Juliana Alves Amaral

Intercâmbio de *Frameworks* especificados em UML-F através de estruturas XML para apoiar o desenvolvimento de software

Belo Horizonte

Pontifícia Universidade Católica de Minas Gerais

2002

Juliana Alves Amaral

Intercâmbio de *Frameworks* especificados em UML-F através de estruturas XML para apoiar o desenvolvimento de software

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica.

Linha de Pesquisa: Sistemas Distribuídos e Inteligência Computacional - SIDIC

Orientador: Prof. Dr. Carlos Alberto Marques Pietrobon

Belo Horizonte

Pontifícia Universidade Católica de Minas Gerais

2002

AGRADECIMENTOS

Ao Rodrigo por seu amor, incentivo, paciência e compreensão;

Aos meus pais, Luciano e Honorina, pelo carinho, compreensão nos momentos de ausência e pela formação que recebi;

Aos meus irmãos Eduardo, Cristiane e Simone pela amizade;

À amiga Claudete, pelo companheirismo e incentivo nessa jornada;

Ao amigo e professor Carlos Pietrobon, pelos sábios conselhos, disponibilidade e dedicada orientação;

Aos professores da PPGEE, pela partilha do conhecimento. Em especial ao Carlos Augusto pelo incentivo e amizade;

À Isabel pelo atendimento cordial;

A Deus, fonte de toda a vida e guia do melhor caminho.

RESUMO

O objetivo principal dessa dissertação é promover o intercâmbio de projetos *frameworks* através do uso de estruturas XML, aumentando a reutilização de software e a utilidade dos *frameworks* entre equipes de desenvolvedores. Esse trabalho combina os benefícios da XML para definir, validar e compartilhar documentos na Web com os benefícios dos *frameworks* e da UML (*Unified Modeling Language*). Nesse trabalho, é proposta a UML-F-X, uma nova extensão da UML-F (uma extensão da UML para o domínio dos *frameworks*), para se obter vantagens dos conceitos de DTD, reduzindo a perda semântica no mapeamento dos *frameworks* para DTDs. Ao somar os esforços da Engenharia de Software com as tecnologias da Internet, esse trabalho pretende ser particularmente útil para os desenvolvedores que pretendem construir e trocar *frameworks* entre ferramentas, utilizando uma abordagem baseada na Web.

ABSTRACT

The main objective of this work is to promote framework design artifacts exchange through the use of XML structures, increasing software reusability and usefulness of frameworks among developers' teams. This work combines the benefits of XML for defining, validating and sharing documents on the Web with the benefits of frameworks and object-oriented Unified Modeling Language (UML). UML-F (an extension of UML for framework domain) is extended by the use of stereotypes and tag values in order to take advantage of DTD concepts, reducing then the semantic loss of framework-DTD conversion process. By joining the software engineering efforts with Internet technologies, this work intends to be quite helpful for developers who want to build and exchange f*rameworks* between software development tools using a Web-based approach.

SUMÁRIO

Lista de Figuras

Siglas

B2B	-	*Business to Business*
CAD/CAM	-	*Computer Aided Design/Computer Aided Manufacturing*
CORBA	-	*Common Object Request Broker Architecture*
DCOM	-	*Microsoft's Distributed Common Object Model*
DDL	-	*Data Definition Language*
DSOM	-	*Distributed System Object Model*
DTD	-	*Document Type Definition*
HTML	-	*Hypertext Markup Language*
IDL	-	*Interface Definition Language*
MER	-	Modelo de entidades e relacionamentos
MFCs	-	*Microsoft's Foundation Classes*
MOF	-	*Meta Object Facility*
MVC	-	*Model-View-Controler*
OCL	-	*Object Constraint Language*
OEM	-	*Object Exchange Model*
OLE	-	*Object linking and Embedding*
OMG	-	*Object Management Group*
OMT	-	*Object Modeling Technique*
OO	-	Orientação a Objeto
OOSE	-	*Object-Oriented Software Engineering*
RMI	-	*Java-Soft's Remote Method Invocation)*
RPC	-	*Remote Procedure Call*
SGML	-	*Standard Generalized Markup Language*
SOAP	-	*Simple Object Access Protocol*
SQL	-	*Structured Query Language*
UML	-	*Unified Modeling Language*
UML-F	-	*Unified Modeling Language Frameworks*
UML-F-X	-	*Unified Modeling Language Frameworks XML*
URI	-	*Uniform resource identifier*
W3C	-	*World Wide Web Consortium*
XMI	-	*XML Metadata Interchange*
XML	-	*Extensible Markup Language*
XP	-	*XML Protocol*
XSL	-	*Extensible Stylesheet Language*

1. Introdução

Esse capítulo apresenta a motivação, os objetivos e a metodologia utilizada nesse trabalho, bem como uma descrição da forma que essa dissertação está organizada.

1.1. Motivação

Novos paradigmas, técnicas e ferramentas de desenvolvimento de software são continuamente procurados pela comunidade de projetistas de software. O incremento nos custos de desenvolvimento e manutenção, o aumento da demanda por informatização na sociedade e a maior complexidade das novas aplicações têm gerado desafios crescentes no processo de desenvolvimento de sistemas de informação. Em muitos casos, as aplicações são caracterizadas por naturezas e domínios diversos e além disso, os seus requisitos mudam rapidamente, inclusive durante o desenvolvimento.

Os métodos clássicos para o desenvolvimento de software não têm sido bem sucedidos em apoiar esse novo paradigma de desenvolvimento. Em contrapartida, a orientação a objeto tem surgido como um poderoso instrumento para o desenvolvimento de sistemas, principalmente no que tange a reutilização, conseguindo suprir algumas das deficiências dos métodos clássicos. Uma das propostas de reutilização, bastante aceita na área de orientação a objetos, é a de se usar *frameworks* para domínios específicos, que poderiam ser instanciados para produzir novos produtos no domínio.

A reutilização de software (código fonte, artefatos de projeto e análise entre outros) tem sido considerada para diminuir os custos de desenvolvimento e manutenção bem como gerar produtos de melhor qualidade, por estar fazendo uso de software já desenvolvido, testado e validado. No entanto, para se alcançar a plenitude dos benefícios prometidos pela orientação a objetos e pela reutilização, é preciso superar obstáculos não só técnicos, mas também gerenciais.

Do ponto de vista gerencial, a prática de reutilizar software exige um alto nível de organização e disciplina de trabalho entre a equipe de desenvolvimento de sistemas. Essa disciplina deve estar presente na criação, documentação e testes de componentes reutilizáveis para que, em um momento futuro, outro desenvolvedor possa localizá-los, adaptá-los e integrá-los numa aplicação específica.

Um dos grandes desafios ligados à reutilização consiste no fato de que essa técnica se opõe à visão individualista e de curto prazo do desenvolvedor solitário que deve concluir rapidamente a codificação do sistema. Os métodos clássicos de desenvolvimento não conseguiram criar uma cultura padronizada de trabalho e o resultado disso é que o processo de construção de software se assemelha mais com o artesanato do que com a engenharia. A reutilização exige uma visão colaborativa de médio e longo prazo, pois os frutos de uma forma de trabalho mais organizada resultarão em benefícios para todo um conjunto de projetistas de software.

Já do ponto de vista técnico, um aspecto problemático do desenvolvimento é a necessidade de integração entre as ferramentas utilizadas nos projetos de software. Se o ambiente de trabalho dos membros da equipe for diferente, as pessoas terão dificuldades técnicas de compartilharem experiências e trocarem informações sobre os projetos. Isto é uma realidade cada vez mais presente atualmente quando pessoas com seus diferentes níveis de conhecimento, localizadas em cidade e países distantes são convidadas a participarem de um mesmo projeto.

Seria interessante que as experiências bem sucedidas em um projeto fossem compartilhadas com equipes de outros projetos ou que diversos participantes de um mesmo projeto pudessem compartilhar a mesma representação. Isto permitiria uma reutilização mais abrangente de software, trazendo ganhos significativos de produtividade. Para que isso seja possível é necessário dispor de mecanismos que permitam o intercâmbio de informações entre as diferentes equipes de projetistas, que não necessariamente trabalham no mesmo local físico.

A integração muitas vezes é problemática pelo simples fato das ferramentas de desenvolvimento serem de fornecedores diferentes. Esta integração pode ser alcançada mapeando os dados manipulados por uma ferramenta diretamente para o formato da outra ou para uma representação intermediária, inteligível por ambas as ferramentas. A XML (*Extensible Markup Language)* tem sido proposta como uma solução para a integração de ferramentas através da representação intermediária. A XML [W3C98] é um padrão aprovado pela W3C *(World Wide Web Consortium*) que deve se tornar um formato universal para a troca de informações na Web.

Além dos problemas dos métodos estruturados de desenvolvimento e dos obstáculos gerenciais e técnicos relacionados com a reutilização de software, uma outra questão pertinente é a manutenção da semântica durante o mapeamento. Ao passar de um modelo para outro, deve-se gerar todas as informações que permitam reconstruir a representação original, sem perda de nenhum elemento modelado. Em muitos processos de conversão de modelos para outros formatos ou linguagens intermediárias, acontece a perda da semântica do modelo original, prejudicando a troca de informações entre os projetistas.

À medida em que as pesquisas sobre Engenharia de Software baseada na Web avançaram, percebeu-se a necessidade de integrar a proposta da XML com a representação de metamodelos [AP2002a], [AP2002b]. A ampliação das oportunidades de intercâmbio entre as equipes poderá contribuir para uma maior reutilização, gerando assim melhorias significativas de produtividade e qualidade de software.

Esse trabalho se propõe a aprofundar o debate ao redor dos problemas técnicos da integração e não das questões gerenciais relacionadas com a disciplina de trabalho colaborativo necessário para a reutilização. Devido à consolidação crescente da Internet como um meio mundial para intercâmbio de dados, optou-se por buscar padrões oriundos da Web na análise de soluções técnicas para o problema da integração.

O presente trabalho se justifica pela necessidade crescente de desenvolver instrumentos que permitam o intercâmbio de informações técnicas de projetos entre equipes de desenvolvedores de software que possivelmente utilizam ferramentas diferentes e que se comunicam via Web. De uma maneira mais específica, esse trabalho irá considerar a adoção de *framework* como o instrumental que permitirá a reutilização de software.

Esse trabalho é pertinente porque, embora a definição de *framewoks* utilizando UML tenha sido considerada por [FPR2000], o intercâmbio dessas arquiteturas entre ferramentas ainda não foi devidamente explorado. Constata-se através de [FPR2000] que os autores se preocuparam em representar a semântica e agregar valores de *framework* à UML em um nível conceitual de especificação, abordando uma situação de um domínio específico: o framework.

Percebe-se, portanto a necessidade de uma nova proposta que contemple tanto o domínio de *framework* quanto o domínio de linguagem de marcação. Esse trabalho se propõe a desenvolver essa proposta que efetuará a integração do domínio dos *frameworks* com o domínio das linguagens de marcação.

1.2. Objetivos

O objetivo principal desse trabalho consiste em desenvolver um roteiro de intercâmbio de estruturas de software (*Frameworks)* entre ferramentas via um documento XML. Para representar *frameworks*, a UML-F será utilizada conforme proposto por [FPR2000]. Para permitir o intercâmbio dessas estruturas, esse trabalho propõe a representação do *framework* expresso em UML-F para um esquema XML (DTD), visto que XML é um padrão para a troca de dados na Web. O roteiro de intercâmbio proposto nessa dissertação envolve a extensão da UML-F para facilitar a representação de *frameworks* em estruturas XML.

Essa nova extensão, que constitui umas das principais contribuições desse trabalho, será denominada UML-F-X, sendo que sua especificação será baseada em propostas de mapeamentos UML-XML existentes na literatura.

A UML-F, de acordo com [FPR2000], se propõe a auxiliar as atividades de projeto e instanciação de *frameworks*. Este trabalho enfatiza apenas o projeto de *frameworks* e não a instanciação ou a implementação desses.

Essa dissertação pretende utilizar três padrões que estão rapidamente se consolidando no mercado: a UML (*Unified Modeling Language*) [OMG99a] como padrão de desenvolvimento de software orientado a objeto, a XML (*Extensible Markup Language*) [W3C98] como padrão de troca de dados e a XMI (XML *Metadata Interchange*) [OMG99b] como padrão de intercâmbio de modelos. É importante frisar que esses três padrões são abertos, isto é, não estão atrelados a nenhum fornecedor específico e tendem a se desenvolver a partir das contribuições da comunidade científica e do meio comercial.

Um objetivo específico desse trabalho é estender a UML-F através dos mecanismos de extensão da UML, permitindo uma representação adequada de *frameworks* em estruturas XML. Essa extensão da UML-F deve ser suficientemente bem estabelecida para minimizar as perdas da semântica dos *frameworks* no mapeamento para esquemas XML.

Outro objetivo específico do trabalho consiste em mostrar a utilidade desse mapeamento na ampliação do emprego de *frameworks* no desenvolvimento de software. É importante estabelecer a diferença entre o projetista do *framework* e o desenvolvedor da aplicação que utiliza o *framework* previamente definido pelo projetista. Neste trabalho, o foco reside no apoio ao projetista do *framework*. Uma vez que os *frameworks* possam ser intercambiados através da Web, a disseminação do uso de *frameworks* como instrumentos de Engenharia de Software torna-se mais fácil, barata e ágil.

1.3. Metodologia

A metodologia desse trabalho consistiu em um estudo bibliográfico seguido da elaboração de uma proposta para permitir o intercâmbio de *frameworks* descritos em UML-F.

Inicialmente, o trabalho envolveu um levantamento bibliográfico sobre *Frameworks* e Padrões de Projeto na literatura existente sobre Engenharia de Software. Esse levantamento foi importante para caracterizar os *frameworks* como uma das mais poderosas técnicas disponíveis para reutilização de software.

A seguir, a pesquisa bibliográfica se concentrou no estudo da linguagem UML. Para tanto, utilizou-se principalmente o material da OMG (*Object Management Group*) e a literatura produzida pelos três criadores da UML (Booch, Jacobson e Rumbaugh). Se a etapa anterior desse trabalho destacou uma técnica de Orientação a Objeto (*framework*), essa etapa enfatizou uma linguagem de modelagem orientada a objetos (UML) e serviu de base para a etapa posterior.

A terceira etapa do trabalho consiste em uma fusão dos conceitos apresentados nas duas etapas anteriores. Essa fusão ocorre através da UML-F, uma extensão da UML voltada para *frameworks.* Nesse ponto, o foco do trabalho se torna mais específico, pois discute a proposta da UML-F. O principal referencial teórico adotado foi a tese de doutorado e os desdobramentos desse trabalho produzidos pelo pesquisador Marcos Fontoura [Fon99, FPR2000].

Devido ao foco Web do trabalho, mostrou-se necessária a realização de uma revisão de literatura sobre intercâmbio de dados e modelos na quarta etapa da presente dissertação. Essa revisão englobou os padrões XML, DTD, XMI e a proposta XML Schema. Nesse momento da pesquisa, foram analisados principalmente os materiais técnicos produzidos pelo organismo padronizador W3C.

Como essa dissertação relaciona a temática do intercâmbio de modelos com a UML, mostrou-se pertinente incluir uma etapa que consistiu na realização de um

levantamento de trabalhos correlatos sobre o mapeamento UML-XML. Apesar da maioria dos trabalhos encontrados não tratarem diretamente do intercâmbio via XML de *frameworks* descritos em UML (objeto de pesquisa dessa dissertação), constatou-se um crescente interesse científico pelo mapeamento UML-XML, resultando em um número relevante de publicações sobre o assunto. Um dos principais trabalhos pesquisados nessa etapa foi o de David Carlson [Car2001] sobre modelagem de aplicativos XML através da UML.

A etapa seguinte consistiu na análise da forma adequada de representação em DTD das características estruturais de um *framework* descrito em UML-F. Para tanto, foi elaborada uma extensão da UML-F, dando origem a UML-F-X. Essa proposta constitui uma das principais contribuições dessa dissertação.

Posteriormente, para fins de verificação, foi realizada a geração de DTDs a partir de um modelo UML-F-X. A geração foi feita de maneira automática, utilizando a ferramenta IBM XMI Toolkit [IBM2000].

A última etapa consistiu na compilação do material produzido, na confecção final e revisão da dissertação.

1.4. Organização da dissertação

Essa dissertação está organizada em cinco capítulos. O segundo capítulo abrange a revisão de literatura, destacando-se a conceituação de *frameworks* e UML. Nesse capítulo, é também apresentada a proposta da UML-F e dos esquemas XML (DTD e XML Schema).

O capítulo 3 inicia com a especificação do mapeamento do diagrama de objetos da UML para documento XML e posteriormente o mapeamento do diagrama de classes para os esquemas XML (DTD e XML Schema). O capítulo apresenta uma breve discussão sobre as possíveis soluções de mapeamento da UML para esquemas XML. O

capítulo finaliza com a argumentação da necessidade de se criar uma extensão da UML-F para representar mais expressivamente *frameworks* em XML.

No capítulo 4, é definida a UML-F-X e são especificadas as características de um metamodelo para representar *frameworks* de uma maneira que facilite o mapeamento futuro para estruturas XML.

No capítulo 5 são apresentadas as conclusões dessa dissertação bem como propostas de trabalhos futuros.

O apêndice 1 contém um estudo de caso que ilustra a geração de DTDs a partir de um modelo descrito em UML-F-X. Nesse estudo de caso, utilizou-se a ferramenta IBM XMI Toolkit [IBM2000] para gerar os DTDs.

2. Revisão da Literatura

Neste capítulo são apresentadas as tecnologias e conceitos envolvidos com o intercâmbio de *frameworks* entre ferramentas via Internet utilizando XML.

2.1. *Frameworks* e Padrões de Projeto

Apesar do avanço das técnicas de desenvolvimento de software, a construção de software continua a ser um processo extremamente complexo. De acordo com [WRN2001], os produtos de softwares têm, em geral, seus cronogramas atrasados, custo maior do que o esperado e apresentam defeitos. Isto resulta em uma série de inconveniências para os usuários e traz consigo uma enorme perda de tempo e recursos.

Uma das abordagens para diminuir a complexidade e aumentar a produtividade e a qualidade tem sido a reutilização de software. Entretanto, esta não é uma tarefa fácil. Segundo [FSJ99], a crescente heterogeneidade de arquiteturas de hardware e software somada à diversidade de sistemas operacionais e plataformas de comunicação dificulta a reutilização de projetos. Pode-se citar também aspectos culturais e de formação que dificultam a reutilização. Pouco se aproveita de sistemas de informação já existentes quando se precisa construir um novo sistema.

De acordo com [GHJV95], uma coisa que os projetistas experientes sabem que não devem fazer é resolver cada problema a partir de princípios elementares ou do zero. A abordagem correta está na reutilização das soluções que funcionaram no passado. No entanto, segundo [PPSS95], apesar das óbvias vantagens da reutilização de software, a maior parte dos sistemas continua a ser desenvolvida a partir do ponto zero. Com isso, pode-se constatar que a metáfora da reinvenção da roda é uma prática freqüente no processo de desenvolvimento de software.

Em ambientes distribuídos, o problema assume dimensões ainda mais críticas. [SF97] denomina esse paradoxo de crise do software distribuído, pois o hardware está

cada vez menor, potente e barato e as redes mais velozes, enquanto que os sistemas distribuídos se tornam maiores, mais lentos e mais caros de desenvolver e de manter.

A orientação a objeto tem surgido como um poderoso instrumento para o desenvolvimento de sistemas de acordo com os princípios propostos pela engenharia de software, principalmente no que tange a reutilização. Os componentes podem ser vistos como os blocos de construção do software moderno. Este é um dos motivos pelos quais a orientação a objetos é um paradigma que tem obtido muito espaço na prática de engenharia de software. Segundo [Jon2001], os objetos tornaram-se os blocos de construção onipresentes do software moderno e a orientação a objetos é o paradigma dominante da prática de engenharia de software contemporânea. O objeto representaria para o software o papel desempenhado pelas placas no hardware, buscando estruturar uma arquitetura baseada em componentes.

A correta representação da estrutura do sistema é uma questão muito pertinente no desenvolvimento de software. [BL2000] afirmam que a necessidade de construir sistemas grandes e complexos faz com que o problema não se restrinja apenas a escolher algoritmos e estruturas de dados, mas sim a projetar e especificar a estrutura do sistema ou seja, sua arquitetura. Para [BL2000], a arquitetura de software envolve a descrição de elementos dos quais os sistemas são construídos, a interação entre esses elementos, os padrões para guiar a composição e as restrições desses padrões. Percebe-se nesse ponto uma forte sintonia entre a modularização da orientação a objeto e a abordagem desejada para a arquitetura de software. De acordo com [MKMG97], a arquitetura de software provê um nível de abstração para que os projetistas possam compreender o comportamento do sistema e conscientizar-se a respeito das diretrizes de evolução do sistema.

Em [BL2000], os autores argumentam que uma base arquitetural para a composição de componentes reutilizáveis auxiliaria muitos dos problemas da construção de software. O benefício básico da reutilização ou reusabilidade de software é uma maior produtividade de software. Para [You95], a reutilização também produz

uma melhor qualidade, pois o componente de software reutilizável sempre requer mais testes e garantia de qualidade, simplesmente porque as conseqüências de um erro são bem mais sérias e o uso contínuo ocasiona uma maior probabilidade de detecção de erros.

Uma das propostas de reutilização bastante aceita na área de orientação a objetos, é a de se usar *frameworks* para domínios específicos, que poderiam ser instanciados para produzir novos produtos no domínio. De acordo com [OFL2001], as restrições de custo e tempo impostas ao desenvolvimento moderno de software obrigam os desenvolvedores a abandonar a prática de se partir do zero. Assim sendo, os desenvolvedores devem aderir a uma abordagem que suporte a reutilização, adotando soluções comprovadas como componentes e *frameworks*.

Segundo [FSJ99], *framework* é uma técnica da orientação a objeto voltada para a reutilização que se beneficia de três características das linguagens de programação orientadas a objeto: abstração de dados, polimorfismo e herança. Um *framework* descreve a arquitetura de um sistema orientado a objeto, os tipos de objetos e as interações entre os mesmos. Para o autor, um *framework* é um esqueleto de uma aplicação que pode ser customizado por um desenvolvedor na construção de um software. Um *framework* modela genericamente uma família de aplicativos semelhantes, permitindo uma maior agilidade que se traduz em uma redução de custos no processo de desenvolvimento de software.

Já [PPSS95] definem f*ramework* como uma arquitetura semi-acabada e passível de reutilização para um domínio de aplicação. [Sch97] conceitua *framework* como sendo uma aplicação genérica que permite a criação de diferentes aplicações em um determinado domínio. Para [BL2000], um *framework* define uma arquitetura para uma família de sistemas, fornecendo partes predefinidas para sua construção e quais partes que devem ser adaptadas para uma função específica. Verifica-se que algumas definições de *framework* enfatizam a estrutura enquanto outras destacam o propósito do *framework*.

De acordo com [FSJ99], o foco intensivo em *frameworks* oferece aos desenvolvedores de software um novo veículo para reutilização e um caminho para capturar a essência de padrões bem sucedidos, componentes bem estruturados e mecanismos de programação eficientes.

Para destacar a importância da reutilização, [OFL2001] propõem a introdução de uma fase explícita de reutilização no processo clássico de desenvolvimento. Segundo os autores, nessa nova fase, que ocorrerá entre a fase de projeto e a codificação, o analista deverá identificar a possibilidade de reutilização de um *framework,* que representa uma solução comprovada para o domínio de aplicação relacionado ao sistema que está sendo desenvolvido.

Com *frameworks* não se busca apenas reutilizar simples componentes de software, mas subsistemas, aumentando assim o grau de reutilização e contribuindo para uma melhor qualidade de software. Segundo [PPSS95], não apenas o código fonte, mas também o projeto da arquitetura é reutilizado em aplicações construídas a partir de *frameworks,* constituindo um avanço nas iniciativas de reutilização de software.

Para [Joh97], a visão original de reutilização de software estava baseada em componentes. Os *frameworks* possuem interfaces mais complexas, mas são de mais fácil customização do que os componentes. [Joh97] percebe *frameworks* e componentes como técnicas diferentes, mas que cooperam entre si, pois um *framework* pode facilitar a construção de novos componentes e fornecer uma interface padrão para os mesmos trocarem dados, manipularem erros e chamarem operações entre eles.

De acordo com [BL2000], em um ambiente orientado a objeto, um *framework* é composto de classes abstratas e concretas e a sua instanciação consiste de composição de herança de classes. Através do recurso da herança proporcionado pela orientação a objetos, é possível definir propriedades (variáveis e métodos) similares de classes em uma superclasse comum. Segundo [PPSS95], uma classe abstrata é um tipo de classe

que define um comportamento comum entre as classes e que não pode ser instanciada diretamente. Assim sendo, a classe abstrata cria uma interface padrão para todas as suas classes descendentes (subclasses). Segundo [FSJ99], como a classe abstrata não tem instâncias, ela é então usada como um molde padrão (*template*) para a criação de subclasses e não de objetos. Por isso, os *frameworks* usam as classes abstratas porque essas definem a interface dos componentes e fornecem um esqueleto que serve de infra-estrutura para estender a implementação dos componentes.

Além de prover a interface, a classe abstrata pode também fornecer parte da implementação para suas subclasses, segundo [FSJ99]. Um método *template* define o esqueleto de um algoritmo em uma classe abstrata, adiando o detalhamento para as subclasses. A classe abstrata pode possuir métodos abstratos e/ou pode fornecer uma implementação *default*, denominada método *hook*. De acordo com o autor, uma classe concreta deve implementar todos os métodos abstratos de sua superclasse e pode implementar qualquer um dos métodos *hook.*

Segundo [Sch97], a utilização de *framework* orientado a objeto difere de aplicativos comuns pois algumas de suas partes, chamadas de *hot spots* ou ponto flexíveis podem apresentar diferentes implementações para cada instância de *framework* e são deixadas incompletas até o tempo de instanciação ou da implementação. Assim sendo, os *frameworks* alavancam o conhecimento e o esforço prévio de desenvolvedores experientes de maneira a evitar o retrabalho de outros desenvolvedores na criação de soluções usuais para aplicativos semelhantes. Para [FPS99], a reutilização derivada do emprego de *frameworks* pode produzir melhorias substanciais na produtividade dos programadores e aumento de qualidade, desempenho, confiabilidade e interoperabilidade de software.

Preocupados em trabalhar com *frameworks* em um nível mais abstrato onde se pudesse especificar e projetar *framework* e reconhecendo as limitações da UML, [FPR2000] propõem uma extensão da UML, denominada UML-F que permite a representação de *frameworks*. Felizmente a UML não é um padrão rígido e imutável,

pois foi concebida de forma a prover um certo grau de liberdade aos usuários para ajustar a linguagem às suas necessidades.

[Pre95] apresenta um *framework* como sendo formado por *hot spots* e *frozen spots*. *Frozen spots* definem a arquitetura global de um sistema, seus componentes básicos e os relacionamentos entre eles, permanecendo inalterados – daí o termo congelado - em qualquer instanciação do f*ramework*. Por outro lado, os *hot spots* são partes específicas para cada sistema, sendo projetados para serem genéricos e poderem ser adaptados de acordo com as necessidades da aplicação. [PPSS95] definem *hot spots* como pontos de refinamentos predefinidos, onde a especialização e a adaptação ocorrem. A especialização de *hot spots* requer a definição de classes adicionais de maneira a sobrepor métodos e/ou configurações de objetos baseados nos componentes já fornecidos pelo *framework*.

De acordo com [Sch97], um subsistema de *hot spot* contém as seguintes partes:

- Uma classe abstrata de base, que define a interface para as responsabilidades comuns;
- Classes concretas derivadas, representando cada uma das diferentes alternativas para os aspectos variáveis;
- Parte opcional com relacionamentos e classes adicionais.

Os primeiros *frameworks* disponibilizados segundo [PPSS95] foram *Smalltalk's Model-View-Controler (MVC) e MacApp.* Outros exemplos são Taligent's *frameworks*, OLE (*Object linking and Embedding*), OpenDoc, Beans, ET++, MFCs (Microsoft's *Foundation Classes*), DCOM (Microsoft's *Distributed Common Object Model*), RMI (Java-Soft's *Remote Method Invocation*) e DSOM (*Distributed System Object Model*). Segundo [Jon97], a maioria dos *frameworks* disponíveis comercialmente são de domínio técnico tais como interface com o usuário e sistemas distribuídos sendo que a maior parte dos *frameworks* específicos de aplicações são proprietários.

A utilização de *frameworks* apresenta os seguintes benefícios, segundo [FSJ99] :

- **Modularização:** é melhorada através do encapsulamento dos detalhes voláteis de implementação atrás de interfaces estáveis. Fica assim mais fácil de localizar o impacto das mudanças no projeto e na implementação, reduzindo o esforço necessário na manutenção do software;
- **Reutilização:** é aumentada através da definição de componentes genéricos que podem ser reaplicados para criar novos sistemas;
- **Extensibilidade:** é favorecida pelo uso dos métodos *hook* que permitem que as aplicações estendam interfaces estáveis. Isto é essencial para assegurar a customização de novas aplicações;
- **Inversão de controle:** o código específico do desenvolvedor é chamado pelo código do *framework*. Dessa forma, o *framework* controla a estrutura e o fluxo de controle dos programas. Para garantir uma maior segurança, o *framework* estabelece quais métodos específicos da aplicação devem ser ativados em resposta aos eventos externos realizados pelos usuários.

Para que seja possível obter os benefícios prometidos pelo *framework* é fundamental investir na qualidade do projeto do *framework*. [Sch97] destaca que o projeto de um *framework* é muito mais complexo do que o projeto de uma aplicação tradicional, devido à flexibilidade inerente e capacidade de variação de um *framework*. [PPSS95] também concordam com esse ponto de vista, ao afirmarem que a complexidade do desenvolvimento e adaptação do *framework* constitui o maior desafio para a adoção dessa avançada técnica de orientação a objeto entre desenvolvedores de software. Construir um *framework* não é uma tarefa fácil. O caminho até a reutilização prometida é árduo, mas os benefícios justificam os desafios.

Um conceito muito próximo de *framework* é o de padrões de projeto (*design patterns*). [GHJV95] definem um padrão de projeto como sendo descrições de objetos e classes comunicantes que são customizados para resolver um problema geral de um

projeto em um contexto particular. Segundo [GHJV95], um padrão de projeto (*pattern*) tem quatro elementos essenciais:

- **Nome do padrão:** referência utilizada para descrever um problema de projeto, suas soluções e conseqüências em poucas palavras. Dar nome a um padrão aumenta o vocabulário de um projeto, eleva o nível de abstração e permite o diálogo entre equipes;
- **Problema:** descreve quando aplicar o padrão, detalhando o problema e explicando seu contexto. Pode incluir uma lista de condições que devem ser satisfeitas para que faça sentido aplicar o padrão;
- **Solução:** descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações. O padrão fornece uma descrição abstrata de um problema de projeto e de como um arranjo geral de classes e objetos resolve o mesmo;
- **Conseqüências:** são os resultados e análises das vantagens e desvantagens da aplicação do padrão. Incluem o impacto sobre a flexibilidade, extensibilidade ou portabilidade de um sistema;

Segundo [BL2000], padrões documentam *frameworks* e ajudam a garantir o correto uso de sua funcionalidade. Para [GHJV95], o objetivo do uso de padrões é capturar a experiência de projetos de software de forma que os projetistas possam usá-la efetivamente. Os padrões ajudam tanto a escolher alternativas de projeto que tornam o sistema reutilizável quanto a evitar alternativas que comprometeriam a reutilização. Além disso, [Joh97] argumenta que a uniformidade reduz o custo de manutenção de software, pois os desenvolvedores podem migrar mais facilmente entre aplicações sem necessitar de aprender tudo de novo.

[GHJV95] destacam que padrões de projetos (*design patterns*) facilitam a reutilização de sistemas e arquiteturas bem sucedidas. Segundo [Joh97], padrões são elementos micro arquiteturais de *frameworks*. Um *framework* usualmente contém

muitos padrões, ou seja, padrões são menores do que *frameworks*, podendo ser vistos como seus blocos construtores. Entretanto, um *framework* pode ser também um padrão.

Como os *frameworks* são descritos através de linguagens de programação, o aprendizado de um *framework* pode ser difícil para um desenvolvedor iniciante, requerendo muito leitura de código e contato com profissionais mais experientes. [Joh97] defende o uso de padrões como uma abordagem para melhorar a documentação de *Frameworks*.

Conforme detalhado acima, *framework* constitui uma das mais promissoras correntes tecnológicas para suportar reutilização de software em grande escala, modelando tanto a parte genérica (*frozen spots*) como as partes variáveis (*hot spots*) de um sistema. Portanto, neste trabalho será utilizada a tecnologia de *framework*, pois essa permite a concepção de arquiteturas mais portáteis e consequentemente passíveis de um maior nível de reutilização.

2.2. UML

Em conformidade com os princípios traçados pela Engenharia de Software, a Orientação a Objeto (OO) tem sido utilizada como uma técnica para o desenvolvimento de sistemas. A Orientação a Objeto constitui uma evolução das técnicas estruturadas tradicionais. Uma das principais características da OO é a fusão da modelagem baseada em dados com a modelagem centrada em processos, permitindo uma maior coesão na análise e no projeto de software. A Orientação a Objeto se baseia em vários conceitos como encapsulamento, herança, classe, mensagem, polimorfismo e outros.

A indústria de software percebeu rapidamente os benefícios do uso da OO. Várias propostas de análise e projeto orientado a objeto surgiram nos meios acadêmicos e empresariais. Essa diversidade de notações demandou a convergência para um padrão único de representação dos conceitos de OO. A UML (*Unified Modeling Language)* constitui a unificação das melhores práticas de OO utilizadas na indústria de software.

A UML é uma linguagem consistente para especificar, visualizar, construir e documentar software, segundo [BRJ98]. Ela é o resultado da convergência das melhores práticas de Orientação a Objeto disponíveis no mercado. Contribuíram para sua construção, entre outras, a proposta de Booch, a OMT (*Object Modeling Technique)* especificada por Rumbaugh e a OOSE (*Object-Oriented Software Engineering)* concebida por Jacobson.

Segundo [Car2001], a UML é um metamodelo para o desenvolvimento de software. O metamodelo, também chamado de modelo de metadados, representa a definição da estrutura e do significado dos dados. Se as ferramentas compartilham o mesmo metamodelo, todos os dados podem ser entendidos e utilizados. O MOF (*Meta Object Facility*) é um padrão adotado pelo OMG (*Object Management Group)* para definir metamodelos. Segundo [CH98], o MOF estabelece um meta-metamodelo com a semântica suficiente para permitir a definição de metamodelos em vários domínios. Dessa forma, a UML pode ser entendida como uma instância do MOF.

O surgimento da UML contribuiu para uma ampliação crescente do uso das técnicas de OO na concepção de software. Segundo [Jon2001], as qualidades mais frequentes observadas em sistemas construídos no modo orientado a objeto são: reutilização, confiabilidade, robustez e extensibilidade.

Para [OMG99a], a UML consiste na especificação semi-formal de um metamodelo para representar a semântica de modelos de análise e projeto orientados a objeto, incluindo modelos estáticos, comportamentais, de uso e arquiteturais. De acordo com [Jon2001], a UML consegue capturar a estrutura dos sistemas orientados a objeto e expressá-la em diagramas que englobam a gama de construções que aparecem em softwares.

Estes diagramas em UML representam a projeção gráfica dos elementos que formam um sistema. No entanto, cabe salientar que o aspecto visual, embora extremamente relevante, não é abordado na definição do metamodelo UML. Segundo

[BJR98], nunca será possível visualizar a estrutura ou comportamento de um sistema através de um único diagrama. Em vez disso, a UML propõe a criação de vários diagramas, sendo cada um com o foco em uma única perspectiva. Esta representação gráfica, ou seja os diagramas, podem permitir a visualização tanto da parte estática quanto da parte dinâmica de um sistema. Os diagramas estáticos contidos na UML são:

- Diagrama de classe
- Diagrama de objetos
- Diagrama de componentes
- Diagrama de implantação

E os diagramas dinâmicos são:

- Diagrama de caso de uso
- Diagrama de seqüência
- Diagrama de colaboração
- Diagrama de transição de estados
- Diagrama de atividades

O modelo de visão 4+1 da arquitetura proposto por [Kru95] descreve uma arquitetura de software utilizando cinco visões concorrentes, sendo que os desenvolvedores utilizam quatro visões para representar suas decisões de projeto e aplicam a quinta visão para fins de ilustração e validação. As visões propostas são as seguintes:

- Visão lógica: suporta os requisitos funcionais e descreve o modelo de objetos através de diagramas de classes;
- Visão de processo: leva em consideração os requisitos não-funcionais como concorrência, sincronização, desempenho e disponibilidade do sistema;
- Visão física: descreve o mapeamento do software para os nós de processamento em termos dos aspectos distribuídos;

- Visão de desenvolvimento: descreve a organização estática do software no seu ambiente de desenvolvimento, detalhando a estruturação dos subsistemas em camadas;
- Visão de cenários: organiza a descrição das quatro visões anteriores através da ilustração com use-cases.

De acordo com [BJR98], os quatro diagramas estáticos ou estruturais existem para visualizar, especificar, construir e documentar os aspectos estáticos de um sistema, isto é, a representação de seu esqueleto e sua estrutura relativamente estáveis. Já os cinco diagramas comportamentais buscam representar as partes dinâmicas do sistema.

De acordo com [Fur98], o diagrama de classes é a essência da UML e o resultado de uma combinação de diagramas propostos pela OMT e pela OOSE. [BJR98] reforçam essa opinião ao afirmarem que os diagramas de classes são os diagramas mais encontrados em sistemas de modelagem orientados a objetos. Essa dissertação se concentra no diagrama estático de classes, pois esse é o diagrama central utilizado pela UML-F para descrever a estrutura dos *frameworks*, conforme será detalhado no item 2.4.

Segundo [Jon2001], os setes objetivos principais definidos pelos criadores da UML são os seguintes:

- Prover aos usuários uma linguagem de modelagem visual expressiva e pronta para uso, de forma que eles possam desenvolver e compartilhar modelos significativos;
- Prover mecanismos de extensibilidade e especialização para ampliar os conceitos centrais;
- Ser independente de linguagens de programação e processos de desenvolvimento particulares;
- Prover uma base formal para entendimento da linguagem de modelagem;
- Estimular o crescimento do mercado de ferramentas orientadas a objetos;

- Suportar conceitos de desenvolvimento de nível mais alto, tais como colaborações, estruturas, modelos e componentes;
- Integrar as melhores práticas.

De acordo com [Fur98], a UML pode ser usada para as seguintes atividades ao longo de um processo de desenvolvimento de software:

- Mostrar as fronteiras de um sistema e suas funções principais utilizando atores e casos de uso;
- Ilustrar a realização de casos de uso com diagramas de interação;
- Representar uma estrutura estática de um sistema utilizando diagramas de classe;
- Modelar o comportamento de objetos com diagramas de transição de estado;
- Revelar a arquitetura de implementação física com diagramas de componente e de implantação;
- Estender sua funcionalidade através de estereótipos.

Em particular, essa dissertação está bastante relacionada com o segundo objetivo da UML entre os sete apresentados por [Jon2001] e a terceira atividade citada por [Fur98].

Nesse trabalho, será utilizada a UML, pois como especificado anteriomente, essa se constitui uma técnica de modelagem bem abrangente. [CSF2000] considera a UML como o elo de conexão entre a Engenharia de Software e o projeto de documentos, pois é possível projetar software OO juntamente com as estruturas XML necessárias.

2.3. Alternativas de Modelagem de Dados

Os modelos de dados tradicionais (Modelo de entidades e relacionamentos - MER, Modelo Relacional dentre outros) não tratam dados complexos (dados aninhados, imagens, etc.). Com o surgimento de novas aplicações como geoprocessamento, ferramentas CAD/CAM (*Computer Aided Design/Computer Aided Manufacturing)* em

projetos de engenharia e sistemas de multimídia que necessitavam manipular essa nova classe de dados, viu-se necessária a criação de um novo modelo para representá-los.

Por outro lado, o rápido crescimento da Internet e a grande disseminação do seu uso possibilitaram a Web a se tornar uma das principais fontes de informação disponíveis e percebeu-se a necessidade da modelagem desses dados para posteriormente fazer o intercâmbio dos mesmo com outros sistemas . A modelagem de dados contida na Web apresenta desafios para as técnicas tradicionais de modelagem, visto a natureza dos dados contidos nela, que são na sua grande maioria semi-estruturados. De acordo com [Bun97], nos dados semi-estruturados a informação é normalmente associada com um esquema que está contido nos dados, que são chamados de auto-descritivos.

Várias abordagens para modelar a Web foram propostas. Algumas procuram modelar a Web como grafos. Em outras propostas de representação de dados, estes são representados como coleção de objetos. O OEM (*Object Exchange Model*) é um exemplo dessa abordagem. Segundo [Suc98] os dados em um modelo OEM formam um grafo em que os nós são os objetos e as arestas são rotuladas com nomes de atributos. Este tipo de grafo é também chamado de multigrafo, pois não existe restrição no conjunto de arcos ligados a um nó nem no tipo de valores dos atributos. Segundo [FLM98], a abordagem anterior apresenta restrições impostas pelo grafo dirigido e rotulado. Já o OEM provê maior flexibilidade e enfatiza o aspecto irregular dos dados.

A XML *(Extensible Markup Language*) também pode ser uma opção para representar um modelo de dados semi-estruturados. A XML é uma linguagem de marcação que constitui uma alternativa para representação de dados na Web. Entretanto, de acordo com [KH2000] , a XML não apresenta um nível de abstração mais alto para modelar documentos na Web. Para [Car2001], os DTDs XML são inadequados para modelar sistemas mais complexos, pois seu vocabulário pode ser compreensível para desenvolvedores experientes, porém não é tão legível para usuários e outros participantes do desenvolvimento de uma sistema. Em virtude disso, o autor sugere o

uso do diagrama de classes da UML para representar visualmente os elementos, relacionamentos e restrições de um vocabulário XML, visto que a UML é um padrão para modelar visualmente sistemas, sendo de mais fácil compreensão entre as pessoas envolvidas no desenvolvimento de sistema. Verifica-se assim que o principal objetivo da XML é o intercâmbio de dados entre aplicações e não propriamente a modelagem dos mesmos.

Entre as abordagens de modelagem apresentadas anteriormente, a UML se diferencia pela sua capacidade de representar tanto aplicações tradicionais como aplicações não convencionais como sistemas baseados na Web, visto que a UML se baseia nos conceitos da Orientação a Objetos, definindo uma notação abrangente para representar dados complexos.

Segundo [Car2001], a Web está modificando fortemente o processo de desenvolvimento de sistemas corporativos. Por outro lado, a UML tem se mostrado capaz de fornecer uma notação uniforme para projetar sistemas das naturezas mais diversas. Assim sendo, optou-se por empregar a UML (descrita no item 2.2) nesse trabalho ao invés de se adotar as alternativas de modelagem apresentadas nesse item.

2.4. UML-F

A tecnologia de *frameworks* tem adquirido importância nos últimos tempos dentro da comunidade de engenharia de software. Como um software complexo, *frameworks* devem ser cuidadosamente projetados. Algumas abordagens usadas para modelar *frameworks* (modificadas de [Fon99]) são apresentadas abaixo:

- Técnicas padronizadas de análise e projeto OO - Técnicas padronizadas como a OMT e a UML modelam visões estáticas (diagramas de classes e objetos, etc.) e visões dinâmicas (diagramas de estado, atividades dentre outros), porém não representam de forma clara os *hot spots* (variabilidade de um método ou extensibilidade de uma classe) nem como os mesmos podem ser instanciados. Assim

sendo, a diferença entre o núcleo de um *framework* (*frozen spots)* e as partes variantes (*hot spots)* não é explicitada nas técnicas padronizadas de análise e projeto OO;

- Padrões de projeto - São normalmente descritos usando notações padrões da OO (OMT e UML), fornecendo uma representação mais clara de *framework*. Através de seus diferentes padrões (ex.: *Strategy, Visitor*), os problemas de variabilidade de um método ou extensibilidade de uma classe, ou seja, os *hot spots* de um *framework*, podem ser melhor representados. No entanto, o usuário projetista da aplicação, necessita conhecer os padrões de projeto para compreender a representação do *framework.* Além disso, para uma mesma classe de *hot spot* pode-se usar diferentes padrões de projetos. Soma-se a isso o fato de que a representação de padrões de projeto através das notações da OO pode resultar em modelos muito complexos, até mesmo para *frameworks* mais simples. Portanto, os padrões de projeto podem até ajudar na representação de *frameworks*, mas não resolvem o problema pois exigem conhecimento prévio de suas estruturas e não foram concebidos para serem linguagens de modelagem (como a UML), mas para serem soluções de problemas comuns em projetos;

- Meta-Programação – É um padrão arquitetural que permite a reconfiguração do sistema em tempo de execução, pertencendo à mesma categoria dos padrões de projeto. A meta-programação é uma alternativa para modelar *hot spots* que requerem instanciação em tempo de execução (*hot spots* dinâmicos), porém os *hot spots* estáticos não são modelados. Da mesma forma que os padrões de projeto, a meta-programação não foi projetada para ser uma linguagem de modelagem;

- Cartões *Hot Spot* e Meta-Padrões – Os cartões *hot spot s*ão utilizados para documentar as informações requeridas no desenvolvimento de um *framework*. Eles representam a semântica de cada *hot spot* de um *framework* e podem ser de dois tipos: cartões de dados ou de funções. Os cartões *hot spot* estão preocupados com a captura do conhecimento para auxiliar o desenvolvimento de *framework*s na fase de

análise de requisitos. Já na fase de projeto, propõe-se o uso de meta-padrões, que também são chamados de essência do padrão de projetos. Os meta-padrões descrevem mecanismos para montar padrões de projeto em domínios específicos. Entretanto, o usuário necessita conhecer os meta-padrões para compreender a representação do *framework*. Assim sendo, essa abordagem apresenta as mesmas limitações que a abordagem de padrões de projeto.

As propostas apresentadas anteriormente não representam todos os aspectos da semântica de um *framework*. Algumas abordagens não representam de forma clara os *hot spots* ou como os mesmos podem ser instanciados. Em outras, o usuário necessita ter o conhecimento dos conceitos dos padrões de projeto ou meta-padrões para compreender a representação do *framework*. Visando contornar essas limitações, [FPR2000] propuseram a UML-F, uma extensão da UML, para modelar *framework*. A UML-F pode ser considerada uma evolução em relação às abordagens anteriores, pois representa e classifica os *hot spots* de *frameworks* de uma forma mais clara e direta.

A própria especificação da UML já previa que seriam necessárias extensões para utilizar a UML em projetos específicos. A UML pode ser avaliada como uma linguagem aberta, pois possui conceitos centrais, mas permite ainda o acréscimo de novos conceitos. [Jon2001] destaca essa flexibilidade da UML e argumenta que os usuários que desejarem customizar a UML precisam:

- Construir modelos que utilizam conceitos centrais sem utilizar mecanismos para extensão na maioria das aplicações normais;
- Acrescentar novos conceitos e notações para temas não cobertos pelo núcleo;
- Escolher entre interpretações variantes de conceitos existentes, quando não houver um consenso;
- Especializar os conceitos, as notações e as restrições para domínios de aplicações particulares.

Verifica-se, portanto, que a UML-F não constitui uma agressão ou desvirtuamento da UML original, mas uma bem-vinda adaptação da UML para o projeto de *frameworks*. Essa flexibilidade da UML permite que os seus próprios conceitos centrais não sejam alterados mais do que necessário e facilita a customização da UML à medida que novas necessidades forem descobertas para domínios específicos.

Para modelar *framework*, é preciso projetar *frozen spots* e *hot spots*. Os primeiros podem ser modelados com os recursos que a UML oferece. Quanto aos *hot spots,* são necessárias extensões na UML para a sua adequada especificação De acordo com [BRJ98], a UML provê os seguintes mecanismos de extensão: estereótipos, *tag values* e restrições.

Os estereótipos são mecanismos para permitir a adicionar novos blocos de construção semelhantes aos existentes, mas específicos a um determinado problema. Um estereótipo é representado graficamente como um nome entre os sinais << e >> (ex.: *<<include>>,<<implementation>>)* colocado acima do nome de outro elemento. A UML apresenta uma relação de palavras reservadas que são utilizadas como estereótipo, sendo que cada palavra se aplica a um determinado símbolo com um significado específico. Segundo [Fur98], esses estereótipos possuem uma semântica predefinida.

Os *tag values* são meios para proporcionar a criação de novas propriedades, permitindo a criação de novas informações na especificação desse elemento. De acordo com [BJR98], um *tag value* é representado como uma seqüência de caracteres entre chaves, colocado abaixo do nome de outro elemento (ex.: {quantidade = 3}). Essa seqüência de caracteres usualmente inclui um nome, um separador (o símbolo =) e um valor. Caso o significado não seja ambíguo, pode-se especificar apenas o valor. De acordo com [FPR2000], a UML permite que os *tags* com valores booleanos possam ser escritos da forma {*tag*}, ao invés de {*tag* = TRUE}. Um *tag value* não é o mesmo que os atributos de uma classe. Os *tag values* devem ser considerados como metadados, pois seus valores se aplicam aos próprios elementos e não às respectivas instâncias. [BJR98]

comparam o uso de estereótipos com o de *tag values* da seguinte forma: "*Utilizando os estereótipos, pode-se adicionar novos itens à UML; usando tag values, pode-se acrescentar novas propriedades.*"

As restrições são mecanismos para especificar uma nova semântica de algum elemento UML. A restrição é representada como uma seqüência de caracteres entre chaves, colocada próxima ao elemento associado. Da mesma forma que nos estereótipos e *tag values*, a UML também prevê uma relação de palavras reservadas com um significado predefinido para serem utilizadas como restrições. Outras restrições podem ser definidas pelos usuários. Segundo [Fur98], a UML ainda inclui uma especificação da linguagem OCL (*Object Constraint Language)* para ser utilizada na definição de restrições.

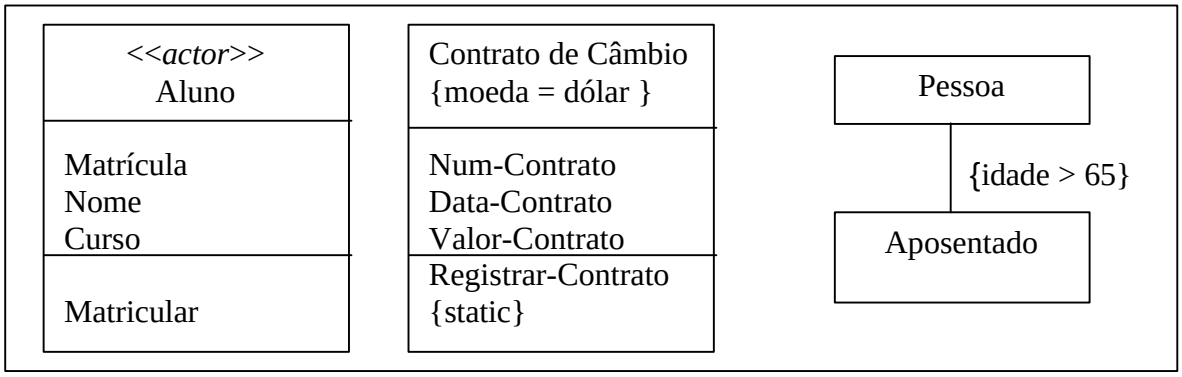

Figura 2. 1– Mecanismos de extensão da UML

A figura 2.1 exibe os três mecanismos de extensão. O primeiro apresentado <<*actor*>> é um estereótipo aplicado a uma classe e significa que a classe Aluno se refere ao ator Aluno descrito no diagrama de casos de uso da UML. Posteriormente foi mostrado o *tag value* {moeda} que assume o valor dólar para a classe Contrato de Câmbio. Por fim, a classe Aposentado possui uma restrição de idade maior do que 65 em relação à classe Pessoa.

Segundo [Fon99] , a UML-F trata três tipos de *hot spots* em um *framework* orientado a objeto: métodos variáveis, classes estendidas e interfaces estendidas. Métodos variáveis são métodos que têm uma interface bem definida, mas que podem

variar em sua implementação de acordo com a instanciação do *framework.* Classes estendidas são classes que podem ser estendidas durante a instanciação do *framework*, recebendo novos métodos, por exemplo. Interfaces estendidas ou classes abstratas permitem a criação de subclasses concretas na instanciação do *framework.* Estas novas classes são chamadas de classe de aplicação, que existem somente quando um *framework* é instanciado.

Os três tipos de *hot spots* acima podem ser estáticos (instanciação não requerida em tempo de execução) ou dinâmicos (instanciação requerida em tempo de execução).

O *hot spot* interface estendida possui mapeamento direto do projeto para uma linguagem de programação OO, porém o método variável e a classe estendida não possuem um mapeamento direto. Entretanto, muitas técnicas de implementação podem ser usadas para transformar o método variável e a classe estendida em interface estendida sem alteração da semântica do *framework.*

Na UML-F, os *hot spots* de um *framework* são representados das seguintes maneiras [Fon99]:

- Extensão do diagrama de classes que identifica explicitamente e classifica os *hot spots* de um *framework.* Os seguintes *tag values* representam essa extensão: *variable, extensible, static, dynamic, incomplete e appl-class. Variable* é aplicado aos métodos de uma classe e representa o *hot spot* métodos variáveis. *Extensible* é aplicado a classe e representa o *hot spot* classes estendidas. Já a *tag incomplete* é aplicado na generalização/especialização e representa o *hot spot* interfaces estendidas. A *tag appl-class* complementa a definição de classes estendidas. Ela é usada para indicar um aditivo na estrutura de um *framework* onde classes específicas da aplicação podem ser ou já foram adicionadas. A *tag incomplete* permite o usuário de um *framework* criar, no momento de instanciação de um *framework,* quantas classes de aplicações forem necessárias dado uma interface extensível. A palavra reservada *incomplete* já existia na UML como um mecanismo de extensão

do tipo restrição. Os *tags static e dynamic* complementam a notação do *hot spot*, indicando se o mesmo requer instanciação em tempo de execução ou não;

- Diagrama de *template* (diagrama de sequência da UML) define as restrições no modelo de instanciação de um *hot spot*. Um valor de tag, *optional*, indica se um evento pode ou não ocorrer. Para cada *hot spot* em um *framework* um conjunto finito de diagramas de *template* podem ser especificados, sendo que pelo menos um deles deve ser seguido por uma instância de *hot spot*. Assim sendo, o diagrama de *template* é usado para descrever um comportamento padrão que deve ser seguido pelas instâncias do *framework;*

- Diagrama de instanciação (diagrama de atividade da UML) é uma representação visual do processo de instanciação de um *framework.* Eles são representados utilizando a sintaxe definida pelo diagrama de atividade da UML. Cada estado de ação é usado para representar um *hot spot* de um *framework.* Todos os *hot spots* devem ser modelados em um diagrama de instanciação.

Percebe-se que o maior número de extensões para representar os *hot spots* se concentra no diagrama de classes. Nesse trabalho, pretende-se considerar os aspectos estruturais e não os comportamentais do *framework*. O somatório desses dois aspectos aliado ao fato de que o diagrama de classes é o principal da UML (vide item 2.2) justifica a opção desse trabalho de se ater ao diagrama de classes com os seus *tag values* propostos pela UML-F. O quadro abaixo resume os *tag values* utilizados nos diagramas da UML-F para representar os *hot spots* em um *framework:*

Nome da extensão	**Tipo da extensão**	**Uso**
{variable}	*Boolean Tag*	Identifica o *hot spot* método variável
{extensible}	*Boolean Tag*	Identifica o *hot spot* classes estendidas
{incomplete)	*Boolean Tag*	Identifica o *hot spot* interfaces

		estendidas
{appl-class}	*Boolean Tag*	Identifica classes que existem apenas na instanciação do *framework.*
{optional}	*Boolean Tag*	Diagrama de seqüência
{dynamic},{static}	*Boolean Tag*	Classifica os *hot spots* quanto ao tempo de execução.

Figura 2.2 – Quadro resumo baseado em [FPR2000]

2.5. XML

2.5.1. Características da XML

A XML (*Extensible Markup Language*) [W3C98] é um padrão aprovado pela W3C *(World Wide Web Consortium*) que deve se tornar um formato universal para a troca de informações na Web. XML é um subconjunto de SGML (*Standard Generalized Markup Language*) que, segundo [Mcg98], pretende tornar a SGML "leve" o suficiente para o uso na Web, ou seja, XML é uma versão simplificada da SGML. Da mesma forma que HTML (*Hypertext Markup Language)*, XML é uma linguagem de marcação nas quais seus documentos são compostos de uma mistura de dados e marcações. SGML é uma metalinguagem, isto é, uma linguagem que descreve outra linguagem. A própria HTML foi derivada da SGML. Segundo [Mcg98], SGML é um padrão muito poderoso e geral, mas à medida que esse poder aumenta, cresce sua complexidade.

Segundo [ABS99], existem três aspectos que diferem XML de HTML:

- Novas *tags* podem ser definidas à vontade;
- Estruturas podem ser aninhadas a profundidades arbitrárias;
- Um documento XML pode conter uma descrição opcional de sua gramática .

O primeiro aspecto torna a XML mais flexível no momento em que *tags* deixam de ser fixas e se tornam mais auto-descritivas. Pode-se assim atribuir nomes mais significativos ao conteúdo referenciado. Já a gramática descrita no terceiro aspecto pode ser usada pelas aplicações que usam XML e que necessitam validar a estrutura do documento. Esta descrição da gramática é chamada esquema e existem atualmente duas propostas: DTD e XML Schema.

Embora HTML seja a linguagem mais utilizada nas publicações eletrônicas, ela trata a informação de forma superficial na medida em que descreve a disposição de textos, imagens e controles em uma página e não o conteúdo. Ao contrário, XML foi projetada para descrever conteúdo, em vez de apresentação. A informação de apresentação em um documento XML é fornecida através da inclusão de folha de estilo (*stylesheet*). Essas folhas de estilo são especificadas em uma linguagem subsidiária da XML, por exemplo a XSL (*Extensible Stylesheet Language*).

Esta diferença agrega uma importante vantagem para XML em relação a HTML. Segundo [Suc98], a XML permite que a troca eletrônica de dados na Web seja legível para o computador, enquanto a HTML permite que a troca de documentos seja legível apenas para o homem. Com a XML há um ganho semântico pois os programas passam a entender a estrutura dos documentos disponíveis na Web, possibilitando assim pesquisas inteligentes das informações armazenadas.

Cada documento XML contém um ou mais elementos, sendo que os limites de cada elemento são delimitados por uma *tag* [1] de início e outra *tag* de fim (ex.: <nome> e </nome>). O elemento é o componente básico da XML. Segundo [ABS99], o texto entre a *tag* de início e a correspondente *tag* de fim, incluindo as *tags* embutidas, é chamada de elemento, enquanto as estruturas entre as *tags* são o conteúdo. O elemento pode possuir elementos aninhados denominados subelementos. A ordem de subelementos e partes de dados é relevante. Na figura 2.3, a *tag* <pessoa> é um subelemento de <população>.

```
<população>
        <pessoa nacionalidade=”brasileiro” >
                <nome>Pedro Bastos</nome>
                <idade>12</idade>
        </pessoa>
</população>
```

Figura 2. 3 - Exemplo de XML

Outro componente do documento XML é o atributo. De acordo com [W3C98], atributos são usados em XML para associar um par de nome e valor a um elemento. Os atributos podem ser usados para:

- Definir um conjunto de atributos pertencentes ao tipo de elemento dado;
- Estabelecer a restrições de tipos para esses atributos;
- Fornecer um valor *default* para atributos.

Na figura 2.3, nacionalidade é um atributo do elemento pessoa. De acordo com [ABS99] existem diferenças entre atributos e *tags*. O atributo pode ocorrer apenas uma vez dentro de uma *tag*, enquanto que subelementos com a mesma *tag* podem ser repetidos. Além disso, o valor associado com um atributo é uma *string*, enquanto a estrutura entre as *tags* pode conter subelementos.

No entanto, essa flexibilidade da XML traz consigo uma certa ambigüidade, pois uma informação pode ser representada como atributo ou subelemento dependendo da visão do projetista. Para minimizar essa situação, um documento XML deve seguir um conjunto básico de regras, referenciadas como um documento bem formado *(Well-Formed Documents)*. Estas regras consistem em *tags* que devem ser aninhadas corretamente e os atributos devem ser únicos [ABS99]. Segundo [Mic2000], um

[1] Segundo [ABS99], tag(s) são marcas combinantes que são balanceadas devendo ser fechadas em ordem inversa à aquelas em foram abertas. O texto entre as marcas é o conteúdo.

documento está bem formado se atender as seguintes regras: um único elemento raiz, *tags* devem ser aninhadas corretamente, consistência entre maiúsculo e minúsculo, elementos não podem sobrepor elementos, o valor de um atributo deve estar entre aspas e um elemento não pode possuir atributos repetidos.

Uma outra forma para assegurar maior confiabilidade a um documento XML é através da sua validação de acordo com restrições de uma determinada gramática (ex.: DTD), porém somente os documentos que possuírem essa gramática poderão ser validados.

2.5.2. Esquema da XML

Para que se possa validar um documento XML e assegurar que ele esteja correto gramaticalmente, deve-se definir de uma forma clara a estrutura desse documento, ou seja, o esquema do documento. Um esquema para um documento XML é basicamente um conjunto de regras predefinidas que descrevem um vocabulário XML. O esquema define o elemento que pode aparecer em um documento XML, tal como os atributos que podem ser associados a um dado elemento. [Mic2000].

Segundo [Car2001], a comunidade XML tem desenvolvido linguagens de esquema para representação do modelo dos documentos. A DTD (*Document Type Definition*) é a linguagem de esquema original incluída na especificação XML 1.0 (W3C98). Segundo [ABS99], a DTD é uma gramática livre de contexto para o documento, podendo servir também como esquema para os dados representados pelo documento em XML.

Embora a sintaxe de um DTD seja relativamente fácil de aprender e possibilite a utilização de *parsers* de validação eficientes e pequenos, a DTD é limitada para descrever esquemas de vocabulários orientados a dados usados em aplicações de comércio eletrônico B2B *(Business to Business),* pois existem as seguintes limitações [Car2001]:

- Os tipos que podem ser expressos usando DTDs não são tão variados quanto os tipos existentes em banco de dados tradicionais;
- Definição inadequada de identificadores usando ID e IDREF;
- Nenhuma restrição na distinção do tipo do elemento de destino de *links*: qualquer definição de ID é aceita;
- Suporte inadequado para restrição de multiplicidade de subelementos;
- Não suporta herança para elementos;
- Suporta somente um *namespace* para todo a DTD ou então deve-se colocar um prefixo explícito, pois a DTD não possui definição para suportar validação de documentos que combinem vocabulários de vários *namespaces*;
- Não suporta um nível de documentação além dos comentários no texto.

Uma proposta para linguagem de esquema alternativa ao DTD é a XML Schema, apresentada pelo grupo de trabalho W3C e recomendada para se tornar um padrão [W3C2001]. Ela apresenta vantagens sobre DTD e supera algumas das suas limitações. As principais características, segundo [Car2001], são:

- XML Schema são instâncias de documentos XML;
- Representação de vários tipos de dados e o refinamento dos tipos de dados;
- Extensão de tipo de elemento (herança);
- Modelo de conteúdo sem ordem;
- Declarações de elemento com escopo local;
- Suporte a *namespaces* XML.

Segundo [W3C99], a linguagem XML Schema pode ser usada para definir, descrever e catalogar vocabulários XML para classes de documentos XML. O papel da XML Schema é similar ao modo que os sistemas de banco de dados relacionais usam a DDL (*Data Definition Language)* da SQL (*Structured Query Language)* para criar esquemas para um conjunto particular de tabelas, atributos, tipos de atributos ou restrições em um banco de dados.

A substituição da DTD por XML Schema ou a coexistência das duas abordagens tem sido uma questão polêmica para o futuro dos dados na Web. Segundo [Car2001], a coexistência entre DTD e XML Schema parece ser a opção mais provável por um bom tempo. XML Schema disponibiliza muitas benefícios convincentes para o uso em aplicações de comércio eletrônico e provavelmente vai ser rapidamente adotada, mas existem outras razões convincentes, como seguem abaixo, para também se utilizar DTD:

- DTD são mais simples e permitem implementações menores e eficientes. A implementação completa da especificação XML Schema requer ferramentas de validação maiores e mais demoradas, o que pode não ser o ideal para todas as aplicações;
- DTDs são ainda uma boa forma para muitas aplicações orientadas a texto;
- Existe um grande e rápido crescimento da base instalada de vocabulários DTD, sendo que estas bases levariam um tempo para ser convertidas para XML Schema.

Para lidar com DTDs e XML Schemas, pode-se usar modelos UML como linguagem de especificação do vocabulário básico. Esta abordagem traz grandes benefícios porque a UML foi desenvolvida como uma linguagem que é independente da linguagem de implementação. Por outro lado, DTDs e XML Schemas são exemplos de linguagens de implementação usadas pelas ferramentas de desenvolvimento em XML.

O mapeamento de modelos UML para XML DTDs é abordado pelo padrão XMI (discutido na próxima sessão) e deve brevemente considerar o mapeamento para XML Schemas.

2.5.3. XMI

Segundo [OMG99b], o padrão XMI (XML *Metadata Interchange*) especifica uma estrutura para intercâmbio de modelos representados em XML. Para [CH98], o objetivo principal da XMI é permitir, em ambientes distribuídos e heterogêneos, a troca de metadados entre ferramentas e entre ferramentas e repositórios de metadados

baseados em MOF. De acordo com [CH98], a especificação XMI consiste basicamente dos seguintes itens:

- Um conjunto de regras de produção DTD XML para transformar metamodelos MOF em DTDs XML;
- Um conjunto de regras de produção de documentos XML para codificar e transferir metadados compatíveis com o MOF;
- Princípios de projeto para DTDs compatíveis com XMI;
- Princípios de geração de documentos XML compatíveis com XMI.

Segundo [OMG99b], as regras de produção podem ser aplicadas em um modelo de um sistema para gerar um documento XML e o inverso das regras pode ser usado para reconstruir o modelo a partir de um documento XML. Na prática, XMI efetua o mapeamento de metamodelos MOF para DTDs e de metadados MOF para documentos XML. Devido ao alinhamento entre a UML e MOF (descrito no item 2.2), XMI se torna uma maneira prática de fazer o mapeamento UML-XML. De acordo com [OMG99b], isso é possível pois a especificação da UML proposta pelo OMG define o meta-modelo UML como um meta-modelo MOF. Assim sendo, a proposta XMI constitui um formato de intercâmbio de modelos UML.

De acordo com [CH98], XMI especifica os requisitos que um DTD XML, chamado de DTD XMI, deve satisfazer, permitindo que a informação de um metamodelo contido em um DTD XMI possa ser verificado através de validações XML. O fato da validação XML poder ser feita por um processador XML tira essa responsabilidade dos programas de importação e exportação, facilitando assim o intercâmbio de dados.

Cada DTD XMI deve obedecer os seguintes requisitos, conforme [CH98]:

- Todos os elementos definidos pela especificação XMI devem ser declarados no DTD;

- Cada construção (classe, atributo ou associação) do metamodelo deve possuir uma declaração de elemento correspondente;
- Todos os elementos que representam extensões ao metamodelo devem ser declarados.

De acordo com [OMG99b], o projeto da proposta XMI seguiu onze mandamentos principais:

- Solução universalmente aplicável: XMI deve prover os meios para intercâmbio de metadados de qualquer meta-modelo MOF;
- Geração automática da sintaxe da transferência: XMI deve especificar a geração de uma sintaxe padrão de intercâmbio de modelos;
- Conformidade com os paradigmas XML: a proposta XMI deve seguir todos os princípios estabelecidos pela XML para o projeto de documentos e a XMI não deve mapear todos os tipos de dados do MOF em elementos distintos do XMI DTD;
- Conhecimentos dos meta-modelos: os lados envolvidos no intercâmbio precisam conhecer o meta-modelo MOF dos metadados envolvidos;
- Codificação completa dos metadados: partindo do pressuposto de que o receptor conhece o meta-modelo, deve ser possível a recuperação das fontes do metadados a partir somente dos documentos XMI;
- Precisão dos meta-modelos MOF: XMI assume que os meta-modelos MOF estão semanticamente corretos para a transmissão posterior dos metadados;
- Fragmentos do Modelos: XMI deve suportar o intercâmbio tanto de partes de um modelo como do modelo inteiro;
- Modelos incompletos ou pouco detalhado: XMI não requer que o modelo esteja completamente formado para que ocorra o intercâmbio. No entanto, o modelo deve estar semanticamente correto;
- Suporte a versões de um modelo;
- Extensibilidade de modelos: XMI permite a transmissão simultânea de metadados de um meta-modelo padrão e de uma ou mais extensões não padronizadas;

- MOF como um modelo de informação: XMI deve ser usada para transmitir tanto dados operacionais quanto metadados.

Uma das vantagens do XMI está na possibilidade de um intercâmbio mais fácil de modelos entre ferramentas, pois não é preciso compreender os detalhes das interfaces CORBA (*Common Object Request Broker Architecture*) definidas pelo MOF. Com XMI, não é preciso conhecer CORBA para trocar modelos. De acordo com [OMG99b], em um contexto mais amplo, XMI pode ser percebida como um formato de intercâmbio de metadados que é independente da tecnologia de *middleware*. Assim sendo, XMI torna-se uma alternativa às interfaces MOF que são dependentes do *middleware* CORBA.

2.5.4. XML como Intercâmbio de Informações

A necessidade de recuperação de dados armazenados na Web impulsiona constantemente o surgimento de novas tecnologias para tratar esses dados. XML pode ser vista como uma dessas tecnologias. A XML também pode ser uma opção para representar um modelo dos dados contidos na Web, porém, de acordo com [KH2000] , a XML não apresenta um nível de abstração mais alto para modelar documentos na Web. Verifica-se assim que o principal objetivo da XML é o intercâmbio de dados entre aplicações e não propriamente a modelagem dos mesmos.

Segundo [CH98], um software precisaria ser apenas capaz de exportar e importar os dados utilizados no formato XML para que conseguisse operar com outras ferramentas compatíveis com XML. A [OMG99b] especifica os quatros seguintes cenários onde o XML pode ser útil:

- Combinação de ferramentas em um ambiente heterogêneo;
- Cooperação através de definições comuns de metamodelos;
- Troca de experiências em um ambiente distribuído com facilidade de publicação e captura de informações;
- Promoção de padrões de projeto e reutilização.

A troca de informações através de XML pode ser particularmente útil entre equipes de desenvolvedores de software que possivelmente utilizam ferramentas diferentes. Baseando-se na XML, o XMI surgiu para resolver o problema da interoperabilidade de ferramentas através do fornecimento de um formato flexível de intercâmbio. A ferramenta precisa ser apenas capaz de gravar e carregar os dados em formato XMI para que possa se integrar com outras ferramentas compatíveis com XMI. Dessa forma, não existe a necessidade de implementar um utilitário separado de importação e exportação para cada combinação de ferramentas que trocam dados entre si.

De acordo com a [OMG99b], a extensão da informação trocada entre ferramentas é limitada pelo nível de compreensão existente entre as ferramentas. Se ambas as ferramentas compartilham o mesmo metamodelo, todas as informações transferidas podem ser entendidas e utilizadas. A capacidade de compartilhar metamodelos é um passo importante na promoção do uso de metamodelos comuns entre os fornecedores de ferramentas. Com o uso da UML-F, este intercâmbio atinge proporções ainda maiores, pois a possibilidade de reutilização de elementos genéricos descritos e documentados em *frameworks* é muito maior do que a chance de reaproveitamento de esquemas específicos.

3. Mapeamento UML- XML

Esse capítulo apresenta várias formas de efetuar o mapeamento da UML para XML. Como essa dissertação enfatiza o uso de *frameworks* modelados em UML-F, é necessário conhecer como os construtores orientados a objetos podem ser representados em uma linguagem de marcação, no caso a XML.

Uma das questões pertinentes do mapeamento é a diferença semântica existente entre o domínio da UML e o da XML. A utilização da UML pode trazer a semântica desejada aos vocabulários em XML, pois a maioria dos vocabulários em XML são incompletos, restringindo-se a uma lista de termos. Segundo [Car2001], um diagrama de classes UML pode ser construído para representar visualmente os elementos, relacionamentos e restrições de um vocabulário XML.

Um vocabulário é uma lista de termos usada para comunicação, porém um vocabulário básico não define sua semântica. A semântica especifica restrições sobre como cada termo pode ou não pode ser usado nas combinações com outros termos. A semântica também especifica uma taxonomia ou uma estrutura de classificação para conceitos abstratos e especialização de termos.

A especificação de DTD XML não foi criada com base nos conceitos da orientação a objetos. Devido a este fato, não existe mapeamento direto das estruturas UML para DTD e portanto um conjunto de regras deve ser aplicado para realizar este mapeamento. Estas regras foram definidas na especificação XMI [OMG99b].

A figura 3.1 ilustra o mapeamento do diagrama de classes em um esquema XML e do diagrama de objetos em um documento XML.

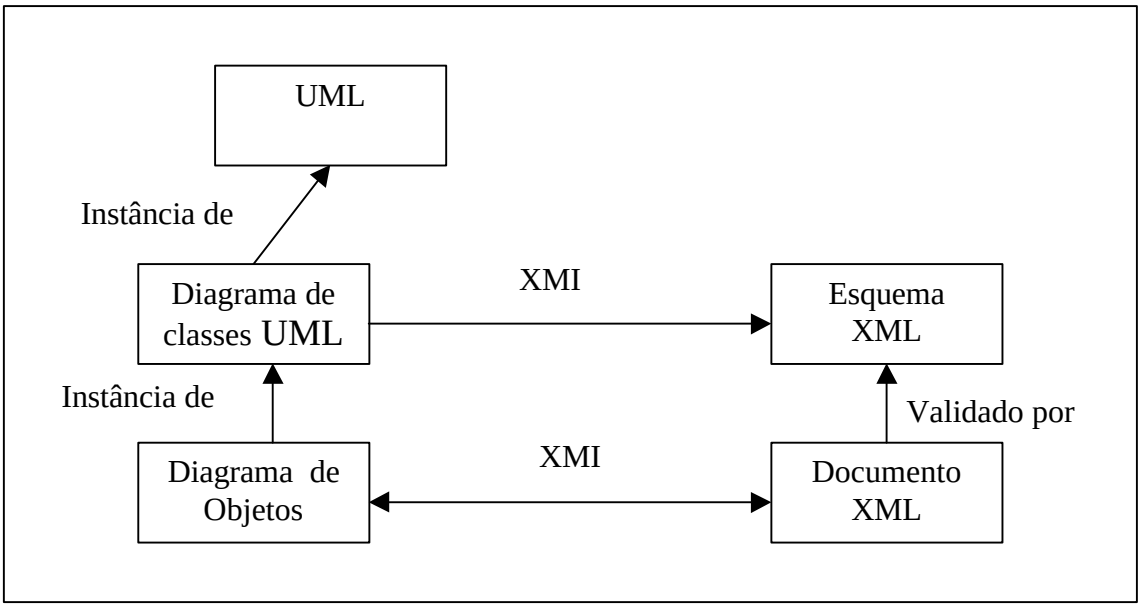

Figura 3. 1 – Mapeamento de diagramas UML para XML [Car2001]

Segundo [Sko99], as regras de conversão de esquemas (mapeamento do diagrama de classes para o esquema XML) só podem ser aplicadas em um único sentido devido às limitações do DTD. Já as regras de conversão de instâncias (mapeamento do diagrama de objeto para o documento XML) podem ser aplicadas nos dois sentidos, permitindo a reconstrução das instâncias a partir do documento XML. A especificação original da XMI [OMG99b] também destaca o fato de que as regras de produção de DTD são aplicadas em um sentido único, enquanto que as regras de produção de um documento XML podem ser utilizadas nos dois sentidos.

Na seção 3.1 será apresentado o mapeamento do diagrama de objeto (instância de um diagrama UML) para um documento XML. Na seção 3.2, discute-se o mapeamento do diagrama de classes para o esquema XML. Nas seções 3.1 e 3.2, discute-se essencialmente a proposta de mapeamento de [Car2001], pois entre os trabalhos pesquisados foi a proposta que apresentou uma especificação mais consistente e sistematizada do mapeamento. Além disso, o trabalho de [Car2001] é recomendado e apoiado pelos criadores da UML: Booch, Jacobson e Rumbaugh. Na seção 3.3, são apresentadas abordagens de outros autores para o mapeamento.

3.1. Mapeamento de Objetos UML para Documentos XML

Para ilustrar o mapeamento, será adotado o seguinte exemplo de uma aplicação bancária.

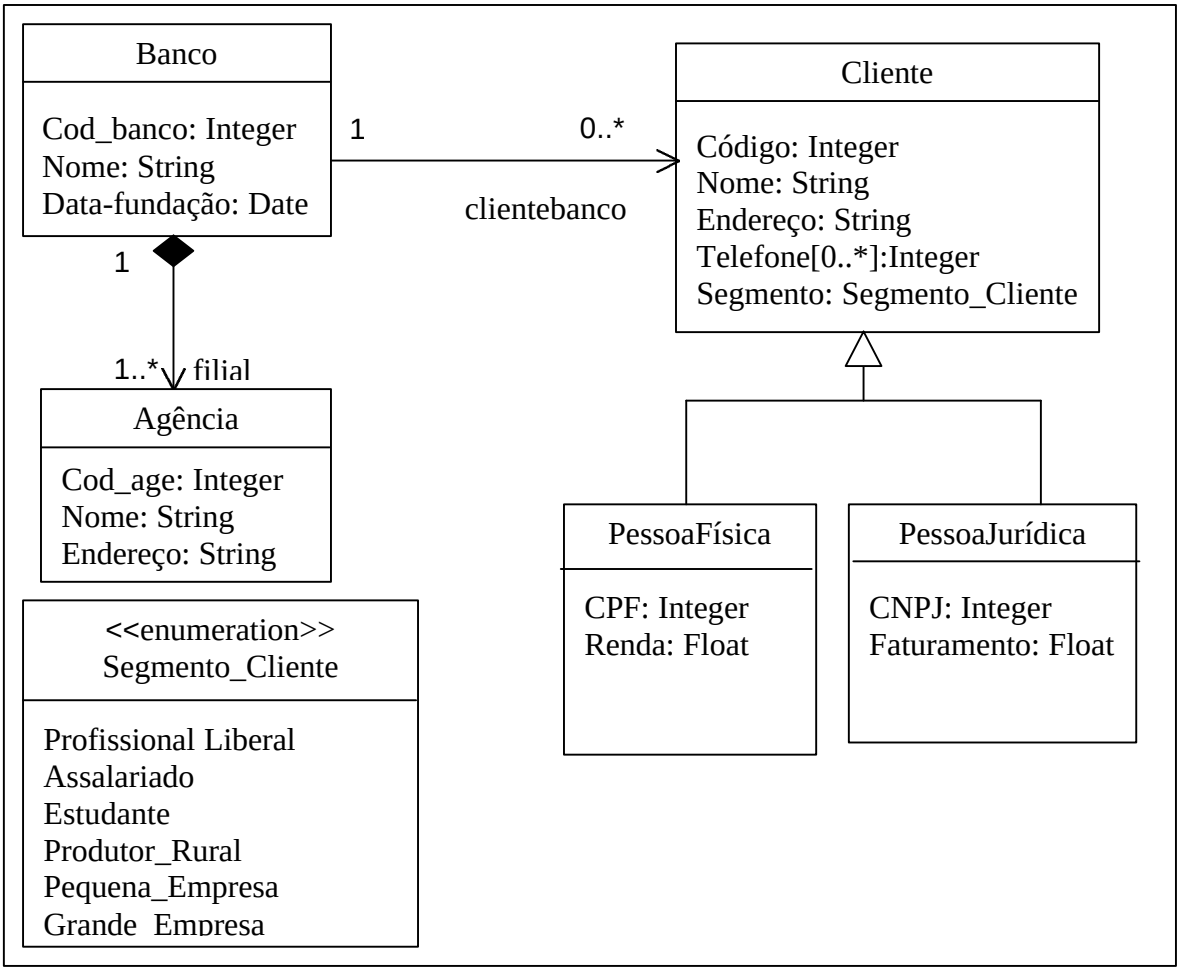

Figura 3. 2 – Diagrama de classes simplificado de uma aplicação bancária

Inicialmente, pretende-se detalhar o mapeamento do diagrama de objetos que é uma instância do diagrama representado na figura 3.2. O diagrama de objetos é representado abaixo:

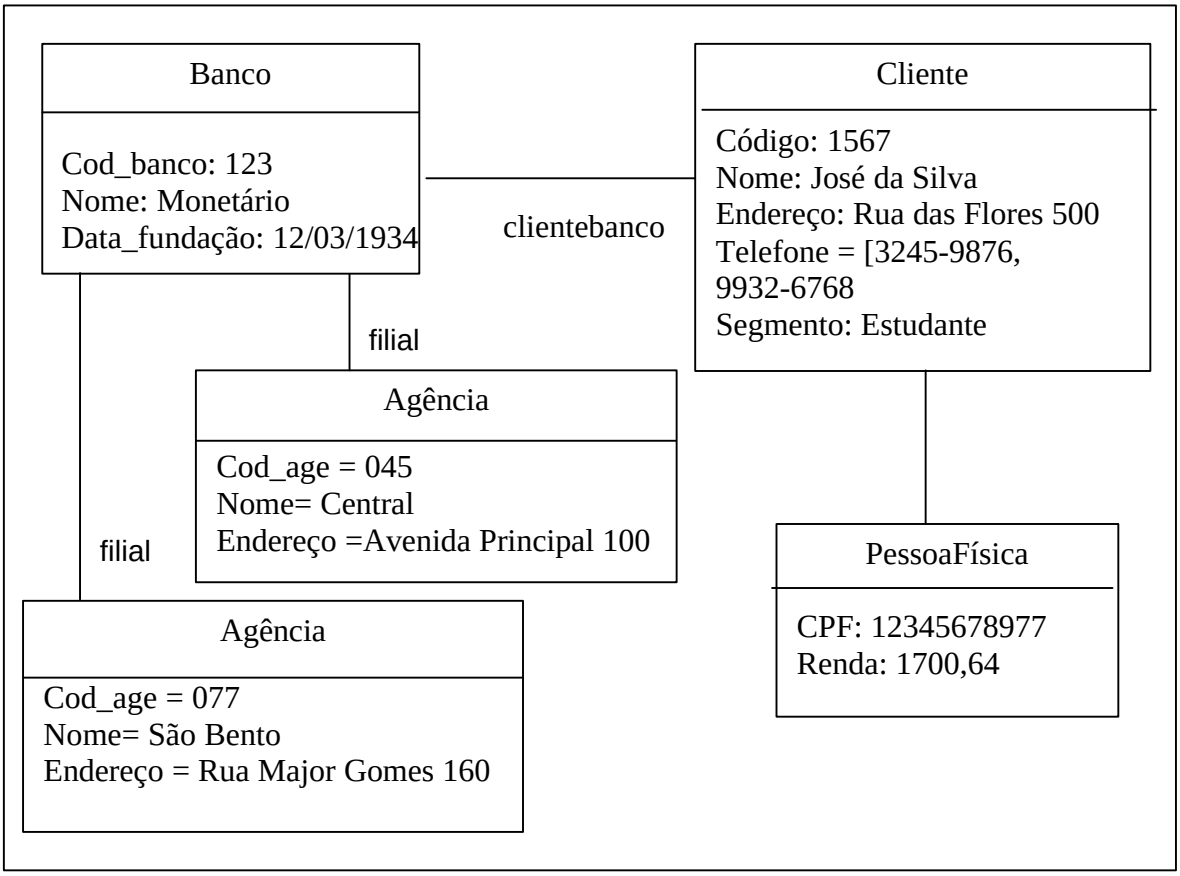

Figura 3. 3 – Diagrama de objetos de um sistema bancário

Para se mapear um diagrama UML para um documento XML, foram identificados os seguintes construtores: classes, herança, atributos, composições e associações.

3.1.1. Classes UML para Elementos XML

Segundo [Car2001], uma classe UML possui características estruturais e comportamentais (métodos), mas apenas os aspectos estruturais são relevantes na definição do vocabulário. As características estruturais englobam os atributos, relacionamentos com nomes dos papéis e as composições. Apesar das composições e dos relacionamentos serem instrumentos independentes de associação, esses são tratados como parte da definição estrutural da classe na qual estão vinculados para facilitar o mapeamento em documentos XML.

Cada instância de uma classe UML (objeto) produz um elemento XML, sendo que o elemento XML funciona como um agrupamento de atributos e associações. No DTD, o nome do tag é o mesmo da classe, respeitando restrições da XML. *Namespaces* também podem ser utilizados. Baseando-se na figura 3.3, o objeto Banco seria representado por uma *tag* de início e uma *tag* de fim.

```
<Banco>
   <Banco.codigo>123</Banco.codigo>
   <Banco.nome>Banco Monetário</Banco.nome>
 ........
</Banco>
```

3.1.2. Herança

A XML padrão não tem mecanismos embutidos para representar a herança. Um DTD não pode representar herança entre elementos. Para solucionar este problema, o padrão XMI especifica o uso de herança por meio de cópia de atributos, associações e agregações. Assim sendo, todos os atributos da super classe aparecem na subclasse, com um prefixo igual ao nome da super classe, conforme o seguinte exemplo:

```
<PessoaFísica>
    <Cliente.código>1567</Cliente.código>
    <Cliente.nome>José da Silva</Cliente.nome>
    <Cliente.endereço>Rua das Flores 500</Cliente.endereço>
    <Cliente.telefone>3245-9876</Cliente.telefone>
    <Cliente.telefone>9932-6768</Cliente.telefone>
    <PessoaFísica.CPF>12345678977</PessoaFísica.CPF>
    <PessoaFísica.Renda>1700</PessoaFísica.Renda>
 ......
</PessoaFísica>
```

3.1.3. Atributos UML para XML

Existem duas possibilidades de mapear atributos de uma classe UML para XML. A primeira compreende na criação de um elemento XML e a segunda de um atributo XML.

Na primeira opção, um atributo UML é mapeado para um elemento XML cujo nome é feito a partir do nome do atributo prefixado com o nome da classe UML. Como não existe representação para atributos multi-valorados em XML, os atributos multi-valorados só podem ser mapeados para elementos XML. O exemplo do item 3.1.2 ilustra esse mapeamento. Nesse exemplo, o atributo telefone é o campo multi-valorado.

Já na segunda opção, um atributo UML pode ser mapeado para um atributo XML. Isso deve ser feito principalmente quando se tratar de tipos primitivos de dados (ex.: *strings*) e tipos enumerados. O uso de atributos XML não é apropriado quando se trata de *strings* extensas. Não é necessário prefixar o atributo XML com o nome da classe UML, pois cada elemento XML define um espaço de nome para seus atributos. No entanto, o valor de cada atributo deve aparecer entre aspas duplas e não pode haver repetição de nomes de atributos dentro de um mesmo espaço de nome. A seguir, tem-se um exemplo do mapeamento de atributos UML para atributos XML:

```
<PessoaFísica código=”1567” nome=”José da Silva” CPF=”12345678977”>
    <Cliente.endereço>Rua das Flores 500</Cliente.endereço>
    <Cliente.telefone>3245-9876</Cliente.telefone>
    <Cliente.telefone>9932-6768</Cliente.telefone>
    <PessoaFísica.Renda>1700</PessoaFísica.Renda>
  ......
</PessoaFísica>
```

Nesse exemplo, os atributos código, nome e CPF foram mapeados em atributos XML. O atributo endereço foi mapeado como elemento por ser uma *string* extensa. Como os atributos em XML não são ordenados, não há como obrigar que o código seja o primeiro atributo.

A especificação da XMI não oferece nenhuma diretriz na escolha de atributo ou elemento XML para representar atributos UML. Devido à falta de critério no uso de atributos ou elementos, surgem dificuldades na interpretação dos documentos. Assim sendo, é aconselhável ser coerente no emprego de atributos ou elementos em um determinado vocabulário.

3.1.4. Atributos Enumerados para XML

Um atributo do tipo enumerado requer que o valor do atributo UML seja obtido de uma lista finita de valores válidos. O padrão XMI especifica duas alternativas de mapeamento de valores enumerados UML para documentos XML.

A primeira alternativa consiste no uso de atributos que aparecem no documento XML da mesma forma que atributos não enumerados, conforme exemplo abaixo:

```
<PessoaFísica segmento=”Estudante”>
    <Cliente.código>1567</Cliente.código>
    <Cliente.nome>José da Silva</Cliente.nome>
   ....
</PessoaFísica>
```

Nesse caso, o DTD se encarrega de validar o conteúdo do atributo, permitindo somente um dos valores válidos. A segunda alternativa envolve a criação de um elemento que possui o atributo xmi.value cujo valor é restringido pelo DTD, de acordo com o seguinte exemplo:

```
<PessoaFísica>
    <Cliente.código>1567</Cliente.código>
    <Cliente.nome>José da Silva</Cliente.nome>
    <Cliente.segmento xmi.value=”Estudante”/>
   ....
</PessoaFísica>
```

3.1.5. Composição UML (Agregação Composta) para XML

Uma instância da composição UML é mapeada para um elemento XML, que é incluído como um subelemento XML. Se o elemento “todo” é excluído, os subelementos “parte” são também excluídos. Em uma composição, o papel é criado como um elemento pertencente ao elemento “todo”. O papel de uma agregação é mapeado da mesma forma que os atributos UML são mapeados para elementos XML, conforme o exemplo a seguir:

```
<Banco>
   <Banco.filial>
```

```
    <Agência código=”045” nome=”Central”>
        <Agência.endereço>Avenida Principal 100</Agência.endereço>
    </Agência>
    <Agência código=”077” nome=”São Bento”>
        <Agência.endereço>Rua Major Gomes 160</Agência.endereço>
    </Agência>
    .....
    </Banco.filial>
</Banco>
```

Essa abordagem restringe que os elementos (todo) e subelementos (parte) estejam no mesmo documento XML. Seria mais flexível permitir a distribuição de elementos e subelementos em vários documentos.

3.1.6. Associações UML para XML

Uma associação UML pode ser mapeada tanto para um elemento quanto para um atributo XML, porém o elemento referenciado deve estar dentro do mesmo documento. A referência é implementada através da introdução do atributo *xmi.id* para identificar o elemento representativo da classe de multiplicidade 0 ou 1 e também para fazer associação entre elementos. No entanto, o *id* deve ser único ao longo do escopo do documento.

No mapeamento da associação UML para elemento XML, deve-se observar as seguintes características :

- Introdução do atributo *xmi.idref* para referenciar um *xmi.id*, ou seja, outro elemento contido no mesmo documento;
- Introdução, caso necessário, do atributo *xmi.label* que pode conter a descrição ou título do elemento referenciado. O *xmi.label* é utilizado para fins de documentação;
- O papel de uma associação é utilizado para nomear um elemento representativo da classe de multiplicidade muitos.

Um exemplo dessa opção de mapeamento segue abaixo:

```
<Cliente código=”1567”>
```

```
    <Cliente.nome>José da Silva</Cliente.nome>
    <Cliente.endereço>Rua das Flores 500</Cliente.endereço>
    <Cliente.telefone>3245-9876</Cliente.telefone>
    <Cliente.telefone>9932-6768</Cliente.telefone>
    <Cliente.clientebanco>
      <Banco xmi.idref =”123” xmi.label =”Monetário” />
    </Cliente.clientebanco>
</Cliente>

<Banco xmi.id =”123”>
   <Banco.Nome>Monetário</Banco>
   <Banco.Data_fundacao>12/03/1934</Banco>
</Banco>
```

Já na alternativa de mapeamento da associação UML para atributos XML, o nome do papel da associação é usado como atributo XML e o valor desse atributo é associado ao valor do elemento *xmi.id.* Segue-se abaixo um exemplo onde essa alternativa é ilustrada:

```
<Cliente código=”1567” clientebanco=”123”>
    <Cliente.nome>José da Silva</Cliente.nome>
    <Cliente.endereço>Rua das Flores 500</Cliente.endereço>
    <Cliente.telefone>3245-9876</Cliente.telefone>
    <Cliente.telefone>9932-6768</Cliente.telefone>
</Cliente>

<Banco xmi.id =”123”>
   <Banco.Nome>Monetário</Banco>
   <Banco.Data_fundacao>12/03/1934</Banco>
</Banco>
```

3.1.7. Partes do Modelo UML para XML

Um modelo UML nem sempre apresenta uma classe que é a raiz da hierarquia de todas as outras classes, porém um documento XML precisa conter exatamente um elemento raiz. O padrão XMI resolve esse problema com a definição de um elemento *<XMI>* que inclui adicionalmente metadados sobre o conteúdo do documento e suporta a troca de partes e incrementos do modelo. Esse elemento *<XMI>* possui um subelemento *<XMI.header>* que contém a documentação e o subelemento *<XMI.content>* onde o conteúdo do modelo é representado.

```
<XMI xmi.version=”1.1”
  <XMI.header>
    <XMI.documentation>
      <XMI.shortDescription> Controle de Clientes Bancários</XMI.shortDescription>
    </XMI.documentation>
    <XMI.metamodel xmi.name=”Modelo bancário genérico”/>
    <XMI.model xmi.name=”Modelo do banco Monetário”/>
  </XMI.header>
  <XMI.content>
    <Cliente código=”1567” clientebanco=”123”>
       <Cliente.nome>José da Silva</Cliente.nome>
        .....
    </Cliente>
    <Banco xmi.id =”123”>
     <Banco.Nome>Monetário</Banco>
       ....
    </Banco>
  </XMI.content>
</XMI>
```

3.1.8. Pacotes UML em *Namespaces* XML

Segundo [BRJ98], em UML, os agrupamentos que organizam um modelo são chamados de pacotes. Um pacote é um mecanismo de propósito geral para organizar elementos em modelos, facilitando assim a compreensão. Os pacotes permitem controlar o acesso aos seus conteúdos, estabelecendo as interfaces existentes entre os sistemas.

A UML permite definir uma hierarquia de pacotes que especificam um *namespace* para os elementos de modelagem. Os elementos contidos em um *namespace* devem possuir nomes diferentes, embora dois elementos possam ter o mesmo nome se pertencerem a diferentes pacotes. No exemplo utilizado acima, a classe Cliente pode aparecer no pacote de Sistema Bancário e no pacote Seguros.

Um dos problemas da especificação original da XML é que essa não incluía uma forma de combinar elementos e atributos de vocabulários diferentes em um mesmo documento. Os *namespaces* da XML foram criados para resolver esse problema, permitindo a reutilização de vocabulários padrões e uso de diferentes vocabulários em um mesmo documento. De acordo com [W3C99], os *namespaces* XML fornecem um

método simples para qualificar nomes de elementos e atributos através da associação desses com *namespaces* identificados por referências URI *(Uniform resource identifier).* Segundo [Car2001], uma URI representa um nome global de recursos acessado através de um serviço de diretórios, enquanto que uma URL referencia um arquivo particular ou uma localização de dados. Assim sendo a URI é um super conjunto dos identificadores de recursos da Internet, que inclui a URL.

Um *namespace* XML consiste em um prefixo aplicado aos atributos ou/e elementos de um vocabulário. A utilização do prefixo *namespace* XML permite a integração de termos (atributos e elementos) de diversos vocabulários em um único documento.

3.2. Mapeamento do Diagrama de classes UML para esquema XML

Um problema encontrado no mapeamento do diagrama de classes UML para esquema XML é que não existe uma maneira padrão de validar documentos XML que combinem elementos de vários DTDs. Um DTD contém apenas um *namespace*, como foi detalhado no item 2.5.2, para todos os elementos que ele define. A solução para essa limitação deve ser resolvida através das regras de mapeamento do modelo UML para DTD.

Apesar de ainda não fazer parte da especificação XMI padronizada pelo OMG, as regras de mapeamento do modelo UML para o XML Schema garantem um nível muito mais exato de representação do modelo em relação ao mapeamento para DTD. O XML Schema permite uma representação mais coerente das declarações de tipos de dados usadas nos elementos das classes UML.

Na figura 3.1, verifica-se que o diagrama de classes da UML é mapeado para um esquema XML com base no padrão XMI [OMG99b]. Dependendo das regras de mapeamento serem mais restritas ou menos restritas, os esquemas gerados podem ser restritos ou relaxados. Um DTD ou um XML Schema restrito é respectivamente um

subconjunto do padrão DTD ou Schema relaxado. Os estereótipos da UML podem ser usados para o controlar o nível de restrição de um dado esquema.

Nas próximas seções, serão apresentadas as combinações desses dois esquemas com as respectivas opções de restrição (relaxado/restrito).

3.2.1. Mapeamento do diagrama de classes UML para DTD relaxado

As características gerais de DTD relaxado são:

- Permissão da troca de partes de um modelo sem restrições. Pode-se assim enviar inicialmente uma parte do modelo e depois enviar seus complementos;
- Modelo do conteúdo desordenado não impondo restrição na geração de documentos. Dessa forma, os subelementos podem aparecer em qualquer ordem dentro de um elemento;
- Escolha livre de elemento ou atributo no mapeamento UML para estrutura XML.

É importante destacar que as regras XMI especificam que tanto atributos quanto elementos XML podem ser produzidos para representar atributos de classe UML e papéis de associação. A produção do DTD relaxado envolve inicialmente, para cada classe UML, a geração de um elemento XML para cada atributo e papel de associação correspondente à classe mapeada.

Posteriormente, cada classe do modelo UML produz um único elemento XML cujo modelo do conteúdo inclui todas as representações de papéis de associação e de atributos da classe UML. Esse elemento XML deve seguir um modelo de conteúdo do tipo escolha (nomes do elementos separados pelo “| “) e o indicador de multiplicidade (“*” no final do grupo de escolha) com isso os elementos podem aparecer diversas vezes. O modelo de conteúdo do tipo escolha não permite mapear para o DTD relaxado as restrições de multiplicidade definidas no diagrama de classes UML.

Por fim, para cada atributo e papel de associação de uma classe UML são produzidos os atributos XML. O resultado disso é que se atinge um alto nível de flexibilidade, permitindo tanto elementos quanto atributos em qualquer ordem e quantidade. Pode-se assim entender o porquê do uso do termo DTD relaxado.

Segundo [BRJ99], papéis e navegações são adornos opcionais nas associações. No entanto, o uso desses recursos enriquece semanticamente o diagrama de classes e orienta o processo de mapeamento do diagrama de classes para esquemas XML.

Na UML, quando uma classe participa de uma associação, ela tem um papel específico a desempenhar nesse relacionamento. Um papel é a face que a classe próxima a uma das extremidades da associação apresenta à classe encontrada na outra extremidade.

Já a navegação é um recurso opcional que especifica a direção prioritária da associação. Quando a navegação não está presente, assume-se que a navegação é bidirecional, permitindo navegar de objetos de um tipo até objetos de outro tipo e vice-versa. Em alguns casos, é interessante limitar a navegação em uma única direção, pois a navegação declara a direção mais eficiente a ser seguida. É importante destacar que a especificação da direção a ser seguida não significa necessariamente que não será possível navegar dos objetos encontrados em uma extremidade até os objetos da outra extremidade.

Para facilitar o mapeamento, [Car2001] utiliza papéis e navegações nos diagramas de classes. O sentido da navegação influencia o mapeamento, pois a classe origem da navegação possuirá o nome do papel como atributo ou elemento de acordo com a opção entre DTD relaxado ou restrito.

Os seguintes critérios também devem ser observados na geração do DTD relaxado:

- No caso de uma classe UML com estereótipo <<*enumeration*>>, devem ser produzidos um elemento e um atributo que descrevam a lista enumerada de valores;
- Quando aparecer herança no diagrama de classe UML, todos os atributos e papéis da associação vinculados à superclasse devem ser copiados para a subclasse, pois os DTDs não suportam a herança;
- Como DTD não suporta a declaração de vários *namespaces*, um modelo UML pode ser representado com um único ou sem nenhum *namespace;*
- A unicidade de um nome de elemento ao longo de um DTD deve ser garantida através do uso do prefixo indicativo do nome da classe;
- Não existe suporte para tipo de dados no DTDs. Permite-se apenas o uso de CDATA para atributos e #PCDATA para elementos;
- ID e IDREF são utilizados para ligações dentro de um mesmo documento, enquanto que o atributo href é usado para associações externas com outros documentos.

Aplicando esses critérios ao exemplo da figura 3.2, produz-se a seguinte definição de DTD:

Classe Cliente

```
<!ELEMENT Cliente.codigo (#PCDATA | XMI.reference)*>
<!ELEMENT Cliente.nome (#PCDATA | XMI.reference)*>
<!ELEMENT Cliente.endereco (#PCDATA | XMI.reference)*>
<!ELEMENT Cliente.telefone (#PCDATA | XMI.reference)*>
<!ELEMENT Cliente.segmento EMPTY>
<!ATTLIST Cliente.segmento xmi.value (Profissional Liberal | Assalariado | Estudante |
Produtor_Rural | Pequena_Empresa | Grande_Empresa) #REQUIRED>

<!ELEMENT Cliente (Cliente.codigo | Cliente.nome | Cliente.endereco |
Cliente.telefone |
Cliente.segmento | XMI.reference) * >

<!ATTLIST Cliente
   codigo CDATA   #IMPLIED
   nome CDATA     #IMPLIED
   endereco CDATA #IMPLIED
   telefone CDATA #IMPLIED
```

```
    segmento (Profissional Liberal | Assalariado | Estudante | Produtor_Rural |
                  Pequena_Empresa | Grande_Empresa) #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
```

O termo #IMPLIED na especificação do DTD significa que o atributo é opcional. No exemplo acima, o atributo Segmento é um tipo enumerado. A seguir, tem-se um novo exemplo onde a característica da herança é apresentada.

Classe PessoaFísica

```
<!ELEMENT PessoaFísica.CPF (#PCDATA | XMI.reference)*>
<!ELEMENT PessoaFísica.renda (#PCDATA | XMI.reference)*>

<!ELEMENT PessoaFísica (Cliente.codigo | Cliente.nome | Cliente.endereco |
Cliente.telefone |  Cliente.segmento | PessoaFísica.CPF | PessoaFísica.renda  |
XMI.reference) * >

<!ATTLIST PessoaFísica
    codigo  CDATA    #IMPLIED
    nome CDATA       #IMPLIED
    endereco CDATA  #IMPLIED
    telefone CDATA   #IMPLIED
    segmento (Profissional Liberal | Assalariado | Estudante | Produtor_Rural |
                  Pequena_Empresa | Grande_Empresa) #IMPLIED
    CPF CDATA    #IMPLIED
    renda CDATA  #IMPLIED
    %XMI.element.att;
    %XMI.link.att;
>
```

3.2.2. Mapeamento do diagrama de classes UML para DTD restrito

As características gerais de DTD restrito são as seguintes:

- Restrição de multiplicidade de acordo com o modelo UML;
- Exigência de modelo de conteúdo ordenado (seqüência) para fazer cumprir a multiplicidade;
- Atributos XML não são gerados, somente elementos XML são produzidos.

Devido às limitações do DTD, a multiplicidade só pode ser garantida através de um modelo de conteúdo do tipo seqüência ordenada (nomes do elementos separados por “,“). A multiplicidade refere-se aos atributos e às associações. A multiplicidade de um atributo diz respeito ao número de vezes que um atributo pode ser informado em uma classe (ex.: atributo multivalorado), enquanto que [BJR98] definem multiplicidade de associação como sendo a quantidade de objetos que podem ser conectados pela instância de uma associação. O modelo de conteúdo do tipo escolha não exige a ordenação, mas também não especifica a multiplicidade.

Os seguintes critérios também devem ser observados na geração do DTD restrito:

- Mapeamento de *namespace*, a unicidade de um nome de elemento, a herança, os tipos de dados e ligações são tratados da mesma maneira que no DTD relaxado;
- As restrições de multiplicidade são garantidas através de caracteres do DTD: espaço (exatamente uma instância), ? (zero ou uma instância), * (zero ou mais) e + (uma ou mais). Adota-se o default [1..1] para os atributos UML sem multiplicidade especificada;
- A ordem dos elementos deve ser obedecida com os atributos das classes UML aparecendo primeiro, sendo seguidos pelos papéis de associação da mesma classe.

Aplicando-se as regras do DTD restrito ao exemplo do item 3.2.1 produz-se uma nova especificação da seguinte maneira:

Classe Cliente

```
<!ELEMENT Cliente.codigo (#PCDATA | XMI.reference)*>
<!ELEMENT Cliente.nome (#PCDATA | XMI.reference)*>
<!ELEMENT Cliente.endereco (#PCDATA | XMI.reference)*>
<!ELEMENT Cliente.telefone (#PCDATA | XMI.reference)*>
<!ELEMENT Cliente.segmento EMPTY>
<!ATTLIST Cliente.segmento xmi.value (Profissional Liberal | Assalariado | Estudante |
Produtor_Rural | Pequena_Empresa | Grande_Empresa) #REQUIRED>
```

<!ELEMENT Cliente (Cliente.codigo, Cliente.nome, Cliente.endereco, Cliente.telefone*, Cliente.segmento, XMI.extension?) ? >

<!ATTLIST Cliente
    %XMI.element.att;
    %XMI.link.att;
>

Classe PessoaFísica

<!ELEMENT PessoaFísica.CPF (#PCDATA | XMI.reference)*>
<!ELEMENT PessoaFísica.renda (#PCDATA | XMI.reference)*>

<!ELEMENT PessoaFísica (Cliente.codigo, Cliente.nome, Cliente.endereco, Cliente.telefone*, Cliente.segmento, PessoaFísica.CPF, PessoaFísica.renda, XMI.extension?) ? >

<!ATTLIST PessoaFísica
    %XMI.element.att;
    %XMI.link.att;
>

3.2.3. – Mapeamento do diagrama de classes UML para XML Schema relaxado

O XML Schema relaxado é concebido de forma que uma instância válida de documento em XML Schema relaxado seja também válida em um DTD relaxado. Assim sendo, as características válidas para XML Schema relaxados são praticamente idênticas aos critérios do DTD relaxado, conforme se verifica abaixo:

- Implementação da multiplicidade através de *minoccurs = 0* (número mínimo de ocorrências) para todos os elementos do esquema;
- Modelo de conteúdo não ordenado (definido por *complex type* e grupo <*all*>)que não impõe nenhuma restrição na geração de documentos;
- Escolha livre de elemento ou atributos no mapeamento da UML para XML Schema.

A única diferença em relação aos DTDs relaxados é que as restrições de multiplicidade permitem que cada elemento seja opcional, mas obrigam que restrição de ocorrência máxima seja especificada.

Cada classe no modelo UML produz um conjunto padrão de declarações *element* do XML Schema, definições *complex type* e definições *attribute group*.

A produção do XML Schema relaxado envolve inicialmente a geração de uma declaração *element* com o nome da classe. Posteriormente, a definição *complex type* é produzida para a classe, contendo três partes: um elemento para cada atributo da classe UML, um elemento para cada papel de associação e um grupo de atributos que referencia a classe e os atributos padronizados XMI. Por fim, um *attribute group* define os atributos XML permitidos no elemento.

Os seguintes critérios também devem ser observados na geração do XML Schema relaxado:

- A unicidade de nome de elemento e as ligações são tratadas da mesma maneira que em DTDs relaxados;
- Ao contrário dos DTDs, XML Schema suporta vários *namespaces* em um mesmo documento XML;
- XML Schema suporta herança através do elemento *complex type* com um subelemento *extension* cujo atributo base referencia a superclasse;
- Permite a utilização de tipos de dados primitivos e a definição de tipos de dados pelo usuário. Os tipos de dados XML são associados com os tipos de dados UML correspondentes. Quando não existir o mapeamento direto, podem ser especificadas regras de compatibilidade entre os tipos UML e XML Schema.

3.2.4. Mapeamento do diagrama de classes UML para XML Schema restrito

Da mesma forma que os DTDs restritos, os XML Schemas restritos são definidos para serem um subconjunto da forma relaxada. No entanto, um XML Schema restrito não necessariamente validará um documento que foi validado por um DTD restrito, porque um XML Schema restrito é capaz de fazer cumprir a multiplicidade do modelo UML sem exigir um modelo de conteúdo ordenado.

As características gerais de um XML Schema restrito são as seguintes:

- Multiplicidade é restringida de acordo com o modelo UML;
- Modelo de conteúdo desordenado é permitido através do grupo de elemento <*all*>;
- Apenas elementos XML são gerados.

Entre as quatro combinações apresentadas (DTD relaxado, DTD restrito, XML Schema relaxado e XML Schema restrito) o XML Schema restrito é aquele que permite o mapeamento mais próximo do modelo UML original.

Os seguintes critérios também devem ser observados na geração do DTD restrito:

- Mapeamento de *namespace*, a unicidade de um nome de elemento, a herança, os tipos de dados e ligações são tratados da mesma maneira que no XML Schema relaxado;
- As restrições de multiplicidade são garantidas através dos atributos *minoccurs* e *maxoccurs* (máximo número de ocorrências) em todas as declarações de elementos.

3.3. Outras Abordagens de Mapeamento

Na literatura analisada, foram encontradas propostas de mapeamento da UML para esquemas XML (DTD ou XML Schema) e também o sentido contrário, ou seja, de esquemas XML para UML . Nas propostas abaixo, percebe-se que o uso da UML ocorreu devido ao fato dos seus diagramas serem um modo de modelar e visualizar esquemas XML em um alto nível de abstração e por ser uma linguagem de modelagem de dados familiar para muitos desenvolvedores e usuários de sistemas. Além disso, a UML é suficientemente robusta para expressar qualquer tipo de documento ou esquema XML. Apesar disso, essas propostas apresentam limitações no tratamento da UML-F, que é o objeto de mapeamento dessa dissertação. Essas propostas serão detalhadas abaixo.

O padrão XMI [OMG99b], descrito no item 2.5.3, é uma proposta da OMG para a geração automática de DTDs para um metamodelo compatível com o MOF, bem como para a geração de documentos XML a partir de metadados que estejam em conformidade com o metamodelo.

Já a abordagem de mapeamento de [Car2001] descrita nas seções anteriores utiliza os mecanismos de extensão da UML para propor perfis UML (UML *profile*), orientando assim os *parsers* na geração dos esquemas XML (DTDs e XML Schema). Nessa proposta, a maioria dos nomes de estereótipos são derivados de construtores básicos da especificação do XML Schema tais como: *Schema, Complextype e Simpletype*. Constata-se que os mecanismos de extensão (ex.: estereótipos) não são objetos do mapeamento nessa proposta, sendo apenas instrumentos internos para guiar o *parser* na geração de XML Schema.

Uma terceira abordagem é apresentada por [CSF2000], que propõem a transformação de diagramas de classes UML em DTDs. Os autores estendem a UML através do uso de estereótipos que orientam o mapeamento da UML para os construtores do DTD. Dessa forma são agregados na UML conceitos dependentes da implementação em DTD. [CSF2000] destacam que os conceitos da UML são bastante próximos do mundo real e por causa disso melhoram o processo de projeto de um DTD e facilitam o entendimento da semântica do DTD. Futuramente, os autores pretendem adaptar a proposta apresentada para XML Schema.

[Car2001] e [CSF2000] trabalharam no domínio específico de linguagem de marcação (XML), agregando informações dessa linguagem à UML. Enquanto [CSF2000] enfatizam o uso da UML para modelar conceitualmente aplicativos que utilizam DTDs, [Car2001] apresenta uma proposta de estender os diagramas de classes UML para orientar a geração de DTD e XML Schema. Já a abordagem de [FPR2000] se preocupa apenas com o domínio específico de *framework*, mas não aborda o domínio de linguagens de marcação.

Uma outra abordagem é mostrada por [KH2000], que define uma conversão de um modelo em UML para um DTD. Os autores representam graficamente através da UML a estrutura do documento XML. Essa abordagem é pertinente, pois se faz necessário um modo conveniente e eficaz para desenvolver e documentar modelos de documentos de uma maneira que facilite a comunicação tanto de usuários como desenvolvedores do sistema. Além disso, esta proposta viabiliza a incorporação desse modelo de documento (XML) ao projeto do sistema como um todo, visto que as outras parte do sistema já foram modeladas utilizando UML. Como a UML não fornece de um modo direto o conteúdo do modelo XML, os autores fizeram uso dos estereótipos para melhor representar o DTD do documento XML.

Já a proposta apresentada por [Sko99] busca mapear modelos de esquema para DTDs e modelos de instância em documentos XML, definindo para tanto regras de conversão. Para alcançar esse objetivo, o autor restringe a riqueza semântica da UML em um subconjunto denominado modelo de esquema, facilitando assim a definição de regras e consequentemente o processo de mapeamento. Entre as restrições adotadas, destaca-se a não representação dos métodos das classes e das heranças no modelo de esquema. Essa abordagem difere de [Car2001], [KH2000] e [CSF2000] pois não estende a UML utilizando estereótipos. Ao invés disso, [Sko99] restringe a UML para simplificar o mapeamento. A proposta de [Sko99] faz parte de um projeto maior de padronização de intercâmbio de informações geográficas, mas seu trabalho é genérico e pode ser adaptado para outros domínios de aplicação.

Uma proposta alternativa é descrita por [JMP2001]. Essa proposta difere das anteriores pois o ponto de partida não é um diagrama UML de uma aplicação, mas uma aplicação em XML que possui um DTD. Portanto, o sentido do mapeamento é o contrário das abordagens acima. Os autores apresentam um algoritmo para construir diagramas UML a partir de dados XML e mostram como usar estes diagramas para o projeto conceitual de um *Data Warehouse* baseado em dados que se encontram na Web. A estrutura dos dados em XML é visualizada pelo diagrama UML derivado a partir do

DTD que descreve a fonte de dados XML. O diagrama UML construído a partir do algoritmo é um subconjunto dos construtores de modelagem descritos na especificação da UML [OMG99a]. Somente são usados nomes de classes, tipos de atributos, cardinalidade e especificação de papéis para associações e agregações.

Uma sétima abordagem é apresentada por [BCFK99]. Da mesma forma que os autores da sexta abordagem, [BCFK99] partem do esquema XML para a UML. Porém o ponto de partida não é o esquema DTD, mas sim XML Schema. Através de uma aplicação em XML que possui um XML Schema, esses autores modelam a aplicação descrita em XML, agregando maior clareza e facilidade para os desenvolvedores de software e usuários na medida em que eles propõem a utilização de uma ferramenta gráfica: a UML. [BCFK99] estendem a UML através dos seus próprios mecanismos de extensão para criar novas classes que possam representar as estruturas do XML Schema, permitindo assim que o mapeamento XML Schema possa ocorrer de forma direta.

Nas propostas apresentadas acima, não se contempla o mapeamento dos métodos do diagrama de classes UML. O padrão XMI não apresenta um mapeamento de forma clara dos métodos da UML para o XML. Nas propostas de [Car2001], [Sko99], [KH2000], [CSF2000] a parte dinâmica, ou seja as operações ou métodos, não foram mapeados visto que o foco dessas abordagens está na troca de dados. Segundo [Car2001], embora as operações de uma classe sejam elementos chaves da especificação do comportamento da mesma no projeto e análise OO, as operações não são requeridas quando se define a estrutura de um documento XML. Por outro lado, as propostas de [BCFK99] e [JMP2001] realizam o mapeamento no sentido contrário e como os métodos não estão presentes na estrutura da XML eles não são levados em consideração.

Outro ponto a ser levantado é que a grande maioria das propostas acima estendem a UML utilizando os seus próprios mecanismos de extensão (estereótipo, *tag*

value e restrição) para orientar o mapeamento UML – esquema XML e vice-versa. Contudo, os mecanismos de extensão não são objetos do mapeamento.

Nessa dissertação, o ponto de partida do mapeamento será a UML-F. Entretanto, as abordagens acima não englobam diretamente todos os aspectos para mapeá-la, já que os princípios essenciais da UML-F se baseiam na representação através dos métodos das classes UML e dos mecanismo de extensão da UML, mais precisamente nos *tag values.*

Para garantir a semântica no mapeamento da UML-F para XML, percebe-se a necessidade de criar novas classes UML para representar explicitamente o domínio especifico da UML-F, utilizando para isto os mecanismos de extensão da UML. A opção por estender a UML foi adotada nessa dissertação porque a grande maioria dos trabalhos descritos acima utilizou com sucesso essa alternativa. Assim sendo, a extensão da UML-F viabiliza o mapeamento da UML-F para o esquema XML sem a perda de semântica correspondente ao *framework.*

No capítulo 4, serão detalhados os procedimentos necessários para realizar o mapeamento da UML-F para XML.

4. UML-F-X – Uma extensão da UML-F para mapeamento em estruturas XML

Quando se trata de mapear um modelo UML para um vocabulário XML, as seguintes dimensões podem ser abordadas: instâncias de objetos para documento XML e definições de classes para esquemas XML. Como foi detalhado no capítulo 3, essas duas opções são fortemente relacionadas, pois o documento XML gerado é validado com base no esquema XML produzido. No entanto, a geração do DTD depende da semântica do modelo UML.

Nesse capítulo são apresentados os requisitos necessários para a extensão da UML-F bem como os procedimentos para especificá-la de forma a facilitar a geração de DTD-XML. Essa extensão da UML-F será denominada UML-F-X, onde a letra X representa a utilização dos padrões XML.

4.1. Escopo do Mapeamento

O objetivo dessa dissertação se relaciona com o intercâmbio de estruturas de projetos que especificam *frameworks.* Como *framework* se trata de um modelo, o foco do trabalho estará no intercâmbio de modelos e não no de dados. O intercâmbio de modelos se situa em um nível de abstração superior ao nível de intercâmbio de dados. Isso se torna mais claro ao compreender as camadas da arquitetura MOF adotada pelo OMG.

O MOF (*Meta Object Facility*) descreve princípios genéricos bastante abstratos para garantir uma padronização mínima na criação de metamodelos. Segundo [OMG99b], o modelo MOF define uma sintaxe abstrata comum para a definição de metamodelos, sendo um modelo para metamodelos. Assim sendo, o MOF pode ser descrito como um meta-metamodelo.

A especificação do MOF é composta de três partes: o modelo MOF, o mapeamento do MOF para a IDL (*Interface Definition Language*) e as interfaces do MOF. As duas últimas partes da especificação dizem respeito à integração do MOF com a arquitetura CORBA e não serão detalhadas pois não constam do escopo desse trabalho. Já a primeira parte da especificação descreve o modelo MOF e o seu detalhamento é pertinente para essa dissertação. Segundo [OMG99b], os principais construtores do MOF são:

- Classes: possuem atributos e operações, sendo que os atributos representam os metadados e as operações realizam funções específicas nos atributos. Tanto os atributos quanto os parâmetros das operações podem ser definidos de maneira ordenada e podem também possuir restrições estruturais de cardinalidade e unicidade. Classes podem herdar propriedades de outras classes;
- Associações : suportam relações binárias entre as classes e definem restrições estruturais de cardinalidade e unicidade;
- Pacotes : são coleções de classes relacionadas e suas associações. Os pacotes podem ser aninhados;
- Tipo de dados: permitem o uso de tipos diferentes dos básicos para parâmetros e atributos;
- Restrições: são usadas para estabelecer regras semânticas entre os elementos de um metamodelo MOF.

Verifica-se uma grande similaridade dos construtores MOF com o núcleo dos construtores UML. De acordo com [OMG99b], existe um alinhamento muito próximo dos conceitos de meta-modelagem do MOF com os conceitos de modelagem da UML. Tanto o MOF como a UML estão em conformidade com o paradigma da orientação a objetos. Convém destacar que a UML apresenta mais recursos do que o MOF já que esse se propõe a ser um especificação abstrata, abrangente e não restritiva. A semelhança de conceitos e a crescente popularidade da UML permitem que a notação gráfica UML se torne a mais adequada para expressar os metamodelos MOF.

O objetivo do MOF é fornecer uma base para o desenvolvimento de repositórios de metadados, de troca de modelos e de outras ferramentas de software. A arquitetura do MOF apresenta quatro camadas, conforme figura 4.1.

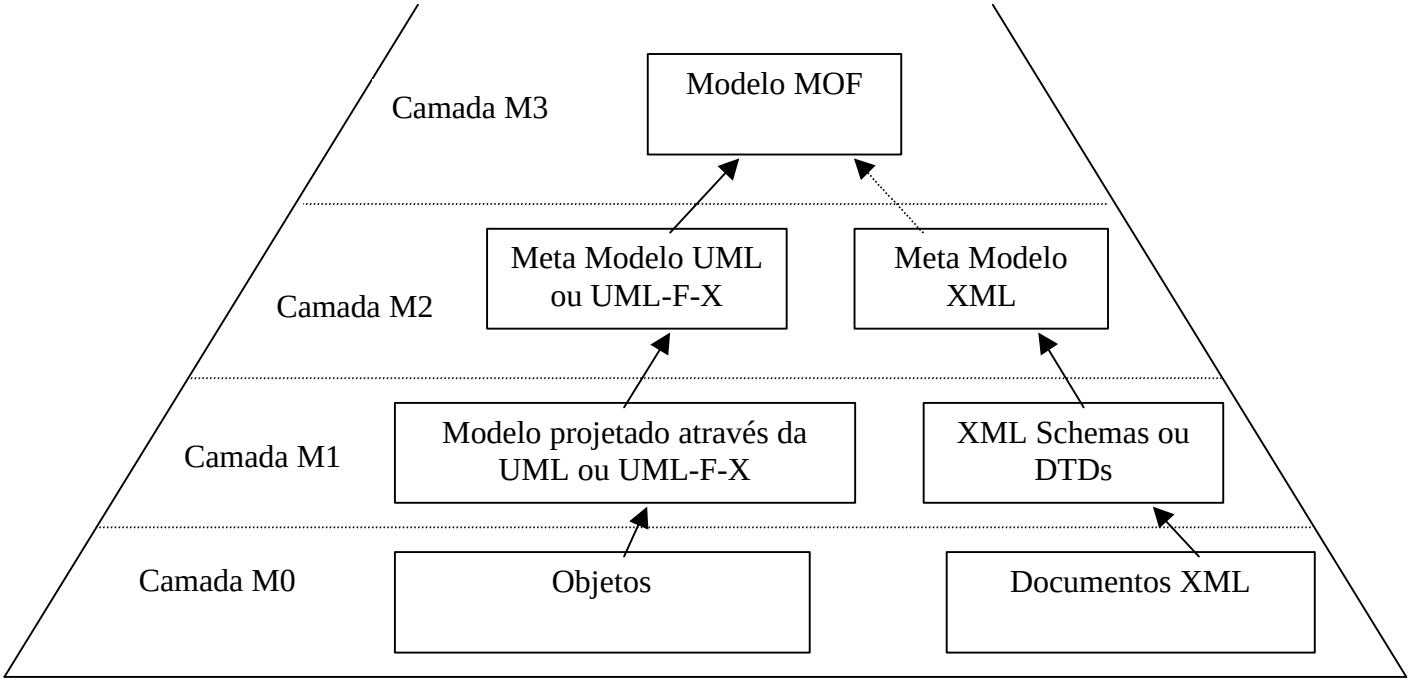

Figura 4. 1 – Arquitetura de camadas MOF (adaptada de [Car2001])

Nessa arquitetura, a camada superior é sempre uma abstração da camada imediatamente inferior. A UML e a UML-F-X (camada M2) podem ser entendidas como instâncias do MOF (camada M3). A UML-F-X foi considerada na camada M2 como um metamodelo MOF, porque atende a especificação dos construtores do MOF. Segundo [BJR98], os mecanismos de extensibilidade de UML permitem estender a linguagem UML de uma maneira controlada, permanecendo fiel aos seus propósitos. De maneira análoga à UML-F descrita no item 2.4 desse trabalho, a UML-F-X também é uma extensão da UML que mantêm fidelidade aos princípios da mesma. Assim sendo, como a UML é um metamodelo MOF, pode-se afirmar que a UML-F-X também é um metamodelo MOF.

O nível de dados (camada M0) é o nível físico e com o menor grau de abstração. É o nível onde são implementados modelos especificados na camada M1, utilizando uma linguagem definida na camada M2.

O lado direito da figura 4.1 apresenta uma estruturação XML semelhante a que é utilizada pela UML. Isso foi feito considerando [Car2001] que trata XML Schema como um possível metamodelo da camada M2. Entretanto, deve-se frisar que o XML Schema ainda não foi definido formalmente pela OMG como sendo uma instância do MOF (camada M3).

Como o objetivo desse trabalho está relacionado com a troca de modelos de software entre aplicações, ou mais exatamente entre ferramentas de modelagem, é preciso especificar recursos que permitam essas trocas. O padrão XMI (descrito no item 2.5.3) permite a troca de metamodelos (camada M2) entre ferramentas, fornecendo regras de produção para a geração de DTDs. Além disso, o padrão XMI fornece regras de produção para codificar e decodificar modelos de níveis da camada M1 para e a partir de documentos XML.

Essas características do padrão XMI são ilustradas na figura 4.2, considerando como metamodelo a UML. Devido o fato do metamodelo UML ser uma instância do MOF, o XMI é utilizado para produzir um DTD para UML. Verifica-se também que o XMI pode ser utilizado para gerar documentos XMI (documentos XML produzidos com regras XMI) a partir de um modelo UML validado por um DTD.

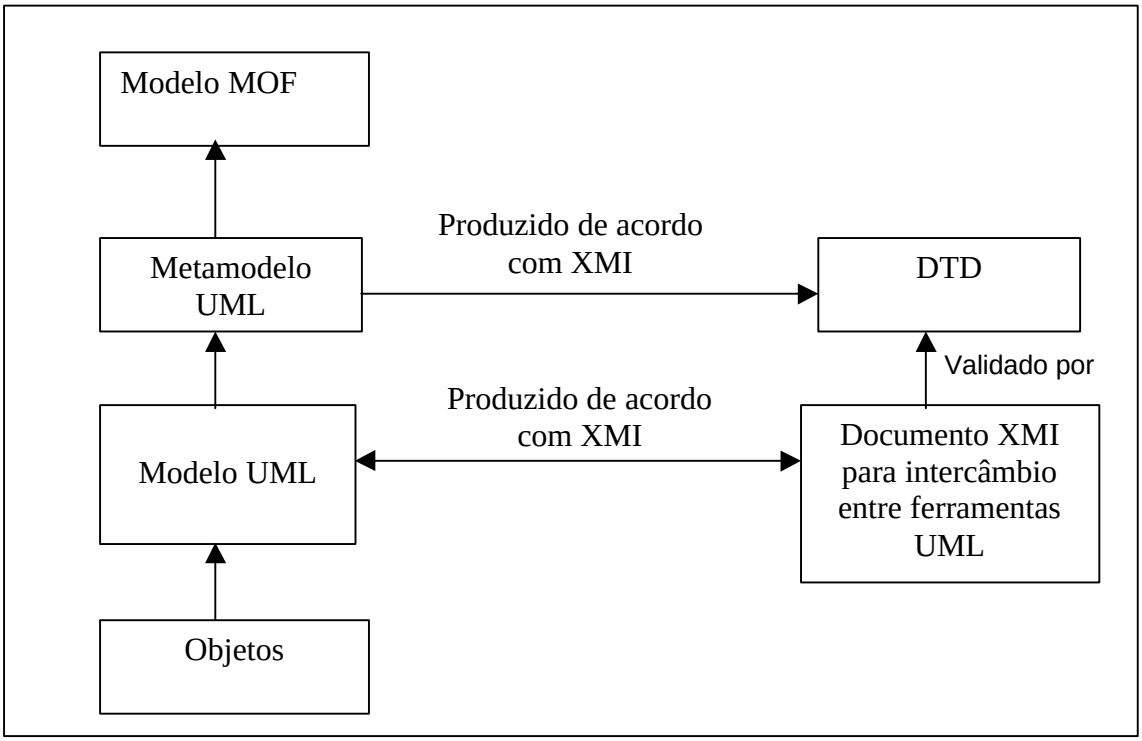

Figura 4. 2 – Regras de produção XMI na arquitetura MOF [Car2001]

Conforme a figura 4.3, a origem do mapeamento proposto nesse trabalho é um modelo de *framework* definido em UML-F-X que se situa na camada M1, porque é uma instância do metamodelo UML-F-X (camada M2). Já o destino do mapeamento será o DTD (camada M2). Deve-se ressaltar que a geração de DTDs a partir de um modelo UML-F-X é possível pois o padrão XMI define regras de geração de DTD para qualquer metamodelo compatível com o MOF, como é o caso da UML-F-X.

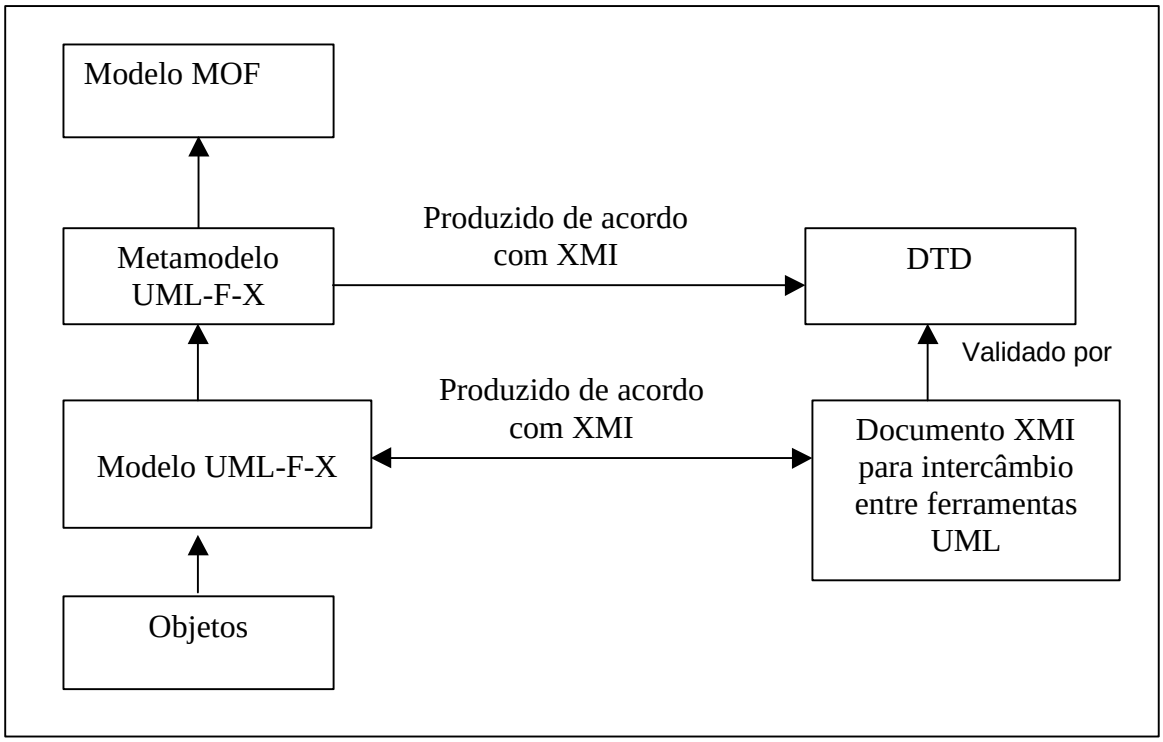

Figura 4. 3 – Mapeamento de UML-F-X para DTD através do XMI (adaptada de [Car2001])

[Car2001] desenvolve uma outra interpretação do mapeamento proposto pela XMI. Na especificação da XMI, as regras XMI podem ser aplicadas para mapear um metamodelo (camada M2) para DTD (vide figura 4.2). Para o autor citado, essas regras também podem ser utilizadas para mapear um modelo (camada M1) para esquemas XML.

De acordo com o autor, essa abordagem é possível pois um modelo desenvolvido segundo as regras de um metamodelo pode ser interpretado como um metamodelo. Portanto ocorre um deslocamento da camada M1 para a camada M2. Esse tipo de deslocamento (*"shift")* já era previsto na própria especificação da XMI, conforme se verifica a seguir:

> *"O uso típico do MOF prevê uma arquitetura de quatro camadas de metadados, mas existem situações onde apenas três camadas são exigidas. Nesses casos, as*

meta-camadas são deslocadas e o modelo se torna o metamodelo." [OMG99b], p. 4-43

Assim sendo, o aparente problema de tentar usar as regras XMI para gerar o DTD a partir de um modelo UML da camada M1 é resolvido através da adaptação de um artifício engenhoso proposto por [Car2001]. O artifício consiste em interpretar o modelo UML como uma instância do modelo MOF.

Aplicando a abordagem de [Car2001] nesse trabalho, um modelo (camada M1) projetado em UML-F-X pode ser interpretado como um metamodelo e assim ser mapeado para DTD, segundo as regras da XMI. A aplicação dessa abordagem é ilustrada pela figura 4.4 que possui apenas três camadas.

Analogamente, [Car2001] interpreta uma instância de um modelo (camada M0) como um modelo (camada M1). Assim sendo, pode-se aplicar as regras da XMI para mapear uma instância de um modelo (objetos UML-F-X) para documentos XML. Entretanto, esse trabalho pretende se concentrar na geração de esquemas XML e não de documentos XML.

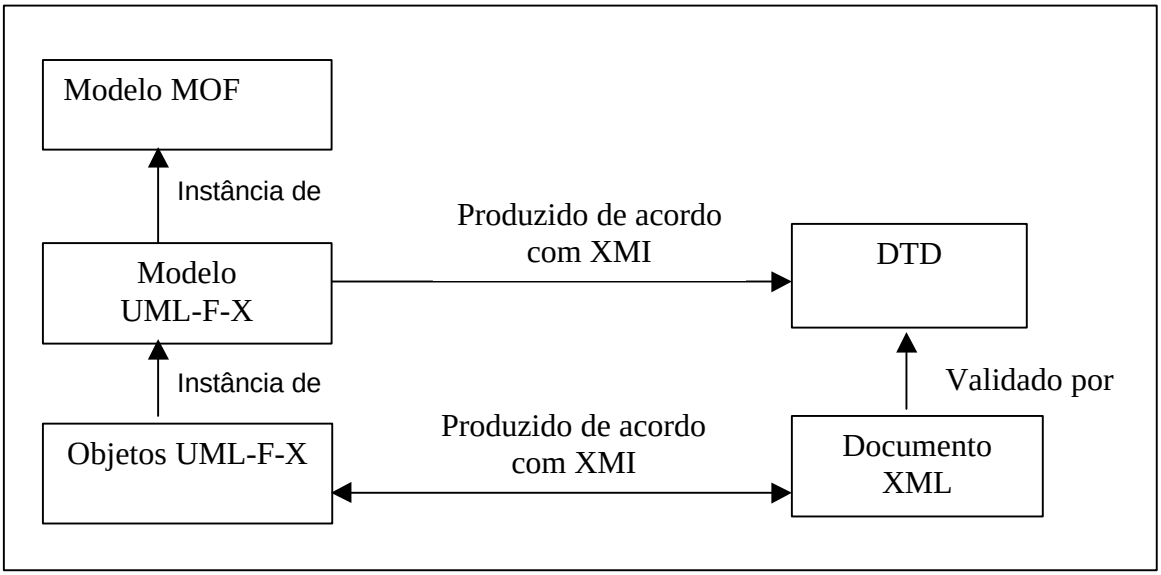

Figura 4. 4 – Mapeamento de UML-F-X para DTD através do XMI (adaptada de [Car2001])

4.2. Tratamento dos Métodos

As abordagens de mapeamento descritas no item 3.4 concentram-se no mapeamento das classes e atributos. Essa dissertação inclui os métodos das classes no mapeamento. É importante destacar que faz parte do mapeamento proposto a assinatura do método, que consiste no nome do método, seus parâmetros e o tipo de retorno. O mapeamento da assinatura do método e dos seus valores de *tag* é suficiente para representar a semântica do projeto de um *framework* feito em UML-F.

A dissertação não irá tratar a chamada, execução e implementação dos métodos, pois pretende-se enfatizar o projeto estrutural de *frameworks* e não a instanciação desses. Com fins ilustrativos, destaca-se que para permitir a chamada remota de métodos através da Web, já existem padrões em desenvolvimento. Segundo [SR2001], o XML-RPC (*Remote Procedure Call)* consiste em uma chamada remota de procedimentos via XML e é o padrão mais simples para ativar serviços Web. De acordo com os autores, os serviços Web são componentes de software reutilizáveis que encapsulam semanticamente funcionalidades discretas e são distribuídos e acessíveis através de programação feita em protocolos padronizados da Internet. Portanto, para se executar remotamente um método, esse deve ser disponibilizado como um serviço Web.

A evolução do XML-RPC derivou na especificação do SOAP (*Simple Object Access Protocol*). [SR2001] caracterizam o SOAP como um protocolo para troca de mensagens e comunicação RPC entre aplicações. O SOAP é baseado na XML e utiliza protocolos como HTTP. O SOAP foi submetido ao W3C e futuramente será denominado XP (XML *Protocol*). Verifica-se que o XML-RPC e o SOAP dizem respeito às questões bem mais físicas e ligadas com a implementação. Essa dissertação se situa em uma camada de abstração superior devido ao fato de se concentrar no intercâmbio de modelos e portanto na documentação dos métodos do *framework* e não na sua implementação.

4.3. Descrição da Semântica

A origem do mapeamento é um modelo descrito em UML-F-X, uma extensão da UML-F criada para facilitar a geração de esquemas XML. Conforme foi apresentado no item 3.4, a grande maioria das propostas estudadas utilizam os próprios mecanismos de extensão da UML (estereótipos, *tag values* e restrições) para estendê-la.

Como um dos objetivos desse trabalho (vide item 1.2) é minimizar as perdas semânticas no mapeamento dos *frameworks* para esquemas XML, será adotada a abordagem de [Car2001] ao invés da proposta XMI [OMG99b]. Segundo [Car2001], as regras de produção XMI permitem apenas a geração de DTDs relaxados e não de DTDs restritos. Essa opção foi adotada pela OMG para permitir o intercâmbio de partes de modelos e por causa das limitações da representação em DTDs.

Entretanto, [Car2001] destaca que em alguns casos é interessante produzir DTDs restritos, pois esses permitem aos projetistas uma melhor representação das restrições semânticas existentes no modelo UML, como por exemplo a garantia das restrições de multiplicidade. Portanto, o DTD restrito garante um nível mais elevado de semântica em relação ao DTD relaxado, que é a opção adotada pelo XMI. Assim sendo, o mapeamento proposto nesse trabalho possibilita a geração de DTDs relaxados ou restritos, permitindo ao projetista uma maior flexibilidade na escolha do nível desejado de semântica.

Nesta dissertação, o recurso de perfis UML proposto por [Car2001] foi utilizado para estender a UML-F. Um perfil UML define um conjunto de estereótipos que estendem os construtores básicos da UML com novos significados e propriedades. Cada estereótipo de um perfil UML pode conter *tag values* e restrições. O objetivo principal de um perfil UML é guiar a geração de esquemas XML (DTD e XML Schema) a partir de diagramas de classes UML. Essa dissertação se concentra na representação, através de perfis UML, da semântica da UML-F-X, pois os procedimentos de geração de esquemas XML a partir de perfis UML já se encontram definidos.

Entre os perfis UML apresentados em [Car2001], esse trabalho adota o perfil <<*enumeration*>>, pois esse é suficiente para representar a semântica da UML-F. Além disso, o estereótipo <<*enumeration*>> foi o escolhido porque trata-se de um elemento padrão da especificação da UML. De acordo com [BRJ98], <<*enumeration*>> aplica-se a uma classe e especifica um tipo enumerado, incluindo seus possíveis valores como um conjunto de identificadores. O estereótipo <<*enumeration*>> é utilizado para orientar o mapeamento da UML para esquemas XML. Os *tag-values* e restrições são opcionais para o estereótipo <<*enumeration*>> e não serão utilizados nesse trabalho, pois não são fundamentais na representação da semântica da UML-F-X.

Conforme descrito no item 2.4, a semântica da UML-F incorpora *tag values* para estender o diagrama de classes da UML. Entre os *tag values* definidos, *appl-class, extensible* e *incomplete* se aplicam às classes e *variable* diz respeito aos métodos, enquanto que *dynamic* e *static* se aplicam tanto para classes quanto para métodos. Nesse trabalho, os *tags values* da UML-F são representados através do perfil UML <<*enumeration*>>. Dessa maneira, a proposta consiste em criar novas classes associadas ao estereótipo <<*enumeration*>>.

4.4. Descrição da extensão UML-F-X

Para que o mapeamento de UML-F-X para DTD fosse possível, optou-se por criar nesse trabalho um metamodelo simplificado que orientasse o projetista na tradução do *framework* originalmente descrito em UML-F para a sua extensão UML-F-X. Sem esse metamodelo, os projetistas ficariam sem um roteiro de apoio no processo de conversão para UML-F-X. Uma vez feita a conversão, a abordagem de [Car2001] pode ser empregada para se gerar DTDs restritos ou relaxados.

No metamodelo, serão especificados os componentes (atributos, métodos, classes e outros) necessários na construção da UML-F-X. Por questões de simplicidade do metamodelo, os elementos da classe *Frozen Class* não serão detalhados, pois se pretende concentrar nos possíveis pontos de variação do *framework*.

No metamodelo UML-F-X, existe o pacote *Simple FW* para organizar os elementos do *framework.* Dentro do pacote *Simple FW*, são utilizadas classes e relacionamentos para especificar os elementos componentes de um *framework.* Uma classe de nome *Framework* ,que representa o próprio *framework*, será criada com os seguintes atributos: *name, creation-date, last-update e version number.* Esses atributos são importantes para que os desenvolvedores que utilizam o *framework* tenham um controle de versões.

Conforme visto no item 2.1, um *framework* é composto por *hot spots* e *frozen spots.* Para o primeiro define-se a classe denominada *Hot Class* e para o segundo elemento uma classe denominada *Frozen Class. O*s *frozen spots* representam os pontos fixos do *framework* e por isso não são detalhados nesse metamodelo. Já os *hot spots* representados na UML-F são elementos centrais desse trabalho.

A classe *Hot Class* possui o atributo *class-type* que é associado à classe *Hot Class Type.* Essa classe por sua vez é rotulada com o estereótipo <<*enumeration*>>, sendo que a lista de valores válidos contém as combinações possíveis de *tag values* para uma classe *hot spot.* A lista de valores é a seguinte:

- *appl-class;*
- *extensible;*
- *extensible static;*
- *extensible dynamic;*
- *incomplete;*
- *incomplete static;*
- *incomplete dynamic;*
- *incomplete extensible;*
- *incomplete extensible static;*
- *incomplete extensible dynamic;*
- *static;*
- *dynamic;*
- *null.*

A *Hot Class* possui zero ou várias operações, que podem ser pontos de variação. Para representar essas operações, é criada a classe *Operation.* Essa classe possui o

atributo *operation-type,* sendo que esse é associado à classe *Hot Operation* estereotipada através de <<*enumeration*>>. O valor *null* é atribuído para o campo *operation-type* dos métodos que não são pontos de variação. A lista de valores válidos para a classe *Hot Operation* é a seguinte:

- *variable;*
- *variable static;*
- *variable dynamic;*
- *static;*
- *dynamic;*
- *null.*

O metamodelo especifica também os tipos de relacionamentos entre as classes. O relacionamento das classes *Hot Class* e *Frozen Class* com a classe *Framework* é representado como uma composição, que é um tipo específico de agregação onde a parte tem um tempo de vida coincidente com o todo [BJR98]. Existem também relacionamentos de agregação, associação e herança entre a classe *Frozen Class* e a classe *Hot Class*. Esses relacionamentos podem ocorrer nos dois sentidos. Por razões de simplificação, o metamodelo UML-F-X não contém as definições que já estão descritas no metamodelo UML, como herança ente classes e atributos relacionados com as assinaturas dos métodos.

A figura 4.5 apresenta, em notação UML, o metamodelo simplificado da UML-F-X de acordo com as características discutidas anteriormente.

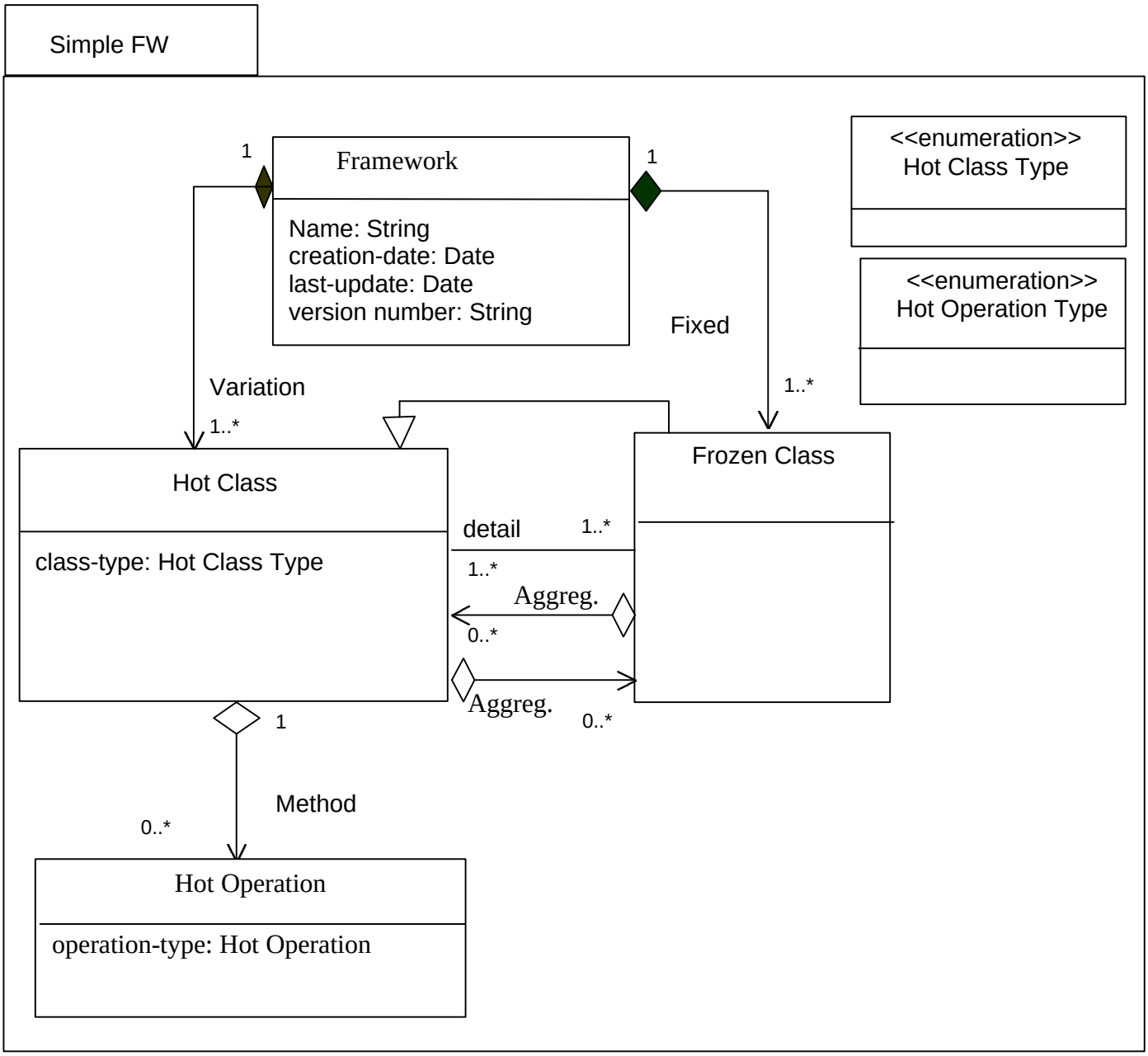

Figura 4. 5 – Metamodelo simplificado UML-F-X

Na figura seguinte, tem-se um exemplo de parte de modelo UML-F obtido de [FPR2000]. Esse modelo descreve um *framework* para aplicativos da área de educação, explicitando pontos de variação através de *tag values.*

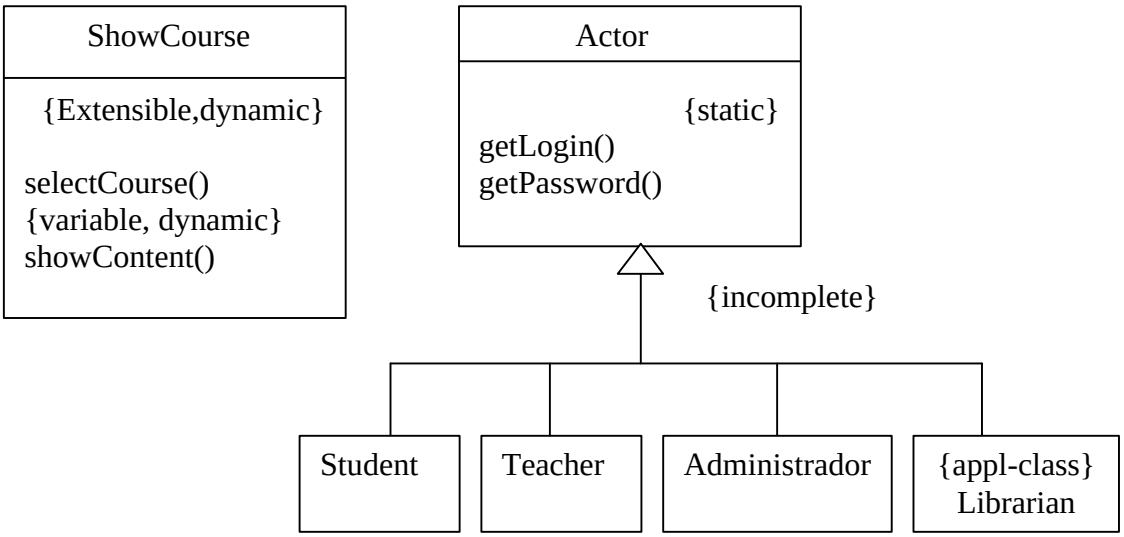

Figura 4. 6 - Diagrama de classes em UML-F [FPR2000]

Na próxima figura, é feita a representação em UML-F-X do modelo descrito na figura 4.6, utilizando para tanto o metamodelo UML-F-X como roteiro para o mapeamento conversão.

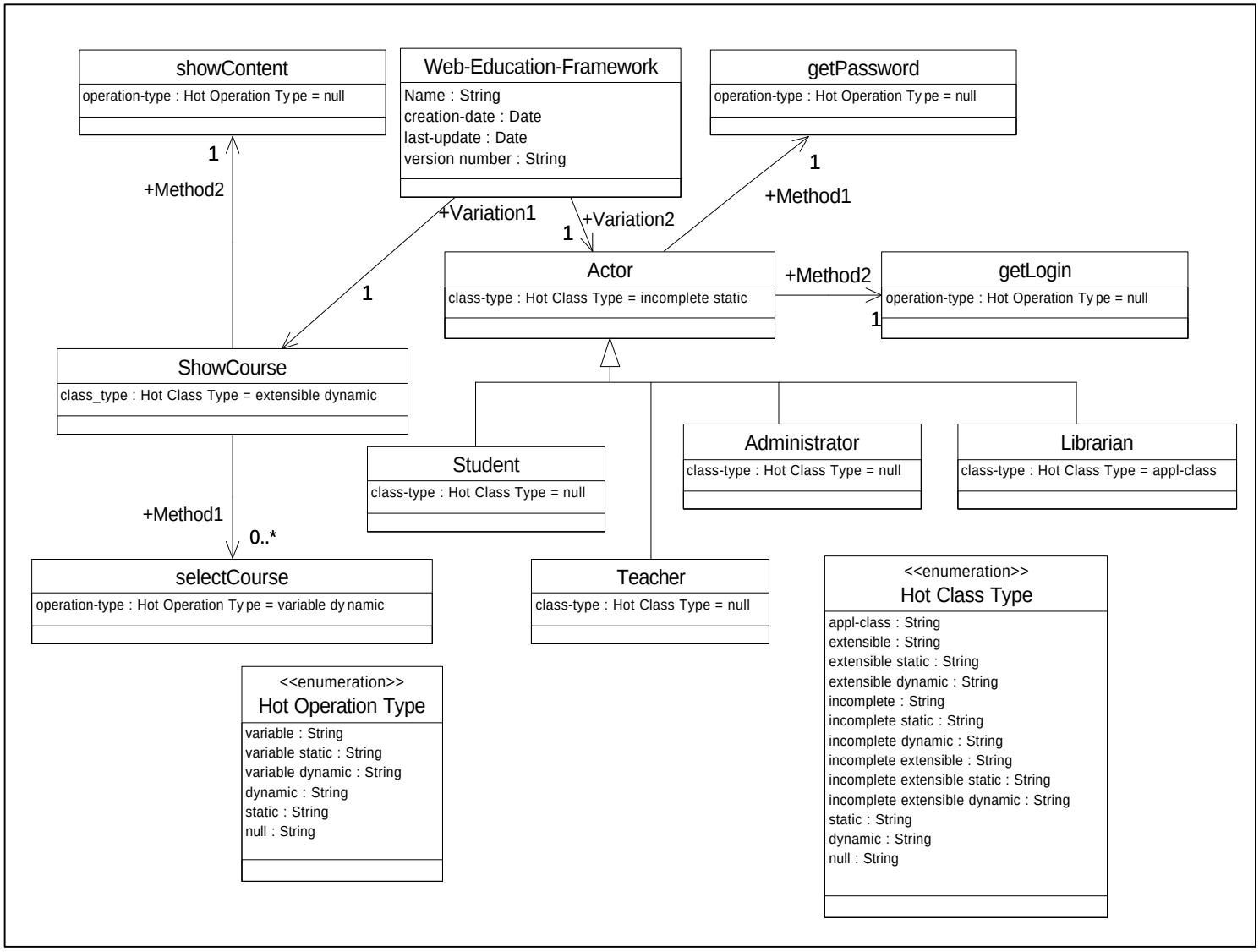

Figura 4. 7 - Diagrama de classes em UML-F-X

4.5. Mapeamento para DTD

Essa dissertação se concentra na definição da semântica da UML-F-X, pois já existem ferramentas que efetuam automaticamente a geração de DTDs / XML Schemas a partir da UML. Como a abordagem de [Car2001] é adotada para o mapeamento dos *frameworks* para esquemas XML, optou-se por utilizar a ferramenta indicada pelo autor na geração de DTDs. Essa ferramenta é a IBM XMI Toolkit desenvolvida pelo *alphaWorks*, que é um grupo de trabalho da IBM sobre tecnologias emergentes. A ferramenta está disponível para *download* em [IBM2000]. Além de gerar DTDs a partir

de modelos UML projetados com Rational Rose (arquivos .mdl), a ferramenta utiliza o padrão XMI para efetuar a conversão entre UML e Java e vice-versa.

No processo de geração de DTDs a partir de diagramas de classes do Rational Rose, a ferramenta IBM XMI Toolkit apresenta algumas particularidades tais como o não mapeamento de notas, métodos das classes e associações sem adornos de nome de papel ou navegação.

A seguir, encontra-se um exemplo de um DTD relaxado referente ao mapeamento da classe *ShowCourse* apresentada na figura 4.7. Os atributos *class-type* e *operation-type* são associados a valores *default* que representam as características dos *hot spots* apresentados na figura 4.6.

<!--CLASS: ShowCourse -->
<!ELEMENT ShowCourse.class-type EMPTY>
<!ATTLIST ShowCourse.class-type xmi.value (*appl-class* | *extensible* | *extensible static* | *extensible dynamic* | *incomplete* | *incomplete static* | *incomplete dynamic* | *incomplete extensible* | *incomplete extensible static* | *incomplete extensible dynamic* | *dynamic* | *static* | *null)* #REQUIRED *extensible dynamic* >
<!ELEMENT ShowCourse.Method2 (showContent)*>
<!ELEMENT ShowCourse.Method1 (selectCourse)*>
<!ELEMENT ShowCourse (ShowCourse.class-type | ShowCourse.Method1 |ShowCourse.Method2 | XMI.reference) * >
<!ATTLIST ShowCourse
class-type (*appl-class* | *extensible* | *extensible static* | *extensible dynamic* | *incomplete* | *incomplete static* | *incomplete dynamic* | *incomplete extensible* | *incomplete extensible static* | *incomplete extensible dynamic* | *dynamic* | *static* | *null)* #IMPLIED *extensible dynamic*
Method1 IDREFS #IMPLIED
Method2 IDREFS #IMPLIED
%XMI.element.att;
%XMI.link.att;

>

A seguir, encontra-se um exemplo de um DTD restrito referente ao mapeamento da classe *ShowCourse* apresentada na figura 4.7.

<!--CLASS: ShowCourse -->
<!ELEMENT ShowCourse.class-type EMPTY>
<!ATTLIST ShowCourse.class-type xmi.value (*appl-class* | *extensible* | *extensible static* | *extensible dynamic* | *incomplete* | *incomplete static* | *incomplete dynamic* | *incomplete extensible* | *incomplete extensible static* | *incomplete extensible dynamic* | *dynamic* | *static* | *null)* #REQUIRED *extensible dynamic* >
<!ELEMENT ShowCourse.Method1 (selectCourse)* >
<!ELEMENT ShowCourse.Method2 (showContent)? >
<!ELEMENT ShowCourse (ShowCourse.class_type?, XMI.extension*, ShowCourse.Method1*, ShowCourse.Method2?)? >
<!ATTLIST ShowCourse
%XMI.element.att;
%XMI.link.att;
>

Para ilustrar o mapeamento para DTDs, projetou-se inicialmente o modelo da figura 4.7 com a ferramenta Rational Rose. No apêndice desse trabalho, encontra-se o DTD completo gerado pela ferramenta IBM XMI Toolkit a partir do modelo da figura 4.7.

5. Conclusão e Trabalhos Futuros

Nesse trabalho foi apresentada uma contribuição para o intercâmbio de *frameworks,* definidos utilizando-se extensões da linguagem UML, através de esquemas XML. De uma maneira mais específica, esse intercâmbio é feito pela representação do *framework* em UML-F-X para assim efetuar o mapeamento para DTD.

Uma das contribuições desse trabalho é a proposta da UML-F-X para modelar *frameworks* de um modo que favoreça uma futura troca dos mesmos através da Web. A revisão de literatura também pode ser considerada uma contribuição desse trabalho, pois pode servir de base para futuros trabalhos relacionados à interseção dos campos da Engenharia de Software com padrões da Internet. Um outro ponto do dissertação que pode ser destacado é a análise crítica das técnicas de mapeamento da UML-XML existentes na literatura. Nesse aspecto, o entrelaçamento dos conceitos do mundo UML com o mundo XML foi uma constante nesse trabalho.

Uma questão pertinente é que essa dissertação pode ser considerada um desdobramento com enfoque Web da tese de doutorado desenvolvida na PUC-RJ por [Fon99]. O intercâmbio de trabalhos, o desenvolvimento de pesquisas conjuntas e a cooperação entre grupos de uma mesma universidade e de universidades diferentes são formas apropriadas de evitar retrabalhos e redundância de pesquisas, otimizando assim os recursos envolvidos e contribuindo para o avanço das pesquisas.

Esse trabalho enfatiza esquemas XML e não de documentos XML. Um trabalho futuro consistiria na aplicação de procedimentos análogos aos utilizados nesse trabalho para mapear uma instância de um modelo (objetos UML-F-X) para documentos XML.

Um trabalho futuro interessante seria validar a UML-F-X para a produção de XML Schema, já que nesse trabalho detalhou-se o mapeamento de *frameworks* descritos em UML-F-X para DTDs. A ferramenta *hyperModel* disponível em [Ont2001] poderia ser utilizada nesse estudo futuro, pois gera automaticamente XML Schema e

HTML a partir de modelos UML. Outra ferramenta interessante para ser avaliada é o ArgoUML [OOT2001], que é uma ferramenta CASE gratuita que utiliza a XMI como formato de gravação de modelos UML.

No mapeamento proposto por esse trabalho apenas a assinatura do método é considerada. Uma possível extensão desse trabalho estaria relacionada com o tratamento da implementação dos métodos. Nesse caso, uma abordagem interessante seria utilizar o protocolo XP para a chamada remota dos métodos.

O metamodelo descrito no item 4.4 detalhou a parte variável (*hot spot)* de um *framework*. Um desdobramento interessante consistira no detalhamento da parte fixa (*frozen).*

O impacto crescente das tecnologias Web no processo de desenvolvimento de sistemas tem provocado uma sinergia entre as pesquisas realizadas na área de padrões da Internet com os trabalhos desenvolvidos pela Engenharia de Software. A consolidação da XML na Web e da UML no campo da análise de sistemas tem acelerado ainda mais esse processo sinérgico. Essa soma de esforços é extremamente positiva, pois evita retrabalhos e estudos em direções antagônicas.

Dessa forma, os benefícios do padrão XML para definir, validar e compartilhar documentos da Web podem ser combinados com os benefícios do padrão UML para especificar, visualizar e documentar sistemas de informações. Utilizando padrões abertos tanto para armazenar quanto para compartilhar informações sobre sistemas orientados a objeto, as equipes de projetistas que trabalham com ferramentas de vários fornecedores podem conseguir colaborar entre si. A troca de dados entre aplicações, ferramentas e repositórios propicia a construção de sistemas distribuídos em um ambiente de desenvolvimento em equipe.

A troca de *frameworks* através da XML situa-se nesse terreno interdisciplinar entre a Web e a Engenharia de Software. Esse terreno tem se mostrado promissor para o

desenvolvimento de uma abordagem da Engenharia de Software coerente com as necessidades atuais de ambientes distribuídos. Nesse contexto, a ampliação das oportunidades de intercâmbio de *frameworks* entre equipes de desenvolvedores poderá contribuir para uma maior reutilização, gerando assim melhorias significativas de produtividade e qualidade de software.

6. Referências Bibliográficas

[ABS99] ABTEBOUL, Serge, BUNEMAN, Peter, SUCIU, Dan. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, Outubro, 1999.

[AP2002a] AMARAL, Juliana A.; PIETROBON, Carlos A. M. *Interchanging Frameworks through XML DTDs.* Submetido ao congresso OOIS'2002 Object-Oriented Information Systems. Montpellier, França, 2002.

[AP2002b] AMARAL, Juliana A.; PIETROBON, Carlos A. M. *Using XML to Improve Frameworks Reuse* . Submetido ao congresso SBES – XVI Simpósio Brasileiro de Engenharia de Software. Gramado, 2002.

[BCFK99] BOOCH, Grady; CHRISTERSON, Magnus; FUCHS, Matthew; KOISTINEN, Jari. *UML for XML Schema Mapping Specification.* Editora Addison Wesley, Agosto, 1999.

[BL2000] BARBOSA, Álvaro C. Pereira; LUCENA, Carlos José P. *Integração de Frameworks de Software.* Monografia em Ciência da Computação, PUC-RJ, Departamento de Informática, 2000.

[BRJ98] BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. *The Unified Modeling Language User Guide.* Editora Addison Wesley,1998.

[Bun97] BUNEMAN, Peter. *Semistructured Data.* PODS, Tucson Arizona,1997.

[Car2001] CARLSON, David. *Modeling XML Apllications with UML.* Editora Addison Wesley,20001.

[CH98] CHANG, Dan; HARKEY, Dan. *Client /Server Data Access with Java and XML.* Editora John Willey & Sons,1998.

[CSF2000] CONRAD,Rainer; SCHEFFENER, Dieter; FREYTAG, J. Christoph. *XML Conceptual Modeling using UML. Conceptual Modeling - ER 2000, 19th International Conference on Conceptual Modeling.* Salt Lake City, Utah, USA, October 9-12, 2000

[FLM98] FLORESCU, Daniele, LEVY,Alon, MENDELZON, Alberto . *Database Tecniques for the World-Wide-Web: a Survey*. SIGMOD Record, vol.27, n.3 – pág.59-74, Setembro, 1998.

[Fon99] FONTOURA, Marcus. *A Systematic Approach to Framework Development.* Tese de Doutorado, PUC-RJ, Departamento de Informática, Rio de Janeiro, 1999.

[FPR2000] FONTOURA, Marcus; PREE, Wolfgang; RUMPE, Bernhard. *UML-F: A Modeling Language for Object-Oriented Frameworks. 14th European Conference on Object Oriented Programming* (ECOOP 2000), *Lecture Notes in Computer Science* 1850, *Springer*, 63-82, Cannes, França, 2000

[FSJ99] FAYAD, Mohamed; SCHMIDT, Douglas; JOHNSON, Ralph. *Building Applications Frameworks*. Editora John Willey & Sons,1999

[Fur98] FURLAN, José Davi. *Modelagem de Objetos através da UML*. São Paulo, editora MAKRON Books, 1998.

[GHJV95] GAMMA, E; HELM, R;JOHNSON,R;VLISSIDES,J. *Design Patterns – Elementes of Reusable Object-Oriented Software.* Addison-Wesley professional computing series, 1995.

[IBM2000] International Business Machines. *XMI Toolkit.* Disponível on-line em http://www.alphaworks.ibm.com/tech/xmitoolkit

[Joh97] JOHNSON, Ralph E. *Frameworks = (Components + Patterns).* Communications of the ACM, Vol. 40, No.10, Outubro 1997, p. 39 – 42.

[Jon2001] JONES, Meilir Page. *Fundamentos do Desenho Orientado a Objeto com UML*. São Paulo, editora MAKRON Books, 2001.

[JMP2001] JENSEN, Mikael Rune; MÜLLER, Thomas Holmgren; PEDERSEN, Torben Bach. *Converting XML Data to UML Diagrams for Conceptual Data Integration. In Proceedings of the First International Workshop at CAiSE*01 on Data Integration Over The Web (DIWeb).* 1st International Workshop at CAiSE*01, Interlaken, Switzerland

[KH2000] KIMBER, W. Eliot; HEINTZ, John. *Using UML to define XML document types.* Markup Languages: Theory & Practice 2/3 (Summer 2000) 295-320

[Kru95] KRUTCHEN, Philippe B. *The 4+1 View Model of Architecture.* IEEE Software, Novembro 1995, p. 42 – 50.

[Mcg98] MCGRTH, Sean. *XML Aplicações Práticas.* Rio de Janeiro, Editora Campus, 1999.

[Mic2000] Microsoft Corporation. *Building XML-Basead Web Aplications Course Workbook*. Microsoft Press, 2000.

[MKMG97] MONROE, Robert; KOMPANEX, Andrew; MELTON, Ralph; GARLAN, David. *Architectural Styles, Design Patterns and Objects.* IEEE Software, Janeiro de 1997, p.43-52

[OFL2001] OLIVEIRA, Toacy C.; FILHO, Ivan M.; LUCENA, Carlos J.P. *Using XML and Frameworks to develop Information Systems.* Third Internacional Conference on

Enterprise Information Systems (ICEIS2001), 2001, Setúbal - Portugal : ICEIS Press,2001. V.2.p.571 – 577.

[Ont2001] Ontogenics Corporation. *HyperModel Application.* Disponível on-line em http://www.xmlmodeling.com.

[OMG99a] *OMG Unified Modeling Language Specification.* Versão 1.3, 1999. Disponível on-line em http://www.rational.com/uml

[OMG99b] *OMG XML Metadata Interchange (XMI).* Versão 1.1, 1999. Disponível on-line em ftp://ftp.omg.org/pub/docs/ad/99-10-02.pdf

[OOT2001] ODUTOLA, Kunle; OGUNTIMEHIN, Anthony; TOLKE, Linus. *ArgoUML Quick Guide*. Versão 0.9.x, 2001. Disponível on-line em http://www.argouml.org.

[PPSS95] PREE, Wolfgang; POMBERGER, Gustav; SCHPPERT, Albert; SOMMERLAND, Peter. *Active Guidance of Framework Development.* Software – Concepts and Tools. Editora Springer-Verlager, 1995, p. 94 – 103.

[Pre95] PREE, Wolfang. Desing Patterns for Object-Oriented Software Development. Addison-Wesley,1995.

[SF97] SCHMIDT, Douglas C. ; FAYAD, Mohamed E. *Building Reusable OO Frameworks for Distributed Software.* Communications of the ACM, Vol. 40, No.10, Outubro 1997, p. 85 – 87.

[Sch97] SCHMIDT, Hans Albrecht. *Systematic Framework Design by Generalization.* Communications of the ACM, Vol. 40, No.10, Outubro 1997, p. 48 – 51.

[Sko99] SKOGAN, David. *UML as a schema language for XML based data interchange*. In Proceedings of the 2nd International Conference on The Unified Modeling Language (UML'99), 1999. Disponível on-line em www.ifi.uio.no/davids/papers/Uml2Xml.pdf.

[SR2001] SLEEPER,Brent; ROBINS, Bill. *Defining Web Services.* Disponível on-line em www.stencilgroup.com , 2001.

[Suc98] SUCIU, Dan, *Semistructured Data and XML*. The 5th International Conference of Foundations of Data Organization (FODO'98), Kobe, Japan, Novembro, 1998

[W3C98] The World Wide Web Consortium. *Extensible Markup Languange (XML) 1.0.,* Fevereiro 1998. Disponível on-line em http://www.w3.org/TR/REC-xml

[W3C99] The World Wide Web Consortium. *Namespaces in XML.* Janeiro 1999. Disponível on-line em http://www.w3.org/TR/REC-xml

[W3C2001] The World Wide Web Consortium. XML Schema Part 0: Primer, 2 May 2001. Disponível on-line em http://www.w3.org/TR/xmlschema-0

[WRN2001] WEBER, Kival.; ROCHA, Ana R.C.; NASCIMENTO, Célia J. *Qualidade e Produtividade em Software.* São Paulo, Makron Books, 2001.

[You95] YOURDON, Edward. *Declínio e Queda dos Analistas e dos Prpgramadores*. São Paulo, Makron Books, 1995.

7. Apêndice

Esse apêndice exibe o DTD completo gerado pela ferramenta IBM XMI Toolkit a partir do modelo da figura 4.7.

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

<!-- XMI Automatic DTD Generation                                    -->
<!-- Date: Sun May 19 22:48:03 GMT-03:00 2002                        -->
<!-- Metamodel: Apendice1                                            -->
<!-- ____________________________________________________________ -->
<!--                                                                 -->
<!-- XMI is the top-level XML element for XMI transfer text          -->
<!-- ____________________________________________________________ -->

<!ELEMENT XMI (XMI.header, XMI.content?, XMI.difference*,
          XMI.extensions*) >
<!ATTLIST XMI
        xmi.version CDATA #FIXED "1.0"
        timestamp CDATA #IMPLIED
        verified (true | false) #IMPLIED
>
<!-- ____________________________________________________________ -->
<!--                                                                 -->
<!-- XMI.header contains documentation and identifies the model,     -->
<!-- metamodel, and metametamodel                                    -->
<!-- ____________________________________________________________ -->

<!ELEMENT XMI.header (XMI.documentation?, XMI.model*, XMI.metamodel*,
                XMI.metametamodel*) >
<!-- ____________________________________________________________ -->
<!--                                                                 -->
<!-- documentation for transfer data                                 -->
<!-- ____________________________________________________________ -->

<!ELEMENT XMI.documentation (#PCDATA | XMI.owner | XMI.contact |
                    XMI.longDescription | XMI.shortDescription |
                    XMI.exporter | XMI.exporterVersion |
                    XMI.notice)* >

<!ELEMENT XMI.owner ANY >
<!ELEMENT XMI.contact ANY >
<!ELEMENT XMI.longDescription ANY >
<!ELEMENT XMI.shortDescription ANY >
```

```
<!ELEMENT XMI.exporter ANY >
<!ELEMENT XMI.exporterVersion ANY >
<!ELEMENT XMI.exporterID ANY >

<!ELEMENT XMI.notice ANY >
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.element.att defines the attributes that each XML element -->
<!-- that corresponds to a metamodel class must have to conform to -->
<!-- the XMI specification.                                       -->
<!-- ____________________________________________________________ -->

<!ENTITY % XMI.element.att
          'xmi.id ID #IMPLIED xmi.label CDATA #IMPLIED xmi.uuid
           CDATA #IMPLIED ' >
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.link.att defines the attributes that each XML element that -->
<!-- corresponds to a metamodel class must have to enable it to   -->
<!-- function as a simple XLink as well as refer to model         -->
<!-- constructs within the same XMI file.                         -->
<!-- ____________________________________________________________ -->

<!ENTITY % XMI.link.att
          'xml:link CDATA #IMPLIED inline (true | false) #IMPLIED
           actuate (show | user) #IMPLIED href CDATA #IMPLIED role
           CDATA #IMPLIED title CDATA #IMPLIED show (embed | replace
           | new) #IMPLIED behavior CDATA #IMPLIED xmi.idref IDREF
           #IMPLIED xmi.uuidref CDATA #IMPLIED' >

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.model identifies the model(s) being transferred          -->
<!-- ____________________________________________________________ -->

<!ELEMENT XMI.model ANY >
<!ATTLIST XMI.model
        %XMI.link.att;
        xmi.name     CDATA #REQUIRED
        xmi.version  CDATA #IMPLIED
>
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.metamodel identifies the metamodel(s) for the transferred -->
<!-- data                                                         -->
<!-- ____________________________________________________________ -->
```

```
<!ELEMENT XMI.metamodel ANY >
<!ATTLIST XMI.metamodel
        %XMI.link.att;
        xmi.name     CDATA #REQUIRED
        xmi.version  CDATA #IMPLIED
>
<!-- _____________________________________________________________ -->
<!--                                                               -->
<!-- XMI.metametamodel identifies the metametamodel(s) for the     -->
<!-- transferred data                                              -->
<!-- _____________________________________________________________ -->

<!ELEMENT XMI.metametamodel ANY >
<!ATTLIST XMI.metametamodel
        %XMI.link.att;
        xmi.name     CDATA #REQUIRED
        xmi.version  CDATA #IMPLIED
>
<!-- _____________________________________________________________ -->
<!--                                                               -->
<!-- XMI.content is the actual data being transferred              -->
<!-- _____________________________________________________________ -->

<!ELEMENT XMI.content ANY >

<!-- _____________________________________________________________ -->
<!--                                                               -->
<!-- XMI.extensions contains data to transfer that does not conform -->
<!-- to the metamodel(s) in the header                             -->
<!-- _____________________________________________________________ -->

<!ELEMENT XMI.extensions ANY >
<!ATTLIST XMI.extensions
        xmi.extender CDATA #REQUIRED
>
<!-- _____________________________________________________________ -->
<!--                                                               -->
<!-- extension contains information related to a specific model    -->
<!-- construct that is not defined in the metamodel(s) in the header -->
<!-- _____________________________________________________________ -->

<!ELEMENT XMI.extension ANY >
<!ATTLIST XMI.extension
        %XMI.element.att;
        %XMI.link.att;
        xmi.extender   CDATA #REQUIRED
        xmi.extenderID CDATA #REQUIRED
```

```
>


<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.difference holds XML elements representing differences to a -->
<!-- base model                                                   -->
<!-- ____________________________________________________________ -->

<!ELEMENT XMI.difference (XMI.difference | XMI.delete | XMI.add |
                  XMI.replace)* >
<!ATTLIST XMI.difference
        %XMI.element.att;
        %XMI.link.att;
>
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.delete represents a deletion from a base model           -->
<!-- ____________________________________________________________ -->

<!ELEMENT XMI.delete EMPTY >
<!ATTLIST XMI.delete
        %XMI.element.att;
        %XMI.link.att;
>
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.add represents an addition to a base model               -->
<!-- ____________________________________________________________ -->

<!ELEMENT XMI.add ANY >
<!ATTLIST XMI.add
        %XMI.element.att;
        %XMI.link.att;
        xmi.position CDATA "-1"
>
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.replace represents the replacement of a model construct  -->
<!-- with another model construct in a base model                 -->
<!-- ____________________________________________________________ -->

<!ELEMENT XMI.replace ANY >
<!ATTLIST XMI.replace
        %XMI.element.att;
        %XMI.link.att;
```

```
        xmi.position CDATA "-1"
>


<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- XMI.reference may be used to refer to data types not defined in -->
<!-- the metamodel                                                -->
<!-- ____________________________________________________________ -->

<!ELEMENT XMI.reference ANY >
<!ATTLIST XMI.reference
        %XMI.link.att;
>
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- This section contains the declaration of XML elements        -->
<!-- representing data types                                      -->
<!-- ____________________________________________________________ -->
<!ELEMENT XMI.TypeDefinitions ANY >
<!ELEMENT XMI.field ANY >
<!ELEMENT XMI.seqItem ANY >
<!ELEMENT XMI.octetStream (#PCDATA) >
<!ELEMENT XMI.unionDiscrim ANY >
<!ELEMENT XMI.enum EMPTY >
<!ATTLIST XMI.enum
        xmi.value CDATA #REQUIRED
>
<!ELEMENT XMI.any ANY >
<!ATTLIST XMI.any
        %XMI.link.att;
        xmi.type CDATA #IMPLIED
        xmi.name CDATA #IMPLIED
>
<!ELEMENT XMI.CorbaTypeCode (XMI.CorbaTcAlias | XMI.CorbaTcStruct |
                        XMI.CorbaTcSequence | XMI.CorbaTcArray |
                        XMI.CorbaTcEnum | XMI.CorbaTcUnion |
                        XMI.CorbaTcExcept | XMI.CorbaTcString |
                        XMI.CorbaTcWstring | XMI.CorbaTcShort |
                        XMI.CorbaTcLong | XMI.CorbaTcUshort |
                        XMI.CorbaTcUlong | XMI.CorbaTcFloat |
                        XMI.CorbaTcDouble | XMI.CorbaTcBoolean |
                        XMI.CorbaTcChar | XMI.CorbaTcWchar |
                        XMI.CorbaTcOctet | XMI.CorbaTcAny |
                        XMI.CorbaTcTypeCode | XMI.CorbaTcPrincipal |
                        XMI.CorbaTcNull | XMI.CorbaTcVoid |
                        XMI.CorbaTcLongLong |
```

```
                    XMI.CorbaTcLongDouble) >
<!ATTLIST XMI.CorbaTypeCode
        %XMI.element.att;
>
<!ELEMENT XMI.CorbaTcAlias (XMI.CorbaTypeCode) >
<!ATTLIST XMI.CorbaTcAlias
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>
<!ELEMENT XMI.CorbaTcStruct (XMI.CorbaTcField)* >
<!ATTLIST XMI.CorbaTcStruct
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>
<!ELEMENT XMI.CorbaTcField (XMI.CorbaTypeCode) >
<!ATTLIST XMI.CorbaTcField
        xmi.tcName CDATA #REQUIRED
>
<!ELEMENT XMI.CorbaTcSequence (XMI.CorbaTypeCode |
                    XMI.CorbaRecursiveType) >
<!ATTLIST XMI.CorbaTcSequence
        xmi.tcLength CDATA #REQUIRED
>
<!ELEMENT XMI.CorbaRecursiveType EMPTY >
<!ATTLIST XMI.CorbaRecursiveType
        xmi.offset CDATA #REQUIRED
>
<!ELEMENT XMI.CorbaTcArray (XMI.CorbaTypeCode) >
<!ATTLIST XMI.CorbaTcArray
        xmi.tcLength CDATA #REQUIRED
>
<!ELEMENT XMI.CorbaTcObjRef EMPTY >
<!ATTLIST XMI.CorbaTcObjRef
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>
<!ELEMENT XMI.CorbaTcEnum (XMI.CorbaTcEnumLabel) >
<!ATTLIST XMI.CorbaTcEnum
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>
<!ELEMENT XMI.CorbaTcEnumLabel EMPTY >
<!ATTLIST XMI.CorbaTcEnumLabel
        xmi.tcName CDATA #REQUIRED
>
<!ELEMENT XMI.CorbaTcUnionMbr (XMI.CorbaTypeCode, XMI.any) >
<!ATTLIST XMI.CorbaTcUnionMbr
```

```
        xmi.tcName CDATA #REQUIRED
>
<!ELEMENT XMI.CorbaTcUnion (XMI.CorbaTypeCode, XMI.CorbaTcUnionMbr*)
>
<!ATTLIST XMI.CorbaTcUnion
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>

<!ELEMENT XMI.CorbaTcExcept (XMI.CorbaTcField)* >
<!ATTLIST XMI.CorbaTcExcept
        xmi.tcName CDATA #REQUIRED
        xmi.tcId   CDATA #IMPLIED
>
<!ELEMENT XMI.CorbaTcString EMPTY >
<!ATTLIST XMI.CorbaTcString
        xmi.tcLength CDATA #REQUIRED
>
<!ELEMENT XMI.CorbaTcWstring EMPTY >
<!ATTLIST XMI.CorbaTcWstring
        xmi.tcLength CDATA #REQUIRED
>
<!ELEMENT XMI.CorbaTcFixed EMPTY >
<!ATTLIST XMI.CorbaTcFixed
        xmi.tcDigits CDATA #REQUIRED
        xmi.tcScale  CDATA #REQUIRED
>
<!ELEMENT XMI.CorbaTcShort EMPTY >
<!ELEMENT XMI.CorbaTcLong EMPTY >
<!ELEMENT XMI.CorbaTcUshort EMPTY >
<!ELEMENT XMI.CorbaTcUlong EMPTY >
<!ELEMENT XMI.CorbaTcFloat EMPTY >
<!ELEMENT XMI.CorbaTcDouble EMPTY >
<!ELEMENT XMI.CorbaTcBoolean EMPTY >
<!ELEMENT XMI.CorbaTcChar EMPTY >
<!ELEMENT XMI.CorbaTcWchar EMPTY >
<!ELEMENT XMI.CorbaTcOctet EMPTY >
<!ELEMENT XMI.CorbaTcAny EMPTY >
<!ELEMENT XMI.CorbaTcTypeCode EMPTY >
<!ELEMENT XMI.CorbaTcPrincipal EMPTY >
<!ELEMENT XMI.CorbaTcNull EMPTY >
<!ELEMENT XMI.CorbaTcVoid EMPTY >
<!ELEMENT XMI.CorbaTcLongLong EMPTY >
<!ELEMENT XMI.CorbaTcLongDouble EMPTY >
```

```
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL CLASS:  ShowCourse                                 -->
<!-- ____________________________________________________________ -->

<!ELEMENT ShowCourse.class_type EMPTY >
<!ATTLIST ShowCourse.class_type
        xmi.value ( appl-class | extensible | extensible static | extensible dynamic |
incomplete | incomplete static | incomplete dynamic | incomplete extensible | incomplete
extensible static | incomplete extensible dynamic | static | dynamic | null ) #REQUIRED
extensible dynamic
>

<!ELEMENT ShowCourse.Method2 (showContent)? >

<!ELEMENT ShowCourse.Method1 (selectCourse)* >

<!ELEMENT ShowCourse (ShowCourse.class_type?, XMI.extension*,
                ShowCourse.Method2?, ShowCourse.Method1*)? >
<!ATTLIST ShowCourse
        %XMI.element.att;
        %XMI.link.att;
>

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL CLASS:  showContent                                -->
<!-- ____________________________________________________________ -->


<!ELEMENT showContent.operation-type EMPTY >
<!ATTLIST showContent.operation-type
        xmi.value ( variable | variable static | variable dynamic | dynamic | static | null )
#REQUIRED null
>

<!ELEMENT showContent (showContent.operation-type?, XMI.extension*)? >
<!ATTLIST showContent
        %XMI.element.att;
        %XMI.link.att;
>
```

```
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL CLASS:  selectCourse                               -->
<!-- ____________________________________________________________ -->


<!ELEMENT selectCourse.operation-type EMPTY >
<!ATTLIST selectCourse.operation-type
        xmi.value ( variable | variable static | variable dynamic | dynamic | static | null )
#REQUIRED variable dynamic
>

<!ELEMENT selectCourse (selectCourse.operation-type?, XMI.extension*)? >
<!ATTLIST selectCourse
        %XMI.element.att;
        %XMI.link.att;
>

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL CLASS:  Actor                                      -->
<!-- ____________________________________________________________ -->


<!ELEMENT Actor.class-type EMPTY >
<!ATTLIST Actor.class-type
        xmi.value ( appl-class | extensible | extensible static | extensible dynamic |
incomplete | incomplete static | incomplete dynamic | incomplete extensible | incomplete
extensible static | incomplete extensible dynamic | static | dynamic | null ) #REQUIRED
incomplete static
>

<!ELEMENT Actor.Method2 (getLogin)? >

<!ELEMENT Actor.Method1 (getPassword)? >

<!ELEMENT Actor (Actor.class-type?, XMI.extension*, Actor.Method2?,
            Actor.Method1?)? >
<!ATTLIST Actor
        %XMI.element.att;
        %XMI.link.att;
>
```

```
<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  Student                                            -->
<!-- ____________________________________________________________________ -->

<!ELEMENT Librarian.class-type EMPTY >
<!ATTLIST Librarian.class-type
        xmi.value ( appl-class | extensible | extensible static | extensible dynamic |
incomplete | incomplete static | incomplete dynamic | incomplete extensible | incomplete
extensible static | incomplete extensible dynamic | static | dynamic | null ) #REQUIRED
null
>

<!ELEMENT Student (Actor.class-type?, XMI.extension*, Actor.Method2?,
            Actor.Method1?)? >
<!ATTLIST Student
        %XMI.element.att;
        %XMI.link.att;
>

<!-- ____________________________________________________________________ -->
<!--                                                                      -->
<!-- METAMODEL CLASS:  Teacher                                            -->
<!-- ____________________________________________________________________ -->

<!ELEMENT Librarian.class-type EMPTY >
<!ATTLIST Librarian.class-type
        xmi.value ( appl-class | extensible | extensible static | extensible dynamic |
incomplete | incomplete static | incomplete dynamic | incomplete extensible | incomplete
extensible static | incomplete extensible dynamic | static | dynamic | null ) #REQUIRED
null
>

<!ELEMENT Teacher (Actor.class-type?, XMI.extension*, Actor.Method2?,
            Actor.Method1?)? >
<!ATTLIST Teacher
        %XMI.element.att;
        %XMI.link.att;
>
```

```
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL CLASS:  Administrator                              -->
<!-- ____________________________________________________________ -->

<!ELEMENT Librarian.class-type EMPTY >
<!ATTLIST Librarian.class-type
        xmi.value ( appl-class | extensible | extensible static | extensible dynamic |
incomplete | incomplete static | incomplete dynamic | incomplete extensible | incomplete
extensible static | incomplete extensible dynamic | static | dynamic | null ) #REQUIRED
null
>

<!ELEMENT Administrator (Actor.class-type?, XMI.extension*,
                Actor.Method2?, Actor.Method1?)? >
<!ATTLIST Administrator
        %XMI.element.att;
        %XMI.link.att;
>

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL CLASS:  Librarian                                  -->
<!-- ____________________________________________________________ -->


<!ELEMENT Librarian.class-type EMPTY >
<!ATTLIST Librarian.class-type
        xmi.value ( appl-class | extensible | extensible static | extensible dynamic |
incomplete | incomplete static | incomplete dynamic | incomplete extensible | incomplete
extensible static | incomplete extensible dynamic | static | dynamic | null ) #REQUIRED
appl-class
>

<!ELEMENT Librarian (Actor.class-type?, Librarian.class-type?,
            XMI.extension*, Actor.Method2?, Actor.Method1?)? >
<!ATTLIST Librarian
        %XMI.element.att;
        %XMI.link.att;
>
```

```
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL CLASS:  Web-Education-Framework                    -->
<!-- ____________________________________________________________ -->

<!ELEMENT Web-Education-Framework.Name (#PCDATA | XMI.reference)* >

<!ELEMENT Web-Education-Framework.creation-date (#PCDATA |
                                XMI.reference)* >

<!ELEMENT Web-Education-Framework.last-update (#PCDATA |  XMI.reference)* >

<!ELEMENT Web-Education-Framework.version number (#PCDATA |
                                XMI.reference)* >

<!ELEMENT Web-Education-Framework.Variation1 (ShowCourse)? >

<!ELEMENT Web-Education-Framework.Variation2 (Actor | Student | Teacher |
                                Administrator |  Librarian)? >

<!ELEMENT Web-Education-Framework (Web-Education-Framework.Name?,
                        Web-Education-Framework.creation-date?,
                        Web-Education-Framework.last-update?,
                        Web-Education-Framework.version
                        number?, XMI.extension*,
                        Web-Education-Framework.Variation1?,
                        Web-Education-Framework.Variation2?)? >
<!ATTLIST Web-Education-Framework
        %XMI.element.att;
        %XMI.link.att;
>
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL CLASS:  getLogin                                   -->
<!-- ____________________________________________________________ -->

<!ELEMENT getLogin.operation-type EMPTY >
<!ATTLIST getLogin.operation-type
        xmi.value ( variable | variable static | variable dynamic | dynamic | static | null )
#REQUIRED null
>

<!ELEMENT getLogin (getLogin.operation-type?, XMI.extension*)? >
<!ATTLIST getLogin
        %XMI.element.att;
        %XMI.link.att;
>
```

```
<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL CLASS:  getPassword                                -->
<!-- ____________________________________________________________ -->


<!ELEMENT getPassword.operation-type EMPTY >
<!ATTLIST getPassword.operation-type
        xmi.value ( variable | variable static | variable dynamic | dynamic | static | null )
#REQUIRED null
>

<!ELEMENT getPassword (getPassword.operation-type?, XMI.extension*)? >
<!ATTLIST getPassword
        %XMI.element.att;
        %XMI.link.att;
>

<!-- ____________________________________________________________ -->
<!--                                                              -->
<!-- METAMODEL:  Apendice1                                        -->
<!-- ____________________________________________________________ -->


<!ELEMENT Apendice1 ((ShowCourse | showContent | selectCourse | Actor |
                Student | Teacher | Administrator | Librarian |
                Web-Education-Framework | getLogin | getPassword)*) >

<!ATTLIST Apendice1
        %XMI.element.att;
        %XMI.link.att;
>
```