupando mapeamentos para classes e atributos.

O pacote `MappingLib` agrupa a diversidade de estereótipos (`ClassMapping` e `AtributeMapping`) que podem ser aplicados a classes de domínio e seus atributos, bem como suas relações de extensão. Sendo obrigatório para todo **Perfil FrameWeb** que o pacote `MappingLib` seja definido e que ele possua pelo menos um `ClassMapping` e um `AtributeMapping`.

Então, uma vez definido por parte do modelador, que, por exemplo, o Sistema de Informação Web usará apenas o `Hibernate` como ORM, quaisquer de seus mapeamentos poderão ser usados no **Modelo de Entidades**. O mesmo é válido para qualquer outra seleção de *framework* ORM feita pelo modelador.

Evidentemente, é necessário acrescentar ao **Metamodelo de Framework** regras gerais, em OCL, para tratar as especificidades da relação entre os pacotes da Figura 32 e os estereótipos a eles pertencentes, semelhantemente ao que foi feito para os outros pacotes presentes no ***FW-15***. A Listagem 4.14 apresenta estas regras.

Os estereótipos em UML são aplicados a classes por meio de associações de extensão (`Extension`) e são usados no **Metamodelo de Framework** para fornecer uma semântica específica quando seus estereótipos forem aplicados.

A Figura 33 apresenta o fragmento do **Metamodelo de Framework** contendo os detalhes das associações de extensão aplicáveis aos estereótipos usados para promover recursos ORM dependente de *framework*. Um `FrameworkProfile` cuja categoria é

Listagem 4.14 – Regras OCL no contexto das figuras 31 e 32.

```
1 -- A MappingLib must have only ClassMapping or DomainExtension or
     AttributeMapping or ControllerExtension
2 context MappingLib
3   inv MappingLibContent:
4     ((self.packagedElement.oclIsTypeOf(DomainExtension)) or
5     (self.packagedElement.oclIsTypeOf(AttributeExtension)) or
6     (self.packagedElement.oclIsTypeOf(ClassMapping) or
7     (self.packagedElement.oclIsTypeOf(AttributeMapping)))
```

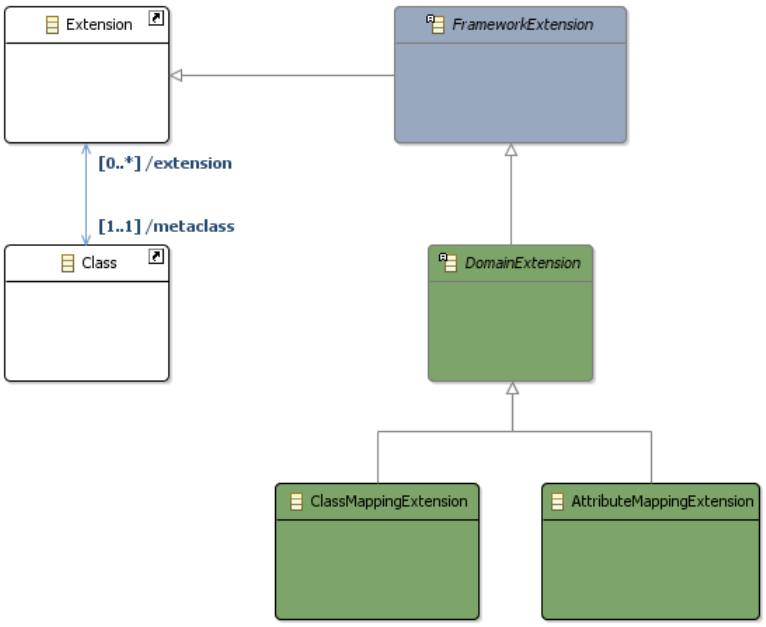


Figura 33 – Fragmento do Metamodelo de Framework do ***FW-15***: relações de extensão ORM.

`ORM`, pode possuir três associações de extensão (especializações de UML `Extension`): `ClassMappingExtension` e `AttributeMappingExtension`.

A extensão `ClassMappingExtension` é usada especificamente para aplicar o estereótipo `ClassMapping` à `DomainClass` e `AttributeMappingExtension` especificamente para aplicar `AtributeMapping` à `DomainAtribute`.

Evidentemente, é necessário acrescentar ao **Metamodelo de Framework** regras OCL para tratar as especificidades da relação de extensão (`Extension`), semelhantemente ao que foi feito para os outros relacionamentos presentes no ***FW-15***. A Listagem 33 apresenta as regras relacionadas às UML `Extension` na parte ORM do **Metamodelo de Framework**.

Usando essa mesma estratégia, diversas especificidades podem ser adicionadas às várias **Definições de Framework** para ajustar a semântica do metamodelo ***FW-15*** de acordo com as necessidades de modelagem.

Listagem 4.15 – Regras OCL no contexto da Figura 33.

```
1 -- A ControllerExtension must be applied only between Controller and
      FrontControllerClass
2 context ControllerExtension
3   inv ControllerExtensionRelation:
4     (self.metaclass.oclIsTypeOf(FrontControllerClass))
5
6 -- A FrontControllerClass extension must be a ControllerExtension
7 context FrontControllerClass
8   inv FrontControllerExtension:
9     (self.extension.oclIsTypeOf(ControllerExtension))
10
11 -- A ResultExtension must be applied only between ResultExtension and Result
12 context ResultExtension
13   inv ResultExtensionRelation:
14     (self.metaclass.oclIsTypeOf(Result))
15
16 -- A Result extension must be a ResultExtension
17 context Result
18   inv ResultExtension:
19     (self.extension.oclIsTypeOf(ResultExtension))
20
21 -- A TagExtension must be applied only between Tag and UIComponent
22 context TagExtension
23   inv TagExtensionRelation:
24     (self.metaclass.oclIsTypeOf(UIComponent))
25
26 -- A UIComponent extension must be a TagExtension
27 context UIComponent
28   inv UIComponentExtension:
29     (self.extension.oclIsTypeOf(TagExtension))
```

4.4 Implementação

Devido às decisões de projeto já mencionadas no Capítulo 3, a implementação da sintaxe abstrata ***FW-15*** produziu um metamodelo bastante extenso, visto que mantém todos os elementos do metamodelo UML e não apenas aqueles que compõem os diagramas de classe.

Essa característica do metamodelo ***FW-15***, em adição ao conjunto de requisitos para a evolução do método, tornou necessário uma observância cuidadosa no que diz respeito às questões relacionadas ao decurso de tempo deste trabalho. Assim o projeto de uma ferramenta para o método necessitou ser tratado em etapas distintas e organizadas.

Primeiramente, e como base para outras etapas posteriores está a formalização do metamodelo ***FW-15***, na sequência ocorre a verificação e validação deste metamodelo, para só então permitir que ferramentas mais elaboradas sejam projetadas e construídas. Entretanto, o processo de verificação e validação só pode ser feito com o apoio de uma ferramenta que apresenta as características mínimas para a construção de modelos ***FrameWeb***. Por isso, além do metamodelo, este trabalho implementou uma ferramenta simplificada com esta finalidade.

Em um segundo momento, com base nas conclusões deste trabalho e dando continuidade a ele, há projetos de graduação nos quais os estudantes farão uso deste recurso para a construção de uma ferramenta gráfica mais completa. Por isso, além do metamodelo este trabalho fornece as diretrizes básicas para o desenvolvimento desta ferramenta.

4.4.1 Preparação do ambiente de trabalho

A implementação de um metamodelo requer o uso de uma ferramenta específica, baseada em uma linguagem gráfica para metamodelagem. Ou seja, uma ferramenta que trabalhe com linguagem para construção de linguagens gráficas.

O processo de implementação do metamodelo ***FW-15*** foi feito por meio do Eclipse EMF (`Luna Service Release 1a - 4.4.1 build 20150109-0600`) e seus "*plug-ins*". Esta não é a versão mais recente da ferramenta, mas é a mais estável e seu "*plug-ins*" para geração de elementos gráficos (*OBEO Designer*) possui uma versão gratuita (*Community*). A versão mais recente desta ferramenta é comercial, portanto adicionaria custos desnecessários ao projeto. Em especial por causa dos projetos de graduação, nos quais os estudantes farão uso deste recurso.

O EMF utiliza três principais recursos para a criação de metamodelos: (i) a linguagem ECore; (ii) diagramas; e (iii) o arquivo `.genmodel`.

A linguagem ECore é composta um conjunto de elementos básicos usados na metamodelagem, seus principais construtos são:

- `EClass:` que representa uma classe, contendo zero ou mais atributos e zero ou mais associações com outras classes;
- `EAttribute:` para representar um atributo de uma classe, podendo ter um nome e um tipo definidos;
- `EReference:` representando a associação entre duas classes, suportando variações para indicar uma composição (associação "*todo/parte*");
- `EDataType:` para representar os tipos possíveis de atributos (EBoolean, EChar, ELong, etc.).

Assim como qualquer linguagem gráfica, a ECore possui um metamodelo. Tal metamodelo está disponível na documentação fornecida pelo Eclipse[7], assim como as informações a respeito de todos os construtos da linguagem.

[7] <http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html>

Um metamodelo produzido pelo EMF com a linguagem ECore é composto além do arquivo *.ecore*, que contém o metamodelo em si, mas também da representação deste metamodelo por meio de um ou mais diagramas, armazenados em um arquivo de formato `.aird`, bem como de um conjunto de configurações armazenadas em um arquivo de formato `.genmodel`. Estes arquivos estão vinculados a um projeto de modelagem (*ECore Modelling Project*) no EMF.

4.4.2 Configuração e aspectos estruturais

No projeto do metamodelo ***FW-15***, foi necessário instalar como recurso adicional o metamodelo UML, no qual este se baseia. O EMF trata este tipo de recurso adicional como um "*plug-in*", que depois de instalado pode ser usado em qualquer projeto.

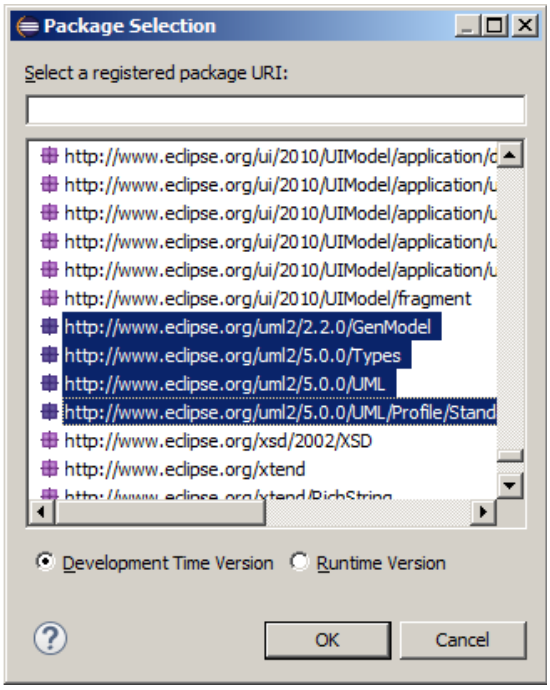
O metamodelo UML utilizado como infraestrutura de base foi obtido diretamente da página do Eclipse[8], sendo composto do SDK, arquivos usados em tempo de execução e exemplos. O SDK é adicionado ao EMF como pode ser observado na Figura 34a e após aplicado ao metamodelo aparece conforme apresentado na Figura 34b. A Figura 34 apresenta também o arquivo `UML.ecore` adicionado como uma dependência de projeto, permitindo que os construtos UML possam ser incorporados aos ***FW-15***.

Como a linguagem ECore possui gramática própria, o metamodelo UML sob ela definido, está sujeito às suas regras gramaticais, consequentemente pode ocorrer certa diferenciação na representação deste metamodelo em comparação com a especificação UML. Em especial porque a linguagem ECore não permite, nesta versão, o uso de redefinição, união, nem especialização de associações (conhecidas como `eReferences`).

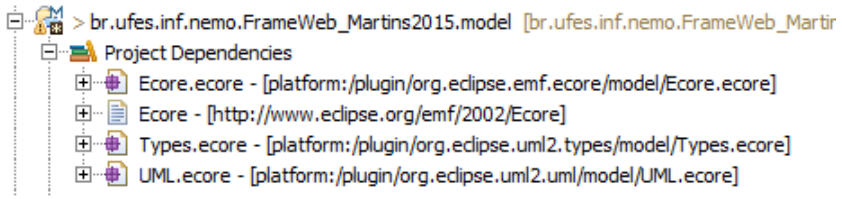
A Figura 35a mostra um exemplo desta característica na observação do metamodelo UML apresentado graficamente em ECore na Figura 35b e sua representação gráfica apresentada na especificação UML.

Por exemplo, na Figura 35a é possível notar a ausência de algumas associações que possuem restrições, como a associação `ownedEnd` entre `Extension` e `ExtensionEnd`, cuja restrição é uma redefinção da associação de mesmo nome que existe entre `Association` e `Property` (`{redefines ownedEnd}`). Aparentemente esta relação não está sendo representada. Porém, na realidade, ocorre que apenas a redefinição não está representada. Por outro lado, a relação original entre `Association` e `Property` existe e é aplicada também entre as classes especializadas `Extension` e `ExtensionEnd`, pela aplicação do princípio de Liskov (LISKOV, 1988). Entretanto, é necessário tratar a cardinalidade da relação quando esta ocorre no contexto das classes especializadas, o que pode ser feito via OCL ou diretamente via código.

[8] <http://www.eclipse.org/modeling/mdt/downloads/?project=uml2>



(a) SDK UML no Eclipse EMF



(b) SDK UML aplicado ao ***FW-15***

Figura 34 – Aplicação do SDK UML ao metamodelo ***FW-15*** no Eclipse EMF.

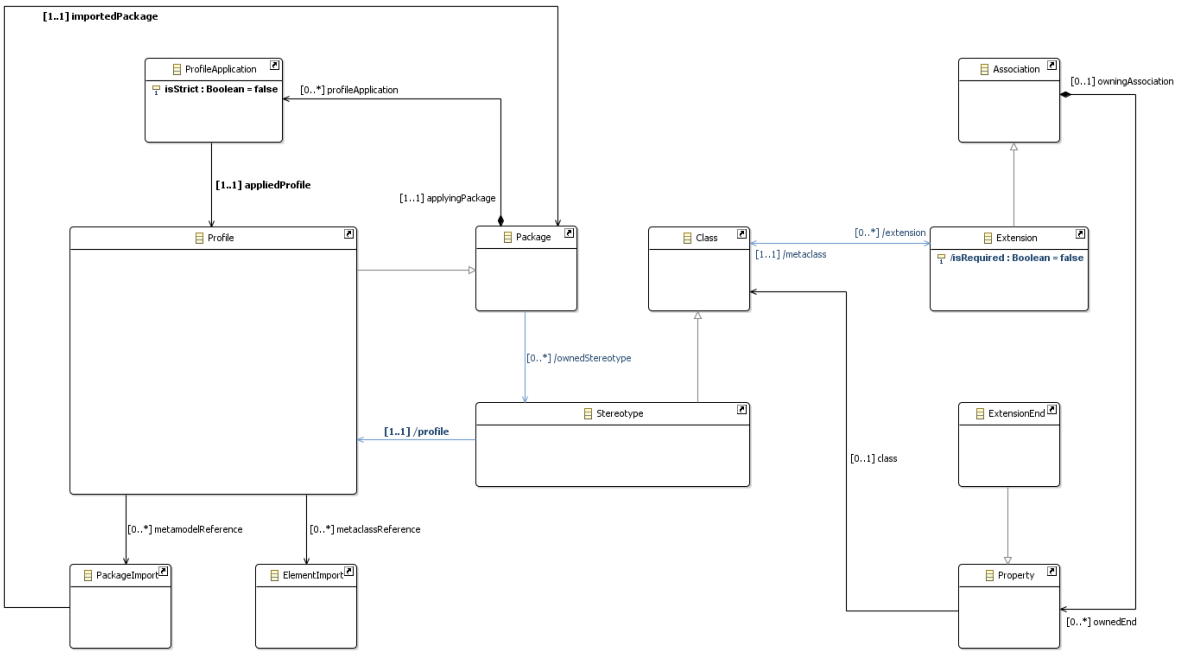
Apesar destas diferenças, o metamodelo UML em ECore é perfeitamente funcional de acordo com a especificação UML, porque as definições necessárias são complementadas via regras OCL. Entretanto, esta abordagem aumenta a complexidade do metamodelo.

Este mesmo tratamento pode ser observado no metamodelo ***FW-15***, quando se faz necessário o uso de regras OCL redefinir ou especializar associações do metamodelo UML no metamodelo ***FW-15***.

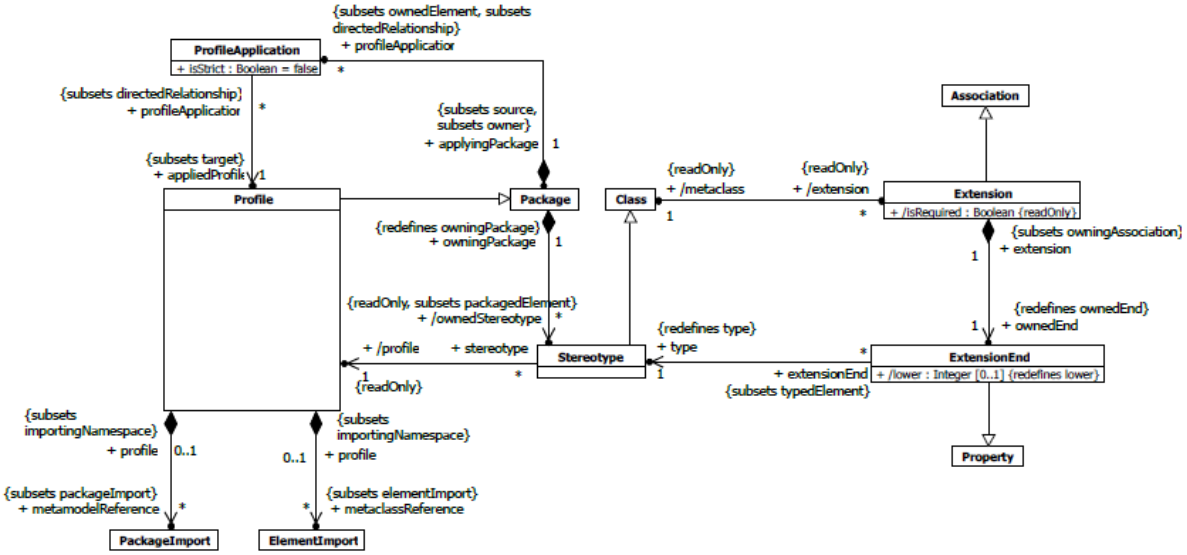
4.4.3 Protótipo de um editor

A partir de um metamodelo em ECore, o EMF é capaz de fazer a geração de código Java tanto deste metamodelo como também de um editor não gráfico, porém suficiente para verificação e validação do metamodelo.

A verificação do metamodelo por meio deste tipo de editor é bastante interessante, porque permite que o metamodelo seja modificado sem que grande esforço de adaptação seja necessário no que diz respeito ao impacto no editor, já que a geração de código é automática em sua maior parte.



(a) Metamodelo UML em ECore



(b) Especificação UML ***FW-15***

Figura 35 – Aspectos do metamodelo UML em ECore em relação à especificação UML (Figura 12.12 Profiles).

Considerando as questões relacionadas à complexidade do projeto já mencionadas anteriormente, optou-se por trabalhar apenas no contexto deste editor, mesmo considerando suas limitações visuais. Esta abordagem permitiu que a maior atenção fosse dispensada com as questões mais importantes para a evolução do método no que diz respeito aos requisitos do projeto, de forma a proporcionar uma base mais sólida para a continuidade do desenvolvimento em futuros trabalhos dos alunos de graduação.

O gerador de código Java do EMF é baseado em duas ferramentas: o JET (*Java Emiter Templates*); e o JMerge (*Java Merge*).

O JET é um "*plug-in*" padrão do EMF, que pode ser usado para geração de código em diversas linguagens (SQL, XML, Java, etc.). O JET permite que modelos (*templates*) para o código Java (no formato `javajet`), produza código Java. O JavaJet é uma linguagem de marcação semelhante ao JSP (*JSP-like*), entretanto seu uso requer bastante tempo de programação, já que o resultado produzido necessita ser verificado e validado em dois níveis de abstração diferentes, o primeiro quanto ao código Java produzido e o segundo no que diz respeito ao resultado da execução deste código Java.

O EMF possui seu conjunto padrão de bibliotecas JET para geração de código Java, mas também permite que modelos específicas sejam adicionadas para executarem geração de código adicional para metamodelos ECore, quando necessário. Esta funcionalidade foi usada na geração do editor para ***FW-15*** de forma a adicionar os códigos específicos relacionados às regras OCL do metamodelo.

O editor ***FW-15*** produzido por meio destes recursos da ferramenta EMF trabalha apresentando os modelos em uma visão de árvore (*tree view*), mas é perfeitamente funcional.

4.4.4 Diretrizes para uma ferramenta gráfica

É possível produzir também uma ferramenta gráfica para o método utilizando como "*plug-in*" do EMF o OBEO Design (*Community*)[9]. Esta ferramenta cria "pontos de vista" (*viewpoints*) para metamodelos em ECore, sendo possível definir objetos gráficos para cada construto do metamodelo. Estes pontos de vista são gravados em arquivos no formato `.odesign`.

Tendo formatado graficamente todos os construtos do metamodelo, é possível vincular este "pontos de vista" (*viewpoints*) ao metamodelo desejado. Então é possível criar "pontos de vista" (*viewpoints*) para atender o metamodelo ***FW-15*** e criar o devido vínculo entre os construtos e sua representação gráfica no EMF.

Como é requisito de projeto que o ***FW-15*** mantenha sua similaridade com a UML, a representação dos construtos do metamodelo devem ser baseadas nas mesmas representações gráficas UML, variando apenas quanto a apresentação de estereótipos já pré-determinados em função dos *frameworks* associados aos modelos.

4.5 Conclusões do Capítulo

A implementação da linguagem ***FW-15*** em um metamodelo capaz de atender a todos os requisitos impostos foi feita utilizando a abordagem ECore com a ferramenta EMF

[9] <http://marketplace.obeonetwork.com/module/OD>

Eclipse. Neste contexto algumas considerações importantes a respeito da implementação devem ser feitas:

- A necessidade de amplo estudo do metamodelo UML no qual o metamodelo ***FW-15*** foi baseado, a fim de permitir a correta representação dos elementos ***FW-15*** como especializações de elementos UML;
- Como a UML é uma linguagem de uso geral e por que a linguagem ECore não permite a implementação especializações de associações (ERererence), a filtragem necessária para permitir sua representação sem especializá-las no metamodelo ***FW-15*** em relação ao UML foi feita por meio de regras OCL;
- Diversas outras regras OCL também foram adicionadas para deflagrar o comportamento desejado ao metamodelo ***FW-15***, não relacionados a aspectos do metamodelo UML, consequentemente o metamodelo ***FW-15*** é bastante carregado destas regras;
- A linguagem ECore não permite o uso de perfis ("*ECore Profiles*"), por isso os recursos de perfil usados pelo metamodelo ***FW-15*** derivam dos perfis UML definidos em seu metamodelo. Esta importante abordagem permite a flexibilidade e extensibilidade desejada;
- Apesar do EMF fornecer recursos para geração automática de código (i.e. "*plug-in*" JET), houve necessidade de alguma codificação complementar, principalmente para tratar questões relacionadas a conversão de tipos ("*casting*");
- Por ser derivado do metamodelo UML, futuras alterações na sua especificação podem eventualmente exigir adequações no metamodelo ***FW-15***.

A implementação do metamodelo ***FW-15*** abre caminho para novas oportunidades de estudo e pesquisa, por exemplo:

- Para a implementação de uma ferramenta gráfica mais completa, capaz de apoiar o desenvolvimento de aplicações com o método ***FrameWeb*** em toda sua plenitude.
- Para a avaliação dos modelos criados em diversos Sistemas de Informação Web com respeito a aspectos relacionados à qualidade do software, além de outros que são importantes no processo de software dentro da WebE;
- Para a extensão deste metamodelo para atingir outras categorias de *frameworks*, bem como para adição de tecnologias relacionadas à Web Semântica.

No Capítulo 5 veremos como o metamodelo ***FW-15*** implementado foi avaliado por meio de uma prova de conceito.

5 Prova de conceito

O protótipo de ferramenta produzido no Capítulo 4 é uma das contribuições do trabalho proposto, com base no processo de formalização e evolução do método ***FrameWeb***: o metamodelo ***FW-15***. Neste capítulo, apresentam-se diversos modelos ***FrameWeb***, implementados com o apoio desta ferramenta como forma de verificar e validar o metamodelo ***FW-15***, de modo a permitir a continuidade dos trabalhos relacionados ao método.

5.1 Avaliação de modelos FrameWeb

Ao longo do trabalho conduzido no curso de mestrado, alunos da graduação desenvolveram modelos com base no método ***FrameWeb***, não só fomentando seu uso como também contribuindo para sua avaliação. Os modelos produzidos nestes trabalhos foram utilizados como prova de conceito do metamodelo por meio do editor proposto neste trabalho. Na sequência desta seção apresentamos os resultados obtidos.

Em Duarte (2014), o primeiro trabalho deste esforço de avaliação, definiu-se o SCAP (Sistema de Controle para Afastamento de Professores) como escopo comum a todos os trabalhos, de forma a permitir uma base comparação para os diversos modelos de projeto produzidos a partir dos mesmos requisitos e análise. A partir desta primeira iniciativa, cada trabalho de graduação foi direcionado a *frameworks* diferentes em suas respectivas fases de projeto.

O propósito do SCAP é ser um Sistema de Informação Web capaz de gerenciar e automatizar o processo para solicitação de afastamento feito pelos docentes do Departamento de Informática da universidade para participação em eventos no Brasil ou exterior.

A Figura 36 apresenta o modelo conceitual do SCAP utilizado nos trabalhos selecionados como prova de conceito do metamodelo ***FW-15***. Este modelo é resultado de uma análise colaborativa entre a equipe de trabalho ***FrameWeb***, composta pelo orientador, um aluno de mestrado e alunos de graduação.

No SCAP pessoas (`Person`) podem ser subdivididas em dois grupos: Secretários (`Secretary`) e Professores (`Professor`), que podem eventualmente ter relações de parentesco (`Kinship`) com outros professores, que devem ser avaliadas para efeito da solicitação de afastamento. Professores podem solicitar afastamento (`LeaveAbsence`) ou podem opinar a respeito dos afastamentos de outros professores por meio de pareceres (`Opinion`). Além disso professores assumem em caráter temporário o papel de chefe do departamento (`HeadMandate`). A partir de uma solicitação de afastamento é instaurado um processo, no

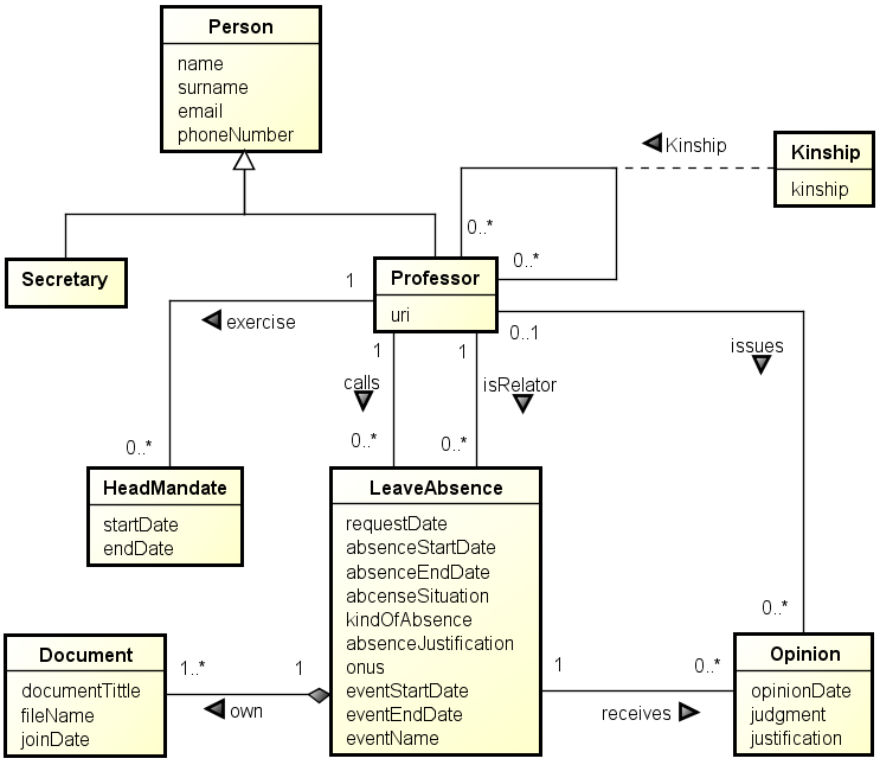


Figura 36 – Modelo conceitual do SCAP (PRADO, 2015), tradução nossa.

qual um professor assume o papel de relator, na sequência diversos documentos (`Document`) são anexados complementando o processo, que será concluído após todos os seus trâmites.

O SCAP é composto de dois sub-sistemas: um para tratar os aspectos principais do sistema que são relacionados ao afastamento dos professores e o segundo para a parte administrativa desempenhada pelo secretário. Neste documento apresentaremos apenas alguns modelos relacionados à solicitação de afastamento para demonstrar o resultado obtido, evitando assim alongar desnecessariamente o texto.

Durante a fase de projeto do Sistema de Informação Web, o modelador deverá usar o modelo conceitual como base para diagramar o sistema segundo o método ***FrameWeb***. Nesta etapa o modelo conceitual serve como base angular para o projeto do **Modelo de Entidades**, refletindo nele as entidades a nível do domínio do Sistema de Informação Web. Então, são diagramados os **Modelos de Persistência** e **Aplicação** que definem a estrutura de armazenamento, o comportamento e as regras do Sistema de Informação Web, e finalmente o **Modelo de Navegação** que trata dos aspectos de interface do sistema. Contudo, não há uma sequência obrigatória para que estes modelos sejam construídos, deixando o modelador livre para trabalhar conforme sua necessidade.

Nas seções 5.2 e 5.3 apresentam-se detalhes sobre o projeto do SCAP apresentados por alunos de graduação utilizando diferentes *frameworks*.

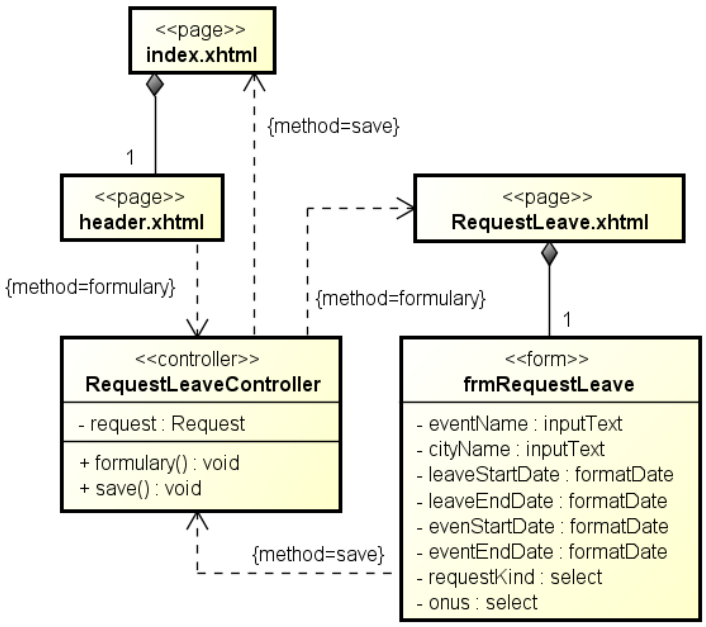


Figura 37 – Representação gráfica UML do **Modelo de Navegação** para afastamentos do SCAP em projeto com o `VRaptor` e `JPA` (PRADO, 2015).

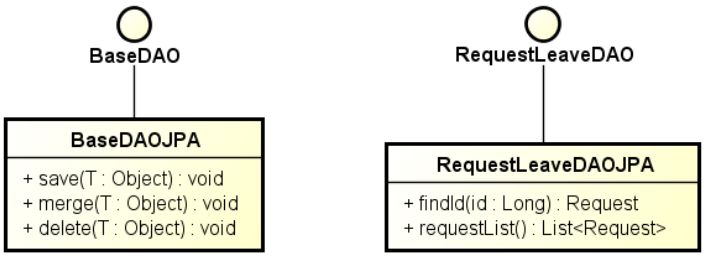
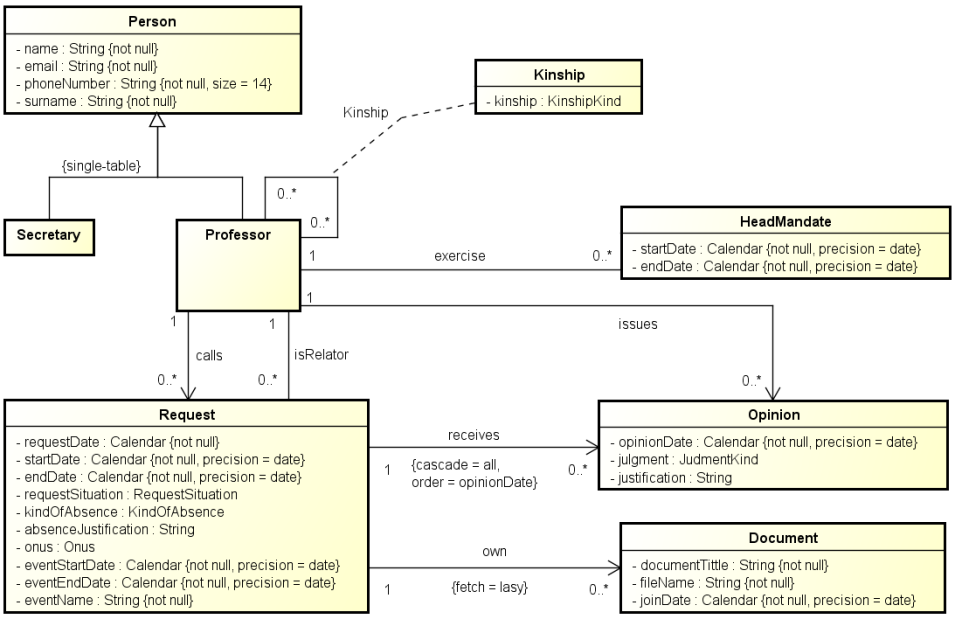


Figura 38 – Representação gráfica UML do **Modelo de Persistência** para afastamentos do SCAP em projeto com o `VRaptor` e `JPA` (PRADO, 2015).

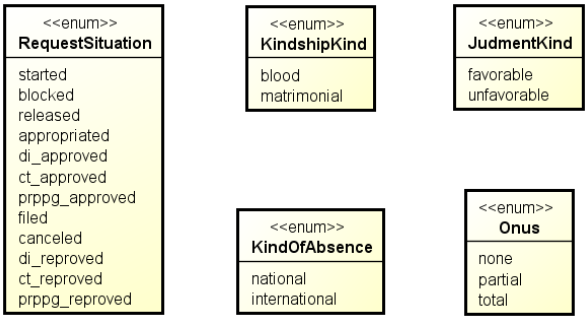
5.2 Aplicação em VRaptor e JPA

O `VRaptor` é um *framework* MVC baseado em ações bastante flexível, que aceita a injeção de dependência das classes declaradas no construtor, trabalha com *cast* automático e outras diversas verificações na comunicação entre as camadas de visão e controle. Pode ser integrado com `JPA` ou `Hibernate`, sendo capaz de validar dados e responder a erros dinamicamente com o seu validador próprio ou usando validadores como o do `Hibernate`.

A Figura 37 apresenta graficamente o fragmento do **Modelo de Navegação *FrameWeb*** para afastamentos do SCAP em projeto com o `VRaptor` e `JPA`, apresentado em (PRADO, 2015), por meio de uma tradicional ferramenta gráfica para UML. As figuras 38 e 39 apresentam, respectivamente, os fragmentos do **Modelo de Persistência** e do **Modelo de Entidades** para este mesmo projeto.



(a) **Modelo de Entidades** SCAP - afastamentos



(b) Enumerações

Figura 39 – Representação gráfica UML do **Modelo de Entidades** para afastamentos do SCAP em projeto com o `VRaptor` e `JPA` (PRADO, 2015).

O mesmo **Modelo de Navegação *FrameWeb*** representado no protótipo do trabalho de Prado (2015), com uma apresentação em árvore (*treeview*), está disponível no repositório deste projeto para consulta. Nele todas as estruturas relacionadas ao caso de uso estudado estão implementadas.

A Figura 40 apresenta um fragmento deste **Modelo de Navegação *FrameWeb*** agora representado no protótipo deste trabalho, com uma apresentação em árvore (*treeview*). Além das estruturas próprias do modelo, é possível notar na Figura 40 a presença das bibliotecas que contem os perfis para o `VRaptor`, `HTML`, `JSTL` e `JPA`. As três primeiras aplicadas ao **Modelo de Navegação** e a última aplicada ao **Modelo de Entidades**. Cada **Definição de Framework** é responsável por definir os estereótipos aplicados aos modelos ***FrameWeb***, dando assim a representatividade desejada por parte do modelador.

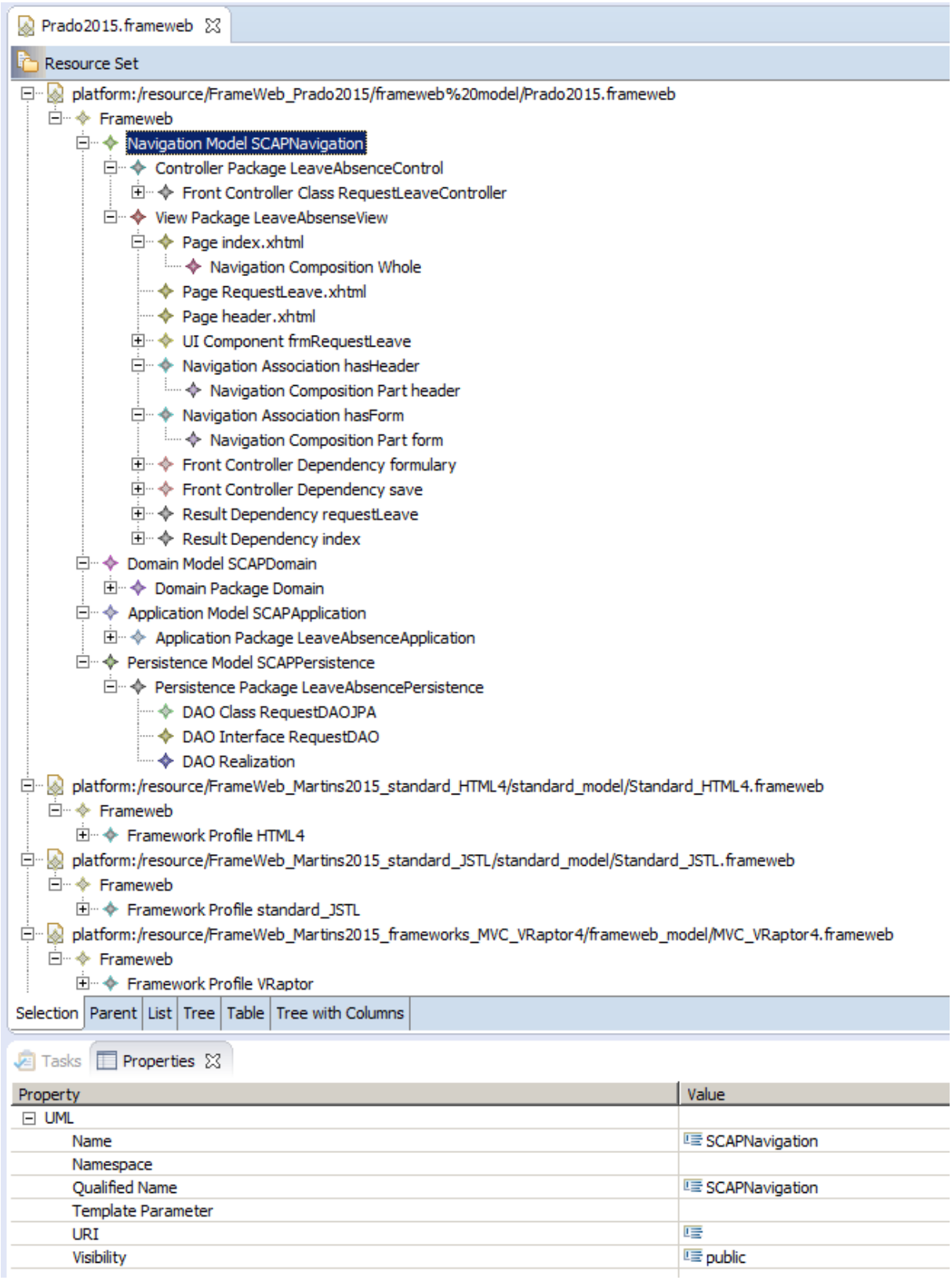
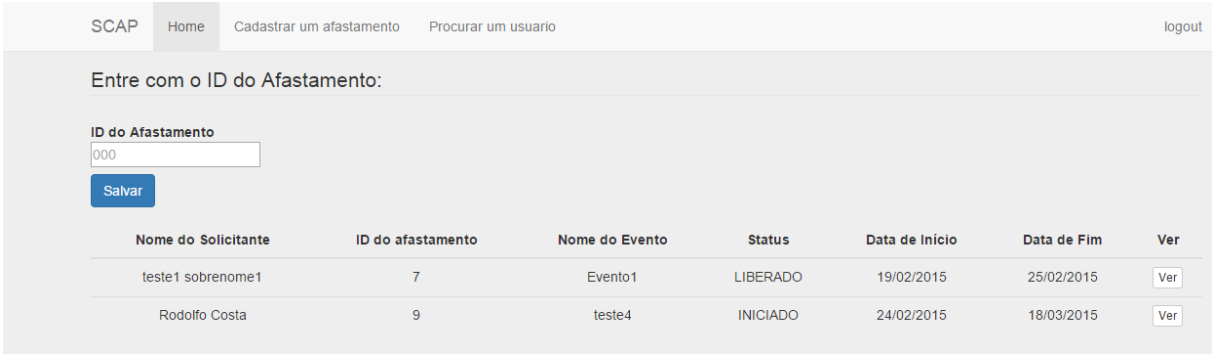
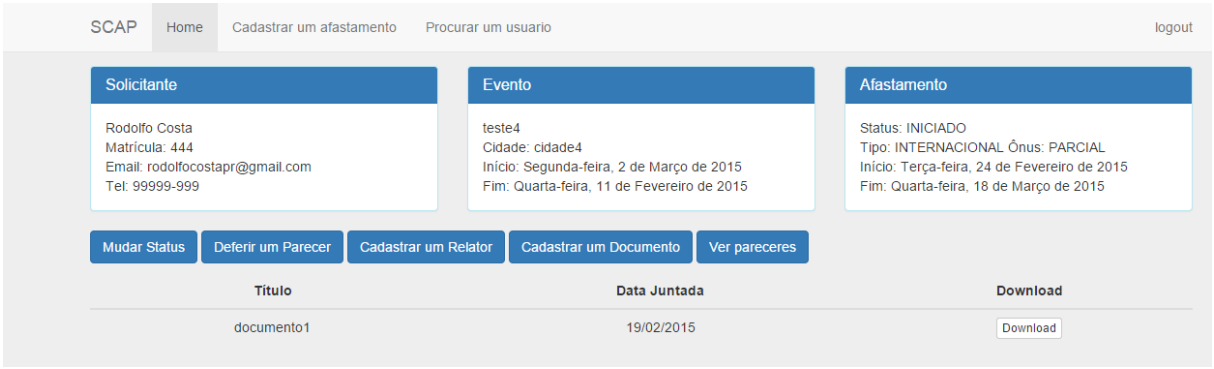


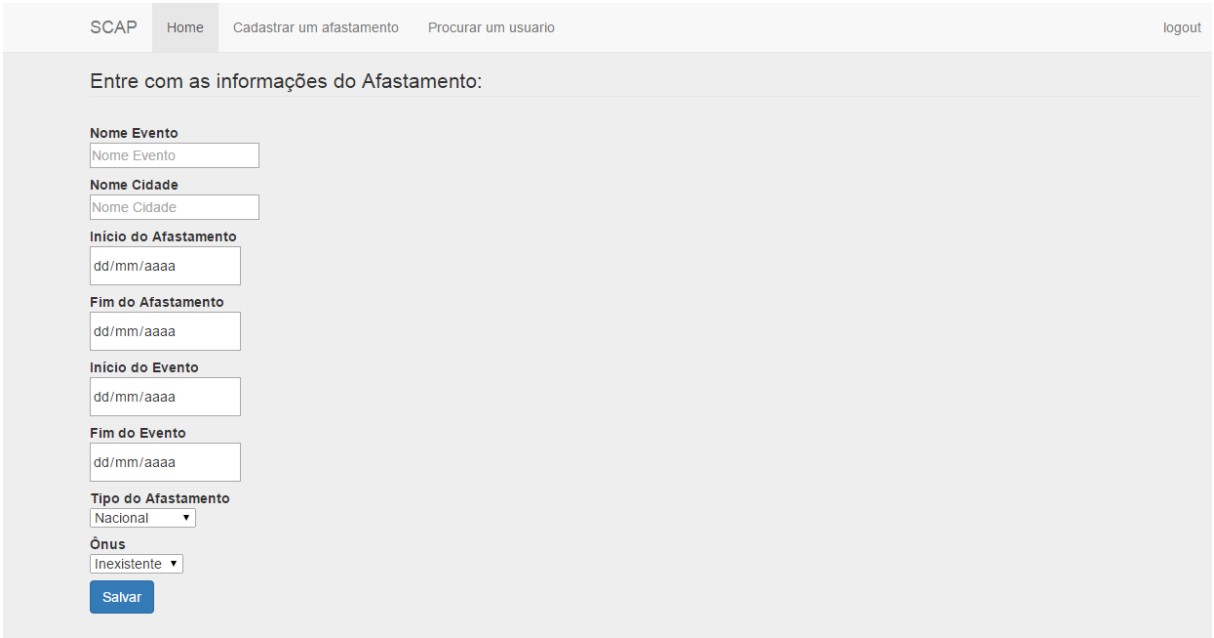
Figura 40 – Representação sob o metamodelo ***FW-15*** dos modelos ***FrameWeb*** para afastamentos do SCAP em projeto com o `VRaptor` e `JPA`.



(a) Lista de afastamentos



(b) Processo de afastamento



(c) Formulario de cadastro de afastamento

Figura 41 – Sistema de Informação Web SCAP - telas de afastamento (PRADO, 2015).

Como o `VRaptor` não possui componentes visuais específicos, o **Modelo de Entidades** do SCAP foi acrescido também das biblioteca `HTML` e `JSTL`, com as quais o modelador definiu os aspectos desejados para a interface do Sistemas de Informação Web.

O resultado obtido em termos de Sistema de Informação Web para os afastamentos do SCAP na visão de Prado (2015) está apresentado na Figura 41. A Figura 41a apresenta

a lista de afastamentos cadastrada para seleção. O processo de afastamento bem como sua lista de documentos aparecem na Figura 41b. Finalmente, o cadastro do afastamento está apresentado na Figura 41c.

A Seção 5.4 apresenta uma análise mais detalhada desses modelos comparativamente.

5.3 Aplicação em JSF e Hibernate

O `JSF` é um *framework* MVC baseado em componentes UI que fornece um conjunto pré-definido de *tags* JSP para representação e composição da interface com o usuário. Entretanto, o `JSF` permite que outros componentes UI sejam adicionados, normalmente por meio de *frameworks* UI (*UI suites*), como o `PrimeFaces` ou `RichFaces`. Também efetua a validação de entrada e conversão de dados bem como promove internacionalização e acessibilidade.

A Figura 42 apresenta graficamente o fragmento do **Modelo de Navegação *FrameWeb*** para afastamentos do SCAP em projeto com o `JSF` e `Hibernate`, apresentado em (DUARTE, 2014), por meio de uma tradicional ferramenta gráfica para UML. As figuras 43 e 44 apresentam, respectivamente, o **Modelo de Persistência** e o **Modelo de Entidades** para este mesmo projeto.

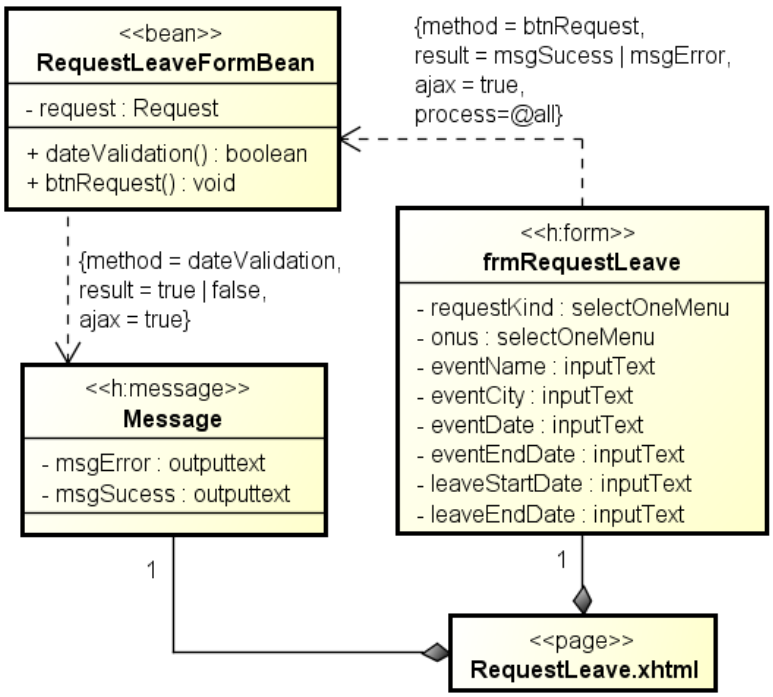


Figura 42 – Representação gráfica UML do **Modelo de Navegação** para afastamentos do SCAP em projeto com o `JSF` e `Hibernate` (DUARTE, 2014).

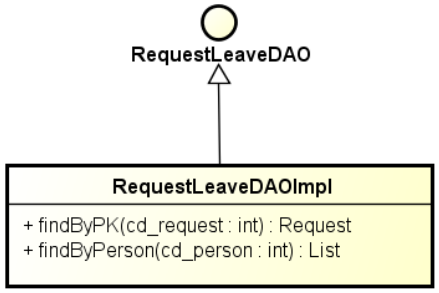
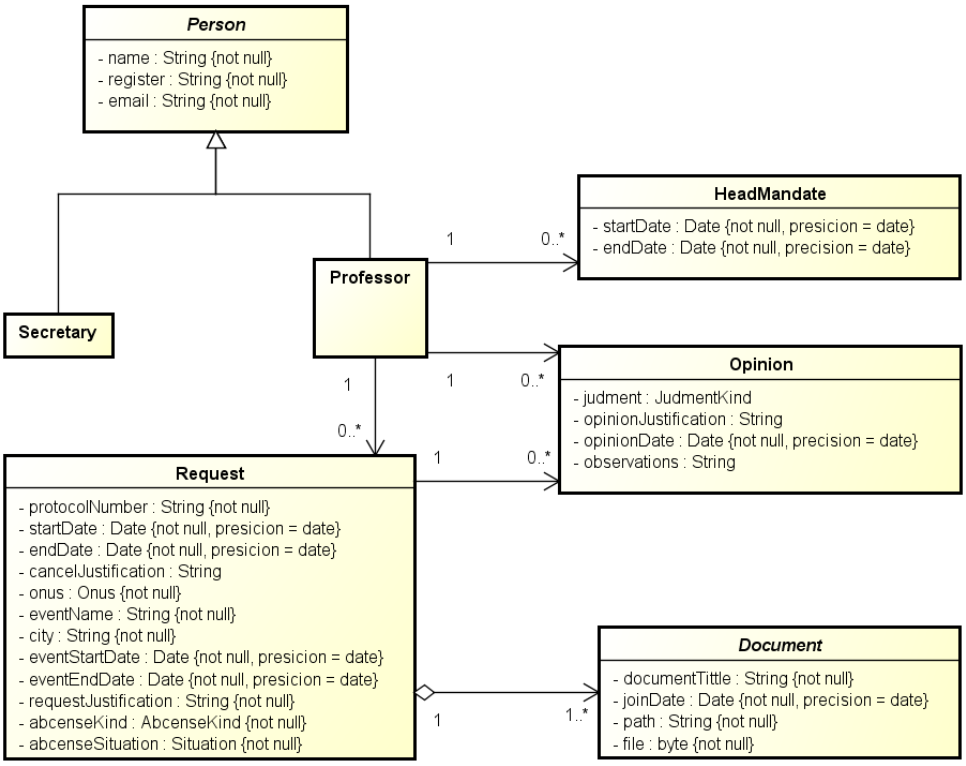
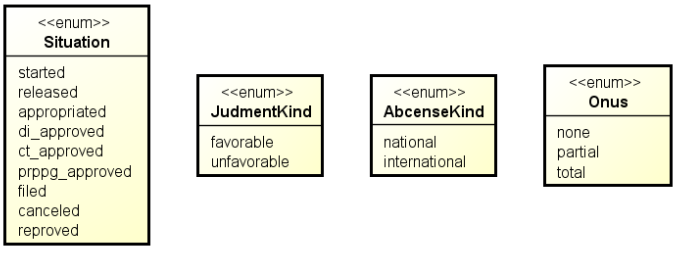


Figura 43 – Representação gráfica UML do **Modelo de Persistência** para afastamentos do SCAP em projeto com o `JSF` e `Hibernate` (DUARTE, 2014).



(a) **Modelo de Entidades** SCAP - afastamentos



(b) Enumerações

Figura 44 – Representação gráfica UML do **Modelo de Entidades** para afastamentos do SCAP em projeto com o `JSF` e `Hibernate` (DUARTE, 2014).

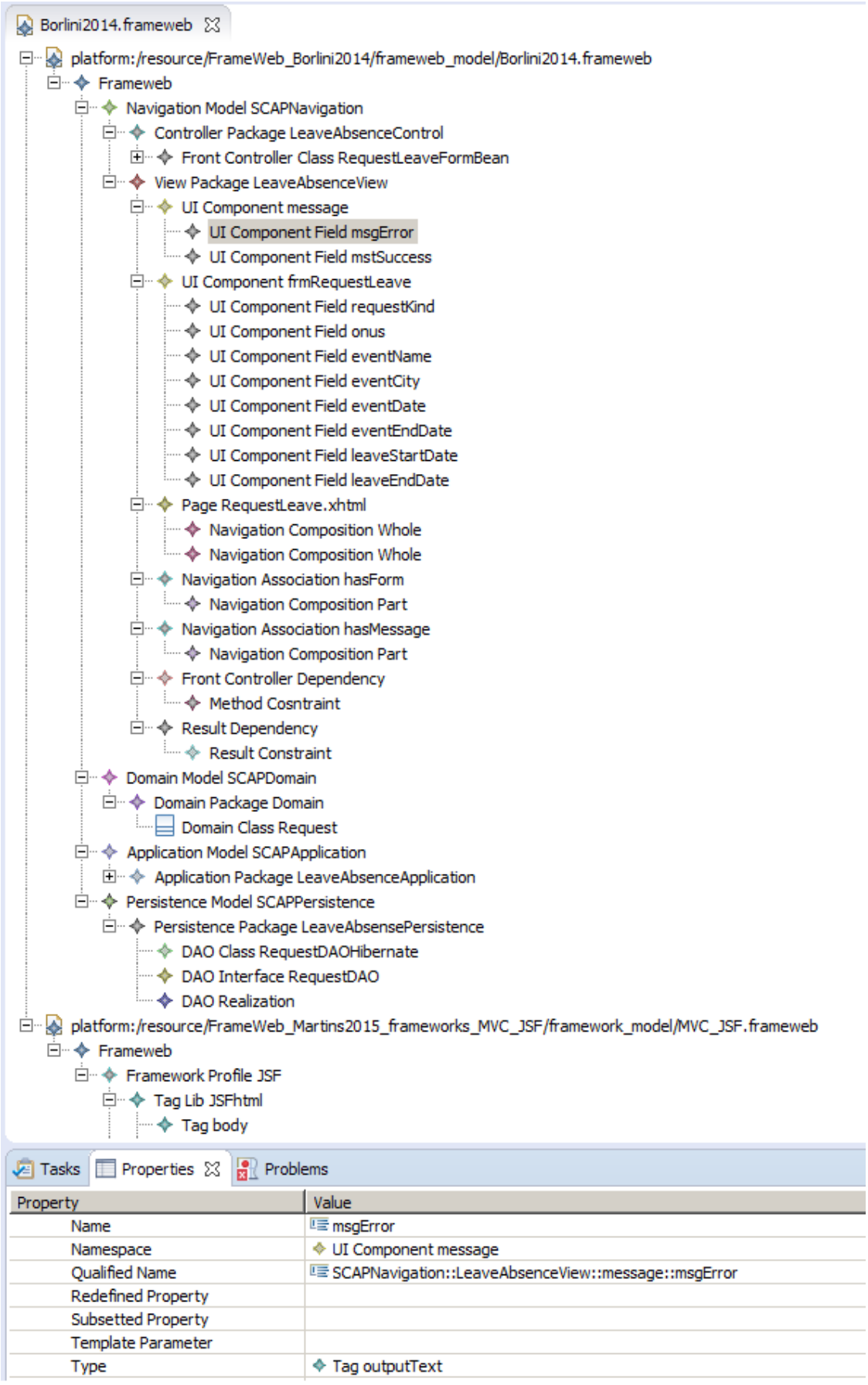


Figura 45 – Representação sob o metamodelo ***FW-15*** dos modelos ***FrameWeb*** para afastamentos do SCAP em projeto com o `JSF` e `Hibernate`.

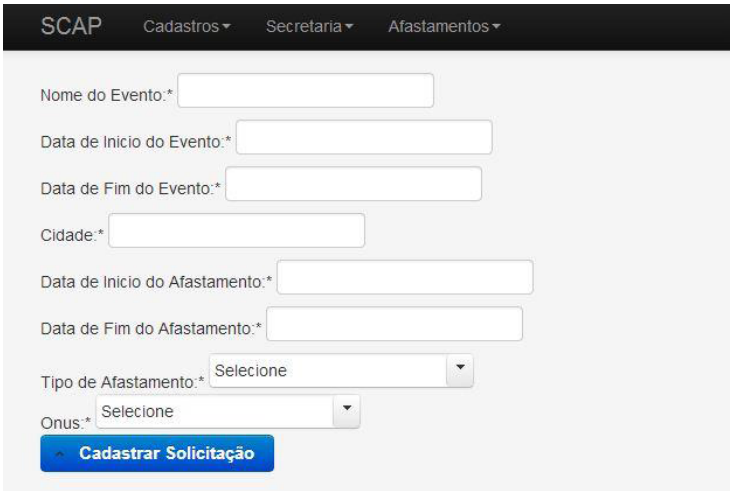
O mesmo **Modelo de Navegação *FrameWeb*** representado no protótipo do trabalho de Duarte (2014), com uma apresentação em árvore (*treeview*), está disponível no repositório deste projeto para consulta. Nele todas as estruturas relacionadas ao caso de uso estudado estão implementadas.

A Figura 45 apresenta um fragmento desse mesmo **Modelo de Navegação *FrameWeb*** agora representado no protótipo deste trabalho, com uma apresentação em árvore (*treeview*).

Além das estruturas próprias do modelo, é possível notar na Figura 45 a presença das bibliotecas que contem os perfis para o `JSF` e `Hibernate`. As primeira aplicada ao **Modelo de Navegação** e a última ao **Modelo de Entidades**. Cada **Definição de Framework** é responsável por definir os estereótipos aplicados aos modelos ***FrameWeb***, dando assim a representatividade desejada por parte do modelador.

O resultado obtido em termos de Sistema de Informação Web para os afastamentos do SCAP na visão de Duarte (2014) está apresentado na Figura 46. A Figura 46a apresenta a lista de afastamentos cadastrada para seleção e manifestação de parecer. O cadastro do afastamento está apresentado na Figura 41b.

(a) Lista de afastamentos



(b) Formulario de cadastro de afastamento

Figura 46 – Sistema de Informação Web SCAP - telas de afastamento (DUARTE, 2014).

A Seção 5.4 apresenta uma análise mais detalhada desses modelos comparativamente.

5.4 Avaliação comparativa

A partir dos das provas de conceito apresentadas nas seções 5.2 e 5.3, foi possível avaliar comparativamente os modelos ***FrameWeb***, quanto sua aplicação com *frameworks* distintos.

Ao observar o **Modelo de Navegação** do SCAP do trabalho de Prado (2015), é interessante notar a necessidade, por parte do modelador, da inclusão de bibliotecas cujo conteúdo é composto de elementos de interface, já que o `VRaptor` não dispõe destas estruturas dada sua característica baseada em ações. Por isso, o modelador fez uso das *tag libraries* `HTML` e `JSTL`, para a montagem dos recursos UI desejados.

Neste caso, há o formulário (instância de `UIComponent`) estereotipado por uma instância de `Tag` denominada `«form»` a partir da biblioteca `HTML` adicionada. O formulário por sua vez, contém como atributos (instâncias de `UIComponentField`) do tipo `UIComponent`, que quando instanciados podem ser tanto um componente definido no modelo como o próprio formulário, quanto um estereótipo.

As dependências apresentam o fluxo modelado refletindo as características do *framework* escolhido, isso fica evidente porque há um conjunto de páginas `XHTML` definidas pelo modelador para compor a estrutura desejada para a interface, por exemplo quando se faz necessário definir uma página com um certo tipo de cabeçalho de forma a ser usado como parte de outras páginas.

Já no **Modelo de Navegação** do trabalho de Duarte (2014), o modelador fez uso da *tag library* proprietária do `JSF`, não necessitando de outros recursos além dos componentes próprios do *framework*. Esses componentes aparecem como atributos do formulário `frmRequestLeave`. Nesta situação o modelador precisou apenas da definição da página e do formulário (instância de `UIComponent`), não havendo necessidade de outros arquivos `XHTML` adicionais para compor a interface.

A partir daí, o formulário é estereotipado por uma instância de `Tag` denominada `«h:form»`, que, por ser obtido da biblioteca `JSF`, apresenta o prefixo "`h`", diferentemente dos estereótipos definidos nas bibliotecas `HTML`. Este formulário também contém como atributos (instâncias de `UIComponentField`) do tipo `UIComponent`, que quando instanciados podem ser tanto um componente definido no modelo, como o próprio formulário, quanto um estereótipo.

É possível observar também o componente «h:message», que é uma outra instancia de UIComponent prefixada de forma semelhante, já que também faz parte da *tag library* JSFhtml da biblioteca JSF. Neste caso os atributos deste componente são *tags* desta mesma *tag library*.

Por outro lado, o modelador poderia adicionar livremente outras bibliotecas complementares (*UI Suites*) para adição de componentes UI, caso fosse necessário ao WIS, entretanto optou por não fazê-lo, porque julgou que o JSF já contia os recursos suficientes para o projeto em questão.

Esta diferenciação se deu exatamente pelo fato do *framework* VRaptor ser baseado em ações enquanto o JSF em componentes, conferindo então escolhas diferentes por parte dos modeladores, não apenas no que diz respeito ao *design* mas também em como utilizar os recursos disponíveis. O ***FW-15***, entretanto, se mostrou flexível ao atender as duas demandas, porque as especificidades de cada proposta não se encontram de forma rígida no metamodelo.

No caso do JSF os fluxos apresentam-se mais simples, porém contem mais detalhes quanto ao que se quer definir. Por exemplo, associação onde o formulário é dependente da classe do controlador frontal estão presentes como restrições: o método, o resultado deste método e as definições AJAX necessárias. Diferentemente do que se apresenta no caso do **Modelo de Navegação** de Prado (2015), onde aparece apenas o método já indicando o fluxo de dependência a ser executado.

No que diz respeito à classe do controlador frontal, o trabalho de Prado (2015) apresenta uma instância de FrontControllerClass estereotipada como «controller» para tratar as funcionalidades do formulário de afastamentos, já no trabalho de Duarte (2014) a instância de FrontControllerClass é estereotipada como «bean», para exercer função semelhante. Esta diferenciação ocorre não só para efeito sintático, mas estes estereótipos possuem semântica diferente, porque obedecem a regras OCL diferentes adicionadas por suas bibliotecas.

Avaliando os dois trabalhos verificou-se a importância da correta representação das definições do método ***FrameWeb***, já que o comportamento do WIS é diretamente dependente das características dos *frameworks* usados. Isso significa dizer que os modelos são reflexo das escolhas de projeto, sendo afetados tanto no que diz respeito aos *frameworks* selecionados mas também quanto ao estilo do modelador e dos requisitos de interface do projeto, estando todas estas coisas intimamente relacionadas.

No que diz respeito ao **Modelo de Persistência** do SCAP, observa-se que há uma diferença clara na nomenclatura das estruturas, as quais fazem referência ao *framework* adotado. Entretanto esta nomenclatura não foi seguida com precisão em (DUARTE, 2014), tendo sido corrigida quando da aplicação do modelo no editor ***FrameWeb***, porque este

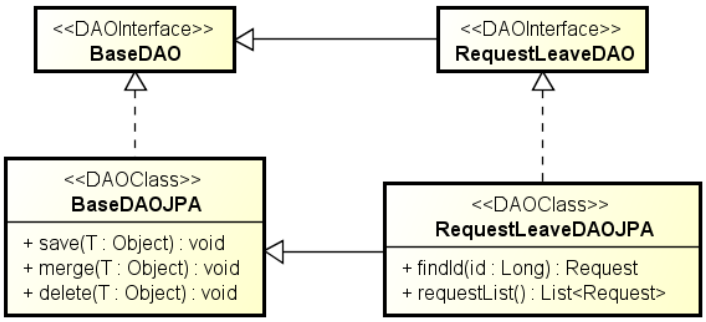


Figura 47 – Representação gráfica UML do **Modelo de Persistência** para afastamentos do SCAP em projeto com o `VRaptor` (PRADO, 2015) revisada.

faz a verificação do modelo. Este um dos requisitos importantes do metamodelo ***FW-15*** que foi devidamente atendido, mesmo sendo dependente da escolha de *framework* por parte do modelador.

Outro aspecto interessante que aparece no **Modelo de Persistência** de Prado (2015), mas não está visível no trabalho de Duarte (2014) apesar de presente enquanto funcionalidade, é o uso de um ancestral comum a todos as interfaces e implementações de persistência (`BaseDAO` e `BaseDAOJPA`). Esta diferença é apenas diagramática, já que o modelador optou por separar este elemento de base em outro diagrama, aqui não apresentado.

Entretanto, no trabalho de Prado (2015) é possível notar a ausência da relação de generalização entre os elementos de base (`BaseDAO` e `BaseDAOJPA`) e suas especializações (`RequestLeaveDAO` e `RequestLeaveDAOJPA`). Já em (DUARTE, 2014) as relações de generalização estão presentes porém não visíveis, que apesar de não ser uma falha, peca por não utilizar a representatividade da linguagem gráfica usada.

O **Modelo de Persistência** de Duarte (2014) por outro lado apresenta uma falha na relação entre `RequestLeaveDAO` e `RequestLeaveDAOImpl`, fazendo uso de uma relação de generalização ao invés de uma realização.

A ferramenta gráfica utilizada nos trabalhos citados peca também por representar as realizações (linha pontilhada com seta fechada) com o mesmo construto de associações comuns (linha contínua), como pode ser visto no modelo apresentado por Prado (2015), quando a interface está representada por um círculo.

Todas estas questões também relacionam-se à verificação dos modelos quando desenvolvidos em uma ferramenta UML de uso geral, tendo sido observadas posteriormente com o apoio do editor ***FW-15***, que apesar de sua limitação gráfica e da transferência dos modelos ter ocorrido de forma manual, foi capaz de detectar estes problemas. Os modelos corrigidos com base na verificação feita com o apoio do editor ***FW-15*** estão mostrados graficamente nas figuras 47 e 48.

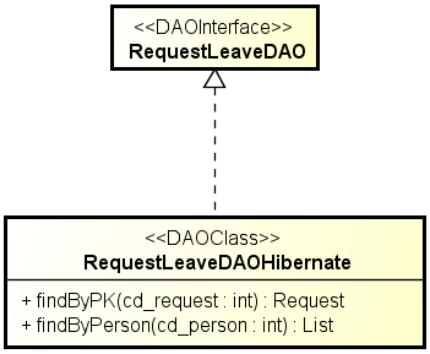


Figura 48 – Representação gráfica UML do **Modelo de Persistência** para afastamentos do SCAP em projeto com o `JSF` (DUARTE, 2014) revisada.

O **Modelo de Entidades** do SCAP apresentado em ambos os trabalhos não foi rigoroso quanto às restrições das associações entre classes de domínio, deixando de explicitar definições importantes para o desenvolvedor. Nos dois casos a falta de uma ferramenta específica capaz de solicitar as informações ausentes fica clara.

Por outro lado as diferenças entre os *frameworks* são pouco perceptíveis já que o `Hibernate` é uma implementação `JPA`. Este foi um dos motivos pelo qual demos mais ênfase neste trabalho aos aspectos navegacionais do método ***FrameWeb***, até porque mesmo outros *frameworks* (não `JPA`) também diferem pouco entre si.

Ambos os trabalhos fizeram uso de `CDI` para a injeção de dependência, portanto não há análises comparativas pertinentes a serem trabalhadas quanto a esta abordagem. Entretanto as partes relevantes associadas ao processo de afastamento do SCAP foram implementadas no editor deste trabalho, sem maiores dificuldades.

Vale salientar, que apesar de os modelos dos trabalhos de Duarte (2014) e Prado (2015) terem sido construídos com base em uma mesma análise, há diferenças entre eles que se devem devido as escolhas dos modeladores (alunos) e não propriamente relacionadas aos *frameworks* escolhidos. Esta variação é permitida e até bem vinda porque demonstra flexibilidade quando a visão de cada modelador no uso do método.

5.5 Verificação sintática de modelos FrameWeb

Como já mencionado na Seção 5.4, a verificação dos modelos é de suma importância para a qualidade da modelagem. Nesse aspecto o metamodelo ***FrameWeb*** é alicerce para permitir que a verificação possa ser executada de acordo com as necessidades impostas pelo método.

A verificação sintática dos modelos ocorre por meio da ferramenta ***FrameWeb***, gerada a partir do metamodelo ***FW-15*** em ECore, conforme explicado na Seção 4.4.3. O

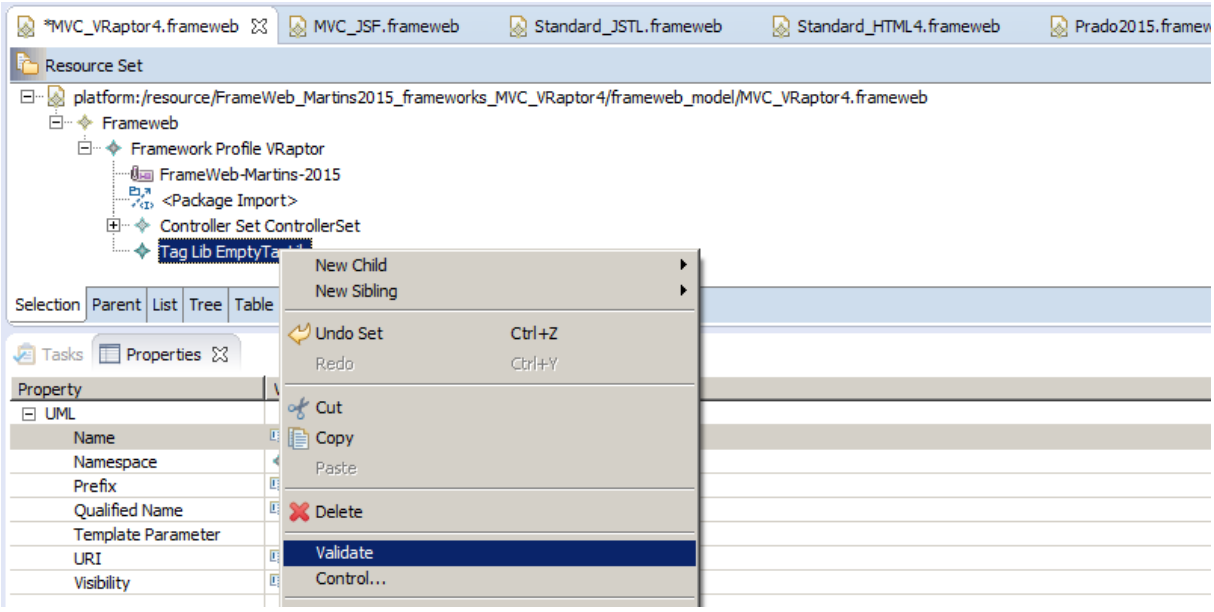


Figura 49 – Chamada para verificação de modelos no protótipo de editor baseado no metamodelo ***FW-15***.

EMF e seus "*plug-ins*" produzem as funções de validação necessárias ao metamodelo no contexto independente de *framework*. Já as verificações dependente de *framework* são feitas pelas regras adicionadas ao metamodelo por meio das bibliotecas, dependendo portanto de sua carga.

Nas duas situações, os modelos ***FrameWeb*** são testados de forma equivalente e uma mensagem padrão é apresentada na tela quando o modelador solicita a verificação. A Figura 49 apresenta a chamada deste recurso no protótipo do editor.

Na Figura 49 a chamada da função de verificação é feita através do menu "*validate*" (validação), porque refere-se à avaliação do modelo quanto os requisitos impostos pelo metamodelo. Todavia a WebE assim como a Engenharia de Software, usa o termo *validação* para identificar se um sistema atende aos seus requisitos originais impostos por parte do cliente e *verificação* no que diz respeito à qualidade e correção do modelo desenvolvido.

Porque o foco deste trabalho está inserido na visão do modelador enquanto projetista de um WIS e não do modelador enquanto desenvolvedor de uma DSL, bem como porque a função executada pelo menu "*validate*" é a de avaliar se o modelo atende à gramática da linguagem, o termo adotado neste trabalho para esta função é *verificação*. Assim a ação executada na chamada do menu "*validate*" é a verificação de modelos.

É interessante notar que as regras impostas ao metamodelo também são executadas de forma semelhante quando a verificação ocorre sob uma **Definição de Framework**. Por exemplo, a Figura 50 apresenta uma mensagem de erro associada a uma instância de `TagLib` denominada `EmptyTagLip` e adicionada especificamente para produzir erro, já que a propriedade `prefix` é de preenchimento obrigatório e está vazia.

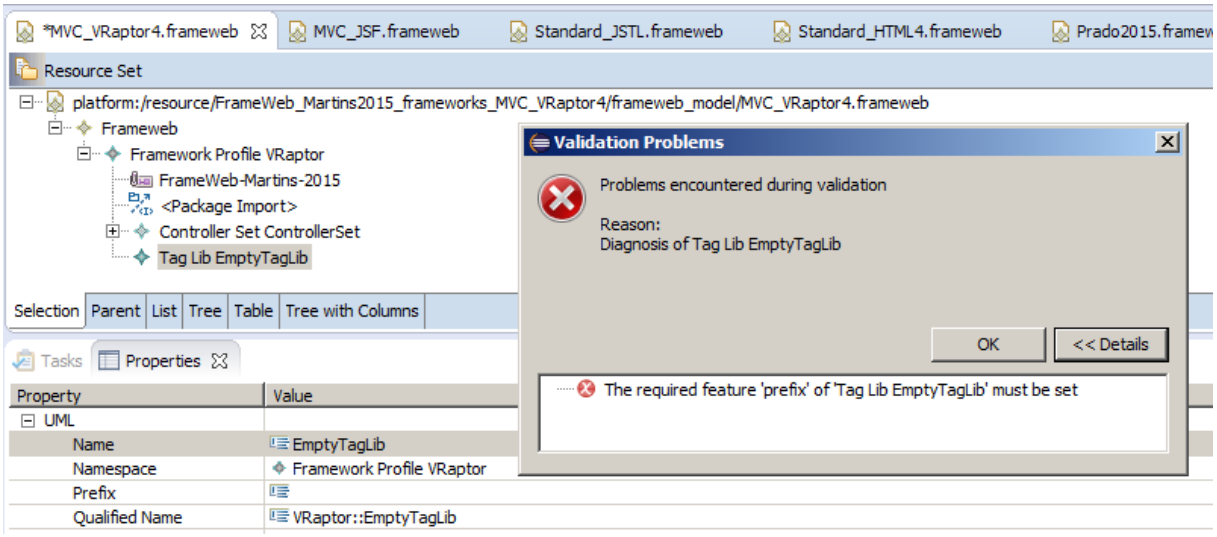


Figura 50 – Exemplo de verificação executada pelo protótipo de editor baseado no metamodelo ***FW-15***.

As verificações podem ser feitas tanto para um único construto do modelo quanto para diversos construtos e até mesmo para todo o modelo. Entretanto, as verificações que abrangem um número muito expressivo de construtos podem trazer uma lista muito grande de falhas, ficando difícil localizar cada problema encontrado. Por isso sugere-se que verificações completas dos modelos devem ser feitas posteriormente às verificações específicas, de forma a garantir uma visão geral sem dificultar a localização de eventuais problemas.

Um grande número de regras estão impostas aos modelos ***FrameWeb*** para facilitar o trabalho do modelador quanto ao uso do método. Todas as verificações que são parte do metamodelo ***FW-15*** podem ser utilizadas em outras ferramentas produzidas com o apoio deste metamodelo.

Não cabe, entretanto, mostrar um grande número testes de verificação no texto deste trabalho para não torná-lo muito extenso, além disso o metamodelo e o protótipo do editor estão disponíveis para execução no repositório do projeto.

Apesar da simplicidade do protótipo de editor em seus aspectos gráficos, sua capacidade de verificação dos modelos mostrou sua importância na aplicação do método ***FrameWeb*** no projeto de WIS, auxiliando muito o modelador e garantindo a qualidade dos modelos por ele produzidos.

6 Considerações Finais

A *web* não está mais restrita aos computadores, estando presente em quase tudo à nossa volta. *Smartphones*, veículos inteligentes, dispositivos vestíveis (*wearables*) estão presentes no dia-a-dia das pessoas definindo o que chamamos de Internet das coisas. Por isso os Sistemas de Informação Web devem ser cada vez mais interoperáveis, ou seja, devem adaptar-se a diversas plataformas e serem passíveis de integração entre si.

A Engenharia Web deve ser aderente a estas tecnologias no sentido de produzir as melhores práticas no desenvolvimento de Sistemas de Informação Web. Normalmente junto a uma nova plataforma são criados diversos recursos para desenvolvimento de Sistemas de Informação Web, incluindo *frameworks*, incrementando em volume e complexidade os diversos recursos disponíveis para o projeto de Sistemas de Informação Web. É importante então, que os métodos para projeto de Sistemas de Informação Web sejam adaptáveis e flexíveis a fim de lidar com esta diversidade.

Em concordância a esta visão, o método ***FrameWeb*** aproxima o projetista do desenvolvedor aproveitando aspectos consagrados pelo estado-da-prática, por isso surgiu a proposta de formalização deste método, não só focando a definição da linguagem e na criação de ferramentas, mas principalmente criando um método para a extensibilidade do método.

Como objetivo principal neste trabalho foi proposto que: (i) a linguagem do método ***FrameWeb*** passasse por um processo de formalização para a partir do qual o metamodelo ***FW-15*** foi definido; (ii) juntamente com este processo de formalização fossem feitas evoluções para permitir o uso de diversos *frameworks*; (iii) uma metodologia para permitir a extensibilidade do método quanto a futuros novos *frameworks* fosse criada; e (iv) como resultado fosse desenvolvido um protótipo de ferramenta simplificado capaz de efetuar a verificação de modelos ***FrameWeb***, permitindo seu futuro uso em outras propostas de trabalho.

Ao longo desta dissertação, uma proposta de atualização do método ***FrameWeb*** e formalização de sua linguagem foi apresentada em conformidade com os requisitos apresentados na Seção 3.3. A linguagem foi definida no contexto do das melhores técnicas Desenvolvimento Dirigido a Modelos e sua formalização resultou no metamodelo ***FW-15***.

A partir da implementação do metamodelo ***FW-15***, foi produzido o protótipo de uma ferramenta simplificada para verificação de modelos ***FrameWeb***, sob o qual alguns experimentos foram conduzidos com vistas ao uso de vários *frameworks* em projetos ***FrameWeb*** executados a partir de uma mesma fase de análise, e posteriormente apresentados como prova de conceito deste trabalho. Como resultado adicional uma metodologia para permitir a extensibilidade do método foi definida e apresentada.

Neste capítulo apresenta-se uma avaliação geral do trabalho (Seção 6.1) e as suas possíveis evoluções (Seção 6.2).

6.1 Avaliação do trabalho

Na avaliação de qualquer trabalho executado encontram-se aspectos fortes e fracos, sendo necessário, portanto, que uma avaliação seja feita para permitir a evolução do conhecimento e dele obter o maior benefício. Revisitando estas questões podemos citar como contribuições deste trabalho:

- A definição formal da linguagem ***FW-15*** por meio da implementação do metamodelo ***FW-15***, mantendo-se a compatibilidade com a versão ***FW-07*** (SOUZA, 2007; SOUZA; FALBO, 2007; SOUZA; FALBO; GUIZZARDI, 2009);
- A linguagem ***FW-15*** agrega características específicas do desenvolvimento *web* à UML, que é uma linguagem padronizada e amplamente conhecida pelos modeladores e desenvolvedores de Sistemas de Informação Web;
- A criação do protótipo de um editor capaz de verificar os modelos ***FrameWeb***, além de servir para permitir que sejam criadas e testadas as diferentes **Definições de Framework** aplicáveis ao método;
- A adição de novas tecnologias ao método (e.g., `AJAX`) e a possibilidade do uso de diferentes padrões como `HTML`, `JSTL` e outros, além de abrir a possibilidade para uso de componentes UI customizáveis.

Dentre estas contribuições, os pontos fortes do trabalho são:

- A definição de uma metodologia para a extensibilidade do método, por meio das **Definições de Framework**, fortalecendo a característica de proximidade das fases de projeto e desenvolvimento dos processos de WebE;
- A capacidade evolutiva do metamodelo ***FW-15***, que por suas características intrínsecas em termos do uso das melhores práticas Desenvolvimento Dirigido a Modelos, permite a adição de futuras categorias de *frameworks*;

- O metamodelo ***FW-15*** permite que *frameworks* MVC baseados em diferentes abordagens (em ações ou componentes UI), sejam modelados em um mesmo ambiente, sem no entanto forçar o modelador a assumir este ou aquele comportamento quando opta por uma ou outra abordagem;

- Partindo de um metamodelo implementado, diversas outras funcionalidades podem ser acrescidas ao método, desde a criação de novas ferramentas até mesmo a geração automática de código.

Algumas reflexões devem ser feitas no que diz respeito ao trabalho executado para avaliar o alcance dos resultados perante os objetivos propostos. Neste contexto e considerando os requisitos levantados e decisões de projeto tomadas algumas limitações devem ser mencionadas tanto para melhor entendimento do trabalho, quanto para promover futuras evoluções e pesquisas. As questões que não foram aprofundadas neste trabalho, constituindo assim limitações da proposta são:

- Por decurso de tempo, não foi possível incluir no escopo deste trabalho a implementação de uma ferramenta gráfica mais apropriada ao método ***FW-15***, entretanto um projeto mais amplo está em desenvolvimento a fim de que alunos de graduação possam colaborar no desenvolvimento desta ferramenta;

- Apesar do trabalho abranger todas as categorias de *frameworks* relacionadas ao método ***FrameWeb***, a ênfase maior foi dada aos *frameworks* MVC, assim não foram implementadas bibliotecas para uma variedade maior de *frameworks* objeto/relacionais e de injeção de dependência;

- A opção de definir o metamodelo a partir do metamodelo UML foi importante para o projeto e trouxe diversos benefícios, contudo esta escolha torna o metamodelo ***FW-15*** dependente da atual versão UML, além disso uma outra consequência desta abordagem foi o grande número de regras OCL necessárias para definir o comportamento desejado para o metamodelo ***FW-15***;

- Apesar do método ter sido avaliado por alunos de graduação durante as aulas de desenvolvimento *web* e *web* semântica, poucos experimentos foram conduzidos no uso do metamodelo e ainda sem o rigor necessário para que conclusões mais bem fundamentadas fossem feitas;

- Ainda é necessário executar mais testes com diferentes *frameworks* para avaliar melhor a extensibilidade e flexibilidade do metamodelo. Os principais testes foram feitos com trabalhos de conclusão de curso (DUARTE, 2014; PRADO, 2015).

6.2 Perspectivas Futuras

Esta dissertação abre as portas para que o método ***FrameWeb*** possa ser usado formalmente em projetos de Sistemas de Informação Web baseados em *frameworks*. Observando os pontos positivos e negativos deste trabalho percebe-se que ainda há bastante a ser feito, tanto nos conceitos metodológicos quanto no desenvolvimento de ferramentas. Sugere-se então como complementação a este trabalho:

- O desenvolvimento de uma ferramenta CASE gráfica para o método ***FrameWeb***, que seja capaz de aproveitar os conceitos formalizados pelo metamodelo ***FW-15***, além de conter outras funcionalidades para integração, verificação e validação de modelos;
- Conduzir experimentos formais com o apoio do editor simplificado já proposto ou a partir de novas ferramentas baseadas no metamodelo ***FW-15***;
- Propor a adição de novas categorias de *frameworks* ao metamodelo ***FW-15***, incluindo autenticação e autorização, geradores de relatórios, gerenciadores de e-mail, integração com recursos bancários e categorias relacionadas a **Web Semântica** que trabalham com *frameworks* como o `Jena`[1] além de *frameworks* NoSQL;

A formalização da linguagem ***FW-15*** é mais um passo para o crescimento e evolução do método ***FrameWeb***, dando continuidade ao trabalho de Souza (2007). Na sequência é possível evoluir o protótipo desenvolvido para uma ferramenta ***FrameWeb*** capaz de executar a geração automática de código a partir de modelos que estejam de acordo com as opções feitas pelo modelador independentemente das tecnologias e *frameworks* a serem empregados, executar a importação de outros modelos em formato UML obtidos de ferramentas CASE de uso geral, dentre outras atividades pertinentes ao método.

[1] <https://jena.apache.org/index.html>

Referências

ALMEIDA, J. P. et al. Platform-independent modelling in MDA: supporting abstract platforms. In: *Model Driven Architecture: European MDA Workshops: Foundations and Applications 2003 and 2004*. The Netherlands and Sweden: Springer, 2005. p. 174–188. Citado na página 41.

ALUR, D.; MALKS, D.; CRUPI, J. *Core J2EE Patterns (paperback): Best Practices and Design Strategies*. Upper Saddle River, USA: Pearson Education, 2013. 462 p. (Prentice Hall Core). ISBN 9780133807462. Citado 3 vezes nas páginas 32, 79 e 88.

AMBLER, S. W. Agile model driven development is good enough. *IEEE Software*, IEEE, v. 20, n. 5, p. 71–73, 2003. Citado na página 35.

AMELLER, D. *Considering Non-Functional Requeriments in Model-Driven Engineering*. 13 p. Tesi de Màster — Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 2009. Citado 2 vezes nas páginas 13 e 36.

BARESI, L.; GARZOTTO, F.; PAOLO, P. Extending UML for Modeling Web Applications. In: *Proceedings of the 34th Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Comput. Soc. Press, 2001. p. 1285–1294. Citado na página 44.

CALVARY, G. et al. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers*, v. 15, n. 3, p. 289 – 308, 2003. ISSN 0953-5438. Citado na página 44.

CÁCERES, P. et al. A MDA-Based Approach for Web Information System. In: CITESEER. *Development, Proceedings of Workshop in Software Model Engineering (WiSME) in UML'2003*. [S.l.], 2004. Citado na página 45.

DUARTE, B. B. *Aplicação do Método FrameWeb no Desenvolvimento de um Sistema de Informação na Plataforma Java EE 7*. Monografia (Projeto de Graduação) — Departamento de Informática, Universidade Federal do Espírito Santo, Vitória (ES), Brasil, 2014. Citado 10 vezes nas páginas 14, 117, 123, 124, 126, 127, 128, 129, 130 e 135.

EMBLEY, D. W.; LIDDLE, S. W.; PASTOR, O. Conceptual-Model Programming: A Manifesto. In: EMBLEY, D. W.; THALHEIM, B. (Ed.). *Handbook of Conceptual Modeling*. Provo, Utah, USA: Springer Berlin Heidelberg, 2011. p. 3–16. ISBN 978-3-642-15864-3. Citado 3 vezes nas páginas 25, 27 e 33.

FONS, J. et al. Applying the OOWS model-driven approach for developing web applications. The Internet Movie Database case study. In: *Web Engineering: Modelling and Implementing Web Applications*. London, UK: Springer Science & Business Media, 2008. p. 65–108. Citado na página 45.

FOWLER, M. *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002. Citado na página 31.

FREEMAN, R. E. *Strategic management: A stakeholder approach*. Cambridge, UK:

Cambridge University Press, 2010. Citado na página 49.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Pearson Education, 1994. ISBN 0201633612. Citado 2 vezes nas páginas 32 e 88.

GARRETT, J. J. et al. *AJAX: A new approach to web applications*. 2005. Disponível em: <www.adaptivepath.com/publications/essays/archives/000385.php>. Acesso em: 12 dec. 2015. Citado na página 57.

GÓMEZ, J.; CACHERO, C.; PASTOR, O. Conceptual modeling of device-independent web applications. *Ieee multimedia*, IEEE, n. 2, p. 26–39, 2001. Citado na página 45.

GRONBACK, R. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Boston, MA, USA: Pearson Education, 2009. 29-54 p. (Eclipse Series). ISBN 9780321635198. Citado na página 42.

GUIZZARDI, G. *Ontological foundations for structural conceptual models*. 17-48 p. CTIT PhD.-thesis — University of Twente, Enschede, The Netherlands, 2005. Citado na página 81.

ISAKOWITZ, T.; KAMIS, A.; KOUFARIS, M. The extended RMM methodology for web publishing. In: CITESEER. NY, USA: Stern School of Business, New York University, 1998. Citado na página 45.

KLEPPE, A.; WARMER, J.; BAST, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston, MA, USA: Addison-Wesley, 2003. (The Addison-Wesley object technology series). ISBN 9780321194428. Citado 2 vezes nas páginas 34 e 37.

KLUGE, J.; KARGL, F.; WEBER, M. The Effects of the AJAX Technology on Web Application Usability. In: *Proceedings of the Third International Conference on Web Information Systems and Technologies - WEBIST*. Barcelona, Spain: SCITEPress, 2007. v. 2, p. 289–294. Citado na página 57.

KOCH, N. et al. Extending UML to model navigation and presentation in web applications. In: YORK, ENGLAND. *Proceedings of Modelling Web Applications in the UML Workshop*. York, UK: Ed. Geri Winters and Jason Winters, 2000. Citado na página 45.

KOCH, N. et al. Web Engineering: Modelling and Implementing Web Applications, chap. UML-Based Web Engineering. In: *UML-Based Web Engineering*. London, UK: Springer London, 2008b. Citado na página 44.

KOCH, N.; KRAUS, A.; HENNICKER, R. The authoring process of the uml-based web engineering approach. In: *First International Workshop on Web-Oriented Software Technology*. Valencia, Spain: [s.n.], 2001. p. 1–29. Citado na página 25.

KOCH, N. et al. Model-driven web engineering. *Upgrade-Novática Journal (English and Spanish), Council of European Professional Informatics Societies (CEPIS) IX*, v. 2, p. 40–45, 2008a. Citado na página 44.

LINAJE, M. et al. Automatic Generation of RIAs Using RUX-Tool and Webratio. In: *Web Engineering: 9^{th} International Conference, ICWE 2009, Proceedings*. San Sebastián, Spain: Springer Berlin Heidelberg, 2009. p. 501–504. ISBN 978-3-642-02818-2. Citado na página 44.

LISKOV, B. Keynote address-data abstraction and hierarchy. *ACM Sigplan Notices*, ACM, v. 23, n. 5, p. 17–34, 1988. Citado na página 111.

MELIÁ, S. et al. A model-driven development for GWT-based rich internet applications with OOH4RIA. In: *Proceedings - 8th International Conference on Web Engineering, ICWE 2008*. Yorktown Heights, NJ, USA: IEEE, 2008. p. 13–23. ISBN 9780769532615. Citado na página 45.

MELLOR, S. J. *MDA distilled: principles of model-driven architecture*. Boston, MA, USA: Addison-Wesley Professional, 2004. Citado na página 36.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and How to Develop Domain-specific Languages. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 37, n. 4, p. 316–344, December 2005. Citado na página 34.

PANACH, J. I. et al. A framework to identify primitives that represent usability within Model-Driven Development methods. *Information and Software Technology*, Elsevier B.V., v. 58, p. 338–354, 2014. Citado na página 45.

PASTOR, O. et al. Model-driven development. *Informatik-Spektrum*, v. 31, p. 394–407, 2008. Citado na página 65.

PASTOR, O. et al. The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. *Information Systems*, Elsevier, v. 26, n. 7, p. 507–534, 2001. Citado na página 45.

PRADO, R. C. d. *Aplicação do método FrameWeb no desenvolvimento de um sistema de informação utilizando o framework VRaptor 4*. Monografia (Projeto de Graduação) — Departamento de Informática, Universidade Federal do Espírito Santo, Vitória (ES), Brasil, 2015. Citado 10 vezes nas páginas 14, 118, 119, 120, 122, 127, 128, 129, 130 e 135.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 8th. ed. USA: McGraw-Hill Higher Education, 2005. (McGraw-Hill higher education). ISBN 9780073019338. Citado na página 27.

PURIFICAÇÃO, C. E. P. d.; SILVA, P. C. d. EngenDSL – a Domain Specific Language for Web Applications. *Proceedings of 10th CONTECSI International Conference on Information Systems and Technology Management*, Tecsi, Sao Paulo, p. 879–899, 2013. Citado na página 45.

SCHMIDT, D. C. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 39, n. 2, p. 25–31, February 2006. Citado na página 36.

SCHWABE, D.; ROSSI, G. An object oriented approach to Web-based applications design. *TAPOS*, v. 4, n. 4, p. 207–225, 1998. Citado na página 25.

SELIC, B. The pragmatics of model-driven development. *IEEE software*, IEEE Computer Society, v. 20, n. 5, p. 19, 2003. Citado na página 34.

SMULLEN III, C. W.; SMULLEN, S. A. An experimental study of AJAX application performance. *Journal of Software*, v. 3, n. 3, p. 30–37, 2008. Citado na página 57.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. Dissertação (Mestrado em Informática) — Universidade Federal do Espírito Santo, Vitória, ES, Brasil, 2007. Citado 6 vezes nas páginas 13, 25, 31, 54, 134 e 136.

SOUZA, V. E. S.; FALBO, R. A. FrameWeb - A Framework-based Design Method for Web Engineering. In: *Proc. of the 2007 Euro American Conference on Telematics and Information Systems*. Portugal: ACM, 2007. Citado 3 vezes nas páginas 31, 54 e 134.

SOUZA, V. E. S.; FALBO, R. A.; GUIZZARDI, G. Designing Web Information Systems for a Framework-based Construction. In: HALPIN, T.; PROPER, E.; KROGSTIE, J. (Ed.). *Innovations in Information Systems Modeling: Methods and Best Practices*. 1. ed. Hershey, PA, USA: IGI Global, 2009. cap. 11, p. 203–237. Citado 4 vezes nas páginas 13, 25, 31 e 134.

THOMAS, D. MDA: Revenge of the Modelers or UML Utopia? *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 21, n. 3, p. 15–17, May 2004. Citado na página 35.

TRASK, B.; ROMAN, A. Using domain specific modeling in developing software defined radio components and applications. Citeseer, 2006. Citado na página 34.

VAN DEURSEN, A.; KLINT, P. Little Languages: Little Maintenance. *Journal of Software Maintenance*, John Wiley & Sons, Inc., New York, NY, USA, v. 10, n. 2, p. 75–92, March 1998. Citado na página 34.

ZHANG, J.; CHUNG, J.-Y. Mockup-driven fast-prototyping methodology for web application development. *Software Practice & Experience Journal, 33 (13), 2003, pp.*, p. 1251, 2003. Citado na página 25.

ZHANG, J.; CHUNG, J.-Y. Mockup-driven fast-prototyping methodology for Web Applications. In: *Proceedings 2003 Symposium on Applications and the Internet*. Orlando, FL, USA: IEEE, 2003. p. 410–413. Citado na página 25.

Apêndices

APÊNDICE A – Regras OCL

Neste apêndice apresenta-se a descrição das principais regras OCL necessárias ao metamodelo ***FW-15*** para consulta. As regras em si podem ser consultadas no repositório do projeto.

Tabela 4 – Descrição das principais regras OCL gerais.

Nome	Descrição
FramewebProjectMinimum	Um FramewebProject necessita obrigatoriamente ter um FrameworkProfile ou FramewebModel, não podendo apresentar ambos simultaneamente
FramewebModelContent	Um FramewebModel só permite pacotes do tipo DomainModel ou NavigationModel ou PersistenceModel ou ApplicationModel ou FrameworkApplication
FrameworkProfileMVCContent	Um FrameworkProfile cuja categoria é FrontController deve conter obrigatoriamente apenas TagLib ou ControllSet ou ResultSet como pacotes
FrameworkProfileORMContent	Um FrameworkProfile cuja categoria é ObjetoRelacional deve conter obrigatóriamente apenas MappingLib como pacotes
FrameworkProfileMVCMinimum	Um FrameworkProfile cuja categoria é FrontController deve conter no mínimo um elemento do tipo TagLib ou ControllSet ou ResultSet
FrameworkProfileORMMinimum	Um FrameworkProfile cuja categoria é ObjetoRelacional deve conter no mínimo um objeto MappingLib

Tabela 5 – Descrição das principais regras OCL do independente de *framework* para **Definição de Framework**

Nome	Descrição
TagLibContent	Uma TagLib deve conter apenas objetos do tipo Tag ou TagExtension
TagLibMinimum	Uma TagLib deve conter no mínimo um elemento do tipo Tag
ResultSetContent	Uma ResultSet deve conter apenas objetos do tipo ResultType ou ResultExtension
ResultSetMinimum	Uma ResultSet deve conter no mínimo um elemento do tipo ResultType
ControllerSetContent	Uma ControllerSet deve conter apenas objetos do tipo Controller
ControllerSetMinimum	Uma ControllerSet deve conter no mínimo um elemento do tipo Controller ou ControllerExtension
MappingLibContent	Uma MappingLib deve conter apenas objetos do tipo ClassMapping ou AttributeMapping
MappingLibMinimum	Uma MappingLib deve conter no mínimo um elemento do tipo ClassMapping ou AttributeMapping
FrameworkApplicationStrict	Um FrameworkApplication deve ter seu atributo FrameworkApplication.isStrict sempre igual a "*verdadeiro*" ("*true*")
FrameworkApplicationProfile	Um FrameworkApplication precisa ser sempre uma aplicação de um FramewebProfile
TagExtensionConstraint	Uma TagExtension só pode ocorrer entre objetos do tipo Tag e UIComponent
ResultExtensionConstraint	Uma ResultExtension só pode ocorrer entre objetos do tipo ResultType e Result
ControllerExtensionExtensionConstraint	Uma ControllerExtension só pode ocorrer entre objetos do tipo Controller e FrontControllerClass
AttributeMappingExtensionConstraint	Uma AttributeMappingExtension só pode ocorrer entre objetos do tipo AttributeMapping e DomainAttribute

ClassMappingExtensionConstraint	Uma ClassMappingExtension só pode ocorrer entre objetos do tipo ClassMapping e DomainClass

Tabela 6 – Descrição das principais regras OCL do independente de *framework* para **Metamodelo de Navegação**

Nome	Descrição
NavigationModelContent	Um NavigationModel pode conter apenas objetos do tipo NavigationPackage
NavigationModelMinimum	Um NavigationModel precisa conter pelo menos um objteto ControllerPackage and um ViewPackage
ControllerPackageContent	Um ControllerPackage pode conter apenas objetos do tipo FrontControllerClass ou NavigationDependency ou NavigationAssociation ou FrameworkApplication quando estes forem aplicações da categoria FrontController
ControllerPackageMinimum	Um ControllerPackage precisa conter pelo menos um objeto
ViewPackageContent	Um ViewPackage pode conter apenas objetos do tipo NavigationClass ou NavigationGeneralization ou NavigationDependency ou NavigationAssociation ou FrameworkApplication quando estes forem aplicações da categoria FrontController
ViewPackageMinimum	Um ViewPackage precisa conter pelo menos um objeto
NavigationCompositionRelation	Uma NavigationAssociation precisa obrigatoriamente conter um objeto do tipo UIComponent como "*parte*" e um do tipo NavigationClass como "*todo*" em sua relação de composição
NavigationCompositionWholeRelation	Uma NavigationAssociation precisa obrigatoriamente ser uma CompositionAssociation e apenas NavigationClass são permidas como "*todo*" na relação
NavigationCompositionPartRelation	Uma NavigationAssociation precisa conter obrigatoriamente apenas objetos do tipo UIComponent como "*parte*" na relação

NavigationClassContent	Uma NavigationClass permite apenas objetos do tipo UIComponentField ou NavigationProperty como propriedades
FrontControllerClassContent	Uma FrontControllerClass permite apenas objetos do tipo IOParameter como atributos e FrontControllerMethod como operações
FrontControllerClassOperation	Uma FrontControllerClass permite no máximo e apenas um único FrontControllerMethod cujo atributo FrontControllerMethod.isDefault é "*verdadeiro*" ("*true*")
UIComponentContent	Um UIComponent permite apenas UIComponentFields como atributos e não permite operações
NavigationGeneralizationContent	Uma NavigationGeneralization deve possuir como source e target objetos do tipo FrontControllerClass
DisplayInjectConstraint	Um UIComponent.name precisa ser igual a um IOParamenter.name em uma relação entre UIComponent e IOParamenter ("*display/inject*")
UIComponentExtensionEnd	Um UIComponent só pode ser extendido por objetos do tipo Tag
FrontControllerExtensionEnd	Uma FrontControllerClass só pode ser extendida por objetos do tipo Controller
ResultExtensionEnd	Um Result só pode ser extendido por obejetos do tipo ResultType
ResultConstraint	Um Result só pode ser participar de relações relações do tipo NavigationAssociation
FrontControllerClassConstraint	Uma FrontControllerClass só pode participar de relações do tipo NavigationAssociation

Tabela 7 – Descrição das principais regras OCL do independente de *framework* para **Metamodelo de Entidades**

Nome	Descrição
EntittModelContent	Um EntityModel pode conter apenas objetos do tipo DomainPackage
EntityModelMinimum	Um EntityModel precisa conter pelo menos um DomainPackage
DomainPackageContent	Um DomainPackage pode conter apenas objetos dos tipos DomainClass ou DomainGeneralization ou DomainAssociation ou FrameworkApplication quando estes forem aplicações da categoria ObjetoRelacional
DomainPackageMinimum	Um DomainPackage precisa conter pelo menos um objeto
DomainAssociationRelation	Uma DomainAssociation precisa ocorrer apenas entre objetos do tipo DomainClass
DomainGeneralizationRelation	Uma DomainGeneralization precisa ocorre apenas entre objetos do mesmo tipo e contendo o mesmo estereótipo
DomainClassContent	Uma DomainClass precisa conter apenas objetos dos tipos DomainAttribute ou DomainOperation
DomainClassAttributeTable	O valor padrão ("*default*") para um nome de tabela (DomainClass.table) em uma DomainClass deve ser igual ao nome da classe (DomainClass.name)
DomainGeneralizationSetRelation	Uma DomainGeneralization precisa estar relacionada apenas com DomainGeneralizationSet
DomainClassExtensionEnd	Uma DomainClass só pode ser extendida por objetos do tipo ClassMapping
DomainAttributeExtension	Um DomainAttribute só pode ser extendido por objetos do tipo AttributeMapping

Tabela 8 – Descrição das principais regras OCL do independente de *framework* para **Metamodelo de Persistência**

Nome	Descrição
PersistenceModelContent	Um PersistenceModel pode conter apenas objetos do tipo PersistencePackage
PersistenceModelMinimum	Um PersistenceModel precisa conter pelo menos um PersistencePackage
PersistencePackageContent	Um PersistencePackage pode conter apenas objetos dos tipos DAOClass ou DAOInterface ou DAOGeneralization ou DAORealization ou FrameworkApplication quando estes forem aplicações da categoria ObjetoRelacional
PersistencePackageMinimum	Um PersistencePackage precisa conter pelo menos um objeto
DAOClassContent	Uma DAOClass precisa conter apenas objetos dos tipos DAOAttribute ou DAOMethod
DAOInterfaceContent	Uma DAOInterfaceClass precisa conter apenas objetos dos tipos DAOAttribute ou DAOMethod
DAORealizationRelation	Uma DAORealization precisa ocorrer apenas entre DAOClass e DAOInterface
DAOGeneralizationRelation	Uma DAOGeneralization precisa ocorre apenas entre objetos do mesmo tipo e contendo o mesmo estereótipo
DAOInterfaceName	O DAOInterface.name precisa ser composto pelo DomainInterface.name + DAOInterface.sufix sendo este sufixo obrigatóriamente “DAO”
DAOClassName	O DAOClass.name precisa ser composto pelo DAOInterface.name + DAOInterface.sufix sendo o sufixo obrigatóriamente formado pelo nome do \framework (Framework.name) utilizado
DAOInterfaceInfix	O DAOInterface.infix deve ser formado pelo nome da DAOClass.prefix que o realiza
DAOClassPrefix	O DAOClass.prefix deve ser formada pel nome da DAOClass.infix por ela realizada