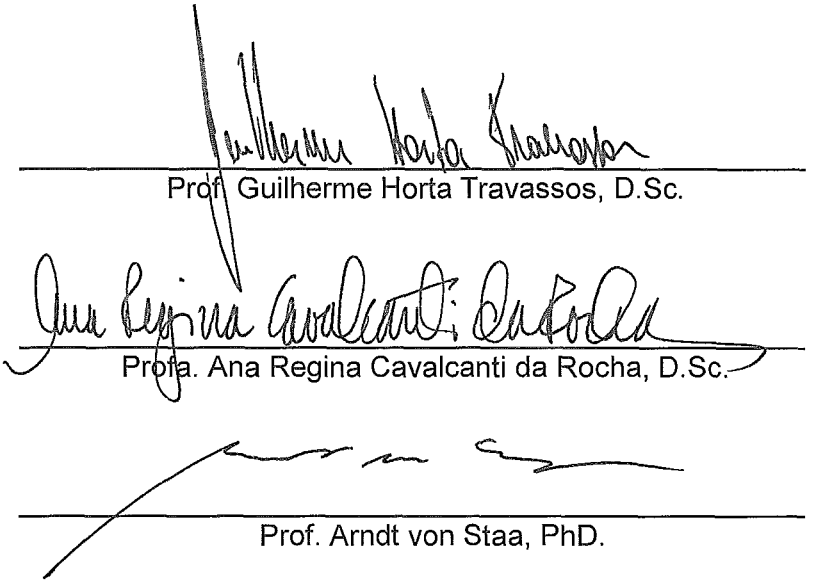
UMA INFRA-ESTRUTURA PARA INTEGRAÇÃO DE FERRAMENTAS CASE BASEADA EM XML, ESQUEMAS E ONTOLOGIAS

Rodrigo Oliveira Spínola

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:



Prof. Guilherme Horta Travassos, D.Sc.

Profa. Ana Regina Cavalcanti da Rocha, D.Sc.

Prof. Arndt von Staa, PhD.

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 2004

SPÍNOLA, RODRIGO OLIVEIRA

Uma Infra-Estrutura para Integração de Ferramentas CASE Baseada em XML, Esquemas e Ontologias [Rio de Janeiro] 2004

VII, 158 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2004)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Integração de Ferramentas
2. Ambientes de Desenvolvimento de Software

I. COPPE/UFRJ II. Título (série)

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UMA INFRA-ESTRUTURA PARA INTEGRAÇÃO DE FERRAMENTAS CASE BASEADA EM XML, ESQUEMAS E ONTOLOGIAS

Rodrigo Oliveira Spínola

Junho/2004

Orientador: Guilherme Horta Travassos

Programa: Engenharia de Sistemas e Computação

Abordagens para integração de ferramentas CASE têm sido desenvolvidas, em particular, na área de ambientes de desenvolvimento de software (ADS). Normalmente, estas abordagens são baseadas no modelo canônico do ADS ou permitem a integração de ferramentas aos pares. Em conjunto com a evolução tecnológica descrita na literatura, configurando novos desafios para a integração de ferramentas, um conjunto de requisitos para a elaboração de uma nova abordagem de integração visando lidar com as heterogeneidades semânticas e estruturais foi definido. Esta abordagem foi implementada na ferramenta xMapper e sua viabilidade avaliada através de um estudo de caso.

Neste contexto, esta Tese apresenta: (1) as dificuldades para integração de ferramentas identificadas na literatura; (2) os requisitos para elaboração de um mecanismo de integração considerando os problemas descritos e a evolução tecnológica; (3) a abordagem para integração de ferramentas proposta; (4) xMapper, uma ferramenta que implementa o mecanismo de integração proposto e, (5) o estudo de caso realizado na análise de viabilidade da abordagem de integração.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

A CASE TOOL INTEGRATION INFRASTRUCTURE BASED ON XML, SCHEMAS AND ONTOLOGIES

Rodrigo Oliveira Spínola

June/2004

Advisor: Guilherme Horta Travassos

Department: Computer Science and System Engineering

CASE Tools integration approaches have been developed into the software development environments (SDE) context. Usually, these approaches are based on the SDE canonical model or peer-application needs. Together with the technological evolution described by the technical literature, we have identified a requirements set to build an innovative integration approach that deals with the semantical and syntactical heterogeneity. Such approach has been implemented by a CASE tool named xMapper. A case study to evaluate its feasibility was also performed.

In this context, this Thesis presents: (1) the tool integration difficulties identified in the literature; (2) the requirements to elaborate an integration mechanism based on difficulties and tecnological evolution; (3) the proposed tool integration; (4) the xMapper, a CASE tool that implements the integration mechanism and, (5) the case study used to evaluate the feasibility of the integration approach.

Aos meus Pais, exemplos de vida.

Aos meus Irmãos, meus melhores amigos.

E à Joice.

Agradecimentos

Primeiramente a Deus, por Seu amor incondicional e por estar comigo em todos os momentos. Nada seria possível sem Ele.

À minha Mãe, Nil, pelo amor, compreensão e apoio irrestrito nas horas mais difíceis.

Ao meu Pai, Edvaldo, pelo carinho, força e por todo exemplo de vida que sempre me deu.

Aos meus Irmãos, Gustavo e Eduardo, pela força e ótima companhia.

À minha Namorada, Joice, pelo amor e carinho. Sua atenção e disponibilidade para as centenas de horas conectados pelo telefone ou internet foram fundamentais.

Aos meus Avós, Tios e Primos pela torcida e apoio constante.

Ao meu Orientador, Guilherme, pela grande dedicação quando mesmo com a agenda apertada sempre conseguia um tempinho para conversarmos. Agradeço também pela orientação, incentivo e por me conduzir durante este trabalho e, por acreditar em mim e no meu trabalho.

Aos professores Ana Regina e Arndt von Staa por participarem de minha banca.

Aos meus Grandes Amigos, Caetano, Gustavo, Márcio, Ricardo, Scheila e Thomaz, virtualmente presentes.

Aos Companheiros de Guerra Ana Cândida, Arilo, Christina, Edgar, Felipe, Gladys, Hamilton, Léo, Marco Antônio, Marco Lopes, Marcos, Paula, Rafael, Sômulo, Tayana e Wladmir, pela amizade, sugestões e ajuda nos momentos que precisei. Em especial a Wladmir pela ajuda nas dúvidas de C++.

Aos Amigos da UNIFACS Atta, Isaac e Manoel aos quais sempre serei grato.

Ao colega Augusto por ter me acolhido em sua casa quando cheguei ao Rio de Janeiro.

À CAPES pelo apoio financeiro.

Sumário

Capítulo 1
Introdução

Neste capítulo são apresentadas as questões que motivaram a realização deste trabalho, seu objetivo principal e a forma como esta dissertação está organizada.

1.1 – Motivação

Aplicações necessitam entender dados disponibilizados por outras aplicações. Porém, fontes de informação representam, muitas vezes, um mesmo conceito do mundo real de maneiras diferentes. Estes conflitos na forma de representação podem ter sua origem, dentre outros, na perspectiva utilizada para a definição da estruturação dos dados e na diferente necessidade de informações relacionadas a um mesmo conceito nas diversas aplicações.

Um cenário típico da necessidade de integração são as ferramentas CASE utilizadas para apoiar as diversas atividades do desenvolvimento de software. É comum duas ferramentas distintas sendo utilizadas para trabalhar com um mesmo artefato (por exemplo, modelo de objetos), mas nada garante que os artefatos estejam estruturados segundo um mesmo formalismo. Também relacionado a este problema, sabe-se que cada ferramenta contribui de forma complementar durante o processo de desenvolvimento de software. Isto implica na necessidade de que muitas das informações geradas por uma ferramenta sejam utilizadas por outra (KARSAI e GRAY, 2000).

A padronização de formatos e conteúdo dos dados é, supostamente, uma solução simples e eficaz para o problema (HASSELBRING, 2000). Entretanto, chegar a um acordo sobre quais conceitos devem estar contemplados e como eles estarão estruturados não é uma tarefa trivial. Visto que os interesses são muitas vezes conflitantes, não existe uma melhor solução (depende do contexto de sua utilização) e ainda assim, os conceitos definidos estão propensos à evolução.

Segundo KARSAI (2000), a tarefa de integração é por natureza complexa e árdua. É um campo de pesquisa presente em diversas áreas da computação

(HASSELBRING, 2000) e muitas pesquisas têm sido realizadas no sentido de encontrar meios para o desenvolvimento de metodologias que permitam a captura e integração da semântica e estrutura distribuídas nos sistemas que se deseja integrar.

Na engenharia de software, abordagens de integração têm sido desenvolvidas, em particular, na área de ambientes de desenvolvimento de software. Existem duas abordagens principais de integração: a baseada em um modelo canônico e a fundamentada na integração de aplicações aos pares. Na perspectiva da utilização de um modelo canônico, todas as ferramentas devem estar projetadas para lidar com um determinado modelo de dados e ainda possuir interfaces para permitir a integração em níveis de apresentação, controle, processo, dado e conhecimento. Caso haja a necessidade de integrar alguma ferramenta que não tenha sido projetada para lidar com o modelo canônico, deve ser feito um mapeamento. Já a abordagem que considera a integração sempre entre pares de aplicações é menos acoplada e, conceitualmente, foi projetada para facilitar o intercâmbio de dados entre aplicações independentes. Entretanto, corre-se um risco muito alto, neste caso, da manutenção do mapeamento dos dados entre as aplicações exigir muito esforço já que há a necessidade de criação de dois conversores para transformação dos dados para cada par de aplicações integradas.

É neste contexto que o trabalho aqui descrito se insere. Ele tenta lidar com os problemas enfrentados pela abordagem do modelo canônico provendo uma solução menos acoplada ao mesmo tempo em que reduz o esforço de manutenção exigido para integração de ferramenta aos pares.

1.2 – Objetivo da Tese

O objetivo fundamental deste trabalho é a definição de uma abordagem baseada no uso de esquemas e ontologias para integração de ferramentas CASE que utilizam artefatos de um mesmo domínio descritos em XML. Desta forma, a proposta deverá lidar com problemas de integração envolvendo a sintaxe, estrutura e semântica dos dados. A proposta é fazer uso das informações semânticas, presentes nas ontologias e, estruturais, presentes nos esquemas, para possibilitar a geração de conversores para transformação de artefatos.

Nesse sentido, a pesquisa inicialmente se direcionou para o estudo de abordagens de integração encontradas na literatura. Analisando estas propostas foi

possível a definição dos problemas envolvendo a integração de dados e, a partir deles, foram definidos os requisitos necessários para uma infra-estrutura de integração. Baseado nestes requisitos, foi elaborada uma abordagem de integração e, definida e implementada uma ferramenta de apoio à integração de artefatos de um mesmo domínio descritos em XML, denominada xMapper. Esta ferramenta se insere no contexto de um conjunto de soluções de apoio à experimentação em engenharia de software da equipe de engenharia de software experimental da COPPE/UFRJ e também intenciona ser integrada em ambientes de desenvolvimento de software, por exemplo, o TABA (TRAVASSOS, 1994).

Por fim, foram realizados alguns estudos de caso com alunos de mestrado da linha de engenharia de software da COPPE/UFRJ visando caracterizar a viabilidade da abordagem proposta nesta tese segundo as variáveis esforço, aderência aos requisitos para integração de dados identificados e perda de informação durante a integração. O resultado do estudo permitiu a avaliação da abordagem proposta e que ajustes fossem feitos no mecanismo de mapeamento.

1.3 – Organização do Texto

Além desta introdução, está tese contém mais seis capítulos e quatro anexos.

No Capítulo 2, *Linguagens de Marcação e a Engenharia de Software*, é feita uma análise da linguagem de marcação XML e de suas correlatas destacando como elas podem ser aplicadas à engenharia de software.

No Capítulo 3, *Integração de Ferramentas*, são discutidas questões envolvendo os problemas e desafios da integração de ferramentas e, são analisadas diversas abordagens de integração enfocando a forma como elas lidam com a integração de dados.

No Capítulo 4, *Requisitos para uma Infra-Estrutura de Integração de Ferramentas CASE com XML*, é feita inicialmente uma análise das dificuldades para integração de ferramentas. A partir da discussão destes pontos, são definidos alguns requisitos para uma infra-estrutura de integração. Então é apresentada uma proposta para integração de ferramentas baseada em XML e um processo definindo as atividades contempladas na proposta.

No Capítulo 5, *xMapper: uma Ferramenta para Integração de Ferramentas CASE com Artefatos em Dados XML*, é apresentado o protótipo desenvolvido para dar apoio às atividades envolvidas na integração.

No Capítulo 6, *Estudo de Caso*, são apresentados alguns conceitos importantes sobre experimentação e como ela se aplica à engenharia de software. Esta introdução ao tema serve também para justificar a importância do capítulo que apresenta o planejamento, a execução e as conclusões alcançadas no estudo de caso elaborado para a avaliação da abordagem proposta no Capítulo 4.

No Capítulo 7, *Considerações Finais*, são apresentadas as conclusões ressaltando suas contribuições, limitações e perspectivas futuras.

No Anexo A, *Escopo Definido da OWL*, é apresentado o sub-conjunto da OWL considerado na construção do mecanismo de integração apresentado no Capítulo 4.

No Anexo B, *Escopo Definido do XML Schema*, é apresentado o sub-conjunto do XML *Schema* considerado na construção do mecanismo de integração apresentado no Capítulo 4.

No Anexo C, *Linguagem para Modelagem de Processos*, é apresentada a linguagem de modelagem utilizada para representar os passos a serem seguidos para a utilização da infra-estrutura de integração.

No Anexo D, *Artefatos Gerados e Utilizados pelo xMapper*, são disponibilizados os artefatos gerados e utilizados no exemplo de uso do mecanismo de integração apresentado no Capítulo 5.

Capítulo 2
Linguagens de Marcação e a Engenharia de Software

Este capítulo descreve e analisa a XML e suas tecnologias correlatas enfatizando seus benefícios para a Engenharia de Software.

2.1 – Introdução

A diversidade de linguagens e protocolos surgidos durante a evolução da Internet permitem ao engenheiro projetar, através da composição destas tecnologias, arquiteturas e ambientes baseados em novos conceitos da computação (BOMPANI *et al.*, 2000). Some-se a isso o fato de que estas linguagens e protocolos continuam em evolução, embora parte deles já esteja com bom nível de amadurecimento. Tendo em mente este cenário, percebe-se que a Internet e as tecnologias surgidas em sua decorrência têm possibilitado muitos avanços em campos variados da ciência, incluindo a Engenharia de Software.

Neste contexto, linguagens de marcação têm sido amplamente estudadas e utilizadas. Duas de suas definições são:

"É um conjunto de símbolos e regras a serem utilizadas ao se fazer a marcação de um documento." (http://www.hyperdictionary.com/)

"É uma linguagem que possui um conjunto de marcadores a serem utilizados para atribuir determinadas características ou significado a uma informação". (CHAUDHRI *et al.*, 2003)

Ambas as definições são equivalentes e, sendo assim, consideradas em conjunto nesta tese.

Embora neste capítulo seja discutida a aplicabilidade de linguagens de marcação na Engenharia de Software, seu escopo (observar Figura 2.1) estará limitado à discussão do padrão e linguagem XML e algumas de suas tecnologias

correlatas. Isto por que dentre as linguagens de marcação existentes, a XML[1] é uma das que mais se destacam devido à sua crescente utilização e grande aplicabilidade na engenharia de software. Essa crescente utilização deve-se basicamente, mas não está limitada, ao fato de:

- estar se tornando o padrão para descrição e intercâmbio de dados entre aplicações e bancos de dados na Internet (FENG *et al.*, 2002);
- possuir caráter semi-estruturado onde dados e metadados (natureza auto-descritiva) são organizados em um único arquivo;
- possibilitar a customização da organização dos dados através de regras de estruturação e semântica definidas pelo usuário;
- o W3C estar trabalhando em tecnologias correlatas para transformação e modelagem de documentos XML e definição de sua semântica;
- ter presença crescente de interfaces XML nas aplicações;
- existirem ferramentas para manipulação de arquivos XML, e;
- ser independente de plataforma. O conteúdo de um documento em XML é texto plano em formato *Unicode*.

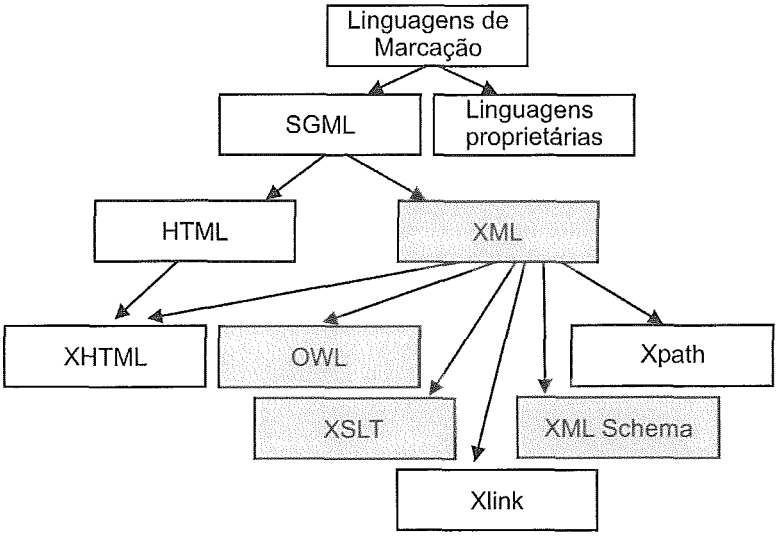


Figura 2.1 – Algumas linguagens de marcação.

A partir de agora veremos os conceitos que fundamentam cada uma das tecnologias em cinza na Figura 2.1 e seu campo de utilização na engenharia de software.

[1] O termo XML é normalmente utilizado para denotar dois significados: a própria linguagem de marcação ou o padrão que especifica o modelo de dados.

2.2 – XML

O padrão XML foi desenvolvido pelo *World Wide Consortium* (W3C) considerando os princípios e convenções da *Stantard Generalized Markup Language* (SGML) (BRYAN, 1992) e provê um simples mas poderoso mecanismo para armazenamento, processamento e distribuição da informação. Quando projetado, alguns dos objetivos a serem alcançados foram (COWAN *et al.*, 2003):

- Facilitar sua utilização na Internet;
- Possuir aplicabilidade abrangente;
- Permitir que programas para processamento de documentos XML fossem facilmente desenvolvidos;
- Permitir que documentos gerados a partir da linguagem XML pudessem ser interpretados tanto por homens quanto por máquinas, e;
- Permitir que documentos XML fossem facilmente criados.

Esses objetivos foram ou vêm sendo rapidamente alcançados. Uma prova disso é o fato do XML estar rapidamente emergindo como o padrão para: (1) intercâmbio de dados na Internet e (2) representação de dados semi-estruturados. Seu potencial também tem sido explorado em outros ramos da comunidade de pesquisa (COLLINS *et al.*, 2002) (HSIAO *et al.*, 2002).

A linguagem XML expressa a informação utilizando basicamente quatro componentes: marcações, atributos, dados e hierarquia. Cada um desses componentes representa uma dimensão diferente da informação e faz parte da sintaxe da XML. A figura 2.2 apresenta um exemplo de documento XML e seus componentes.

O texto em negrito representa os dados contidos no documento. Para atribuir significado a esses dados são utilizadas marcações (texto sublinhado). Os atributos (representados em itálico), por sua vez, são representados na própria marcação e seu objetivo é prover informações sobre como interpretar os dados dentro da marcação na qual ele esteja inserido. Assim, tendo os dados, sabendo o que eles são (marcações) e como interpretá-los (atributos); resta então o componente hierarquia. Este define como os outros três componentes estão relacionados, ou seja, define o contexto no qual os dados estão inseridos.

```
<curriculo>
   <dadosPessoais>
      <nome>Rodrigo Oliveira Spínola</nome>
      <dataNascimento>27/07/1980</dataNascimento>
      <nacionalidade>Brasileira</nacionalidade>
      <endereco tipo="residencial">
         <rua>Barata Ribeiro </rua>
         <complemento7></complemento>
         <bairro>Copacabana</bairro>
         <cidade>Rio de Janeiro</cidade>
         <estado>RJ</estado>
         <pais>Brasil</pais>
      </endereco>
      <endereco tipo="profissional">
         <rua></rua>
         <complemento></complemento>
         <bairro>Ilha do Fundão</bairro>
         <cidade>Rio de Janeiro</cidade>
         <estado>RJ</estado>
         <pais>Brasil</pais>
      </endereco>
   </dadosPessoais>
</curriculo>
```

Figura 2.2 – Exemplo de documento XML.

Ainda sobre a sintaxe de documentos XML, pode-se perceber na Figura 2.2 que as marcações ocorrem aos pares para que seu conteúdo seja definido internamente e que deve existir um único elemento raiz, neste caso, *curriculo*.

Embora a XML seja definida como uma linguagem de marcação, pode também ser considerada uma meta-linguagem. Isto porque permite definir novas linguagens de marcação, a partir da criação de um conjunto de marcadores, geralmente associadas a um contexto específico. Alguns exemplos são:

- ebXML[2] (Eletronic Business eXtensible Markup Language): é uma especificação que define um padrão baseado no XML para possibilitar que sistemas de comercio eletrônico de diferentes organizações possam se comunicar.
- NewsML[3]: é um padrão para representação e gerenciamento de notícias durante seu ciclo de vida.

[2] Maiores detalhes consultar http://www.ebxml.org/.
[3] Maiores detalhes consultar http://www.newsml.org/.

Como uma linguagem de marcação que suporta a definição de marcadores pelo usuário e separa o conteúdo do documento da forma como ele será apresentado possibilitando a reutilização da informação, XML ainda facilita a automatização do processamento da informação - requisito básico para aplicações que constantemente manipulam dados como sistemas de comércio eletrônico.

2.2.1 Modelando Documentos XML

O fato de documentos XML serem usados não apenas para armazenar dados mas também, por exemplo, para manter informações de configuração de ambientes e semântica de um determinado domínio, implica na necessidade de defini-los de forma mais criteriosa. Entretanto, a linguagem XML se limita a definir o que podemos fazer, por exemplo: criar novas marcações, atributos, especificar relações de hierarquia. Não define como utilizar estes elementos em conjunto para termos um documento estruturado consistente com suas necessidades. Assim, além da preocupação com a sintaxe (regras de formação de um documento XML) é preciso estar atento à sua modelagem pois é justamente nesta etapa que decisões importantes deverão ser tomadas.

Documentos XML modelados corretamente garantem bom desempenho das aplicações que o manipulam, principalmente quando atividades de armazenamento, busca e gerência da informação são consideradas. Não é difícil construir corretamente documentos XML. O problema se dá pelo fato da linguagem XML ser bastante flexível tornando possível sua má utilização. A tabela abaixo apresenta alguns erros cometidos e suas possíveis conseqüências (CHAUDHRI *et al.*, 2003):

	Erros cometidos	Conseqüências
a	Uso incompleto de marcações	Definição incorreta do contexto de um elemento de dado
b	Uso incompleto de atributos	Elementos de dados podem ser interpretados de maneira incorreta
c	Uso incorreto de atributos	Atributos podem assumir o papel de um elemento de dado
d	Uso de possíveis elementos de dados como metadados	As marcações são definidas incorretamente causando problemas para interpretação do documento
e	Má construção da hierarquia	Definição de marcações desnecessárias, incorretas ou redundantes

Tabela 2.1 – Alguns problemas comuns na definição de um documento XML.

Uma sugestão (CHAUDHRI *et al.*, 2003) para evitar que problemas como estes ocorram é a aplicação de um conjunto de questionamentos no momento em que o documento esteja sendo modelado (a letra no final de cada questão associa-a ao tipo de problema apresentado na tabela 2.1):

- Este elemento é um dado ou metadado? (b,d)
- Este atributo auxilia na interpretação ou representa um elemento de dado? (c)
- O contexto para estes elementos de dados está bem definido? (a,b,e)
- Esta marcação está de acordo com o elemento de dado inserido nela? (a)
- Todos os membros de um grupo estão relacionados com o conceito descrito pelo nó pai? (b,e)
- Existe algum tipo de ambigüidade na definição da hierarquia? (a,b,e)

Para esclarecer um pouco mais esses pontos, seguem dois exemplos envolvendo alguns dos problemas relatados e como as questões acima auxiliam evitá-los. Vejamos a Figura 2.3 (gerado a partir de algumas alterações do XML apresentado na Figura 2.2):

```
<curriculo>
   <dadosPessoais>
      <nome>Rodrigo Oliveira Spínola</nome>
      <dataNascimento>27/07/1980</dataNascimento>
      <nacionalidade>Brasileira</nacionalidade>
      <endereco tipo="residencial" rua="Barata Ribeiro"
       complemento="" bairro="Copacabana" cidade="Rio de
       Janeiro" estado="RJ" pais="Brasil"/>
   </dadosPessoais>
</curriculo>
```

Figura 2.3 – Problemas na definição de atributos.

Analisando o marcador *endereco,* podemos perceber que foi definido um conjunto de atributos incorretamente. Um atributo deve ser definido apenas quando ele agrega alguma informação que auxilie na interpretação de um conjunto de nós, filhos do nó ao qual ele faz parte. Neste exemplo, dos atributos definidos, o único utilizado corretamente é *tipo*. Todos os outros devem estar representados como filhos de *endereco* uma vez que representam elemento de dados e não informação sobre esses

elementos. Um possível questionamento utilizado para identificar este problema seria: *Este elemento é um elemento de dado ou metadado?*

Para o segundo exemplo será utilizado o fragmento de documento XML da Figura 2.4.

```
<curriculo>
   <dadosPessoais>
      <nome>Rodrigo Oliveira Spínola</nome>
      <dataNascimento>27/07/1980</dataNascimento>
      <nacionalidade>Brasileira</nacionalidade>
      <endereco tipo="residencial"> </endereco>
      <rua>Barata Ribeiro </rua>
      <complemento></complemento>
      <bairro>Copacabana</bairro>
      <cidade>Rio de Janeiro</cidade>
      <estado>RJ</estado>
      <pais>Brasil</pais>
   </dadosPessoais>
</curriculo>
```

Figura 2.4 – Má construção da hierarquia.

Neste caso, a má definição da hierarquia causa, basicamente, dois problemas: ambigüidade e fraca coesão na definição dos conceitos. Ambigüidade pelo fato do entendimento da informação contida no documento estar comprometida: o que deve ser colocado como elemento de dado do marcador *endereco*? Fraca coesão pelo fato dos elementos estarem mal distribuídos, sendo definidos em locais que não lhes são próprios, prejudicando a semântica do documento. Uma questão que poderia ser feita para identificar esse tipo de problema seria: *O contexto para estes elementos de dados está bem definido?*

2.2.2 Manipulação de Documentos XML

Para uma aplicação processar o conteúdo de documentos XML, seus elementos devem ser antes identificados e disponibilizados. Para isto, existem duas interfaces de programação de aplicação (API) que podem ser utilizadas: *Document Object Model* (DOM) e *Simple API for XML* (SAX) (HORS, 2003) (MARCHAL, 2000).

DOM é uma API que permite aos programas acessar e atualizar dinamicamente o conteúdo e a estrutura de um documento XML (APPARAO *et al.*, 1998). Para isto, ela define: (1) uma estrutura lógica em árvore na memória a partir dos elementos de um documento e (2) a forma como eles serão acessados e

manipulados. A estrutura em árvore é utilizada pelo fato de um documento XML possuir uma estrutura hierárquica.

A Figura 2.5 apresenta parte da estrutura em árvore gerada e mantida em memória pelo DOM quando o documento XML da Figura 2.2 estiver sendo manipulado. Nesta figura os retângulos representam os objetos gerados a partir dos elementos, losangos seus atributos e as ovais, o conteúdo. Assim, para acessar cada elemento do documento basta percorrer a árvore montada.

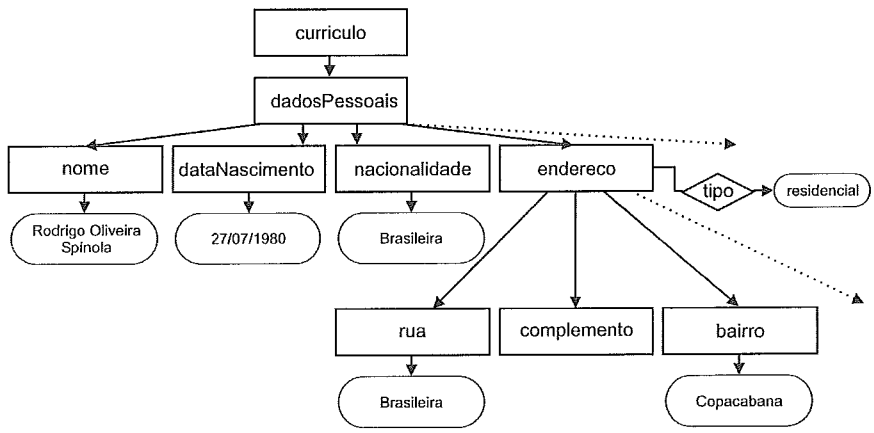


Figura 2.5 – Fragmento da árvore gerada pelo DOM.

Já o SAX faz uma leitura seqüencial do documento e gera eventos, para cada elemento do documento acessado. A partir desses eventos, cabe à aplicação interpretá-los e montar uma estrutura de dados para manipular o documento. A Tabela 2.2 apresenta parte da seqüência de eventos gerados quando uma aplicação fizer a leitura do fragmento de um documento XML utilizando a API SAX (MARCHAL, 2000).

Documento XML	Eventos Gerados

<?xml version="1.0"?>
<curriculo>
  <dadosPessoais>
  <nome>**Rodrigo Oliveira Spínola**</nome>
  ...
  <endereco *tipo="residencial"*>
  <rua>**Barata Ribeiro** </rua>
  ...
  <pais>**Brasil**</pais>
  </endereco>
  </dadosPessoais>
</curriculo>
	Início do documento
	Início do elemento currículo
	Início do elemento dadosPessoais
	Início do elemento nome
	Fim do elemento nome
	...
	Início do elemento endereco
	Início do elemento rua
	Fim do elemento rua
	...
	Início do elemento pais
	Fim do elemento pais
	Fim do elemento endereco
	Fim do elemento dadosPessoais
	Fim do elemento Currículo

Tabela 2.2 – Seqüência de eventos gerados pela API SAX.

As principais diferenças entre DOM e SAX estão relacionadas:

- às suas origens: diferente do DOM, o SAX não foi uma iniciativa do W3C;
- ao escopo: o SAX foi projetado para lidar, especificamente, com documentos XML. Por este motivo, ele possui algumas funcionalidades não contempladas no DOM (projetado inicialmente para o HTML);
- ao estilo de programação: para propiciar a manipulação de documentos XML, o DOM cria uma estrutura em árvore na memória refletindo a organização hierárquica do documento enquanto que o SAX gera um conjunto de eventos para cada elemento acessado durante a leitura do documento.

Estudos que analisam as vantagens e desvantagens de cada alternativa têm sido feitos. Shekhar *et al.*(2001) apresentam as conclusões de um experimento envolvendo as duas interfaces. O objetivo do estudo foi verificar a influência da quantidade de nós a serem acessados e a aleatoriedade desses acessos no tempo de resposta para processamento de documentos XML. A importância deste estudo se deve ao fato da medição do tempo de resposta no processamento do documento XML sinalizar qual API utilizar em contextos específicos.

Para chegar às conclusões, o tempo de resposta foi medido em duas situações. Na primeira, o número de elementos do documento permanecia constante enquanto a quantidade de vezes que seus elementos eram acessados variava. Para este caso, mediu-se o tempo de resposta para duas quantidades de elementos distintas. Os resultados podem ser vistos nos gráficos das Figuras 2.6a e 2.6b.

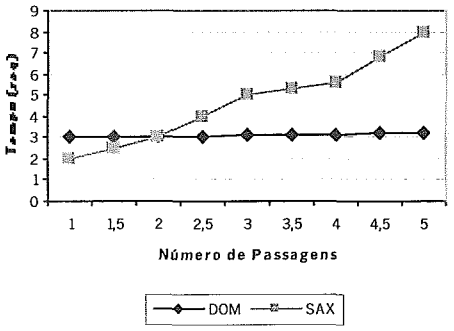


Figura 2.6a

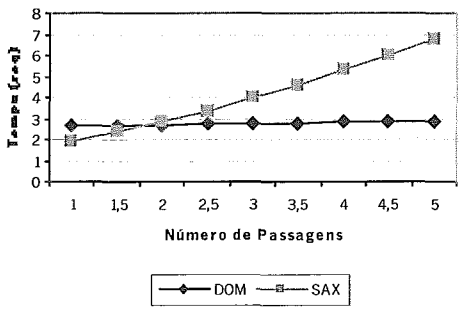


Figura 2.6b

Para a segunda situação, o número de elementos do documento variava enquanto a quantidade de acesso a eles permanecia constante (para dois patamares). Os resultados podem ser vistos nos gráficos das figuras 2.7a e 2.7b.

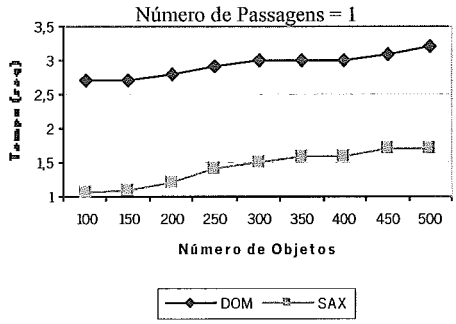


Figura 2.7a

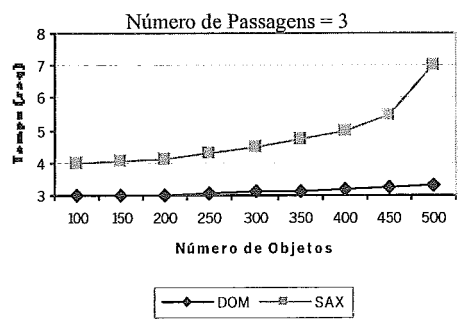


Figura 2.7b

Cada uma das abordagens possui suas vantagens e limitações. A partir dos gráficos apresentados pode-se perceber que o SAX dificulta o acesso aleatório aos diferentes elementos de um documento XML. Isto por que o SAX acessa cada elemento apenas uma vez durante a leitura do documento e caso o usuário queira retornar a um determinado nó é preciso fazer novamente a leitura do documento até encontrar o elemento desejado ou, montar uma estrutura de dados para manipular os elementos do documento capturados durante a primeira leitura (esforço desnecessário já que o DOM faz isso).

O principal problema do DOM é seu alto consumo de memória, já que uma estrutura em árvore contendo todos os elementos do documento XML deve ser montada em memória para que a navegação nos nós possa ser feita.

Uma decisão equivocada de qual abordagem utilizar pode resultar em esforço de implementação desnecessário (quando, por exemplo, acessos aleatórios são comuns e estar se utilizando o SAX) ou baixo desempenho na manipulação dos dados (quando, por exemplo, o tamanho do documento manipulado é grande e estar se utilizando o DOM).

2.2.3 Aplicações do XML na Engenharia de Software

Existem dois aspectos distintos que caracterizam o relacionamento entre essas duas áreas. O primeiro diz respeito à utilização da linguagem XML e tecnologias relacionadas na construção de ambientes e ferramentas de engenharia de software. O segundo foca a exploração do XML no desenvolvimento de uma nova geração de arquiteturas de software e *middlewares* criando novas possibilidades em termos de integração, segurança e interoperabilidade de sistemas (MASCOLO *et al.*, 2001).

A engenharia de software faz uso de ferramentas, técnicas, procedimentos e paradigmas objetivando aumentar a qualidade dos softwares produzidos (PFLEEGER, 2001). As ferramentas utilizadas durante o processo de desenvolvimento lidam, muitas vezes, com os mesmos conceitos embora manipulem os dados de forma independente. A principal conseqüência disto é um conjunto de ferramentas que não são capazes de compartilhar informação. Neste contexto, a XML permite ao engenheiro de software definir um padrão comum de como os diversos artefatos gerados durante o ciclo de desenvolvimento serão descritos possibilitando uma maior interoperabilidade entre as ferramentas. Um exemplo é o uso do XMI (*XML Metadata Interchange*) como padrão para intercãmbio de artefatos de projetos de sofwatre. XMI define um esquema para a criação de arquivos XML que representam diagramas UML (*Unified Modeling Language*).

O desenvolvimento de software para a internet também vem sofrendo alterações. Sabe-se que a interoperabilidade entre aplicações e repositórios heterogêneos é necessária para este tipo de software. O XML vem sendo utilizado como um meio para permitir esta integração.

Documentação é outra área de grande aplicação da linguagem XML. Engenheiros de software fazem uso de documentação como um mecanismo para incrementar o entendimento dos diversos níveis de abstração envolvidos durante o desenvolvimento de sistemas (HARTMANN *et al.*, 2001). Sem esta documentação, a única fonte de informação seria o código fonte. Entretanto analisar código fonte leva tempo e torna a análise bastante susceptível a erros principalmente quando se considera a quantidade de informação a ser assimilada e o conhecimento do domínio necessário. Neste contexto, pode-se perceber que documentações de projetos de software seguem, muitas vezes, estruturações pré-definidas. Estas são geralmente definidas a partir de práticas já consolidadas na comunidade de engenharia de software.

Um exemplo disto é o padrão para especificação de requisitos definidos em (IEEE Std 830-1998, 1998). Para este caso, poderiam ser definidos um conjunto de marcações especificando as diversas informações a estarem contempladas em uma especificação de requisitos e as regras de estruturação destes marcadores (ver seção 2.3). Uma justificativa para utilizar XML neste contexto seria o ganho em facilidade para manipular um documento de requisitos. Um exemplo de aplicação seria a construção de ferramentas para apoiar a inspeção de documentos de requisitos.

Existem ainda outras áreas em que é possível utilização da XML mas, por estarem fortemente relacionadas com suas tecnologias correlatas, serão descritas nas próximas seções.

2.3 – XML *Schema*

Documentos XML podem ser classificados em bem formados ou válidos. Um documento é considerado bem formado quando segue as regras definidas pela sintaxe da XML. Para ser válido, além de seguir a sintaxe da XML, é necessário que o documento esteja de acordo com algum esquema previamente definido. O objetivo do XML *Schema* é, justamente, definir os blocos legais de construção de um documento XML. Neste esquema podem ser definidos: os elementos presentes no documento, os atributos que podem aparecer, quais elementos são filhos de outros elementos, a seqüência em que os elementos filhos podem ser utilizados, a quantidade de elementos filhos, se um elemento pode ser vazio ou conter texto, os tipos de dados para os elementos e atributos e valores padrão para atributos dentre outros.

O XML *Schema* foi originalmente proposto pela Microsoft e é, atualmente, uma recomendação do W3C. Alguns pontos que têm apoiado sua utilização são:

- suporte a tipo de dados facilitando descrever o conteúdo do documento;
- a utilização da sintaxe da linguagem XML para definir o esquema. Isto possibilita sua manipulação utilizando os mesmos princípios de um documento XML além de permitir a aplicação de regras de transformação definidas em XSLT (ver seção 2.4).

A Figura 2.8 apresenta um exemplo de definição de esquema para o documento XML da Figura 2.2. Os conceitos básicos que devem ser entendidos são os de elemento (*element*) e tipo complexo (*complexType*). A marcação *element* deve aparecer para cada elemento do documento XML. É nela que são definidas as possíveis propriedades de cada nó de um documento XML como: nome (*name*), tipo (*type*), atributos (*attribute*) e cardinalidade (*minOccurs, maxOccurs* e *occurs*). Tendo identificado os elementos, é preciso definir seu tipo. Este pode ser um tipo simples (string, data, inteiro) ou complexo. Neste último caso, utilizamos a marcação *complexType* para especificar como é estruturado o elemento em questão (FALLSIDE, 2001).

Assim, para o documento da Figura 2.8 temos o seguinte entendimento. Existe um elemento raiz (*curriculo*) do tipo complexo que deve ter como filho um elemento *dadosPessoais*. Este por sua vez, é também do tipo complexo e é composto por uma seqüência de filhos cujos nomes das marcações são *nome* (do tipo *string*), *dataNascimento* (do tipo *date*), *nacionalidade* (do tipo *string*) e *endereco* (de tipo complexo). Este último pode ocorrer várias vezes (*maxOccurs*). Mais abaixo se encontra a definição do elemento *endereco* o qual possui o atributo *tipo* e uma seqüência de elementos (*rua*, *complemento*, *bairro*, *cidade*, *estado* e *país*) do tipo *string*.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
    <xs:element name="curriculo">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="dadosPessoais"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="dadosPessoais">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="nome" type="xs:string"/>
                <xs:element name="dataNascimento" type="xs:date"/>
                <xs:element name="nacionalidade" type="xs:string"/>
                <xs:element ref="endereco" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="endereco">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="rua" type="xs:string"/>
                <xs:element name="complemento" type="xs:string"/>
                <xs:element name="bairro" type="xs:string"/>
                <xs:element name="cidade" type="xs:string"/>
                <xs:element name="estado" type="xs:string"/>
                <xs:element name="pais" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="tipo" use="required" type="xs:string"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figura 2.8 Exemplo de XML *Schema*.

2.3.1 Aplicações do XML *Schema* na Engenharia de Software

Por ser uma linguagem para modelagem de documentos XML, o XML *Schema* pode ser utilizado para documentação de artefatos XML. Atualmente também tem se estudado muito sobre boas práticas para definição de documentos XML (ELMASRI *et al.*, 2002).

Por definir como um arquivo XML estará organizado estruturalmente, o XML *Schema* é também o ponto de partida para criação de mecanismos para integração de ferramentas e mapeamento de documentos XML para uma base de dados relacional (BOURRET, 2001) (HA, 2001). Este último ponto ganha destaque uma vez que é preciso criar meios de integrar dados legados com o novo padrão para intercâmbio de informação, XML.

2.4 – XSLT

A XSLT (XML *StyLesheeT*) é uma linguagem para definição de regras de transformação em documentos XML. Estas regras são definidas em folhas de estilo (do inglês *stylesheet*) seguindo a sintaxe da linguagem XML.

O termo folha de estilo é utilizado pelo fato de que um dos principais papeis do XSLT é adicionar informações de apresentação a um documento XML através de sua transformação para uma linguagem orientada à apresentação como HTML, XHTML (*eXtensible* HTML) ou SVG (*Scalable Vector Graphics*). Estas transformações são executadas por um processador XSLT cuja responsabilidade é aplicar as regras definidas na folha de estilos aos nós correspondentes de um documento XML. Dentre as possíveis transformações, uma que vem ganhando importância e não está relacionada com formatação para apresentação são as de um documento XML para outro (KAY, 2003).

O princípio de funcionamento do XSLT (observar Figura 2.9) segue os seguintes passos: (1) um documento XML é enviado para o processador XSLT; (2) o processador XSLT utiliza um documento XSLT no qual estão presentes as informações sobre como deverá ser feita a conversão do documento; (3) a partir destas informações, um novo documento é gerado pelo processador.

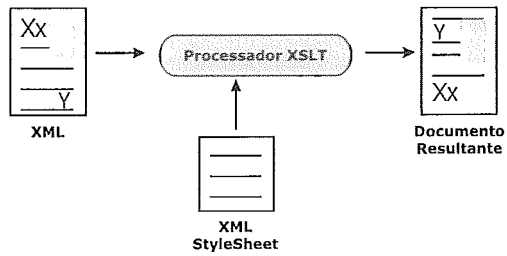


Figura 2.9 – Princípio de funcionamento do XSLT.

A definição das regras de transformação baseia-se no conceito de *templates*. Nestes são especificadas as mudanças a serem efetuadas em um determinado conjunto de marcações. Tendo as regras definidas (nos *templates*) basta, então, procurar no documento XML onde (cláusula *match*) elas serão aplicadas e efetuar as transformações estabelecidas (KAY, 2003).

Para demonstrar a funcionalidade do XSLT, consideremos a folha de estilos da Figura 2.10. Existem três *templates*: raiz (encontra o nó *curriculo*), *dados* e *nome*. Dentro de cada um deles são descritas as alterações que estarão presentes no documento gerado e informações que auxiliarão o processador XSLT. Pontos que merecem destaque são as palavras *match*, *call-template*, *value-of* e *select*. O *match* é utilizado para informar ao processador a qual elemento do documento XML será aplicado o *template* em questão. O *call-template* é usado para encadear a ordem na qual os diversos *templates* serão chamados durante a aplicação da folha de estilos. Já as palavras *value-of* e *select* podem ser utilizadas em conjunto para recuperar o valor de um determinado elemento XML e colocá-lo no documento gerado a partir das transformações especificadas.

Assim, o documento apresentado na Figura 2.10 especifica as seguintes informações a serem utilizadas pelo processador XSLT no documento apresentado na Figura 2.2 para geração do documento HTML da Figura 2.11:

- **Linhas 1 e 2:** especificação que o documento em questão é uma folha de estilo e definição de mais alguns parâmetros considerados pelo processador XSLT;
- **Linhas 3 a 10:** definição do *template* a ser aplicado quando o elemento raiz (*curriculo*) for encontrado;
- **Linha 7:** chamada para o próximo *template* a ser aplicado. Neste caso, o nome do próximo *template* é *dados*;
- **Linhas 11 a 15:** definição do *template* a ser aplicado quando o elemento dadosPessoais for encontrado;

- **Linha 13:** chamada para o próximo *template* a ser aplicado. Neste caso, o próximo *template* é *nome*;
- **Linhas 16 a 21:** definição do *template* a ser aplicado quando o elemento *nome* for encontrado;
- **Linha 19:** recupera o valor da marcação *nome* do documento XML e insere-o no documento HTML que está sendo gerado.

```
1: <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
2:  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
3.  <xsl:template match="curriculo">
4:      <html>
5:          <head/>
6:          <body>
7:              <xsl:call-template name="dados"></xsl:call-template>
8:          </body>
9:      </html>
10: </xsl:template>
11: <xsl:template  name="dados" match="./dadosPessoais">
12:     <table width="566" border="1" align="left">
13:         <xsl:call-template name="nome"/>
14:    </table>
15: </xsl:template>
16: <xsl:template  name="nome" match="./nome">
17:    <tr>
18:         <td>Nome</td>
19:        <td><xsl:value-of select="//nome"/></td>
20:    </tr>
21: </xsl:template>
22:</xsl:stylesheet>
```

Figura 2.10 Exemplo de um documento XSLT.

```
<html>
    <head/>
    <body>
        <table width="566" border="1" align="left">
            <tr>
                <td>Nome</td>
                <td>Rodrigo Oliveira Spínola</td>
            </tr>
        </table>
    </body>
</html>
```

Figura 2.11 Documento HTML gerado

2.4.1 Aplicações do XSLT na Engenharia de Software

Por ser uma tecnologia correlata ao XML, o XSLT é aplicado à engenharia de software apoiando o emprego do XML. Um exemplo bastante interessante está relacionado aos artefatos de software.

Sabemos que os artefatos gerados durante o ciclo de desenvolvimento do software possuem diferentes audiências. Isto implica na necessidade de representação da documentação em diferentes níveis de abstração para, por exemplo, o usuário da ferramenta, o desenvolvedor, a equipe de teste e o gerente de projeto. Cada um desses usuários desempenha tarefas que tendem a se complementar embora, na maioria das vezes, possuam sua própria visão e entendimento sobre um mesmo artefato. Então se faz necessário criar mecanismos que permitam combinar as diversas perspectivas com objetivo de gerar versões dos documentos compatíveis com o papel desempenhado pelo usuário do artefato (HARTMANN *et al.*, 2001). Tendo a documentação descrita em XML, é possível utilizar o XSLT para gerar os artefatos para uma perspectiva desejada.

Um outro exemplo de aplicação do XSLT está relacionado com a integração de ferramentas. Este assunto será discutido em maiores detalhes no capítulo 4.

2.5 – Web Ontology Language (OWL)

A OWL é uma linguagem para definição de ontologias fundamentada na tecnologia XML (HARMELEN *et al.*, 2003). O termo ontologia teve sua origem na Filosofia onde era utilizada para tentar descrever e, algumas vezes, desvendar domínios naturais. Já na Inteligência Artificial, ontologias têm um caráter um pouco distinto; são utilizadas para descrever domínios já consagrados como Medicina e Engenharia. Assim, o que se busca com uma ontologia em Inteligência Artificial é firmar um acordo sobre o vocabulário do domínio de interesse, a ser partilhado por agentes que conversam sobre ele (FALBO, 1998).

Existem muitas definições para o termo ontologia. Uma das mais conhecidas é a definição feita por Gruber (1995): "Uma ontologia é uma especificação explícita e formal de uma conceituação compartilhada". Neste contexto, conceituação significa uma forma abstrata de como uma pessoa entende as coisas do mundo. Dizer que a conceituação é explícita é afirmar que os conceitos e relacionamentos de um modelo abstrato são definidos através de termos e definições explícitas. Ou seja, ontologia é

uma descrição de conceitos e relações que podem existir para um agente ou uma comunidade de agentes. Além disso, uma ontologia não descreve apenas conhecimento imediato, isto é, conhecimento que pode ser obtido diretamente a partir da observação do domínio, descreve também conhecimento derivado (aquele obtido através de inferência sobre o conhecimento imediato disponível) (FALBO, 1998).

Em geral, elas são utilizadas por pessoas, bancos de dados e sistemas que necessitam compartilhar informações de um domínio[4]. Em detrimento às particularidades de cada abordagem, seus objetivos iniciais eram (GRUNINGER e LEE, 2002):

- ajudar na compreensão de uma certa área de conhecimento;
- ajudar pessoas a atingir um consenso no seu entendimento sobre uma área de conhecimento;
- permitir a comunicação entre aplicações, pessoas e entre aplicações e pessoas;
- possibilitar inferência computacional;
- permitir o reuso e a organização do conhecimento via sua estruturação e organização em bibliotecas ou repositórios de planos, planejamentos e informações de domínio.

Atualmente, pesquisas envolvendo ontologias são bastante difundidas e sua importância é reconhecida por diversos grupos de pesquisa e áreas de aplicação incluindo engenharia do conhecimento, projeto e integração de bancos de dados, entre outros. Devido às peculiaridades de cada uso, Guarino (1998) classificou-as em:

- *ontologias genéricas*: descrevem conceitos bastante gerais como, espaço, tempo, matéria, objeto, evento, ação, etc., que são independentes de um problema ou domínio particular. Isto é, procuram construir teorias básicas do mundo, de caráter abstrato e aplicáveis a qualquer domínio;
- *ontologias de domínio*: expressam conceituações de domínios particulares, descrevendo seu vocabulário;
- *ontologias de tarefas*: expressam conceituações sobre a resolução de problemas, independentemente do domínio em que ocorram. Isto é, descrevem o vocabulário relacionado a uma atividade ou tarefa genérica,

[4] Domínio é um conceito importante que norteia qualquer principio para atividades ou objetos do mundo. Por exemplo, sistemas envolvendo software são desenvolvidos em um contexto particular determinado a

tal como, diagnose ou vendas. Segundo Falbo (1998), a principal motivação deste tipo é facilitar a integração dos conhecimentos de tarefa e domínio em uma abordagem mais uniforme e consistente.

- *ontologias de aplicação*: descrevem conceitos dependentes do domínio e de tarefa particulares. Estes conceitos freqüentemente correspondem a papéis desempenhados por entidades do domínio durante a realização de uma atividade;
- *ontologias de representação*: explicam as conceituações que fundamentam os formalismos de representação de conhecimento. Ou seja, conceitos e relacionamentos definidos em outras ontologias são considerados instanciações e conceitos contemplados nas ontologias de representação.

Independente de sua classificação e domínio, a complexidade envolvida na construção de ontologias é grande e, portanto, algum mecanismo deve ser usado para facilitar seu desenvolvimento e posterior avaliação (GRUBER,1995). Alguns itens que devem ser considerados durante sua construção são:

- **Clareza:** uma ontologia deve comunicar efetivamente o significado planejado dos termos definidos. As definições devem ser objetivas, documentadas em linguagem natural e independente do contexto social ou computacional.
- **Coerência:** uma ontologia deve ser coerente, isto é, deve comportar apenas inferências consistentes com as definições. Coerência deve ser observada, também, em relação a conceitos definidos informalmente. Se uma sentença for passível de ser inferida a partir dos axiomas da ontologia contradiz uma definição ou exemplo dado informalmente, então a ontologia é incoerente.
- **Extensibilidade:** Uma ontologia deve ser projetada para antecipar usos do vocabulário compartilhado e, portanto, sua representação deve possibilitar extensão e especialização.
- **Generalidade:** uma ontologia deve ser capaz de ser compartilhada entre atividades distintas como projeto e análise;
- **Completude:** a completude de uma definição em uma ontologia depende do nível de granularidade da ontologia;
- **Concisão:** uma ontologia deve capturar apenas as informações necessárias e suas definições. Redundâncias devem ser, assim, evitadas;

---

partir de vários fatores como a organização para a qual está sendo desenvolvido, o problema que busca

- **Compromissos de codificação mínimos:** A conceituação deve ser especificada no nível de conhecimento sem depender de uma tecnologia particular para representação deste;
- **Compromissos ontológicos mínimos:** uma ontologia deve ser suficiente para apoiar as atividades pretendidas de compartilhamento de conhecimento. Isto é obtido através da especificação de uma teoria considerando apenas a definição dos termos que são essenciais para a comunicação de conhecimento consistente com esta teoria e, definição mínima de restrições possíveis sobre o mundo que está sendo modelado.

De maneira geral, qualquer que seja o domínio, a complexidade envolvida na construção de ontologias é grande e, portanto, deve existir algum mecanismo que torne possível e facilite sua definição.

2.5.1 OWL

Criada pelo W3C, a OWL é uma forte candidata a padrão para descrição de ontologias na web. Atualmente, o W3C provê três subconjuntos da linguagem OWL que variam de acordo com seu poder de expressão. O objetivo desta classificação é limitar a complexidade de definição de ontologias para atender aos tipos variados de usuários. Os subconjuntos são:

- **OWL Lite:** suporta as necessidades primárias dos usuários provendo mecanismos de classificação em hierarquia e algumas características simples de restrição. Por exemplo, este subconjunto suporta apenas valores de cardinalidade iguais a 0 ou 1.
- **OWL DL:** suporta os usuários que necessitam de forte poder de expressão limitando este poder ao fato da definição da ontologia ser computacionalmente completa e decidível.
- **OWL Full:** suporta os usuários que necessitam de forte poder de expressão mas não limita este poder ao fato da definição da ontologia ser computacionalmente completa e decidível.

---

atender e mesmo a tecnologia que utiliza (OLIVEIRA, 1999).

Cada uma dessas sublinguagens é extensão de uma mais simples. Assim, valem as seguintes relações:

- Toda ontologia definida em OWL Lite é uma ontologia definida em OWL DL;
- Toda ontologia definida em OWL DL é uma ontologia definida em OWL Full.

Independente de sua classificação, para propiciar a definição de ontologias, ela faz uso basicamente de três elementos: classes, propriedades e instâncias. Classes são os conceitos a serem contemplados, propriedades são os atributos destes conceitos e instâncias são classes com os valores de seus atributos especificados. A Figura 2.12 apresenta um fragmento de uma ontologia definida em OWL. Neste exemplo, é definida, parcialmente, a semântica de um currículo. Este é composto basicamente de três conceitos (em negrito): *curriculo*, *dadosPessoais* e *endereco*. São definidas então restrições (sublinhado) que nos informam como eles estão relacionados entre si. Por fim são definidas as propriedades (em itálico) com suas respectivas definições, tipos e conceitos às quais fazem parte. Por exemplo, a propriedade *nacionalidade* faz parte (*domain*) do conceito *dadosPessoais*.

Os conceitos são definidos através do marcador *class* e para cada um podem ser definidas, dentre outras:

- restrições sobre suas propriedades, marcador *restriction*. As restrições podem ser de valor e/ou cardinalidade;
- relações de interseção (*intersectionOf*), complemento (*complementOf*) e união (*unionOf*) com outros conceitos;
- axiomas que representam relações de sub conceitos (*subClassOf*), conceitos equivalentes (*equivalentClass*) e conceitos disjuntos (*disjointWith*).

É importante perceber que em OWL as propriedades não são definidas nos conceitos dos quais faz parte. São definidas separadamente através do marcador *ObjectProperty* e associadas aos conceitos através do marcador *domain*. Essa é uma característica interessante pois evita a repetição desnecessária de definições de propriedades. Para cada propriedade podem ser definidas, dentre outras:

- características lógicas como simetria (*SymmetricProperty*) e transitividade (*TransitiveProperty*);
- relacionamento com outras propriedades através de associações de equivalência (*equivalentProperty*) e inversão (*inverseOf*).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF xml:lang="en" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:map="http://www.rodrigospinola.com">
    <owl:Ontology rdf:about="http://www.cos.ufrj.br/~ros/template#">
        <rdfs:comment>Descreve os conceitos de um curriculo</rdfs:comment>
        <owl:versionInfo>1.0</owl:versionInfo>
    </owl:Ontology>
    <owl:Class rdf:ID="curriculo">
        <owl:Restriction>
            <owl:onProperty rdf:resource="#dados"/>
            <owl:cardinality>1</owl:cardinality>
            <owl:allValuesFrom rdf:resource="#dadosPessoais"/>
        </owl:Restriction>
    </owl:Class>
    <owl:Class rdf:ID="dadosPessoais">
        <owl:Restriction>
            <owl:onProperty rdf:resource="#endereco"/>
            <owl:minCardinality>1</owl:minCardinality>
            <owl:allValuesFrom rdf:resource="#endereco"/>
        </owl:Restriction>
    </owl:Class>
    <owl:Class rdf:ID="endereco"/>
    <owl:ObjectProperty rdf:ID="nome">
        <rdfs:domain rdf:resource="#dadosPessoais"/>
        <rdfs:range rdf:resource="#String"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="dataNascimento">
        <rdfs:domain rdf:resource="#dadosPessoais"/>
        <rdfs:range rdf:resource="#Date"/>
    </owl:ObjectProperty>
    ...
    <owl:ObjectProperty rdf:ID="nacionalidade">
        <rdfs:domain rdf:resource="#dadosPessoais"/>
        <rdfs:range rdf:resource="#String"/>
    </owl:ObjectProperty>
    ...
</rdf:RDF>
```

2.5.2 Aplicações da OWL na Engenharia de Software

Alguns benefícios do uso de ontologias na construção de software são (USCHOLD, 1998):

- **reusabilidade:** a ontologia é uma base formal para capturar entidades, atributos, processos e seus relacionamentos em um determinado domínio.

Esta representação formal pode tornar-se um componente reutilizável ou compartilhado entre ferramentas;

- **busca:** uma ontologia pode ser utilizada como um metadado para indexação de um repositório de informação;
- **confiabilidade:** uma representação formal torna possível a automatização para checar a consistência tornando o software mais confiável;
- **especificação:** uma ontologia pode auxiliar no processo de identificação de requisitos e definição de uma especificação para uma ferramenta;
- **aquisição de conhecimento:** uma ontologia pode ser utilizada como ponto de partida para guiar um processo de aquisição de conhecimento.

Entretanto, a aplicabilidade de ontologias na Engenharia de Software não se restringe às apresentadas acima. Para apresentar alguns dos pontos específicos, será utilizada parte de um *framework* proposto por Jasper *et al.* (1999) para classificar os diversos campos de utilização de ontologias.

O *framework* apresenta quatro áreas principais para agrupamento dos diferentes tipos de aplicação de ontologias: (1) autoria imparcial; (2) ontologia como especificação; (3) acesso comum à informação e; (4) busca baseada em ontologia. Entretanto serão considerados aqui apenas os grupos (2), (3) e (4). Isto por que o grupo autoria imparcial (1) é bastante semelhante ao grupo acesso comum à informação.

A idéia básica do cenário de utilização de ontologias como especificação está no fato de se criar uma ontologia que modela o domínio de uma determinada aplicação e provê um vocabulário para especificação dos requisitos. Benefícios desta abordagem incluem documentação, confiabilidade e (re)utilização do conhecimento definido.

Um exemplo de aplicação deste grupo é a análise de domínio. Neste caso, o conhecimento do domínio é organizado em um modelo chamado Teoria do Domínio, cuja característica principal é o uso de ontologia do domínio[5], para ser utilizado durante todo o processo de desenvolvimento de software (VAN HEIJST *et al.*, 1997). Um exemplo concreto desta abordagem foi implementada no TABA (FALBO, 1998) (OLIVEIRA, 1999) tornando-o um ambiente de desenvolvimento de software orientado

[5] Uma ontologia do domínio expressa uma conceituação para um domínio particular, definindo restrições (expressas na forma de axiomas formais) na estrutura e conteúdo do conhecimento desse domínio.

a domínio. Ainda no contexto do TABA, Falbo (1998) introduziu a noção de Servidor de Conhecimento como uma estrutura capaz de promover a integração de conhecimento servindo de apoio para a construção de sistemas baseados em conhecimento. Basicamente, um Servidor de Conhecimento provê resolvedores genéricos de problemas e bases modulares de conhecimento baseadas em ontologias permitindo a reutilização de conhecimento de domínio. Uma vez que o Servidor torna disponível conhecimento sobre um universo de discurso, o engenheiro de software pode se utilizar deste conhecimento para compreender este universo e para propor uma solução inicial para o problema.

Uma outra aplicação está relacionada à documentação de software. Ontologias podem ser utilizadas na construção de um modelo de informação que permita a exploração do espaço da informação em termos de itens que são representados por suas associações, propriedades e definições (HEFLIN, 2003).

Já o cenário de acesso comum à informação fundamenta-se na idéia de que aplicações podem fazer uso de ontologias para permitir acesso a fontes de informações heterogêneas. A maior parte dos exemplos contidos neste grupo referem-se a questões de interoperabilidade entre ferramentas. Podemos citar também a web semântica[6] para a qual as ontologias são consideradas, por Tim Berners-Lee (2001), um ponto crítico. Isto por que as estruturas conceituais que definem internamente uma ontologia são a chave para que os dados possam ser corretamente interpretados pelo software e esta interpretação é o fundamento para a construção da web semântica. Através da definição formal de conceitos e teorias de um domínio, ontologias ajudam pessoas e máquinas se comunicarem considerando a semântica da interação ao invés da sintaxe apenas (MAEDCHE e STAAB, 2001).

Por fim, a idéia básica do grupo de busca baseada em ontologia está no fato da utilização de ontologias para tornar mais eficientes os mecanismos de busca em repositórios de informação. Este é um ponto bastante importante na construção, por exemplo, de sistemas de gerência do conhecimento.

[6] Web Semântica é a extensão da web atual na qual é dado um significado bem definido à informação possibilitando uma melhor cooperação entre computadores e pessoas (BERNERS-LEE *et al.*, 2001).

2.6 – Considerações Finais

Neste capítulo foram discutidos o XML e algumas de suas tecnologias correlatas (XML *Schema*, XSLT e OWL). Eles podem ser considerados uma grande inovação para a construção de softwares baseados na Internet.

Foram apresentadas também algumas de suas aplicações na Engenharia de Software. Uma das mais importantes envolve a definição de uma sintaxe abstrata – criada com a customização de marcadores – para integrar diferentes ferramentas que são utilizadas em um determinado contexto.

Analisando atentamente as tecnologias apresentadas neste capítulo, pode-se perceber que elas lidam com perspectivas diferentes para descrever uma informação: a XML para especificar a informação propriamente dita, o XML *Schema* para definir sua estrutura e a OWL com a semântica. Todas utilizando o mesmo padrão sintático, o XML. Isto é uma vantagem pois torna possível o desenvolvimento de aplicações utilizando um mesmo formalismo para lidar com as diversas perspectivas envolvidas na manipulação de informação. Outro ponto positivo que deve ser considerado é o fato de todas elas serem padronizadas pelo W3C o que aumenta a confiança em sua continuidade e evolução. Dessa forma, optamos por utilizar estas tecnologias na elaboração do mecanismo para integração de ferramentas proposto nesta tese.

Por fim, a discussão aqui realizada serve de base para o entendimento do mecanismo para integração de ferramentas apresentado no capítulo 4.

Capítulo 3
Integração de Ferramentas

Este capítulo introduz os conceitos envolvidos com a integração de ferramentas apresentando seus desafios e algumas soluções desenvolvidas com enfoque na integração de dados.

3.1 – Introdução

Existe uma crescente demanda no acesso e integração de fontes de informações heterogêneas e distribuídas. Entretanto, criar mecanismos que tornem possível a comunicação integrada de ferramentas é um grande desafio (BIRD *et al.,* 2000) (EMMERICH *et al.,* 2001) (PINTO *et al.,* 2003) (WÖB e PÜHRETMAIR, 2001) (KARSAI, 2000). A dificuldade está concentrada, principalmente, na manipulação dos diversos esquemas e entidades que representam as necessidades de cada aplicação. Por exemplo, o desenvolvimento de software é uma atividade que pode ser apoiada por um conjunto de ferramentas definidas em um processo de software. Em alguns casos, duas ferramentas distintas podem ser utilizadas para trabalhar sobre um mesmo artefato, mas nada garante que o artefato a ser compartilhado considere a estrutura de organização utilizada pelas duas ferramentas. Sabe-se também que cada ferramenta contribui de forma complementar durante o processo de desenvolvimento de software tornando, algumas vezes, necessário que muitas das informações geradas por uma ferramenta sejam utilizadas por outra (KARSAI e GRAY, 2000).

A questão da integração também é importante para o cenário atual de desenvolvimento de software baseado em reuso. Isto pelo fato de um número crescente de aplicações não serem mais desenvolvidas "do zero" mas, utilizando componentes, algumas vezes, comerciais (EMMERICH *et al.*, 2001). Estes, por sua vez, geralmente não são construídos seguindo modelos já padronizados de integração mas sim, produzidos por diferentes fabricantes que possuem padrões próprios às vezes incompatíveis com os demais. Neste caso, a heterogeneidade pode envolver questões relacionadas à linguagem de programação utilizada, plataforma em que está

disponível e uso de diferentes representações para o intercâmbio dos dados. Yakimovich *et al.* (1999) apresentam cinco tipos de problemas envolvendo a integração de componentes:

- **Funcional:** função necessária ausente ou implementada incorretamente no componente;
- **Não funcional:** não conformidade com requisitos não funcionais (como, confiabilidade, manutenibilidade e usabilidade dentre outros) especificados para a aplicação na qual o componente será integrado;
- **Arquitetural:** neste caso estão classificados alguns tipos de problemas como: empacotamento (disparidades entre a sintaxe de desenvolvimento do componente e do sistema alvo), controle do fluxo de informação, sincronização entre o componente e o sistema alvo e, comunicação entre componentes (deve haver um "protocolo" para comunicação).
- **Conflitos:** neste grupo estão classificados problemas relacionados a alocação de recursos para os componentes;
- **Interface:** aqui estão agrupadas as diferenças relacionadas à, por exemplo, estruturação e semântica dos dados e, quantidade de parâmetros para chamada de funções nos componentes.

De forma mais genérica, alguns trabalhos discutem outros aspectos a serem considerados na integração de ferramentas (WASSERMAN, 1990) (TRAVASSOS, 1994) (SATTLER, 1997). Neste conjunto, eliminando as repetições, temos:

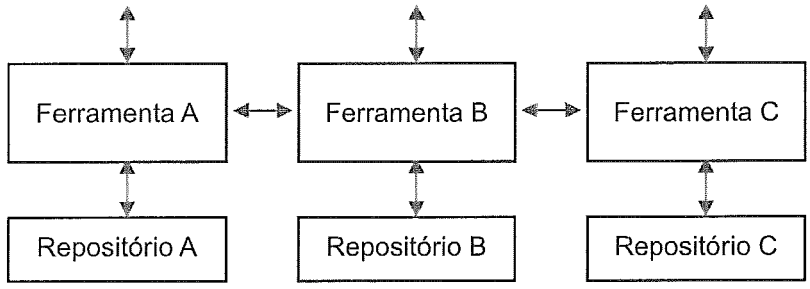
- **Dados:** demonstra como deverá ser a utilização das informações pelas ferramentas e assegura que todas as informações são gerenciadas de forma consistente. Assim, a integração de dados preocupa-se com a interoperabilidade das ferramentas descrevendo o que deve ser feito para que os dados gerados por uma ferramenta possam ser manipulados por outra;
- **Controle:** relaciona-se com a comunicação e a interoperabilidade das ferramentas sendo responsável por permitir a flexibilidade de combinações de funcionalidades que as ferramentas em conjunto suportam, de acordo com as preferências de projeto;
- **Apresentação:** refere-se a questões de interface com o usuário. Neste caso, o ideal é que as ferramentas sigam a um padrão que facilite seu uso e

entendimento e que dêem a impressão de que todas elas foram projetadas seguindo um mesmo princípio;

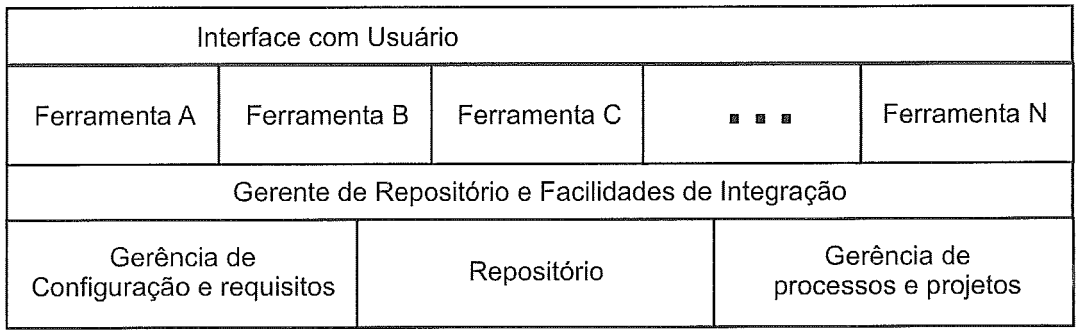
- **Plataforma:** refere-se aos serviços básicos utilizados pelas ferramentas como os de rede e sistemas operacionais;
- **Processo:** trata o papel que cada ferramenta exercerá no contexto do desenvolvimento de software, assegurando que elas interagirão para o suporte ao processo de desenvolvimento definido. Esta faceta da integração está bastante relacionada com o controle;
- **Conhecimento:** trata a semântica dos dados integrados e os possíveis conhecimentos adquiridos durante a utilização da aplicação.

Neste contexto, vários ambientes e modelos de referência foram desenvolvidos. A maioria dessas abordagens baseava-se na utilização de repositórios centrais onde todas as informações eram mantidas. Entretanto, cada vez mais tem se tornado claro que a abertura desses ambientes à integração de ferramentas independentes é um dos principais pré-requisitos para o sucesso do mecanismo de integração o que tem influenciado no desenvolvimento de abordagens que considerem a distribuição das fontes de dados (SATTLER, 1997). A Figura 3.1 apresenta, de forma simplificada, essas duas abordagens (SHARON e BELL, 1995). Na Figura 3.1a é apresentada uma arquitetura na qual as ferramentas integradas são fracamente acopladas e dependem de mecanismos que efetuem transformações nos dados para que seja possível a troca de informações entre elas. É uma perspectiva que devido ao contexto atual de desenvolvimento de software e as tecnologias disponíveis têm se tornado a mais viável. Ela lida: (1) com a integração de componentes que foram projetados sem levar em conta a integração com outros componentes e, (2) com a possibilidade da distribuição das fontes de dados.

Já a perspectiva apresentada na Figura 3.1b considera a utilização de um repositório centralizado. Nesta perspectiva todas as ferramentas devem estar projetadas para lidar com aquele modelo de dados e ainda possuir interfaces para permitir a integração em níveis de apresentação, controle, processo, dado e conhecimento. Embora atinja todos estes níveis, é uma solução que dificulta e limita a integração de ferramentas desenvolvidas por terceiros.



(a)



(b)

Figura 3.1 – Adaptada de (SHARON e BELL, 1995).

A partir de agora será dada maior ênfase à questão da integração de dados, objetivo maior de discussão neste capítulo. Uma boa classificação dos diferentes tipos de problemas envolvendo interoperabilidade entre aplicações em nível de dados pode ser encontrada em (SHETH, 1998) e (YAKIMOVICH *et al.*, 1999). A partir de seus trabalhos podem ser identificados quatro níveis de heterogeneidade:

- **Sistema:** inclui incompatibilidades de *hardware* e sistemas operacionais;
- **Sintático:** refere-se a diferenças em linguagens e mecanismos para representação dos dados;
- **Estrutural:** refere-se a diferenças no modelo de dados e esquemas utilizados;
- **Semântico:** refere-se ao significado dos termos utilizados como metadados.

Muitas tecnologias têm sido desenvolvidas para lidar com esses tipos de heterogeneidade. Para os conflitos envolvidos na classe sistema, podem ser utilizados *middlewares* como CORBA e DCOM. Estes lidam com problemas de heterogeneidade de *hardware*, sistemas operacionais, linguagens de programação e ainda provêm meios de comunicação que podem ser utilizados pelas aplicações (EMMERICH,

2000). Por lidar apenas com conflitos da classe sistema, ele pode ser considerado uma tecnologia a ser utilizada em complemento a outras abordagens de integração.

Para a questão da integração sintática, a XML tem sido aceita como padrão para intercâmbio de dados (FENG *et al.*, 2002). Entretanto, é importante lembrar que a XML é uma linguagem para descrever a estrutura sintática de um documento, não seu significado.

Já o problema da integração estrutural vem sendo constantemente tratado através da definição de esquemas que especificam padrões para certos domínios de aplicação. Um exemplo é a utilização do XML *Schema* para definição de como estão estruturados documentos XML para atender a aplicações de comércio eletrônico (ebXML). Embora a definição de esquemas possa, com sucesso, ser utilizada para se chegar a um acordo sobre qual vocabulário será utilizado como metadado, é muito difícil se chegar a um ponto em que todos estejam de acordo quanto aos conceitos considerados.

Por fim, ainda resta o problema da heterogeneidade semântica. Um bom exemplo é o uso de sinônimos onde diferentes termos são utilizados para representar o mesmo conceito. Outros tipos de problemas envolvendo a integração da semântica dos dados podem ser encontrados em (VISSER *et al.*, 1998). Uma solução para ele seria equipar as aplicações a serem integradas com um mecanismo que as tornasse possível compartilhar e trocar informações de forma semanticamente consistente. Isto pode ser alcançado de diversas formas. Uma delas, e a mais natural, seria desenvolver manualmente conversores entre a terminologia utilizada para cada par de sistemas. Entretanto, esta solução só seria viável para um pequeno conjunto de aplicações (CUI *et al.*, 2001a). Uma outra solução, mais viável que a primeira, é criação de mecanismos automatizados ou semi-automatizadas que considerem a estrutura e a semântica das informações a serem integradas. Esta semântica pode ser formalmente representada com o uso de ontologias. Dessa forma, esquemas e ontologias desempenham um papel importante na integração de fontes de informações heterogêneas.

A partir de agora serão apresentados na seção 3.2 alguns mecanismos para integração de ferramentas considerando suas arquiteturas. Será dado um maior enfoque á questão da integração de dados. Será feita também uma breve discussão das principais vantagens e desvantagens de cada abordagem. Já na seção 3.3 serão descritas, resumidamente, outras abordagens e na seção 3.4, as conclusões deste capítulo.

3.2 – Mecanismos para Integração de Ferramentas

Existem ou encontram-se em desenvolvimento diferentes propostas para integração de ferramentas. Estas têm sido construídas segundo diversas abordagens e perspectivas de utilização que acabam influenciando o tipo de arquitetura e integração adotados. Serão apresentadas agora algumas dessas abordagens enfocando seus mecanismos para integração de dados.

3.2.1 – TABA

3.2.1.1 – Visão Geral

A Estação TABA (ROCHA *et al.,* 1990) é um meta-ambiente capaz de gerar, através de instanciação, ambientes de desenvolvimento de software (ADS) adequados às particularidades de processos de desenvolvimento e projetos específicos. Assim, ela possui facilidades para a definição de processos, métodos e ferramentas CASE a serem utilizadas no processo de desenvolvimento. Estes elementos estão organizados em um modelo de integração do ambiente, permitindo que diferentes ambientes sejam definidos e instanciados.

O projeto TABA foi criado no início da década de 90, a partir da constatação de que domínios de aplicação diferentes possuem características distintas, e que estas devem incidir nos ambientes de desenvolvimento através dos quais os engenheiros de software desenvolvem aplicações (ROCHA *et al.,* 1990).

3.2.1.2 – Abordagem de Integração

A arquitetura da Estação TABA foi definida como um conjunto de componentes integrados que possuem controle sobre a sua existência, suas informações, estados e funcionalidades básicas (ver Figura 3.2).

A idéia que se quer passar na figura 3.2 é a de que um ADS está inserido no contexto de um produto de software e é responsável, com seu conjunto de componentes, por traduzir (segundo a necessidade do usuário) uma representação do mundo real para o mundo computacional, que é o próprio produto de software. Desta forma, um ADS tem que diminuir a distância entre o problema do mundo real e o

mundo computacional, facilitando para o usuário do ADS a construção de produtos de software (TRAVASSOS, 1994).

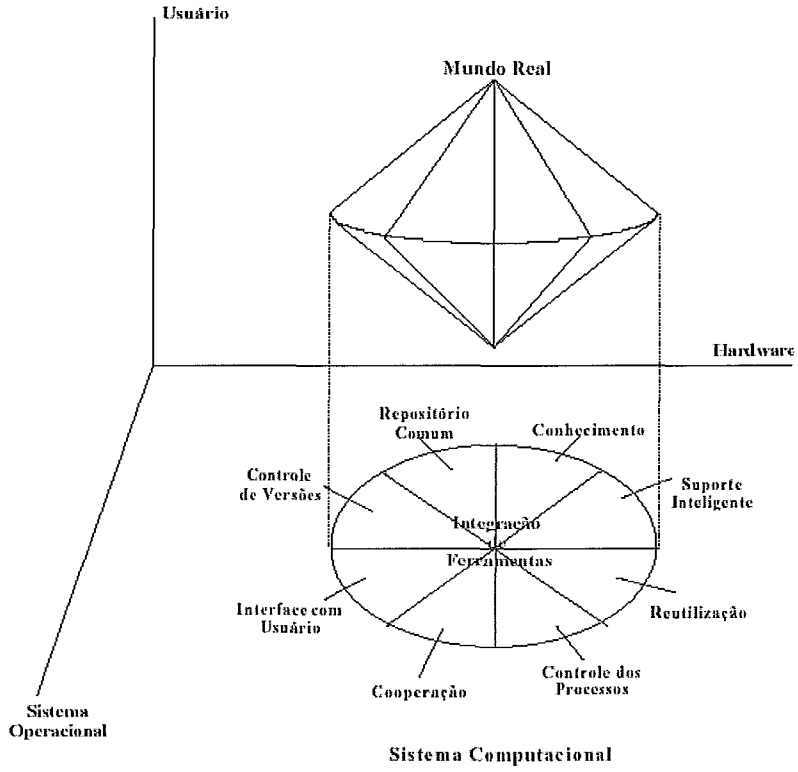


Figura 3.2 – Estrutura de um ADS na Estação TABA (TRAVASSOS, 1994).

Os componentes básicos da Estação TABA são (TRAVASSOS, 1994): repositório comum, controle de versões, interface com o usuário, cooperação, controle dos processos, reutilização, suporte inteligente e conhecimento. O componente repositório comum é o responsável pelo controle, gerenciamento e armazenamento dos objetos manuseados pelo ambiente e meta-ambiente. O controle de versões como o próprio nome já diz, controla e gerencia versões de documentos e itens de software produzidos.

O componente interface com o usuário gerencia e controla a interação do usuário com o ADS garantindo, quando possível, a consistência da apresentação das ferramentas. O componente cooperação trata da comunicação entre os usuários e suas necessidades de interação com outros membros da equipe. Já o componente controle de processos é o responsável pelo controle e gerenciamento do processo de desenvolvimento. Por fim, temos os componentes reutilização, suporte inteligente e conhecimento. O primeiro provê o ADS de mecanismos que possibilitam a reutilização de trabalhos anteriores. O segundo fornece inteligência global ao ambiente, assistindo

o usuário na utilização do meta-ambiente e dos ADSs, através de assistentes inteligentes. Já o componente conhecimento incorpora mecanismos para o armazenamento e a utilização de conhecimento.

A utilização em conjunto destes componentes permite a integração de ferramentas ao ambiente e definem, como um todo, a filosofia de integração do TABA e seus ambientes instanciados (TRAVASSOS, 1994). Fundamentando-se nestes componentes, Travassos (1994) definiu o Modelo Canônico para Integração de Ferramentas da Estação TABA e dos seus ambientes instanciados considerando quatro tipos de integração (observar Figura 3.3). **Apresentação e Interação:** responsável por proporcionar ao usuário a sensação de integração no ambiente. **Gerenciamento e Controle:** responsável por prover serviços e funcionalidades que permitam o funcionamento de forma organizada, ao longo do processo de desenvolvimento. **Dados:** responsável por prover serviços básicos de armazenamento e gerenciamento de estruturas de informação. **Conhecimento:** responsável por possibilitar os serviços básicos de armazenamento, gerenciamento e utilização do conhecimento descrito e adquirido ao longo do processo de desenvolvimento.

Figura 3.3 – Enfoque de Integração da Estação TABA (TRAVASSOS, 1994).

A integração de dados está fundamentada em um modelo de grafos. Neste modelo, construído utilizando conceitos de orientação a objetos, foram definidas um conjunto de classes que atuando em conjunto com uma linguagem de descrição e manipulação de métodos[7] são capazes de representar quaisquer informações necessárias ao ambiente (TRAVASSOS, 1994). Vale destacar aqui que há uma similaridade entre a linguagem de manipulação de métodos e a XML. A linguagem para definição de métodos é composta por um conjunto de "marcações" que poderão conter informações sobre o artefato que esteja sendo representado pelo modelo de

[7] Definida por Travassos (1994), ela fundamenta-se em um conjunto de elementos do paradigma estruturado e orientado a objetos para a descrição da semântica de um documento mantido no TABA.

grafos. Embora estas marcações, ao contrário do que acontece com a XML, sejam fixas e pré-determinadas, seu objetivo é o mesmo, descrever a informação que contêm. Fazendo um paralelo, essa linguagem é similar a arquivos de configuração definidos em XML e bastantes presentes em aplicações atuais. Por fim, vale informar que os dados, representados utilizando esta proposta, são mantidos em um repositório centralizado acessado apenas pelo ambiente.

3.2.1.3 – Comentários

A proposta de (TRAVASSOS, 1994) baseou-se na idéia de que um ADS está inserido no contexto de um produto de software e é o responsável por traduzir uma representação do mundo real para o computacional através de um conjunto de componentes que consideram aspectos relacionados ao *hardware*, interação, interface com o usuário, gerência de processos e armazenamento de dados. Entretanto, mesmo esta estrutura atendendo à integração de ferramentas internas[8], há uma dificuldade muito grande para a integração de ferramentas externas[9]. Estas devem "conhecer" a estrutura de organização interna da informação no ambiente que é baseada em um modelo de grafos. Com o passar do tempo, melhorias foram e vêm sendo feitas ao modelo: (1) o componente "conhecimento" foi definido (FALBO, 1998); (2) foi estendido de forma a possibilitar a captura, armazenamento e compartilhamento de conhecimento do domínio da aplicação (OLIVEIRA, 1999).

Atualmente, tem se trabalhado no conceito de Ambiente de Desenvolvimento de Software Orientado à Organização (ADSOrg) (VILLELA, 2004). Entretanto, a questão da integração de dados sofreu pouca ou nenhuma alteração ao mesmo tempo em que surgiram novas tecnologias e necessidades. Alguns exemplos são, trabalho distribuído, a internet e as linguagens de marcação para estruturação de artefatos, dentre outros.

[8] Internas: desenvolvida levando-se em consideração a arquitetura da Estação TABA.
[9] Externas: desenvolvida sem levar em consideração a arquitetura da Estação TABA.

3.2.2 – GEHR

3.2.2.1 – Visão Geral

No cenário atual de assistência médica, ter acesso à informação (por exemplo, dados clínicos) é um ponto chave no gerenciamento da saúde de pacientes. Padrões EHR[10] (do inglês, *Electronic Health Record*) auxiliam no acesso e integração dessas informações (BIRD *et al.*, 2003). Entretanto, esta abordagem é ineficaz. O problema está no fato de existirem diversos padrões EHR especificando como os dados clínicos são organizados. Muitas vezes, a adoção de padrões distintos em departamentos dificulta até mesmo a troca de informações dentro de uma organização.

Neste contexto, pelo fato de instituições de assistência médica terem cada vez mais que compartilhar informações presentes em seus sistemas sobre a saúde de pacientes, torna-se imperativo o desenvolvimento de mecanismos que possibilitem a troca de dados de forma mais transparente. Com isto em mente, foi desenvolvido o GEHR (do inglês, *Good Electronic Health Record*). Ele é um mecanismo de apoio a sistemas de acompanhamento da saúde de pacientes e seu principal objetivo é prover uma infra-estrutura de informação baseada em metadados que possibilite a utilização de dados clínicos (de padrões distintos) provenientes de diferentes organizações. Para isto, ele provê uma arquitetura aberta e um formato padrão para registros eletrônicos de acompanhamento médico (BIRD *et al.*, 2003).

3.2.2.2 – Abordagem de Integração

Os princípios básicos que vêm norteando o desenvolvimento desta abordagem são: (1) elaborar um mecanismo que consiga integrar sistemas de EHR e, (2) permitir a evolução do modelo de integração com objetivo de captar novos conceitos à medida que estes forem surgindo. Esta necessidade de evolução constante implicou na elaboração de uma arquitetura configurável, baseada em meta-modelagem.

A arquitetura definida no GEHR está fundamentada, basicamente, em dois componentes (BIRD *et al.*, 2000a):

[10] EHR é uma coleção longitudinal de informações sobre saúde de pessoas identificadas por centros de assistência médica e distribuídas para serem acessadas por outros sistemas (HINA, 2000).

- Modelo de referência: conhecido como GOM (GEHR *Object Model*), especifica como capturar, organizar, agrupar e armazenar informações clínicas provenientes de sistemas de EHR.
- Arquétipos: criados para lidar com a qualidade dos dados manipulados e garantir que as informações armazenadas são válidas em termos de conhecimento clínico.

A Figura 3.4 apresenta o relacionamento entre esses componentes e as instâncias de cada um. Estas são as configurações feitas em cada um dos componentes para que eles possam conter (instância de um EHR) e representar (arquétipo) um tipo de dado clínico.

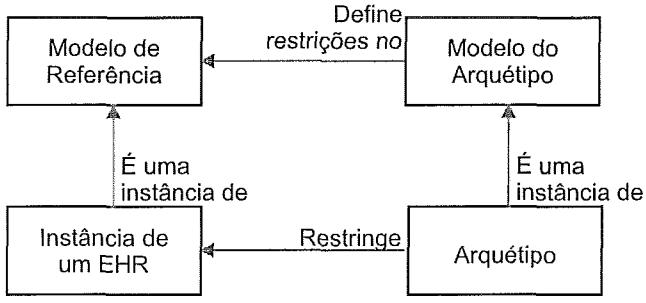


Figura 3.4 – Adaptada de (BIRD *et al.*, 2003).

O objetivo do GOM é definir um modelo abstrato que faz uso da linguagem XML para o armazenamento das informações. Neste modelo de referencia, ele contempla mecanismos que dêem suporte aos principais conceitos envolvidos em projetos de sistemas de EHR (BIRD *et al.*, 2000a):

- **Transação:** os registros presentes em um EHR podem ser persistentes ou transacionais. Persistentes são aqueles em que os dados devem permanecer constantes ao longo do tempo enquanto que os transacionais sofrem atualizações e suas versões não podem ser descartadas. Neste caso, deve existir um mecanismo para controle de versões;
- **Navegação:** os dados de um sistema de EHR devem ser mantidos segundo uma coleção de itens de conteúdo hierarquicamente agrupados. Para isto, devem ser utilizados organizadores[11] para facilitar o acesso às informações que, em geral, são acessadas de forma agrupada;

[11] *Organizers* no original (BIRD *et al.*, 2000a).

- **Conteúdo:** estruturas genéricas são utilizadas para definir as diversas informações que um dado clínico pode ter. Estas são organizadas de forma hierárquica para que o contexto atribua parte do significado à informação.
- **Tipo de dados:** definição dos tipos de dados suportados pelo sistema.

Analisando os conceitos contemplados pelo GOM, percebe-se que não são consideradas a semântica e a estruturação de modelos clínicos específicos. Estas questões são solucionadas pelo mecanismo voltado diretamente para a integração de dados nesta arquitetura, os arquétipos. Resumidamente, estes são esquemas (descritos em documentos XML *Schema*) que definem a estrutura da informação e, em parte, sua semântica. Os arquétipos provêm um meio de tornar a arquitetura extensível permitindo a inclusão de novos modelos clínicos e customização dos já existentes quando houver necessidade. A importância disto está relacionada ao fato de que representar todos os conceitos presentes em modelos clínicos e a forma como estes são representados é inviável (BIRD *et al.*, 2000b).

Para entendermos melhor o conceito de arquétipos pensemos na seguinte metáfora: considerando o modelo de referência como sendo a gramática da língua portuguesa e tendo um vocabulário qualquer (por exemplo, o contido no dicionário Aurélio), é preciso que haja algum mecanismo para definir como as palavras podem ser organizadas para formar uma oração com sentido. Neste ponto se enquadra a funcionalidade dos arquétipos (BIRD *et al.*, 2000b). Eles estariam definindo as regras da gramática. No contexto do GEHR, um arquétipo define um conceito clínico especificando o conteúdo e estruturação que os dados devem possuir para representá-lo e possibilitar seu armazenamento no GOM.

Como ilustrada na figura 3.5, a arquitetura para a integração de dados do GEHR é composta pelo GEHR *Kernel* e por uma infra-estrutura responsável pela manipulação dos arquétipos. As funções do GEHR *Kernel* são: configurar as estruturas de armazenamento (GOM) para um modelo clínico descrito por um arquétipo, inserir e recuperar dados. Para que novos tipos de modelos clínicos possam ser utilizados pelo GEHR, é necessário criar um arquétipo correspondente. Isto pode ser feito através do editor de arquétipos.

A infra-estrutura montada para manipulação de arquétipos é composta de componentes para seu armazenamento, desenvolvimento, disponibilização e tradução. Este último é responsável por configurar estruturas de armazenamento (GOM) para o modelo clínico descrito pelo arquétipo.

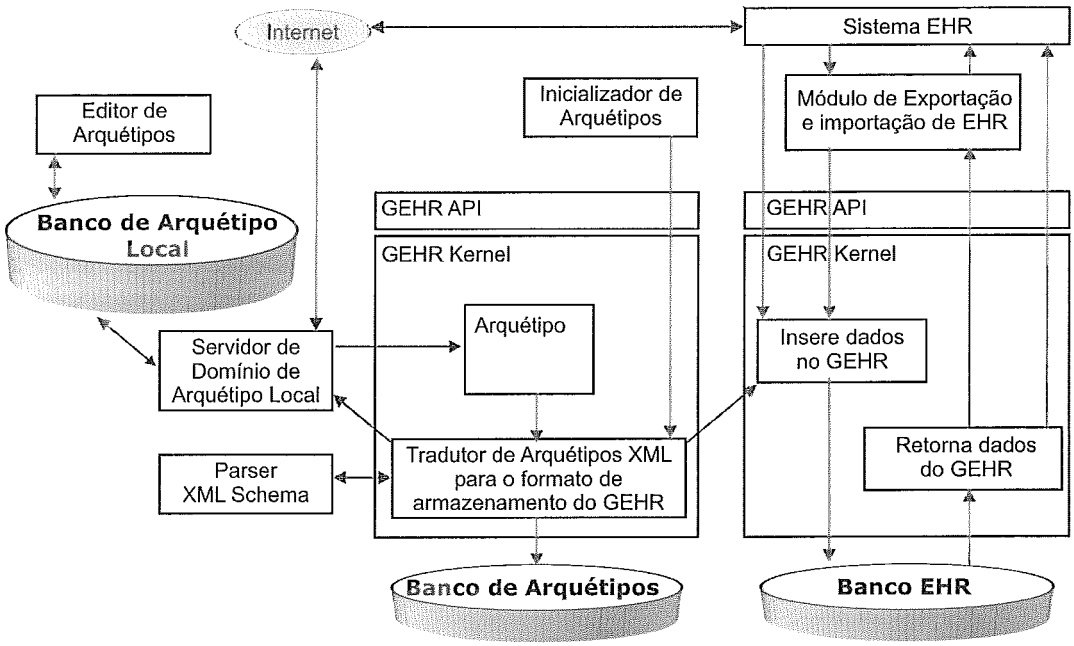


Figura 3.5 – Arquitetura para Integração de Ferramentas do GEHR. Adaptada de (BIRD *et al.*, 2000b).

É perceptível que a arquitetura descrita permite a inclusão e customização de modelos clínicos solucionando o problema em se definir previamente todos os tipos de dados clínicos a serem manipulados pelo sistema. Entretanto, isto não resolve o problema da integração com sistemas legados. Para isto, foram desenvolvidos conversores utilizando a linguagem XSLT. Como a tecnologia utilizada para armazenamento é o XML, basta criar regras de transformação que mapeiem os dados de uma ferramenta externa para o modelo de dados definido pelo GOM e que esteja de acordo com um determinado arquétipo (observar Figura 3.6).

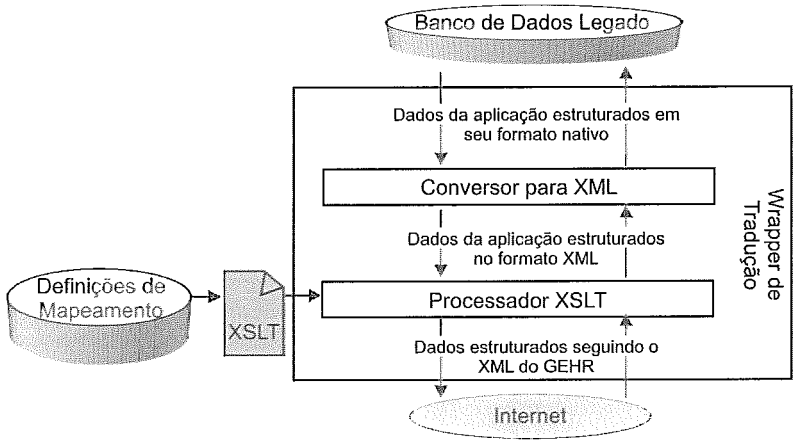


Figura 3.6 – Transformação dos documentos no GEHR. Adaptada de (BIRD *et al.*, 2000b).

2.2.2.3 – Comentários

O projeto GEHR apresenta um exemplo prático de mecanismo para integração de dados baseado em conceitos de meta-modelagem. O foco desta abordagem está na separação do problema de integrar aplicações em dois pontos: (1) desenvolvimento de um *modelo semântico formal abstrato* e (2) *especificação da estrutura* e da semântica dos dados em arquétipos. Este último, mais relacionado com a questão da integração de dados, apresentou uma solução interessante para o problema.

Pode-se destacar também a utilização da linguagem XML para armazenamento de dados, XML *Schema* para validar e definir a estrutura dos dados armazenados no sistema e, da linguagem XSLT para mapear duas estruturas de documentos. Isto demonstrou a aplicabilidade destas tecnologias em problemas de integração. Entretanto, o processo de geração de conversores definidos em XSLT ainda é manual e bastante propenso a erros. Outro problema é a falta de uma representação formal da semântica dos dados uma vez que as estruturas responsáveis por isso, os arquétipos, estão mais relacionados com a estruturação das informações.

3.2.3 – TIGRA

3.2.3.1 – Visão Geral

O TIGRA é uma proposta de arquitetura reutilizável para integrar aplicações (EMMERICH *et al.*, 2001). Reutilizável porque, embora tenha sido concebida para integração de aplicações financeiras, sua idéia pode ser utilizada em outros domínios. Dentre suas características, três das que mais se destacam são o fato de contemplar questões referentes à manutenção da semântica dos dados, o processo utilizado no seu desenvolvimento e a separação entre representação e transporte de dados.

Ele foi motivado pela necessidade em facilitar a integração de diversas (cerca de 120) aplicações financeiras divididas em ferramentas de *front-end* e *back-end*. Estas eram integradas utilizando uma abordagem *ad hoc*. Para cada uma das aplicações deveriam existir duas ou mais interfaces responsáveis pela conversão dos dados. Entretanto, pelo fato do processo ser manual e o esforço gasto em seu desenvolvimento ser alto, estas interfaces eram implementadas apenas para as aplicações mais ativas do sistema financeiro. Assim, ainda existiam conversões de dados efetuadas manualmente. Um dos principais problemas desta abordagem era,

além do alto esforço envolvido, a baixa qualidade na conversão dos dados que muitas vezes implicava em retrabalho.

3.2.3.2 – Abordagem de Integração

Como dito na seção anterior, um dos pontos de destaque do projeto TIGRA foi o processo utilizado na sua elaboração. Ele foi desenvolvido de forma iterativa e incremental com objetivo de mitigar os riscos e perceber os benefícios conseguidos com a arquitetura rapidamente (EMMERICH *et al.*, 2001). Os passos seguidos no processo foram:

- **Análise dos Requisitos:** onde foram especificadas as necessidades do negócio, requisitos funcionais e, principalmente, não funcionais. Os requisitos não funcionais que mais influenciaram a elaboração do TIGRA estavam relacionados à: escalabilidade, desempenho, confiabilidade, disponibilidade, segurança, extendibilidade e utilização de tecnologias não proprietárias;
- **Prototipação Exploratória da Arquitetura**[12]**:** no qual são testadas as idéias de solução para a arquitetura através do desenvolvimento de protótipos;
- **Seleção do *Middleware*:** através da análise dos protótipos elaborados e, principalmente, dos requisitos não funcionais identificados;
- **Projeto Piloto:** desenvolvimento da Arquitetura considerando o *middleware* selecionado e os requisitos identificados. Nesta etapa foi solucionado o problema da integração de um número delimitado;
- **Distribuição:** onde a arquitetura de integração passou a ser utilizada pela organização.

Uma das premissas a serem consideradas durante o processo de elaboração da arquitetura foi o fato das aplicações de *front-end* e *back-end* não poderem ser modificadas, ou seja, a arquitetura tinha que propiciar sua integração de forma transparente. Esta necessidade implicou na construção de adaptadores de importação e exportação para que o mecanismo de integração pudesse interagir com as ferramentas do ambiente. O padrão adotado para o intercâmbio de dados foi o XML. A Figura 3.7 apresenta a solução concebida. Ela é formada basicamente de adaptadores

[12] Do inglês *Explorative Architecture Prototyping*.

de entrada e saída e um roteador central. No geral, a solução baseia-se nas transformações efetuadas sobre documentos XML e na criação de um modelo centralizado onde é possível representar as informações a serem integradas entre as aplicações.

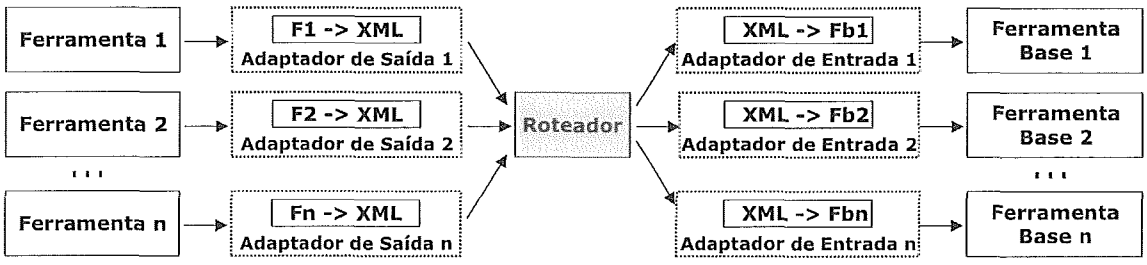


Figura 3.7 – Arquitetura do TIGRA adaptada de (EMMERICH *et al.*, 2001).

A função dos adaptadores de saída é obter os dados de um componente ou ferramenta externa e convertê-lo na representação semântica comum. Já os adaptadores de entrada são responsáveis por recuperar os dados da base de dados centralizada e disponibilizá-los para um determinado componente em sua representação nativa. Estes adaptadores utilizam componentes de mapeamento baseados em XSLT para conversão dos dados. Já o componente Roteador é responsável por encaminhar os dados para seus devidos receptores. Perceba que com isso é possível utilizar os *front-ends* e de forma transparente os dados sofrerão as transformações necessárias e serão encaminhados para os sistemas de *back-end*.

A importância na definição de uma representação semântica comum é justificada pela redução na quantidade de componentes de mapeamento a serem desenvolvidos que cai de uma razão quadrática $O(n^2)$ para linear $O(n)$, sendo n o número de aplicações a serem integradas.

Outra característica importante da arquitetura é o fato dela ter sido construída considerando a separação entre representação e transporte de dados. Os dados são manipulados no formato XML enquanto que a questão do transporte é resolvida com a utilização de um *middleware*. A escolha da linguagem XML foi motivada principalmente pela disponibilidade de padrões para intercâmbio de dados no setor financeiro (FpML, 1999) e a presença crescente de interfaces XML nos programas. O XML foi utilizado em conjunto com o XSLT para prover a camada de integração. Assim, os dados gerados pelas ferramentas de *front-end* e *back-end* deverão inicialmente estar no formato XML. Tendo isto, o dado será transformado (utilizando XSLT) em um XML seguindo a representação semântica centralizada. A partir desse XML central,

mecanismos de conversão poderão ser criados para exportar os dados para uma determinada aplicação.

O *middleware* foi escolhido depois de terem sido desenvolvidos protótipos utilizando cada opção disponível para observar o comportamento de cada uma segundo os requisitos identificados. A necessidade deste procedimento foi justificada pela quantidade de soluções de *middleware* presentes. Baseado nos requisitos não funcionais (segurança, confiabilidade e outros) o CORBA se mostrou ser a opção mais adequada.

3.2.3.3 – Comentários

O ponto forte da abordagem de integração no projeto TIGRA foi a utilização complementar da linguagem de marcação XML e de um *middleware*. Esta combinação permitiu a separação entre o mecanismo de integração e o de transporte dos dados. Vale ressaltar também a importância da definição de uma representação semântica comum no mecanismo de integração pois este reduziu bastante a quantidade de componentes de mapeamento a serem desenvolvidos. Pode-se destacar ainda o processo utilizado no desenvolvimento da arquitetura. A abordagem iterativa e incremental foi decisiva na mitigação dos riscos e verificação gradual dos benefícios da arquitetura.

Entretanto, uma dificuldade presente nesta solução é o fato dos *drivers* de mapeamento terem de ser implementados manualmente. Embora a quantidade tenha sido reduzida, isso ainda exige bastante esforço por parte da equipe de manutenção.

3.2.4 – DIXSE

3.2.4.1 – Visão Geral

O DIXSE (*Data Integration for XML based on Schematic Knowledge*) é um sistema semi-automatizado para a integração de dados XML. Ele trata o problema da integração levando em conta a modelagem dos conceitos envolvidos nas fontes de dados heterogêneas, ou seja, ele considera os esquemas de cada base de dados na geração das informações necessárias para prover a integração (RODRÍGUEZ-GIANOLLI, 2001). Para isto o DIXSE, através de um processo semi-automatizado, deriva uma descrição semântica comum (formalizada em um esquema conceitual) a

partir de um conjunto de entrada formado por DTDs[13] (*Document Type Definition*). Esta descrição semântica criada ainda poderá ser posteriormente aperfeiçoada pelo usuário.

Feito o mapeamento dos DTDs no esquema conceitual, o DIXSE poderá gerar automaticamente *wrappers* para documentos XML que estejam de acordo com os DTDs mapeados. Isso permitirá a integração das diversas fontes de dados.

3.2.4.2 – Abordagem de Integração

A arquitetura proposta para integração de ferramentas no DIXSE fundamenta-se no conceito de meta-modelagem e utiliza a linguagem Telos[14] para representar seu modelo de dados e o esquema conceitual gerado. Para geração do esquema conceitual, o DIXSE possui um arcabouço responsável por efetuar sua derivação a partir de vários DTDs.

Os esquemas conceituais são representados no DIXSE como coleções de tipos de entidades e seus atributos. Os principais conceitos suportados pelo modelo são: *entity class*, *entity attribute*, *mapping* e *document type*. Em conjunto, eles estabelecem os objetos nos quais os elementos de um DTD poderão ser mapeados. Este mapeamento é feito a partir da linguagem de mapeamento DIXml. Esta agrega informações a um DTD com instruções de como transformar seus elementos em seus correspondentes no esquema conceitual (RODRÍGUEZ-GIANOLLI e MYLOPOULOS, 2001).

A arquitetura do DIXSE é formada por dois componentes principais: o *Schema Engine* e o *Document Loader*. O primeiro permite que usuários registrem os esquemas provenientes dos DTDs no repositório e o segundo, possibilita o armazenamento de documentos XML cujos DTDs tenham sido registrados. Para armazenar as informações seguindo o esquema conceitual, o DIXSE faz uso de um banco de dados orientado a objetos, o ConceptBase (ConceptBase, 1998). Este banco foi escolhido por que ele implementa um modelo de dados que suporta a linguagem Telos. A Figura

---

[13] DTD é uma linguagem para modelagem de documentos XML. Assim, ele é um mecanismo utilizado para descrever todo objeto (elementos, atributo) que compõe um documento XML. Entretanto, o DTD é uma solução limitada para a descrição das regras de estruturação de documentos XML. Por exemplo, ele não define tipos como inteiros e data o que causa problemas no momento em que há necessidade de definir o tipo de conteúdo que partes de um documento pode aceitar. Além desta limitação sintática, existe também a dificuldade em manipular documentos DTD os quais não seguem a sintaxe XML.

[14] A TELOS é uma linguagem que provê facilidades para construção, busca e atualização de bases de conhecimento estruturadas (MYLOPOULOS *et al.*, 1990).

3.8 apresenta estes componentes, como eles se relacionam e os sub-componentes de cada um.

O *Schema Engine* é formado por cinco componentes:

- ***Parser* DTD:** responsável por efetuar a manipulação dos elementos que compõem um documento DTD;
- ***Parser* XML:** responsável pela manipulação dos elementos que compõem um documento XML;
- **Derivador de Esquema:** aplica um conjunto de diretivas para derivar o esquema conceitual a partir de documentos DTD. Estas diretivas são fruto de um conjunto de heurísticas previamente definidas e da interação do usuário durante a definição do modelo semântico;
- **Gerador de Esquema:** recebe o esquema conceitual gerado pelo componente derivador de esquema e utiliza-o para gerar o modelo de dados utilizando a linguagem Telos a ser instanciado no repositório central. Informações acerca do esquema gerado também são armazenadas no catálogo do repositório através do componente gerente de catálogo;
- **Gerador de XSL *Wrapper*:** recebe o esquema conceitual criado pelo componente derivador de esquema e utiliza-o na geração do documento XSLT onde estarão definidas as regras de transformação necessárias para a conversão dos documentos XML, cujos DTDs já estejam registrados em instâncias do esquema conceitual. Informações acerca do *wrapper* gerado também são armazenadas no catálogo do repositório através do componente gerente de catálogo

Já o *Document Loader* é formado por dois componentes:

- **Integrador de Dados:** onde são manipuladas informações necessárias no momento das transformações executadas em documentos XML;
- **Processador XSL:** responsável por efetuar as transformações nos documentos XML.

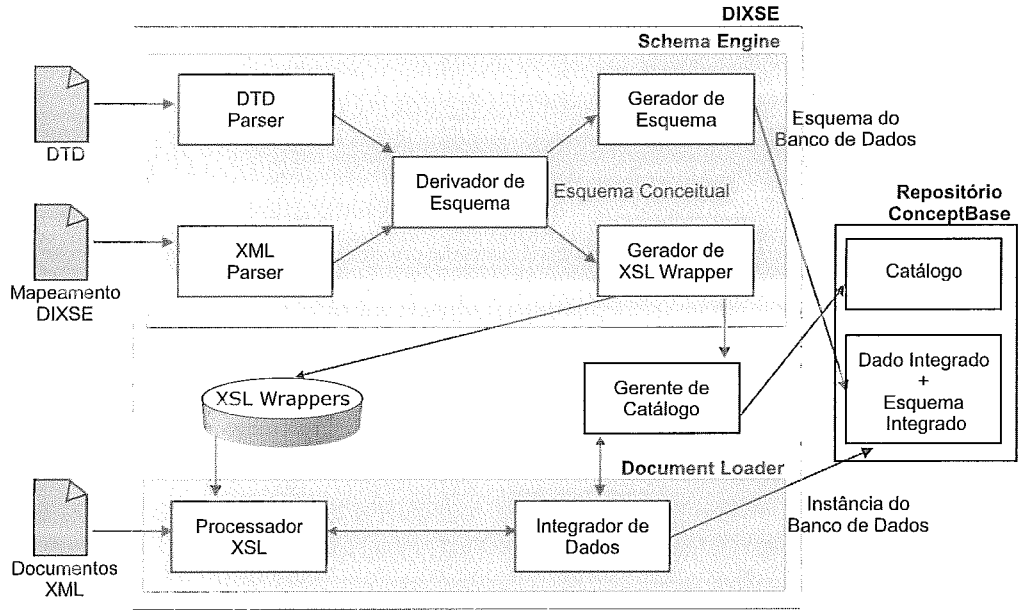


Figura 3.8 – Adaptado de (RODRÍGUEZ-GIANOLLI e MYLOPOULOS, 2001).

A comunicação entre os dois principais componentes, *schema engine* e *document loader*, é feita através do gerente de catálogo e do repositório de XSL *Wrapper*. O primeiro responsável por manter as informações do catálogo do banco de dados atualizadas e o segundo por armazenar os *wrappers* XSLT.

3.2.4.3 – Comentários

Vimos que o DIXSE provê um mecanismo semi-automatizado para geração de um esquema conceitual onde as informações provenientes das fontes de dados, que tiveram seu DTD mapeado, são integradas e mantidas. Para isto, o DIXSE:

- **considera a semântica da informação a ser integrada:** baseada em heurísticas definidas em regas de mapeamento e informações providas pelo usuário, o DIXSE extrai a semântica de documentos DTD e gera um modelo conceitual;
- **utiliza a integração de esquemas para prover integração dos dados:** o mapeamento dos dados é feito baseado nas informações contidas nos DTDs e no conhecimento do usuário sobre o domínio dos dados;
- **gera de forma automatizada transformadores:** arquivos XSLT são gerados automaticamente para efetuar o mapeamento entre o documento XML que está de acordo com o esquema conceitual e o utilizado pelas ferramentas integradas.

Entretanto, esta abordagem também possui alguns problemas:

- **o conhecimento é extraído de esquemas:** o objetivo de esquemas não é atribuir semântica aos dados mas sim, especificar a forma como estão estruturados;
- **utilização de DTDs para representação de esquemas:** embora cumpra seu papel, DTD é uma solução limitada para definir como documentos XML estão estruturados. Além disso, por não seguirem a sintaxe XML, é difícil de ser manipulado. Esta dificuldade implicou no desenvolvimento de um componente voltado apenas para a manipulação de arquivos DTDs.

3.2.5 – DOME

3.2.5.1 – Visão Geral

Prover suporte a transações de comércio eletrônico envolve, muitas vezes, a recuperação e integração de informações provenientes de múltiplas fontes de dados. Para resolver este problema é preciso especificar formalmente o significado da terminologia utilizada em cada fonte de informação e definir mecanismos de tradução entre elas. Motivado por este problema, foi iniciado o projeto DOME (*Domain Ontology Management Environment*) (CUI *et al.*, 2001a).

O DOME é um sistema baseado em ontologias para apoiar a integração de dados. Para isto, ele provê apoio ferramental para as atividades de definição e validação de ontologias (representando a semântica de cada fonte de informação), e mapeamento entre ontologias e entre elas e esquemas criando assim um ambiente propício para a integração das fontes de informação (CUI *et al.*, 2001a).

3.2.5.2 – Abordagem de Integração

A arquitetura do DOME (ver Figura 3.9) é formada por um conjunto de componentes: servidor de ontologia, servidor de mapeamento, interface de administração, interface com o usuário, "motor" de busca[15], diretório de recursos e extração de *wrappers* e ontologias.

[15] Do inglês *query engine.*

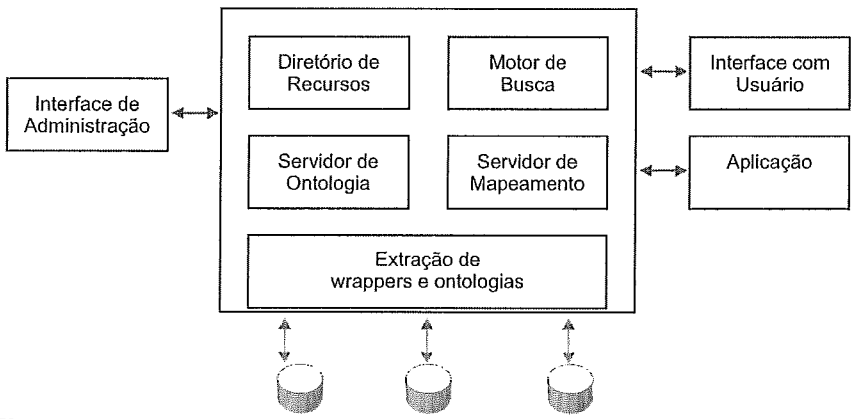


Figura 3.9 – Arquitetura do DOME. Adaptado de (CUI *et al.*, 2001b).

O componente interface de administração provê suporte para a execução de atividades envolvidas com a criação de ontologias. Ele é formado por ferramentas para: (1) a extração semi-automatizada de ontologias a partir de sistemas legados; (2) definição de ontologias; (3) definição do mapeamento entre ontologias e entre estas e esquemas. É neste componente que as atividades centrais relacionadas com a integração de dados são efetuadas.

Este componente permite o desenvolvimento de ontologias seguindo duas abordagens: *top-down* e *bottom-up*. Na abordagem *top-down* o processo se inicia com a análise do domínio para identificar os conceitos chaves e mais genéricos. Esta abordagem deve ser utilizada quando se pretende elaborar a ontologia onde serão mapeadas as informações contidas nas diversas fontes de dados começando pela definição dos conceitos mais genéricos e estáveis. Já a abordagem *bottom-up* é iniciada com a extração, semi-automatizada, de conceitos presentes nos esquemas das bases de dados a serem integradas. Tendo extraídos os diversos conceitos, passa-se então a escolher quais são os mais estáveis e abrangentes para poder ser definida uma ontologia mais estável e genérica.

Tendo definido as ontologias, passa-se então a elaborar os mapeamentos necessários para que a semântica distribuída entre os conceitos utilizados em cada fonte de dados possa ser integrada. Neste ponto, são mapeadas informações semânticas entre ontologias e também estruturais onde informações provenientes de esquemas são mapeadas nas ontologias. Embora de fundamental importância no processo de integração, esta tarefa é executada manualmente o que a torna propensa a erros. Definidas as ontologias e os mapeamentos necessários, estes serão

armazenados nos componentes servidor de ontologia e servidor de mapeamento, respectivamente.

O componente interface com o usuário é o responsável por permitir que o usuário tenha acesso às informações distribuídas pelas fontes de dados de forma transparente, ou seja, independente de onde elas estejam, de sua estrutura, sintaxe e semântica. Para isto, ele fornece ao usuário dados descrevendo as informações que podem ser manipuladas. Com estas informações, o usuário define quais dados deseja ter acesso e a consulta criada é enviada para o componente motor de busca.

Ao receber a consulta, o motor de busca decide inicialmente em quais fontes de dados buscar aquelas informações. Para isto, ele faz uma consulta ao componente diretório de recursos que o informará quais sítios possuem dados mais relevantes para aquela consulta e onde eles estão localizados. Baseado nesta informação, o motor de busca decompõe a consulta em sub-consultas. Cada uma destas direcionadas a uma das fontes de dados a serem acessadas.

Tendo definido as sub-consultas e seus destinos, ambos são enviados para o componente de extração de *wrappers* e ontologias onde as consultas serão transformadas na linguagem de consulta utilizada pela fonte de dados de destino. Esta transformação nas consultas é feita através das informações previamente mapeadas nas ontologias e armazenadas no componente servidor de mapeamento.

Ao receber os resultados das consultas, estes são transformados pelo componente de extração de *wrappers* e ontologias para o formato utilizado internamente pelo DOME. Por fim, os resultados da busca são integrados pelo componente motor de busca e apresentados para o usuário.

3.2.5.3 – Comentários

O projeto DOME demonstrou a aplicabilidade de ontologias na integração de fontes heterogêneas de informação. É uma proposta bastante interessante uma vez que considera a integração levando em conta aspectos estruturais e, principalmente, semânticos dos dados. Para isto, disponibiliza ao usuário uma infra-estrutura que atende desde requisitos relacionados à manipulação da semântica dos dados a questões de transparência na recuperação de dados armazenados em bases distribuídas.

3.3 – Outras Propostas

Serão apresentadas agora outras propostas (KARSAI, 2000) (PINTO *et al.*, 2003) (WÖB e PÜHRETMAIR, 2001) analisadas. Entretanto, elas não estão descritas de forma detalhada como as anteriores pelo fato do conjunto contemplado na seção 3.2 ser abrangente e terem sido utilizadas como base para a elaboração da proposta apresentada no capítulo 4.

3.3.1 – X-Arc

O X-Arc é parte de uma arquitetura para integração de dados espaciais de um sistema de informação geográfico (PINTO *et al.*, 2003). Ele foi desenvolvido no contexto do SPeCS, um arcabouço que provê um ambiente de trabalho colaborativo para atividades que envolvam o intercâmbio de informações geográficas (MEDEIROS *et al.*, 2000).

A camada de integração, implementada pelo X-Arc, permite a integração e o compartilhamento de fontes de dados heterogêneas utilizando técnicas de mediação (PINTO *et al.*, 2003) e ontologias. Mediação é uma das técnicas de integração de bases de dados que provê acesso a fontes heterogêneas e distribuídas de dados. Seu princípio de funcionamento é baseado na utilização de um componente global, o mediador, e vários componentes locais, os *wrappers*. O mediador acessa as informações compartilhadas pelos *wrappers* e torna possível o acesso às fontes de dados locais (WIEDERHOLD, 1995). Neste contexto, as ontologias servem de base para organizar uma taxonomia com objetivo de prover aos usuários informações sobre o domínio das fontes de dados integradas. Estas informações são fundamentais, por exemplo, para construção de buscas e tradução de buscas entre as fontes de dados.

A figura 3.10 apresenta o X-Arc e seus principais componentes: *Manager* e *Partner*. Cada um deles possui um conjunto de metadados de onde são extraídas as informações para as ontologias. Este conjunto é global para o componente *Manager* e local para cada *Partner*. Simplificadamente, os componentes *Partner* são os responsáveis pela publicação das informações, por tornar possível o acesso às fontes de dados e também por prover *wrappers* para efetuar as traduções necessárias durante o acesso às fontes de dados. Já o componente *Manager*, centraliza os

metadados e tem como uma de suas funções traduzir buscas e encaminhá-las para o *Partner* correspondente.

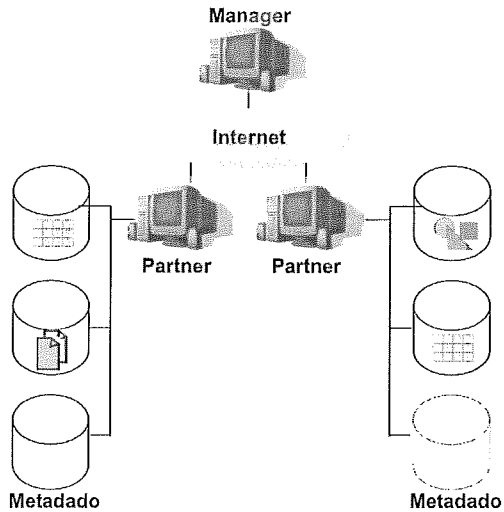


Figura 3.10 – Arquitetura do X-Arc. Adaptada de (PINTO *et al.*, 2003).

3.3.2 – Proposta de (KARSAI, 2000)

(KARSAI, 2000) apresenta uma proposta de integração baseada em três pontos principais: semântica da informação, modelo integrado de dados e utilização de um *middleware* para transporte dos dados.

A semântica da informação e o modelo integrado de dados estão bastante relacionados. A idéia do modelo integrado de dados é possibilitar o armazenamento, de forma centralizada, das informações provenientes das ferramentas integradas. A partir deste modelo é definida a semântica compartilhada entre as ferramentas. Já o objetivo do *middleware* é tornar o acesso das ferramentas ao mecanismo de integração transparente.

A Figura 3.11 apresenta a arquitetura da solução. Ela possui dois componentes principais: o servidor do modelo integrado (IMS, do inglês *integrated model server*) e o adaptador de ferramentas (TA, do inglês *tool adapter*). Estes componentes são os responsáveis, respectivamente, por: efetuar transformações semânticas e armazená-las no modelo integrado de dados e; efetuar transformações sintáticas para que as informações possam ser transmitidas pelo *middleware*.

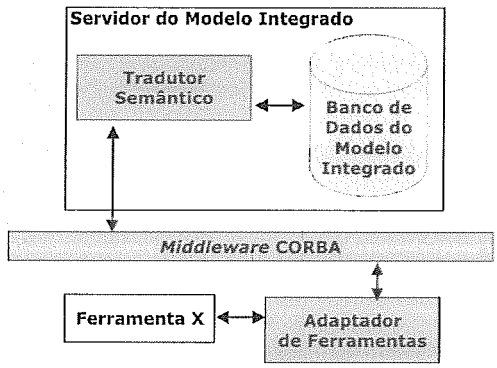


Figura 3.11 – Arquitetura de Integração. Adaptada de (KARSAI, 2000).

3.3.3 – Proposta de (WÖB e PÜHRETMAIR, 2001)

Em (WÖB e PÜHRETMAIR, 2001) é apresentada uma proposta baseada em XML para integração de informações heterogêneas sobre turismo e entre estas e um sistema de informação geográfica. Neste caso, será discutida aqui apenas a questão da integração entre as fontes heterogêneas de informação. A proposta fundamenta-se na separação entre metadados e o processo de transformação dos documentos a serem integrados. Isto torna possível a evolução do domínio de documentos a serem integrados sem a necessidade de modificar o mecanismo de geração de transformadores implementado. Para isto, o mecanismo de integração é formado basicamente por três componentes (ver Figura 3.12):

- Base de conhecimento: onde são armazenados os metadados, informações estruturais descritas em DTDs, das informações (documentos XML) a serem integradas;
- Editor de mapeamento de conhecimento: responsável por prover ao usuário funcionalidades de administração da base de conhecimento dentre as quais se destaca o mapeamento entre as estruturas dos documentos representadas pelos DTDs;
- Processador de integração de dados: responsável pelas transformações nos dados. Neste caso, ao receber a solicitação de transformação, o processador busca na base de conhecimento os metadados, do documento alvo, necessários e efetua as alterações com base nestas informações.

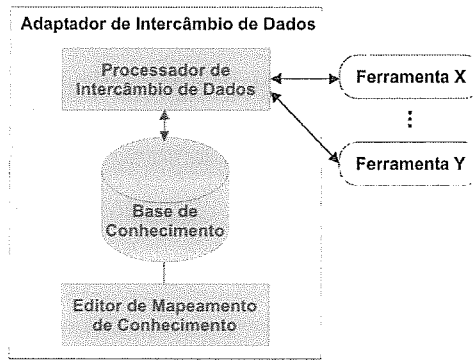


Figura 3.12 – Arquitetura de Integração. Adaptada de (WÖB e PÜHRETMAIR, 2001).

3.4 – Considerações Finais

Neste capítulo foram apresentadas diversas abordagens de integração de ferramentas. O foco principal das discussões estava voltado para a questão da integração de dados, uma das perspectivas de integração. O conjunto de soluções apresentadas é bastante abrangente, variando de abordagens fundamentadas em repositórios centralizados até as mais distribuídas, que fazem uso de *middlewares* para propiciar a comunicação entre os componentes do sistema integrado. Vale destacar também que atualmente, com o amadurecimento da web como principal veículo para intercâmbio de dados, o interesse em criar mecanismos de integração está, em sua maioria, com foco em dados semi-estruturados. Daí a criação de mecanismos de integração que utilizam a XML como tecnologia base de suas abordagens.

Por fim, foi visto também que os pontos chaves para a elaboração de um mecanismo de integração de dados são as disparidades sistêmicas, sintáticas, estruturais e semânticas. As duas primeiras podem ser facilmente resolvidas com a adoção de *middlewares* e de uma linguagem para intercâmbio de dados como XML. Já, as questões estruturais e semânticas são complexas e devem ser consideradas em conjunto. Uma complementa a outra. Neste cenário, ganha destaque a utilização de esquemas para representar a estrutura dos dados e ontologias, sua semântica. Embora ainda não muito utilizada com esta finalidade, existem muitas vantagens na utilização de ontologias para a integração de dados (BARLOW, 2000) (ADAMS *et al.*, 2000):

- as ontologias provêm um rico e pré-definido vocabulário que serve de base para a criação de uma interface conceitual estável com os dados independente de seus esquemas;
- o conhecimento representado pelas ontologias auxilia no processo de transformação dos dados;
- as ontologias apóiam o gerenciamento e o reconhecimento de inconsistências nos dados.

A importância desta revisão bibliográfica está no fato de ter criado as bases para entender o problema da integração investigando algumas de suas diferentes soluções. Cada uma das abordagens vista aqui colaborou para o desenvolvimento da proposta de integração apresentada no capítulo 4.

Capítulo 4

Requisitos para uma Infra-Estrutura de Integração de Ferramentas CASE com XML

Este capítulo apresenta alguns problemas para integração de ferramentas e baseado neles, define um conjunto de requisitos para uma infra-estrutura de apoio à integração de ferramentas CASE. Com estes requisitos especificados, é então apresentada uma abordagem para a integração de artefatos de um mesmo domínio definidos em XML baseada na utilização de esquemas e ontologias.

4.1 – Introdução

Criar mecanismos que possibilitem o intercâmbio de dados entre ferramentas não é uma tarefa trivial. Os problemas estão em sua grande parte relacionados a heterogeneidades nos esquemas e conceitos que representam as necessidades de cada aplicação, na sintaxe utilizada para a manipulação dos dados e em incompatibilidades geradas a partir da escolha de qual plataforma e linguagem de programação utilizar no desenvolvimento das ferramentas.

Por causa de sua natureza autodescritiva, o XML rapidamente emergiu como o padrão para intercâmbio de dados na Internet e representação de dados semi-estruturados (FENG *et al.*, 2002). Entretanto o XML não pode ser considerado uma "bala de prata", existem várias dificuldades que devem ser contornadas.

Este capítulo apresenta uma abordagem para a integração de artefatos de um mesmo domínio definidos em XML. O mecanismo proposto efetua mapeamentos semânticos e estruturais entre artefatos que documentam uma atividade específica, por exemplo, descrição de um processo de desenvolvimento de software ou um relatório de discrepância gerado através de inspeção de software. A proposta está fundamentada na utilização de tecnologias (discutidas no capítulo 2) que servirão como base para lidar com os problemas advindos da heterogeneidade sintática,

estrutural e semântica. Essas tecnologias são XML, XML *Schema* e OWL, respectivamente.

Não será considerada a utilização da tecnologia XMI (*XML Metadata Interchange*). Esta define um esquema para a criação de arquivos XML que representam diagramas UML a serem compartilhados entre ferramentas de modelagem UML. Esta decisão foi tomada a partir da análise dos seguintes fatores:

- Embora seja um padrão para compartilhamento de modelos UML entre ferramentas CASE, o XMI permite que marcações específicas às necessidades de cada aplicação sejam inseridas. Isto causa perda de informação durante a troca de informação entre ferramentas impedindo, muitas vezes, que o modelo seja apresentado visualmente;
- Um documento XMI é por natureza também XML. Assim, ele está no escopo da abordagem de integração apresentada neste capítulo. Ou seja, ele é um caso específico de integração apoiado pela infra-estrutura.

Além desta introdução, este capítulo possui mais 5 seções. Na 4.2 são apresentados alguns problemas para integração de dados classificando-os em quatro categorias principais. A seção 4.3 apresenta os requisitos levantados para a infra-estrutura de integração de ferramentas CASE baseados nos problemas discutidos na seção anterior. Feito isto, a seção 4.4 apresenta a abordagem de integração proposta explicando sua arquitetura e componentes. Na seção 4.5 é apresentado um processo de integração baseado nesta abordagem. Por fim, a seção 4.6 descreve as conclusões deste capítulo enfocando como a abordagem proposta lida com as dificuldades explicitadas na seção 4.2.

4.2 – Problemas para Integração de Dados

Existem diversas propostas de solução para o problema da integração de dados: texto ASCII, repositório compartilhado, interfaces e representações padronizadas, transformação e adaptação, integração baseada em eventos e *frameworks* (HARRISON *et al.*, 2000). Mas existem alguns pontos cruciais em todas as abordagens. Dois dos mais importantes são: lidar com a integração dos esquemas das diversas fontes e contornar a heterogeneidade semântica existente. Relacionados a estes pontos já discutidos no capítulo 3, outros critérios que devem ser considerados

para obtenção de sucesso no desenvolvimento de um mecanismo para integração de dados mas que dificultam seu desenvolvimento são (HARRISON *et al.*, 2000) (MADNICK, 2001):

- **Distribuição:** é natural termos ferramentas executadas em locais separados sendo utilizadas em conjunto para atender a uma necessidade específica (por exemplo, modelagem e codificação de um software) podendo implicar na necessidade de troca de dados entre elas. Assim, é fundamental que exista algum mecanismo responsável por efetuar as transformações necessárias nos artefatos. Neste contexto, o problema da distribuição está relacionado com o acesso das ferramentas ao mecanismo de integração. O ideal é que este acesso seja transparente;
- **Ferramentas Externas:** a utilização de ferramentas desenvolvidas por terceiros na realização das atividades durante, por exemplo, o desenvolvimento de um sistema é uma atitude corriqueira nas organizações. Podem existir mais de uma ferramenta com o mesmo propósito sendo utilizadas em uma mesma atividade por questões de comodidade do usuário ou custo da ferramenta. Ou até mesmo, duas ferramentas contemplando atividades diferentes mas complementares durante o desenvolvimento, nas quais os artefatos gerados por uma ferramenta são ponto de partida para a utilização da outra ferramenta. Assim, o mecanismo para integração deve possibilitar que estas aplicações troquem dados. No contexto de ambientes de engenharia de software isso significa manter o ambiente aberto a ferramentas externas;
- **Evolução Semântica:** o mecanismo de mapeamento entre dados heterogêneos deve ser passível de evolução. Isto por que o domínio de um artefato pode evoluir em decorrência de novos conceitos ou até mesmo, complemento ao modelo do domínio que tenha sido definido de maneira incompleta. Esta questão influencia diretamente na tecnologia adotada para a construção do mecanismo de integração. Caso tenhamos, por exemplo, uma notação para definir ontologias, é preciso que esta notação possibilite o acréscimo de novos conceitos sem perda ou necessidade de reestruturação dos conceitos previamente definidos. É interessante notar também que o problema da evolução semântica está relacionado principalmente com a manutenção do mapeamento estrutural e semântico entre os dados. Uma mudança na semântica pode incorrer na necessidade de redefinição dos

mapeamentos feitos e na definição de novas regras de transformação para cada documento afetado;

- **Integridade Semântica:** quando são definidos os conceitos presentes em um domínio e o esquema dos artefatos relacionados a este domínio, estão também sendo definidas restrições para que o artefato tenha um significado correto ao ser interpretado. Neste caso, durante o processo de mapeamento entre as fontes de dados, é necessária a utilização da semântica e dos esquemas para apoiar a definição, manutenção e transformação dos artefatos fazendo com que a semântica e o esquema especificado para cada um deles não sejam "burlados". Ou seja, depois de serem manipulados, os artefatos devem permanecer válidos de acordo com sua semântica e seu esquema;
- **Abstração de Dados:** este é um critério básico, mas também um dos mais difíceis de serem alcançados para aplicações que se proponham a possibilitar a integração de dados. Abstrair dados é poder manipulá-los independente da aplicação que os criou. Neste caso, o significado da palavra independente deve ser amenizado a depender do objetivo do mecanismo de integração em questão;
- **Contexto da Informação:** considerar o contexto no qual a informação está inserida é também um critério básico a ser contemplado em mecanismos de integração. Por exemplo, a palavra manga por si só não traz consigo seu significado; ela pode significar uma fruta como também parte de uma roupa. Assim, sempre que uma informação puder ser interpretada de mais de uma maneira, o que nos informará seu verdadeiro significado é o cenário na qual estiver inserida. Podemos nomear este cenário como sendo o domínio do assunto sendo discutido. Assim, na criação de mecanismos de integração é preciso estar definido o domínio do artefato a ser integrado.

Estes problemas podem ser classificados em quatro grupos e ter sua origem:

- Nos possíveis sistemas operacionais, protocolos de rede e *hardware* utilizados por cada ferramenta, **discrepância sistêmica**. Neste grupo se encontra o item distribuição;
- Nas possíveis abordagens para armazenamento e manipulação dos dados, **discrepância sintática**. Neste grupo se encontram os itens ferramentas externas e abstração de dados;

- Nas diversas formas possíveis de se organizar um documento, **discrepância estrutural**. Neste grupo se encontram os itens ferramentas externas, abstração de dados e integridade semântica;
- Nas variadas possibilidades para definição de um mesmo conceito, **discrepância semântica**. Alguns exemplos são: a utilização de sinônimos e níveis variados de granularidade na definição de um mesmo conceito. Neste grupo se encontram os itens ferramentas externas, abstração de dados, contexto da informação, evolução semântica e integridade semântica.

Neste contexto, a utilização de XML tem sido "pregada" como uma solução para todas as dificuldades de integração. Entretanto o XML possui alguns problemas. Reconhecendo suas limitações, várias comunidades, incluindo a W3C, têm trabalhado em tecnologias correlatas (SELIGMAN e ROSENTHAL, 2001).

A proposta mais simples para intercâmbio de informações em XML é baseada na definição e adoção de um conjunto de marcações para organizar o conteúdo do documento. Entretanto, chegar a um acordo sobre quais marcações devem estar contempladas e como elas estarão estruturadas não é uma tarefa trivial. Visto que os interesses são muitas vezes conflitantes, não existe uma melhor solução (depende do contexto de sua utilização) e ainda assim, as marcações definidas estão propensas à evolução.

Outros desafios baseados nos citados anteriormente e que se encontram nos grupos de discrepância estrutural e semântica, no contexto da utilização da XML, são (MADNICK, 2001):

- **Múltiplos Padrões:** como visto, a existência de variados conjuntos de marcações definidos para atender a um mesmo domínio de aplicação dificulta a troca de dados. Para lidar com isto, é possível o desenvolvimento de regras de transformação definidas em XSLT para efetuar o mapeamento entre as marcações. Entretanto, a especificação e manutenção destas regras de mapeamento não é uma tarefa trivial. Basta atentar para o fato da quantidade de regras crescer exponencialmente – é definida uma regra para cada elemento presente em um documento XML.
- **Semântica das Marcações:** imaginemos agora uma situação em que um conjunto de marcações definidas para estruturar o conteúdo de um documento tenha sido definido. Embora já seja um grande passo para a comunicação entre ferramentas, ainda assim o significado que cada

marcação possui para cada aplicação é variável. Por exemplo: imaginemos um documento descrevendo a configuração de um computador. A capacidade de armazenamento poderia estar definida pela marcação <*espacoDisco*> mas, a unidade de medida seria *megabytes* ou *gigabytes*? Este é um problema de difícil solução, principalmente se o número de aplicações a serem integradas for grande.

4.3 – Requisitos para uma Infra-Estrutura de Apoio à Integração de Ferramentas CASE

Baseado no cenário da seção 4.2, este tópico apresenta os requisitos identificados para a elaboração de uma abordagem de integração para possibilitar a interoperabilidade entre ferramentas e lidar com os quatro grupos de discrepância envolvendo a integração de dados. Vale ressaltar aqui, antes de discutir cada grupo de discrepância, que os requisitos para lidar com os problemas mais genéricos como os citados nos itens ferramentas externas e abstração de dados da seção 4.2 são o conjunto total de requisitos definidos para cada um dos grupos de discrepância.

4.3.1 – Discrepância Sistêmica

Neste caso, é importante contemplar em uma solução de integração mecanismos que tornem possível o envio de informações entre as aplicações e entre estas e o integrador (BOMPANI *et al.*, 2000) (EMMERICH, 2000) (MADNICK, 2001).

A necessidade de incorporar facilidades de comunicação entre aplicações pode variar de acordo com a arquitetura de integração utilizada. Para arquiteturas baseadas em repositórios centralizados de dados, prover internamente facilidades para envio de informações pode ser considerado um requisito básico. Para arquiteturas menos acopladas, a solução de comunicação pode ser feita de forma mais independente.

Já a comunicação entre as aplicações e o mecanismo de integração deve estar presente em ambos os tipos de arquiteturas.

4.3.2 – Discrepância Sintática

Neste caso, é necessário definir uma sintaxe comum a ser utilizada pelas ferramentas a serem integradas para representação dos seus artefatos (HARRISON *et al.*, 2000) (MADNICK, 2001) (MASCOLO *et. al.*, 2001) (RODRÍGUEZ-GIANOLLI, 2001). Assim, deve-se selecionar uma tecnologia de definição de documentos que esteja sendo amplamente utilizada por ferramentas em geral.

Embora a definição de uma tecnologia comum para representação das informações seja um passo fundamental, vale ressaltar que a definição resulta em um apoio parcial ao problema da discrepância sintática. Isto por que podem existir sistemas legados que não exportem seus dados no formato definido, tornando necessária a criação de conversores entre as diferentes representações sintáticas.

4.3.3 – Discrepância Estrutural e Semântica

Estes são dois pontos chave do problema da integração e estão discutidos em conjunto por serem fortemente relacionados; a estrutura de um documento ajuda na descrição de sua semântica. Para lidar com as disparidades semânticas e estruturais entre artefatos, é preciso criar mecanismos que tornem possível a transformação nos dados (ADAMS *et al.*, 2000) (CUI *et al.*, 2001b) (MADNICK, 2001) (RODRÍGUEZ-GIANOLLI, 2001). Por ser uma tarefa complexa e muitas vezes repetitiva, o ideal é que um mecanismo de integração disponibilize apoio semi-automatizado a esta atividade.

Este apoio semi-automatizado implica em outras necessidades como notações para especificar e representar a estrutura e semântica dos artefatos. Frente às dificuldades apresentadas na seção 4.2, uma solução de integração para lidar com:

- **Contexto da informação:** deve limitar o escopo dos artefatos a serem integrados para minimizar problemas como ambigüidade e a definição de um termo com mais de um significado possível;
- **Evolução semântica:** deve possibilitar que a semântica definida para um artefato seja evoluída. Ou seja, a notação para representação da semântica deve ser extensível. Além disso, deve minimizar o esforço de mapeamento e geração de transformadores caso a semântica do artefato seja alterada;

- **Integridade semântica:** deve limitar e guiar a interação do usuário durante o mapeamento entre as fontes de informação para minimizar o problema de definições incorretas durante o mapeamento.

Por fim, outro problema que deve ser considerado é a construção de conversores. Esta deve ocorrer de forma automatizada com base em informações estruturais e semânticas previamente mapeadas.

Na próxima seção será apresentada a proposta de apoio à integração de dados detalhando seu funcionamento e como lida com os requisitos levantados nesta seção e os problemas descritos no tópico anterior.

4.4 – Proposta de Apoio Ferramental para a Integração de Ferramentas CASE

Antes de entrar em detalhes de cada componente da infra-estrutura de integração e como eles lidam com os requisitos identificados na seção anterior, analisemos de forma abstrata a idéia que norteou a elaboração desta proposta. A Figura 4.1 apresenta a metáfora de uma infra-estrutura de integração com baixo acoplamento para artefatos de um mesmo domínio descritos em XML. Basicamente, a infra-estrutura visa permitir que usuários possam, a partir de sua estação de trabalho, utilizar softwares que lhes sejam mais familiares na execução de suas tarefas podendo compartilhar informações com outros usuários envolvidos em atividades semelhantes mas que estejam utilizando ferramentas diferentes.

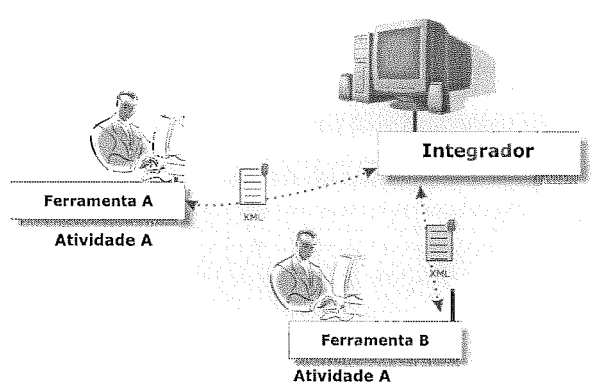


Figura 4.1 – Idéia simplificada da Arquitetura para Integração de Ferramentas.

Para realizar a integração, esta metáfora utiliza o seguinte princípio de funcionamento. Um mapeamento é efetuado entre os artefatos das ferramentas que se deseja integrar e a partir dele um conversor, que permite a transformação entre esses artefatos, é gerado (ver Figura 4.2).

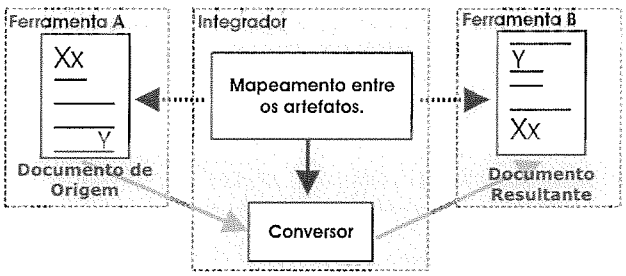


Figura 4.2 – Princípio de Funcionamento.

Com base nestes princípios e nos problemas relacionados às discrepâncias sistêmicas, sintáticas, estruturais e semânticas, foi elaborada a infra-estrutura apresentada na Figura 4.3.

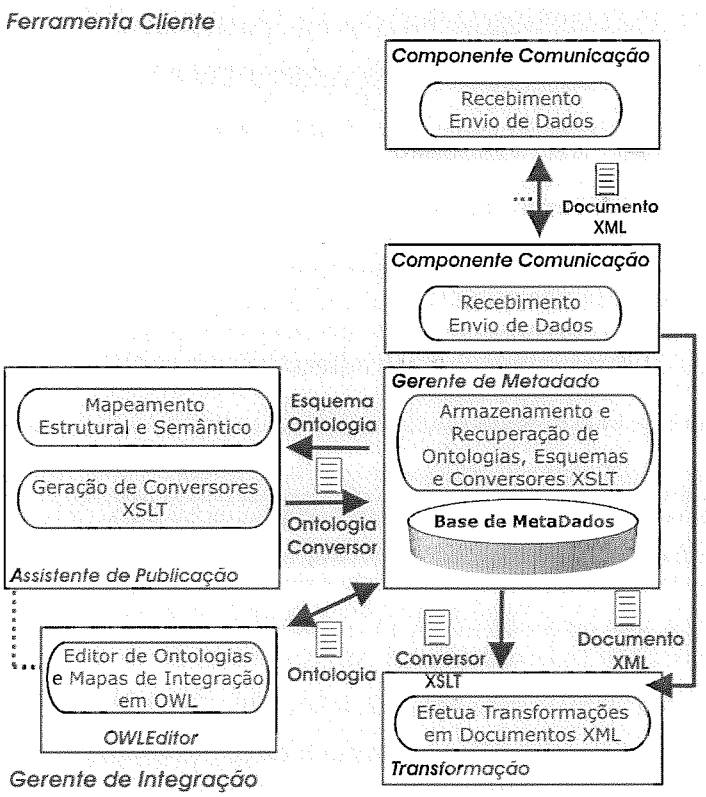


Figura 4.3 - Abordagem para Integração de Ferramentas.

Para possibilitar o intercâmbio de informação, a proposta é composta de duas ferramentas principais: Cliente e Gerente de Integração. A ferramenta cliente será responsável, através do componente comunicação, por permitir ao usuário, ter acesso às funcionalidades do Gerente de Integração de forma transparente e poder, assim, exportar seus dados para estes serem utilizados por outra ferramenta.

O Gerente de Integração é o foco da arquitetura de integração e é nele que serão tratadas as questões referentes aos mapeamentos estruturais e semânticos dos dados para seja possível a geração de regras de mapeamento através das quais os dados serão integrados. Para isto, o Gerente de Integração é formado por cinco componentes: comunicação, gerente de metadado, assistente de publicação, editor de ontologia e mapa de integração (OWLEditor) e, transformação. Em linhas gerais, eles atuam em conjunto na definição da semântica dos documentos a serem integrados, no mapeamento das informações necessárias para construção de conversores, no armazenamento e recuperação dessas informações e na comunicação com a ferramenta Cliente.

O Gerente de Integração irá mapear os esquemas dos documentos a serem integrados em um mapa de integração. Este é um documento criado a partir de uma ontologia, descrevendo a semântica de um artefato, acrescida de informações estruturais provenientes dos esquemas dos artefatos. Ele centraliza informações semânticas e estruturais e é a base para a geração de conversores. Assim, é possível fazer uso da semântica descrita em uma ontologia para a definição de vários mapas de integração entre ferramentas que se deseja integrar (ver Figura 4.4).

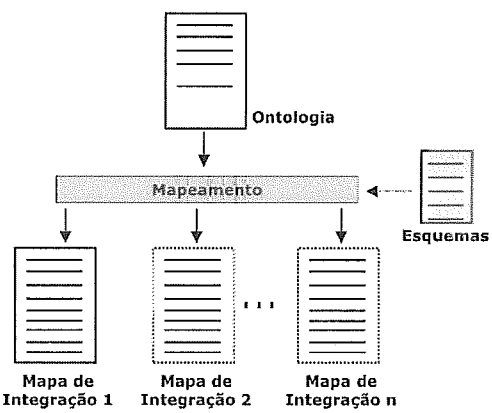


Figura 4.4 – Criação de Mapas de Integração.

Simplificadamente, os componentes comunicação da Ferramenta Cliente e do Gerente de Integração lidam com a discrepância sistêmica, os componentes Assistente de Publicação e Transformação do Gerente de Integração lidam com as discrepâncias estruturais e semânticas e os demais componentes servem de apoio ao funcionamento destes.

Algumas das principais características da infra-estrutura proposta são:

- utilização da XML como padrão para intercâmbio de informações;
- uso das tecnologias correlatas ao XML para permitir a integração de fontes de dados heterogêneas;
- separação entre o mecanismo de integração e transporte dos dados (ver Seção 4.4.1.5);
- consideração da semântica das informações a serem integradas através do uso de ontologias;
- mapeamento semi-automatizado das informações estruturais dos esquemas nas ontologias;
- geração automatizada dos conversores.

Alguns fatos que apoiaram a decisão de utilizar XML e suas tecnologias correlatas foram:

- a disponibilidade de uma grande quantidade de ferramentas que auxiliam a manipulação de arquivos XML (EMMERICH et al., 2001);
- a presença crescente de interfaces para importação e exportação de dados descritos em XML nas aplicações (EMMERICH et al., 2001);
- o fato desta tecnologia ser o padrão para intercâmbio de dados na Internet (COLLINS *et al.*, 2002) (HASSELBRING, 2000);
- o suporte do W3C que tem trabalhado em tecnologias correlatas como XSLT, XML *Schema* e *Web Ontology Language* (OWL) (FALLSIDE, 2001) (KAY, 2003) (HARMELEN *et al.*, 2003);
- o fato da utilização da linguagem XML servir de base sintática para o intercâmbio de informações entre as aplicações.

Este último ponto é de grande importância pois lida parcialmente com a discrepância sintática. Parcialmente pelo fato de poderem existir sistemas legados que não exportem seus dados no formato XML. Neste caso, é necessária a criação de um

conversor para transformar os dados legados para o formato XML. Este último ponto não é contemplado nesta proposta uma vez que o escopo definido é o de artefatos de um mesmo domínio descritos em XML.

A partir de agora serão apresentados, em detalhes, os componentes que formam a infra-estrutura.

4.4.1 – Gerente de Integração

Parte integrante da infra-estrutura proposta neste trabalho, a ferramenta Gerente de Integração possui as seguintes responsabilidades: (1) efetuar transformações nos documentos XML; (2) gerenciar metadados (esquemas, conversores, ontologias e mapas de integração); (3) estabelecer mecanismos para mapeamento semi-automatizado entre informações estruturais e semânticas; (4) criar de forma automatizada transformadores (XSLTs); (5) possibilitar a definição da semântica dos artefatos a serem integrados através de ontologias e; (6) tornar transparente o acesso ao mecanismo de integração para seus usuários.

É nesta ferramenta que são contempladas as facilidades responsáveis por tratar os problemas de integração descritos no tópico 4.2. O Gerente de Integração é formado por cinco componentes (ver Figura 4.3): Transformação, Gerente de Metadado, Assistente de Publicação, OWLEditor e Comunicação.

4.4.1.1 – Gerente de Metadado

Todas as ações efetuadas na utilização do mecanismo de integração têm como resultado ou pré-requisito um documento XML seja ele especificando um esquema (XML *Schema*), uma ontologia (OWL), um mapa de integração (ontologia + informações de esquema), um conversor (XSLT) ou mesmo o dado propriamente dito (XML). Para lidar com estes documentos tornando possível seu armazenamento e recuperação, foi criado o componente Gerente de Metadado. Suas principais funcionalidades são: (1) organizar a estrutura de armazenamento; (2) salvar os variados tipos de documentos manipulados pelo Gerente de Integração e seus componentes; (3) tornar possível a remoção destes documentos; (4) disponibilizar interfaces para interação com os demais componentes do Gerente de Integração.

Tendo definido estas facilidades, foi feita uma análise das abordagens de armazenamento de documentos XML disponíveis. Três alternativas foram

consideradas: sistema de arquivos, banco de dados relacional e banco de dados XML nativo.

Na estratégia de armazenamento em sistemas de arquivos, os documentos XML são mantidos como documentos texto em estruturas de diretórios. O principal benefício desta abordagem é a facilidade de implementação sem a necessidade de uso de um mecanismo mais robusto (banco de dados) para gerência dos documentos. Entretanto, existem alguns problemas com esta abordagem (TIAN et al., 2002): (1) dificuldade em efetuar buscas e atualizações nos elementos que compõem os documentos; (2) necessidade de colocar os documentos integralmente em memória antes de serem acessados.

Já o emprego de SGBDs (Sistema Gerenciador de Banco de Dados) relacionais para o armazenamento de documentos XML usufrui da maturidade, segurança e do desempenho de tecnologias já consolidadas além de permitir integração entre as informações provenientes dos dados em XML com as das aplicações legadas. Entretanto, não há um mapeamento direto entre o modelo de dados XML para o modelo de dados de um SGBD relacional (SUCIU, 2001). Por definição, o modelo de dados relacional é normalizado, não aninhado e fragmentado em diversas relações enquanto que dados XML são aninhados, monolíticos e não normalizados. Fazer este mapeamento significa então, identificar e ajustar as discrepâncias. Este processo pode incorrer em perdas semânticas e ou estruturais e no aumento de complexidade para representar os dados XML. Por fim, esta complexidade compromete a clareza e o desempenho do acesso aos dados. A Figura 4.5 apresenta uma visão simplificada do problema: Como armazenar estruturas irregulares sendo as tabelas estruturas regulares por definição? Como traduzir consultas escritas para retornar dados em XML para consultas relacionais? Como, tendo localizado os dados pesquisados, transformá-los em XML? (SUCIU, 2001).

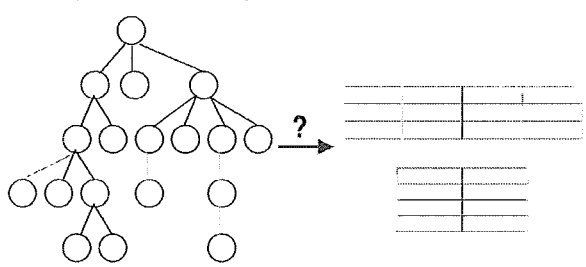


Figura 4.5 O problema do armazenamento de documentos XML em um modelo de dados relacional (SUCIU, 2001).

Outras discrepâncias entre dados XML e bancos de dados relacionais podem ser encontradas em (CHAMPION, 2001), (KAPPEL, et. Al., 2001), (SHANMUGASUNDARAM, et. Al., 1999) e (VARLAMIS, 2001). Para eles, a principal dificuldade é causada pela heterogeneidade existente entre os modelos de dados do banco e do padrão XML.

Por fim, foram analisados os bancos de dados XML nativos. Como o próprio nome já diz, ele utiliza o padrão XML como seu modelo de dados o que traz grandes benefícios para a manipulação de documentos XML. Eles possuem mecanismos que facilitam a busca, armazenamento e atualização dos elementos que compõem um documento XML. Apesar deste diferencial, estes bancos ainda não estão maduros e são implementados muitas vezes utilizando tecnologias proprietárias que podem ser descontinuadas.

Feita está análise inicial, partiu-se para a decisão de qual abordagem utilizar. Os principais fatores considerados foram: a facilidade de uso da estratégia de armazenamento e o atendimento aos requisitos definidos para o Gerente de Metadado. Como não havia necessidade de mecanismos mais elaborados para lidar com a manipulação dos documentos e estes sempre serem manipulados por inteiro sem necessidade de mecanismos de quebra de arquivo e busca de informações específicas de dados, a estratégia escolhida foi a baseada em sistema de arquivos. Dessa forma, procurou-se pela abordagem que resolvesse o problema corretamente da forma mais simples.

Para lidar com esta estratégia de armazenamento, foram implementados dois módulos no Gerente de Metadado: manipulação dos arquivos e configuração. O módulo de configuração permite ao usuário especificar o local onde serão mantidos os documentos. Quando um diretório é escolhido, são criados automaticamente quatro sub-diretórios para armazenar os diferentes tipos de documentos mantidos: conversores, esquemas, ontologias e mapas de integração. Estas informações são persistidas em um arquivo de configuração do Gerente de Metadado.

Já o módulo onde foram implementadas as funções para manipulação dos arquivos é responsável por: (1) apresentar, a depender do contexto (ontologia, mapa de integração, esquema ou conversor) de sua utilização, os arquivos mantidos; (2) disponibilizar interfaces para salvar documentos em suas devidas pastas e; (3) remoção dos documentos. Esta última funcionalidade é utilizada diretamente no Gerente de Metadado enquanto que as duas primeiras são utilizadas pelos componentes Assistente de Publicação, OWLEditor e Transformação.

4.4.1.2 – OWLEditor

Ontologias podem auxiliar na formalização da semântica presente em artefatos. Entretanto, construí-las não é uma tarefa trivial. Envolve a definição de conceitos, propriedades, axiomas e restrições. Isto implica na necessidade de um apoio ferramental para seu desenvolvimento (LASSILA *et al.* 2000).

Segundo Falbo *et al.* (1998), um processo para definição de ontologias é composto de seis atividades:

- **Identificação de Propósito e Especificação de Requisitos:** onde são identificados o propósito, a aplicabilidade e, feito isto, os requisitos da ontologia.
- **Captura da Ontologia:** onde é capturada a conceituação do universo de discurso, com base no propósito e requisitos identificados na fase anterior.
- **Formalização da Ontologia:** nesta etapa, deve-se estabelecer um formalismo para representar explicitamente a conceituação capturada na fase anterior em uma linguagem formal.
- **Integração com Ontologias Existentes:** esta etapa deve ser considerada quando durante o processo de captura e/ou formalização surja a necessidade de integrar a ontologia em questão com outras já existentes, visando aproveitar conceituações previamente estabelecidas.
- **Avaliação:** a ontologia deve ser avaliada durante todo o processo para verificar se satisfaz os requisitos estabelecidos na especificação.
- **Documentação:** todo o desenvolvimento da ontologia deve ser documentado.

Na integração de dados, ontologias são úteis no compartilhamento da informação através da definição de um vocabulário comum. Neste contexto, foi desenvolvido o componente OWLEditor. Ele é uma ferramenta para definição de ontologias em OWL e apóia duas das atividades desse processo: captura e formalização. Para isto, ele possui três módulos:

- **Definição de Metadados:** no qual são manipuladas informações que descrevem a ontologia e os engenheiros que têm trabalhado na sua definição/manutenção.
- **Definição dos Conceitos:** no qual são manipulados os conceitos e, as restrições e axiomas relacionados a ele. Este módulo possui também

algumas funcionalidades que impedem que o engenheiro do domínio defina relações inconsistentes ou incoerentes na ontologia. Por exemplo, o módulo não permite que um conceito seja ao mesmo tempo equivalente e disjunto a um outro conceito.

- **Definição das Propriedades:** no qual são manipuladas as propriedades e as restrições relacionadas a elas. Seguindo o mesmo princípio de funcionamento do módulo de definição dos conceitos, do lado esquerdo são listadas as propriedades e do direito, as restrições e axiomas que podem ser manipuladas. Este módulo também possui uma funcionalidade que impede a definição inconsistente ou incoerente das propriedades. Neste caso, não é possível definir uma propriedade como sendo equivalente e inversa a uma mesma propriedade.

Entretanto, por causa da complexidade decorrente dos variados tipos de marcações presentes na OWL e nas diversas possibilidades de combinação entre elas, foi necessário definir previamente um conjunto de marcações e uma forma de combinação entre elas a serem consideradas. O conjunto de marcações consideradas da linguagem OWL pode ser visto no Anexo A.

Vale destacar aqui também que este componente também auxilia a manutenção dos mapas de integração. Como estes são gerados a partir de ontologias previamente definidas, a manipulação dos conceitos e propriedades presentes no mapa de integração se torna possível.

Por fim, este componente interage com outros dois: Assistente de Publicação e Gerente de Metadado. Para o primeiro, disponibiliza serviços para definição de conceitos e propriedades necessários durante o processo de mapeamento. Para o segundo, utiliza os serviços de armazenamento para ter acesso às ontologias e efetuar as manipulações necessárias nos arquivos.

4.4.1.3 Assistente de Publicação

Este é o componente chave da infra-estrutura de integração. Nele são efetuados os mapeamentos estruturais e semânticos e, gerado o conversor para transformações entre os artefatos. Para criar este mecanismo foram utilizadas as tecnologias XML *Schema*, OWL e XSLT (ver capítulo 2). As duas primeiras voltadas para o mapeamento, a terceira para transformações.

Para lidar com as discrepâncias estruturais e semânticas, poder-se-ia pensar em uma solução onde para cada relacionamento entre aplicações houvesse a necessidade de mapeamento entre seus artefatos. Isto teria como conseqüência a necessidade de desenvolvimento manual de dois *drivers* para importação e exportação dos dados para cada par de aplicações. Entretanto, a manutenção de uma solução seguindo este princípio seria ineficaz e ineficiente. Para isto, basta atentarmos para o fato da quantidade de mapeamentos e *drivers* a serem mantidos crescerem a uma razão quadrática. Por exemplo, para duas aplicações deveriam ser efetuados dois mapeamentos, para três seis mapeamentos, para quatro dez mapeamentos e assim por diante. Percebe-se facilmente que a quantidade de mapeamento a serem feitos obedece à expressão $n^2 - n$ onde n é o número de aplicações a serem integradas (ver Figura 4.6a).

O mecanismo aqui proposto para lidar com este problema é fundamentado na utilização de ontologias para representar a semântica dos artefatos e esquemas contendo as informações estruturais. De forma simplificada, a idéia é mapear os esquemas dos documentos, a serem integrados, na ontologia. A partir do momento que este mapeamento é realizado, a ontologia passará a ser denominada Mapa de Integração. Este conterá informações semânticas e estruturais que guiarão a geração automatizada dos conversores entre as aplicações cujo esquema tenham sido mapeados. Esta solução lida diretamente com o problema citado no parágrafo anterior: (1) a quantidade de mapeamentos a serem feitos passa a crescer de forma linear uma vez que as informações necessárias para a criação dos conversores só precisam ser mapeadas uma vez para cada aplicação (ver Figura 4.6b) e, (2) os conversores são gerados de forma automatizada diminuindo consideravelmente o trabalho para a integração das ferramentas.

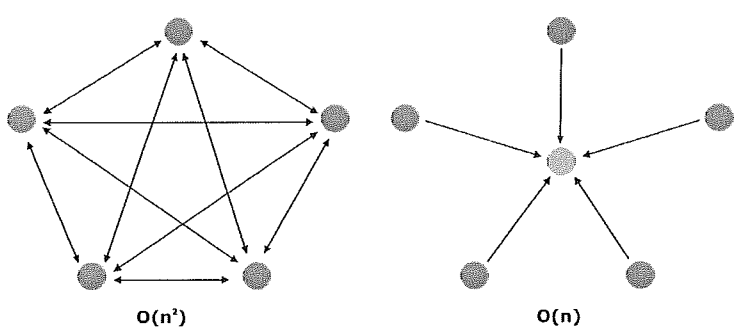


Figura 4.6 – Quantidade de mapeamento a serem feitos.

Serão apresentados agora os mecanismos de mapeamento entre artefatos, de geração dos conversores e remoção das informações mapeadas no mapa de integração.

4.4.1.3.1 – Mapeando Esquemas no Mapa de Integração

O mapeamento tem como entradas o Mapa de Integração definindo a semântica do artefato e o esquema especificando sua estruturação. O resultado do processamento será a inclusão de um conjunto de informações no Mapa de Integração que permitirão a geração dos conversores.

Como dito anteriormente, as linguagens escolhidas para definição de esquemas e Mapas de Integração foram a XML *Schema* e a OWL, respectivamente. Assim como a OWL, a XML *Schema* também teve seu escopo limitado (ver Anexo B). Tendo especificado o escopo da OWL e do XML *Schema*, foi necessário definir algumas diretrizes que serviram de base para a elaboração do algoritmo de mapeamento das informações estruturais no mapa de integração:

- **Diretriz 01:** elementos de tipo complexo devem ser mapeados para conceitos na ontologia;
- **Diretriz 02:** elementos de tipo nativo (aqueles previamente definidos como inteiro, data e *string* dentre outros) devem ser mapeados para propriedades na ontologia;
- **Diretriz 03:** um atributo de um elemento de tipo complexo ou nativo deve ser mapeado como atributo para seu conceito ou propriedade correspondente e seu conteúdo pode ter origem em um outro atributo ou conceito do documento de origem;
- **Diretriz 04:** uma seqüência de elementos definidos pela marcação *sequence* no esquema deve ser mapeado na ontologia como uma seqüência de elementos dentro do conceito ao qual o elemento de tipo complexo corresponde;
- **Diretriz 05:** a quantidade de vezes (uma ou mais que uma) que um elemento poderá aparecer definido pela marcação *maxOccurs* ou *cardinality* no esquema deve ser mapeada para o elemento correspondente na ontologia, seja ele conceito ou propriedade;

- **Diretriz 06:** gerar um identificador a ser inserido nos elementos mapeados na ontologia a partir de seu esquema de forma que seja possível identificar de qual esquema são provenientes as informações mapeadas.

Como a OWL não especifica marcações próprias para efetuar este mapeamento, foi criado um conjunto de marcações de acordo com as necessidades identificadas a partir das diretrizes formuladas a serem inseridas no mapa de integração durante o mapeamento. Estas marcações guiarão a geração dos conversores definidos em XSLT. Como estas marcações serão inseridas em conceitos ou propriedades do mapa de integração, teremos dois sub-conjuntos de marcadores. As Figuras 4.7 e 4.8 apresentam as marcações definidas para conceitos e propriedades respectivamente.

```
1: <map:mapFile map:id="">
2:         <map:schema>
3:             <map:attribute element=""></map:attribute>
4:             <map:name></map:name>
5:             <map:sequence>
6:                 <map:next moreOne="0"></map:next>
7:                 <map:next moreOne="1"></map:next>
8:             </map:sequence>
9:         </map:schema>
10: </map:mapFile>
```

Figura 4.7 - Marcações definidas para mapeamento nos conceitos do mapa de integração.

Analisando a Figura 4.7 tem-se o seguinte:

- **Linha 1:** é informado que existe uma informação de esquema mapeada no mapa de integração e de qual esquema as informações são provenientes (*map:id*);
- **Linha 2:** empacota as informações provenientes do esquema para organizar as marcações de mapeamento;
- **Linha 3:** informa quais atributos pertencem àquele conceito no esquema mapeado. Neste caso, pode existir mais de um atributo e, por conseqüência, mais de um marcador *map:attribute*. Este marcador pode possuir também um atributo *element* que indica onde será buscado o conteúdo para o atributo em questão;

- **Linha 4:** informa o nome que aquele conceito representa no artefato do esquema mapeado;
- **Linha 5:** indica que aquele conceito possui propriedades ou outros conceitos e que estes estão organizadas em uma determinada seqüência;
- **Linhas 6 e 7:** apresentam os elementos (propriedades ou outros conceitos) que compõem o conceito atual. Cada marcador *map:next* informa qual o próximo elemento – conceito ou propriedade – a ser mapeado e o atributo *moreOne* indica se aquele elemento mapeado do esquema pode (1) ou não (0) se repetir. A quantidade de vezes que o marcador *map:next* pode aparecer é ilimitado.

Estes marcadores contemplam as diretrizes 01, 03, 04, 05 e 06. Agora, analisando a Figura 4.8, tem-se o seguinte:

- **Linha 1:** é informado que existe uma informação de esquema mapeada no mapa de integração e de qual esquema as informações são provenientes (*map:id*);
- **Linha 2:** empacota as informações provenientes do esquema para organizar as marcações de mapeamento;
- **Linha 3:** informa o nome que aquela propriedade representa no artefato do esquema mapeado;

```
1: <map:mapFile map:id="">
2:     <map:schema>
3:         <map:name></map:name>
4:     </map:schema>
5: </map:mapFile>
```

Figura 4.8 - Marcações definidas para mapeamento nas propriedades do mapa de integração.

Este conjunto de marcadores dá suporte às diretrizes 02, 03 e 06.

Tendo definido as diretrizes e conjunto de marcações necessárias, foi então desenvolvido um algoritmo que efetuasse o mapeamento estrutural/semântico. O algoritmo está fundamentado na comparação entre os nomes dos conceitos e propriedades do esquema e do mapa de integração para efetuar o mapeamento. Para dar inicio ao processamento, pega-se no esquema o elemento que representa a raiz do documento XML. É necessário começar pela raiz pelo fato dos demais elementos

serem seus descendentes e isto facilitar o procedimento uma vez que a navegação no esquema se torna mais simples. A partir deste elemento do esquema, é buscado no mapa de integração algum conceito ou propriedade que possua o mesmo nome de acordo com as regras estabelecidas nas diretrizes 01 e 02. Ou seja, para elementos de tipo complexo são percorridos apenas os conceitos; para elementos de tipo nativo são percorridas as propriedades. Tendo identificado uma similaridade entre o nome de um elemento complexo e um conceito, as informações definidas na Figura 4.7 são inseridas no mapa de integração.

A partir do momento que um primeiro elemento tenha sido mapeado, as informações contidas no marcador *map:sequence*, ou seja, os nós filhos do elemento mapeado, guiarão o restante do mapeamento. Para cada elemento especificado no marcador *map:next* é verificado no esquema se ele é um elemento de tipo complexo ou nativo. Em ambos os casos, é verificada a possibilidade dele ocorrer mais de uma vez e são inseridas informações referentes à repetição de elementos. Caso seja complexo, procura dentre os conceitos definidos no mapa de integração algum que possua o mesmo nome e o procedimento descrito até aqui se repete. Caso seja nativo, procura no mapa de integração uma propriedade cujo nome seja o mesmo do elemento e insere as informações definidas na Figura 4.8.

Entretanto, como o mapeamento é baseado em similaridades entre os nomes dos elementos, propriedades e conceitos, existe a possibilidade do algoritmo não efetuar o mapeamento de alguns elementos. Por este motivo, a abordagem de mapeamento aqui proposta é semi-automatizada e os nomes dos conceitos e das propriedades que não foram possíveis de serem mapeadas são armazenadas em uma lista para que o engenheiro de software que esteja configurando o ambiente de integração informe como será realizado seu mapeamento. O algoritmo abaixo representa o processo de mapeamento descrito:

1. Receba a ontologia e o esquema que serão utilizados no mapeamento;
2. Adicione um cabeçalho na ontologia contendo a legenda e um identificador para esquema que está sendo mapeado;
3. Percorra os elementos (filhos da raiz) do esquema e para cada marcação *xs:element*:
    a. Procure na ontologia com base no nome do elemento o local onde as informações de mapeamento serão inseridas.
        i. Se achar, insira informações de mapeamento na ontologia;
        ii. Se não achar, coloque na lista de conceitos não mapeados;

b. Verifique se existe algum atributo a ser mapeado. Caso positivo adicione informações de mapeamento sobre atributos;
c. Verifique se existe uma seqüência (marcador *xs:sequene* do esquema) de elementos a ser mapeada. Caso positivo: para cada elemento da seqüência:
    i. Insira informações de mapeamento na ontologia;
    ii. Verifique se pode ocorrer mais de uma vez. Caso positivo, insira as informações necessárias na ontologia;
    iii. Efetue o mapeamento do elemento ao qual o nó atual faz referência na ontologia.
        1. Se o nó ao qual faz referência é um elemento de tipo complexo volte para o passo 3.a.
        2. Caso o elemento ao qual o nó faz referência seja de um tipo nativo, procure na ontologia a propriedade correspondente ao elemento atual.
            a. Caso encontre, insira na propriedade da ontologia as informações de mapeamento necessárias;
            b. Caso não encontre, insira o nome do elemento atual na lista de propriedades não mapeadas.
4. Salve a ontologia com as informações mapeadas;
5. Solicite ao usuário que finalize o mapeamento informando da lista de conceitos os elementos correspondentes na ontologia.
    a. Se necessário crie um novo conceito definido pelo usuário na ontologia;
6. Repita o algoritmo a partir do passo 3.a passando as informações obtidas com o usuário por parâmetro.
7. Solicite ao usuário que finalize o mapeamento informando da lista de propriedades os elementos correspondentes na ontologia.
    a. Se necessário crie uma nova propriedade definida pelo usuário na ontologia;
8. Repita o algoritmo a partir do passo 3.c.iii.2 passando as informações obtidas com o usuário por parâmetro.
9. Salve o mapa de integração com as informações mapeadas e finalize o mapeamento.

4.4.1.3.2 – Gerando Conversores em XSLT

Este módulo do componente Assistente de Publicação é o responsável pela geração automatizada dos conversores descritos em XSLT. Para isto, ele executa um algoritmo que utiliza as informações semânticas e estruturais armazenadas no mapa de integração durante o procedimento descrito na seção anterior.

O algoritmo em questão possibilita a escolha de um documento de origem e vários alvos. A geração dos conversores sempre se dá aos pares, documento origem → documento alvo, uma vez que as regras de transformação definidas em arquivos XSLT deverão buscar alguma informação no documento de origem e estruturá-la de

outra maneira no documento alvo. Vale destacar que, como o processo de geração de conversores se dá sempre para cada par de aplicações, para cada conversor gerado haverá a necessidade mínima de mapeamento de dois esquemas; um para o documento origem e outro para o alvo.

Embora as marcações mapeadas a partir dos dois esquemas sejam utilizadas, as que servirão de base para definição do documento gerado a partir do conversor serão provenientes do esquema do documento alvo. Isto por que são elas que informarão a estrutura que o artefato alvo deverá possuir. As informações do esquema de origem serão utilizadas apenas para informar quais marcadores do documento de origem se transformarão em seu correspondente no documento alvo. Assim, para cada elemento do esquema do artefato alvo que tenha sido mapeada, será procurado seu correspondente nas informações do esquema de origem. Caso esta informação esteja disponível, é gerada uma regra para transformações entre os dois elementos e uma instrução para recuperar o valor do documento de origem e colocá-la no artefato alvo. Caso a informação não esteja disponível, será criada uma regra que definirá o marcador necessário no artefato alvo e seu conteúdo será vazio já que esta informação não está presente no artefato de origem. O algoritmo abaixo representa o processo de geração das regras de transformação descrito aqui:

1. Receba a legenda dos esquemas dos artefatos origem e alvo para os quais será gerado o conversor;
2. Crie um arquivo XSLT para armazenamento das regras;
3. Procure no mapa de integração os identificadores correspondentes a cada legenda;
4. Para cada conceito ou propriedade do mapa de integração faça:
    a. Recupere o nó que contém as informações do esquema do artefato alvo;
    b. Recupere o nó que contém as informações do esquema do artefato de origem;
    c. Recupere o nome do conceito ou propriedade em questão do esquema do artefato alvo;
    d. Recupere o nome do conceito ou propriedade em questão do esquema do artefato de origem;
    e. Crie o *template* para o conceito ou propriedade em questão com as informações obtidas dos dois esquemas;
    f. Verifique se há atributos no esquema do artefato alvo. Se houver, acrescente a regra de transformação necessária;
    g. Para a seqüência de elementos da marcação *map:sequence* do esquema do artefato alvo, crie um conjunto de chamada de *templates*;
        i. Para cada elemento contido na marcação *map:sequence* verifique a possibilidade de haver repetições daquele elemento no artefato alvo. Se houver, crie a instrução de repetição para o nó em questão;
5. Salve o arquivo XSLT finalizando a geração do conversor.

Esta funcionalidade complementa o módulo responsável por efetuar o mapeamento no sentido de lidar com as questões envolvendo discrepâncias estruturais e semânticas. A partir destes dois módulos é possível efetuar a integração de ferramentas distintas que queiram compartilhar artefatos de um mesmo domínio descritos em XML.

4.4.1.3.3 – Removendo Informações Mapeadas

Este módulo do componente Assistente de Publicação é o responsável por remover informações mapeadas no mapa de integração. É um ponto importante se levarmos, principalmente, em conta que esquemas podem evoluir implicando na necessidade de um novo mapeamento.

4.4.1.4 – Componente Transformação

Junto com o Assistente de Publicação, este componente completa a principal funcionalidade do Gerente de Integração, prover a integração de dados XML. O componente Transformação é o responsável por efetuar as transformações nos dados utilizando para isso as regras de transformação criadas no Assistente de Publicação.

De todos os componentes que fazem parte do Gerente de Integração, este é o mais independente. Isto por que as transformações nos documentos XML são efetuadas via XSLT e para isto, é necessária a utilização de um processador XSLT (vide seção 3.4). Neste contexto, duas restrições foram consideradas: suportar a versão 1.0 do XSLT e ser *open source*. O processador escolhido foi o *Xalan* versão 1.7 para plataforma Windows. Ele é resultado do projeto Apache XML[16] e está disponível na Internet para utilização com as linguagens C++ e Java. Com isto, o componente Transformação é apenas uma interface para o processador *Xalan*.

Para efetuar as transformações nos documentos, basta informar para o processador onde os documentos XML e XSLT podem ser encontrados e onde será gerado o artefato alvo. O processador se encarregará de efetuar as transformações necessárias (ver Figura 4.9). Estes passos definem as funcionalidades presentes no componente transformação:

- Permitir que o usuário selecione o documento XML a ser transformado;

[16] Pode ser acessado em http://xml.apache.org/.

- Permitir que o usuário defina o documento XSLT que contém as regras de transformação necessárias;
- Recuperar informações no Gerente de Metadado sobre onde o artefato alvo deverá ser mantido;
- Enviar estas informações para o processador XSLT (*Xalan*) para que sejam efetuadas as transformações e gerado o novo artefato.

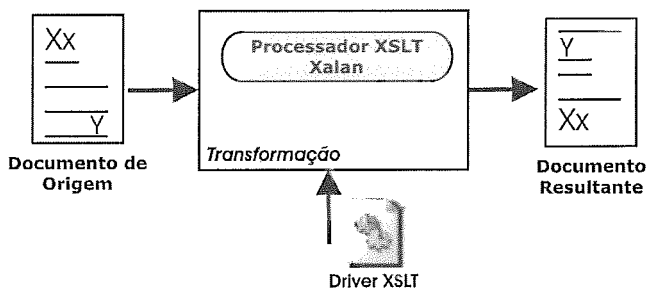


Figura 4.9 – Funcionamento do Componente Transformação.

Por fim, o componente Transformação embora utilize artefatos gerados pelo componente Assistente de Publicação, ele interage diretamente apenas com o Gerente de Metadado onde os documentos necessários para o processador XSLT são gerenciados.

4.4.1.5 – Componente Comunicação

Foi visto na seção 4.2 que tornar o acesso das ferramentas ao mecanismo de integração transparente é um ponto chave na construção e para o sucesso dele. Para lidar com esta questão, foi considerada a inclusão de um componente responsável por esta comunicação. A partir desta idéia inicial foram agregadas outras funcionalidades e hoje temos as seguintes funcionalidades a serem contempladas no seu desenvolvimento:

- tornar o acesso das ferramentas ao mecanismo de integração transparente;
- disponibilizar interfaces das funcionalidades providas pelo Gerente de Integração relacionadas à transformação de documentos XML para que esta possa ser solicitada pelo usuário;
- tornar possível o transporte de documentos XML entre a Ferramenta Cliente e o Gerente de Integração para que este possa efetuar as transformações necessárias;

- encaminhar o documento resultante da transformação para o cliente que a solicitou;
- interagir com o componente Comunicação da ferramenta Cliente;
- lidar com questões de integração em nível de sistema utilizando para isso um *middleware*.

Através destas funcionalidades, o componente Comunicação permitirá a separação entre o mecanismo de integração e o de transporte de dados.

4.4.2 – Ferramenta Cliente

Parte integrante da infra-estrutura proposta neste trabalho, a ferramenta Cliente será responsável por permitir a comunicação do usuário com o Gerente de Integração. Para prover estas funcionalidades, a Ferramenta Cliente possui um componente responsável pela comunicação com o Gerente de Integração (ver Figura 4.3).

4.4.2.1 – Componente Comunicação

Este módulo complementará as funcionalidades do componente Comunicação da ferramenta Gerente de Integração. Junto a ele, proverá uma camada para separar o mecanismo de integração do de transporte de dados tornando transparente a comunicação entre a ferramenta Cliente e o Gerente de Integração.

Muitas das funcionalidades que deverão ser contempladas neste componente são compartilhadas com seu correspondente no Gerente de Integração. Estas funcionalidades são:

- tornar o acesso das ferramentas ao mecanismo de integração transparente;
- disponibilizar para a ferramenta Cliente o acesso às funcionalidades disponibilizados pelo Gerente de Integração;
- tornar possível o transporte de documentos XML entre a Ferramenta Cliente e o Gerente de Integração;

4.5 – Processo de Integração

O processo apresentado nesta seção tem como objetivo apoiar a abordagem de integração proposta nesta tese. Ele apresenta as alternativas possíveis de serem seguidas durante a integração explicitando os artefatos gerados em cada atividade e quem é o responsável por executá-la. Os requisitos levantados para a definição das atividades do processo foram especificados durante a elaboração da abordagem de integração. Os requisitos são:

- Permitir a definição da semântica contida nos artefatos das ferramentas a serem integradas através de ontologias;
- Permitir o mapeamento do esquema dos artefatos na ontologia;
- Permitir a geração de conversores a partir das informações mapeadas na ontologia;
- Garantir que as informações necessárias ao mapeamento possam ser mantidas e recuperadas quando necessário.

A Figura 4.10 apresenta o processo a ser seguido para configuração do ambiente de integração. Ele é composto de seis atividades:

- **Analisar Ferramentas a serem Integradas:** nesta atividade, os engenheiros de domínio e de software devem fazer um levantamento sobre as discrepâncias semânticas e estruturais entre os artefatos das ferramentas a serem integradas. É uma atividade complexa visto que é aqui que deverão ser definidos os esquemas dos artefatos a serem integrados e os conceitos contemplados em sua semântica. É realizada parcialmente fora do contexto da ferramenta xMapper, e representa uma atividade preparatória para a integração;
  Produto: definições dos problemas estruturais e semânticos que deverão ser tratados. Definição da ontologia e esquemas a serem construídos;
  Responsável: Engenheiro de Domínio e Software;
- **Configurar Gerente de Metadado:** nesta atividade, o engenheiro de software define o local onde serão mantidas as informações necessárias e geradas pela infra-estrutura de integração;
  Produto: arquivo de configuração do gerente de metadado;
  Responsável: Engenheiro de Software;

- **Definir Ontologia:** nesta atividade, o engenheiro do domínio define os conceitos e propriedades referentes ao domínio do artefato das ferramentas a serem integradas;
  Produto: ontologia definida;
  Responsável: Engenheiro do Domínio;
- **Mapear Semântica e Estrutura:** nesta atividade, o engenheiro do domínio ou de software efetua o mapeamento de artefatos heterogêneos através da inserção de informações estruturais provenientes dos esquemas representando os documentos fonte e alvo na ontologia;
  Produto: mapa de integração;
  Responsável: Engenheiro do Domínio ou de Software;
- **Gerar Conversor:** nesta atividade, o engenheiro de software gera de forma automatizada o conversor a partir das informações (semânticas e estruturais) contidas no mapa de integração;
  Produto: conversor gerado;
  Responsável: Engenheiro de Software;
- **Transformar Documento:** nesta atividade o engenheiro de software seleciona o artefato a ser transformado e o conversor contendo as regras de transformação.
  Produto: artefato alvo;
  Responsável: Engenheiro de Software;

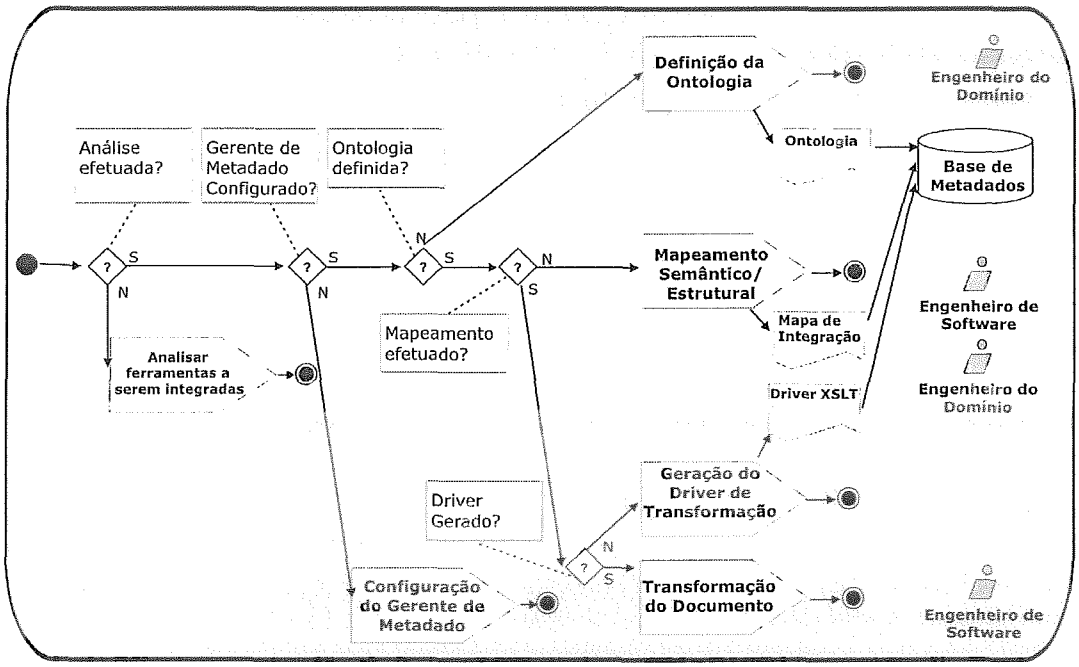


Figura 4.8 – Processo de configuração do ambiente de integração.

Este processo foi elaborado baseado na notação de modelagem de processos proposta por (VILLELA, 2004) e descrita no Anexo C.

4.6 – Considerações Finais

Este capítulo discutiu inicialmente problemas da integração de dados e com base neles, definiu os requisitos da abordagem de integração proposta nesta tese. A partir destes requisitos, apresentou a abordagem para a integração de artefatos de um mesmo domínio descritos em XML baseada na utilização de informações estruturais e semânticas contidas em esquemas e ontologias.

Dois dos problemas discutidos na seção 4.2 são básicos para qualquer infra-estrutura que se proponha a lidar com a integração de ferramentas. Estes tópicos são **abstração de dados** e **ferramentas externas**. Para a questão da abstração de dados, nossa proposta limita o escopo de integração a artefatos de um mesmo domínio descritos em XML. Já para lidar com ferramentas externas, basta que estas exportem seus dados no formato XML.

Frente às demais dificuldades explicitadas na seção 4.2, a abordagem apresentada traz os seguintes benefícios:

- **Distribuição.** A solução aqui apresentada contempla mecanismos que tornam transparente a comunicação entre as ferramentas e o mecanismo de integração. Isto é feito através dos componentes de comunicação da ferramenta cliente e do gerente de integração.
- **Evolução Semântica.** A linguagem utilizada para definição de ontologias - OWL - permite o acréscimo de conceitos através de relações de sinônimos, antônimos e herança dentre outros. Isto ajuda a tornar a abordagem proposta extensível e lida com parte do problema. A principal dificuldade presente neste tópico é conseqüência da própria evolução do domínio do artefato. Caso as modificações afetem as informações já mapeadas, haverá a necessidade de mapear novamente as informações estruturais no mapa de integração e gerar novos *s* de transformação. Como a abordagem aqui proposta permite ao usuário realizar isso forma de forma semi-automatizada e automatiza, respectivamente, o esforço de manutenção é bastante reduzido.
- **Integridade Semântica.** Este requisito não é totalmente contemplado na abordagem aqui apresentada. Ele é difícil de ser atingido pelo fato do algoritmo para mapeamento criado ser executado de forma semi-automatizada. Isto dá margem à quebra da integridade nos momentos em que uma decisão deva ser tomada pelo usuário. Embora não lide diretamente com este problema, a solução apresentada tenta minimizá-lo através de controles de integridade durante a definição da semântica dos artefatos e evitando, quando possível, interação com o usuário durante o mapeamento semântico/estrutural.
- **Contexto da Informação:** o escopo para solução de integração apresentada está relacionada diretamente com este problema. O fato dos artefatos considerados para integração serem de um mesmo domínio e terem sua semântica descrita através de ontologias minimiza ou elimina a possibilidade de ambigüidade ou interpretações incorretas das informações contidas nos documentos.
- **Múltiplos Padrões:** neste quesito, a abordagem proposta minimiza o esforço do integrador de ferramentas na criação e manutenção dos conversores necessários através de um procedimento semi-automatizado.
- **Semântica das Marcações:** esta abordagem não traz nenhum ganho quanto a essa característica.

Por fim, foi apresentado o processo a ser seguido para a integração de dados utilizando a abordagem proposta. No próximo capítulo será apresentado o protótipo da infra-estrutura de integração para a abordagem aqui descrita.

Capítulo 5

xMapper: uma Infra-estrutura para Integração de Ferramentas CASE com Artefatos em XML

Este capítulo apresenta o protótipo da infra-estrutura para integração de ferramentas cujos artefatos sejam de um mesmo domínio e descritos em XML através de um exemplo de uso. Discute sua implementação e como ele permite realizar a integração de forma semi-automatizada e baseada na tecnologia XML e suas correlatas.

5.1 – Introdução

Buscando apoiar a abordagem de integração de dados apresentada no capítulo 4, a ferramenta xMapper foi definida e implementada. O xMapper baseia-se fundamentalmente no processo de integração apresentado na seção 4.5 permitindo que o usuário realize de forma semi-automatizada cinco das seis atividades necessárias para a integração. A atividade não apoiada é Analisar Ferramentas a serem Integradas. É uma atividade de análise e por isso não está contemplada embora a formalização das ontologias, um dos produtos da atividade, possa ser efetuada utilizando xMapper.

Como apresentado na seção 4.4 o mecanismo de integração proposto nesta tese é formado de vários componentes. Destes, dois estão relacionados diretamente com o problema da integração (destaque em azul da Figura 5.1): Assistente de Publicação e Transformação. Os demais componentes (destaque em verde da Figura 5.1), Gerente de Metadado e OWLEditor, provêem apoio às atividades desempenhadas pelos componentes centrais da infra-estrutura. A partir desta constatação, estes foram os componentes do xMapper escolhidos para serem implementados (destaque em vermelho da Figura 5.1). Desta forma, optou-se por não implementar a ferramenta Cliente e os componentes responsáveis pela comunicação entre esta e o Gerente de Integração.

O diagrama apresentado na Figura 5.2 mostra, de forma simplificada, os componentes escolhidos, quais métodos foram implementados em cada um deles e de quais componentes externos eles dependem para execução de suas funcionalidades. Embora o componente Assistente de Publicação possua apenas três métodos principais, sua complexidade de desenvolvimento foi alta sendo que os métodos *mapearEsquema()* e *gerarXSLT()* representam os algoritmos apresentados nas seções 4.4.1.3.1 e 4.4.1.3.2, respectivamente.

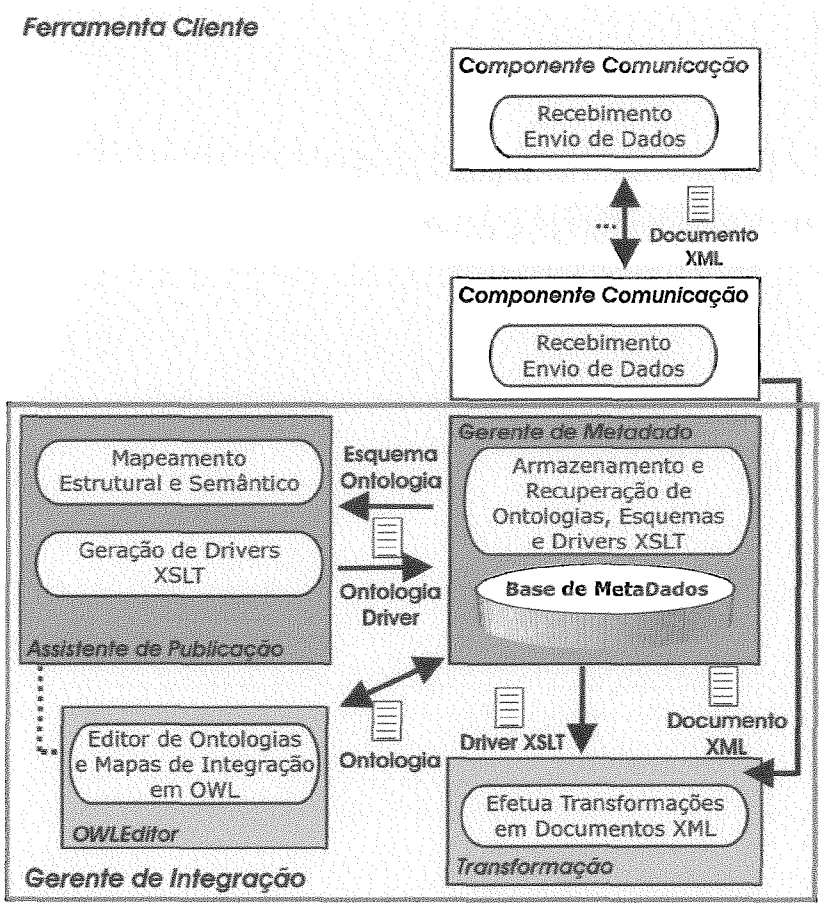


Figura 5.1 – Componentes Implementados do xMapper.

Já o componente OWLEditor permite a manipulação de artefatos descritos em OWL. Neste caso, estes artefatos podem ser ontologias ou mapas de integração. Tanto o OWLEditor quanto o Assistente de Publicação por trabalharem diretamente sobre arquivos com sintaxe XML, necessitam de um *parser* que disponibilize uma interface para manipulação dos elementos do documento. Para isto, eles utilizam o MSXML. O componente Tradutor é apenas uma interface para o Xalan, processador

XSLT responsável por efetuar as transformações nos dados. Neste caso, foi necessária a utilização do Xerces (*parser* XML) pelo fato do Xalan fazer internamente, uso dele. Por fim, tem-se o componente Gerente de Metadado que possui a responsabilidade de organizar o armazenamento dos artefatos.

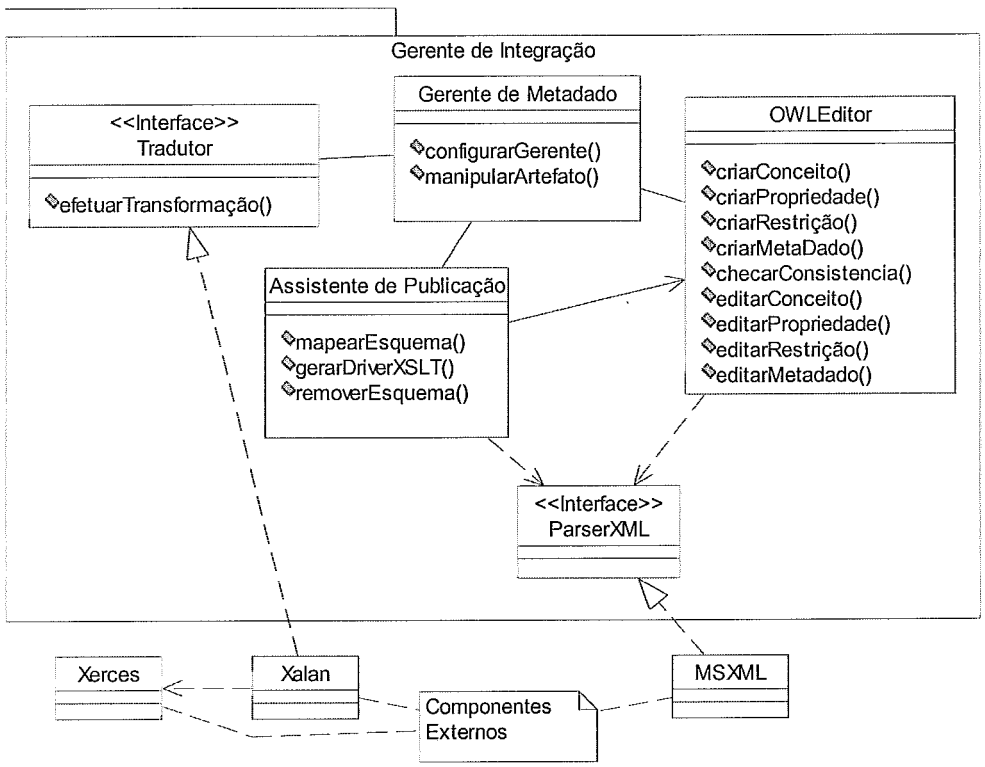


Figura 5.2 – Componentes do xMapper.

Além desta introdução que apresentou quais componentes da abordagem proposta de integração foram implementadas, este capítulo também descreve detalhes técnicos da implementação do xMapper, o protótipo implementado e um conjunto de diretrizes para sua utilização.

5.2 – Detalhes da Implementação

A ferramenta foi desenvolvida na linguagem C++ utilizando o ambiente de programação Borland C++ Builder 6.0 na plataforma Windows. A escolha deste ambiente de programação se deu por ele ser bastante intuitivo para a criação de interfaces com o usuário e, principalmente, por prover diversos componentes para

manipulação de arquivos XML. Dentre esses componentes, ganha destaque a escolha de qual interface de programação utilizar para a manipulação de documentos XML. Neste caso, foi escolhido o DOM e sua implementação MSXML disponível no ambiente de programação C++ Builder. Os fatores que motivaram esta decisão foram as necessidades de acesso aleatório, repetitivo e constante atualização dos elementos dos documentos XML e a facilidade de manipulação dos elementos dos documentos XML (estes tópicos foram discutidos em detalhes na seção 2.2.2).

A utilização da linguagem XML e suas correlatas implicaram em preocupações acerca de problemas de incompatibilidade entre suas diferentes versões. Assim, foram escolhidos: XML 1.0, XSLT 1.0, XML *Schema* 1.0 e OWL 1.0.

5.3 – Apoio às Atividades do Processo de Integração

O apoio dado por xMapper às atividades do processo de integração apresentado na seção 4.5 está descrito nas seguintes seções onde um exemplo de integração entre uma ferramenta de apoio à aplicação de PBR, PBR *Tool* (SILVA *et al.*, 2004) e uma de apoio ao processo de inspeção, ISPIS (KALINOWSKI *et al.*, 2004), é apresentado. Neste caso, o artefato a ser compartilhado entre as aplicações é o relatório de discrepância gerado durante a inspeção de software e o sentido do intercâmbio da informação é da PBR *Tool* para ISPIS. As Figuras 5.3 e 5.4 apresentam os artefatos utilizados por cada uma das ferramentas.

```
<formDiscrepancy type="pbr">
    <perspective>USER</perspective>
    <project>ESE</project>
    <inspector>103137506</inspector>
    <start>7/9/2003 09:40:56</start>
    <artifact>C:\Program Files\PBRTool\SRS_Documents\ABC_Video_System.rtf</artifact>
    <discrepancyList>
        <discrepancy status="updated">
            <classification>Omissão</classification>
            <description>Como calcular a taxa de aluguel? Que outras taxas existem?</description>
            <identTime>7/9/2003 10:44:44</identTime>
            <step>Participants - View Document</step>
            <line>300</line>
            <requirement>Functional Requirement 8</requirement>
        </discrepancy>
    </discrepancyList>
    <PBR_help>0</PBR_help>
    <Tool_help>0</Tool_help>
    <end>7/9/2003 12:13:47</end>
    <totalTime>02:32:50</totalTime>
</formDiscrepancy>
```

Figura 5.3 – Fragmento do artefato gerado pela PBR *Tool*.

```
<formDiscrepancy type="pbr">
    <discrepancyList type="pbr">
        <inspectionStart>7/9/2003 09:40:56</inspectionStart>
        <discrepancy>
            <location>Functional Requirement 8 – Linha: 300</location>
            <classification>Omissão</classification>
            <severity/>
            <description>Como calcular a taxa de aluguel? Que outras taxas existem?</description>
        </discrepancy>
        <inspectionEnd>7/9/2003 12:13:47</inspectionEnd>
    <discrepancyList>
</formDiscrepancy>
```

Figura 5.4 – Fragmento do artefato utilizado por ISPIS.

5.3.1 – Analisando as Ferramentas a serem Integradas

Nesta atividade é feito um estudo das disparidades semânticas e estruturais presentes nos artefatos das ferramentas que serão integradas. Desta forma, o principal insumo desta atividade são os dois artefatos cuja semântica e estrutura deverão ser captados para possibilitar o uso de xMapper. A análise da semântica se concentrará nos conceitos e propriedades que definem o significado do artefato a ser integrado. Vale destacar que conceitos particulares de cada aplicação devem ser desconsiderados. Para o exemplo de integração entre as ferramentas PBR *Tool* e ISPIS, por exemplo, deve-se concentrar nos conceitos de um relatório de discrepância.

Já a análise da estrutura considera os detalhes de organização de cada documento e tem como resultado a criação de um esquema definindo a estruturação para cada a ser integrado. Assim, para os artefatos apresentados nas figuras 5.3 e 5.4, há a necessidade de omitir algumas marcações (*perspective, project, inspector, artifact*, dentre outras), modificar o nome de algumas (*start*, *end*, dentre outras), acrescentar a marcação *severity* além de efetuar junções de conceitos (*requirement* e *line*).

Esta atividade exige bastante esforço e atenção por parte do integrador. Ela é importante para que as demais atividades do processo sejam executadas corretamente. Serão apresentados agora as atividades que deverão ser efetuadas durante a utilização do xMapper para a integração das ferramentas ISPIS e *PBR Tool*.

5.3.2 – Apoio à Gerencia de Metadados

Inicialmente é preciso configurar o local onde a infra-estrutura manterá as informações necessárias. Isto é feito no Gerente de Metadado utilizando o módulo de configuração (ver Figura 5.5). Este componente também possui funcionalidades (abertura e exclusão de artefatos) para manutenção das informações utilizadas pela infra-estrutura.

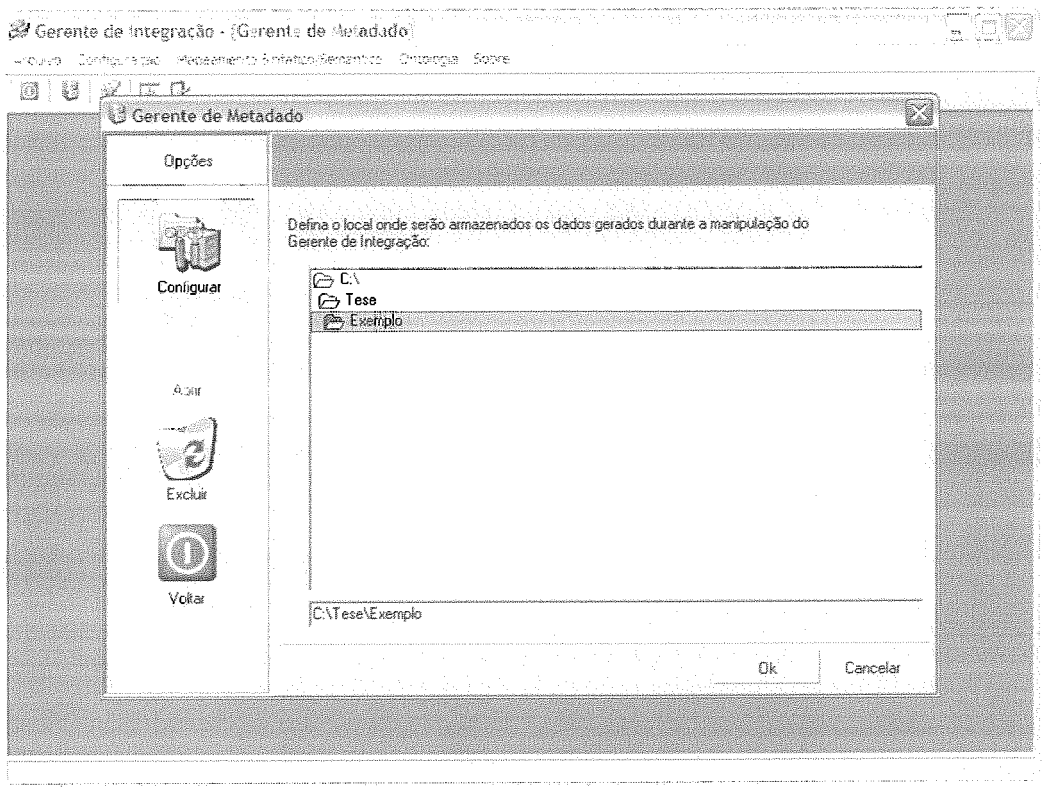


Figura 5.5 – Selecionando o local de armazenamento no Gerente de Metadados.

5.3.3 – Apoio à Definição de Ontologia e Manutenção de Mapas de Integração

Antes de dar início ao mapeamento entre os artefatos, é preciso definir sua semântica. Isto é feito através do componente OWLEditor. Ele possui funcionalidades para definição de metadados que descrevem a ontologia ou mapa de integração, conceitos, propriedades e restrições. A Figura 5.6 apresenta a janela onde são definidos os conceitos, relações entre conceitos e as restrições aplicadas às

propriedades associadas a cada conceito. Do lado esquerdo são listados os conceitos criados e do lado direito, todas as restrições e axiomas que podem ser definidos. Ao clicar em cada conceito, suas respectivas restrições e axiomas são mostrados e poderão ser manipulados. A metáfora de visualização e especificação dos metadados e propriedades é semelhante à utilizada para definição dos conceitos.

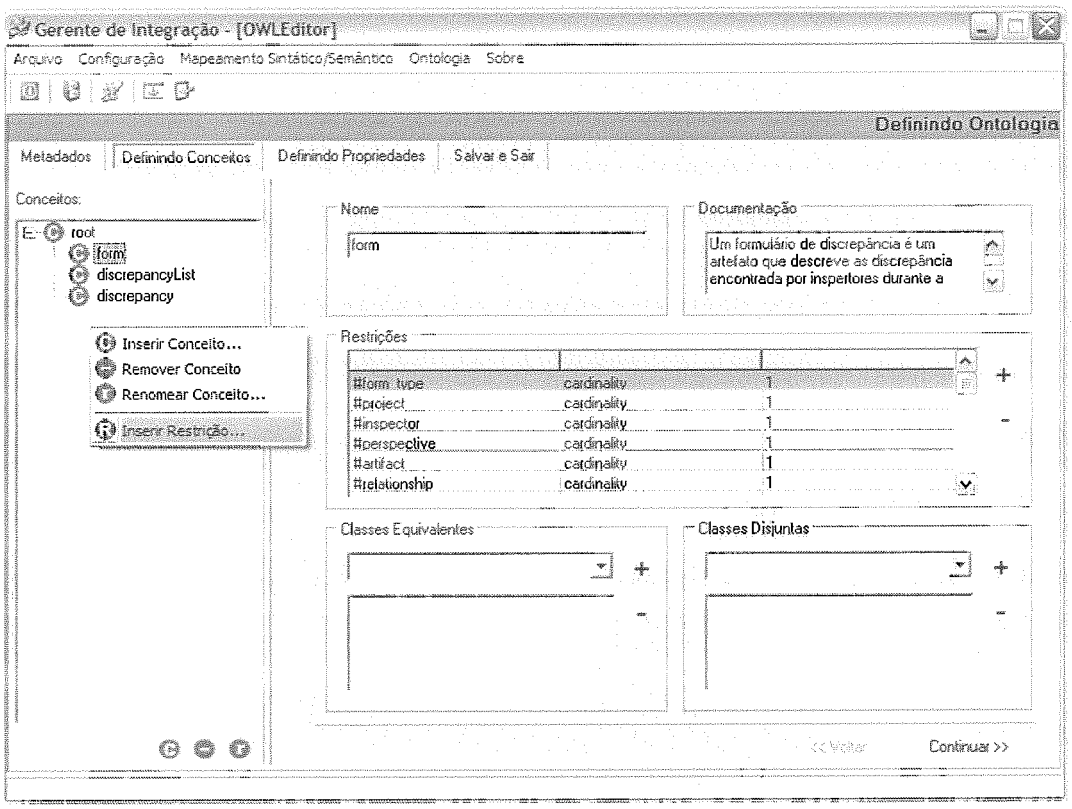


Figura 5.6 – Definição dos Conceitos.

Ao se definir uma ontologia, deve-se preocupar com os principais conceitos que explicitam a semântica, neste caso, de um artefato. Desta forma, não se deve considerar informações que somente são relevantes para ferramentas específicas. Estas informações mais pontuais serão acrescidas à ontologia durante o processo de mapeamento e neste momento, a ontologia passará a ser considerada um mapa de integração.

Neste exemplo, esta se definindo uma ontologia descrevendo a semântica de um relatório de discrepância. Para esta ontologia foram definidos três conceitos:

- **form:** representando o formulário de discrepância como um todo;
- **discrepancyList:** representando uma lista de discrepância;

- **discrepancy:** definindo uma discrepância;

E oito propriedades:

- **project:** definindo o conceito de projeto;
- **inspector:** definindo o conceito de inspetor;
- **artifact:** definindo o conceito de artefato;
- **totalTime:** definindo o conceito de tempo de duração da inspeção;
- **requirement:** definindo o conceito de requisito;
- **classification:** definindo a classificação que uma discrepância pode ter;
- **description:** definindo a descrição de uma discrepância encontrada.

A Figura 5.7 apresenta um trecho da ontologia, gerada nesta etapa, representando a semântica do conceito *form*. A partir desta ontologia, serão gerados os mapas de integração necessários para os mapeamentos entre os artefatos como discutido na seção 4.4.

```
<owl:Class rdf:ID="form">
    <rdfs:comment>Um formulário de discrepância é um artefato... de artefatos de software.</rdfs:comment>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#form_type"/>
        <owl:cardinality rdf:datatype="#NonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#project"/>
        <owl:cardinality rdf:datatype="#NonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#inspector"/>
        <owl:cardinality rdf:datatype="#NonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#perspective"/>
        <owl:cardinality rdf:datatype="#NonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#artifact"/>
        <owl:cardinality rdf:datatype="#NonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#totalTime"/>
        <owl:cardinality rdf:datatype="#NonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
</owl:Class>
```

Figura 5.7 – Conceito *form* definido em OWL.

Foram definidas outras restrições não descritas aqui. A ontologia definida nesta etapa pode ser vista no Anexo D.2.

5.3.4 – Apoio ao Mapeamento Semântico/Estrutural

Tendo definido a semântica do artefato a ser integrado, passa-se então a efetuar o mapeamento de informações estruturais gerando dessa forma, o mapa de integração. Para esta atividade, inicialmente são escolhidos o esquema a ser mapeado, o mapa de integração onde o esquema será mapeado e uma legenda para facilitar a manipulação das informações mapeadas. Como se pretende efetuar o mapeamento semântico e estrutural entre os artefatos de relato de discrepância utilizados por duas ferramentas, este passo deverá ser efetuado duas vezes; uma para cada esquema especificando a estruturação utilizada pelo artefato de cada ferramenta. Neste exemplo serão efetuados os mapeamentos dos esquemas dos artefatos das ferramentas PBR *Tool* e ISPIS (ambos os esquemas estão apresentados no Anexo D.3).

Dado início ao mapeamento, o algoritmo apresentado na seção 4.4.1.3.1 é executado e os elementos que não foram mapeados são apresentados para que o usuário auxilie na finalização deste procedimento. Esta etapa é executada em três passos: (1) mapeamento dos elementos complexos representando os conceitos no mapa de integração; (2) mapeamento dos elementos de tipo nativo representando as propriedades no mapa de integração e; (3) mapeamento de um atributo para um determinado conceito. As Figuras 5.8, 5.9 e 5.10 apresentam o apoio que a ferramenta dá a cada um destes passos, respectivamente.

Pode-se perceber através da Figura 5.8 que o algoritmo não fez a associação entre os conceitos *form* e *formDiscrepancy*. Como o conceito *form* já existia no mapa de integração, basta ao usuário efetuar a associação.

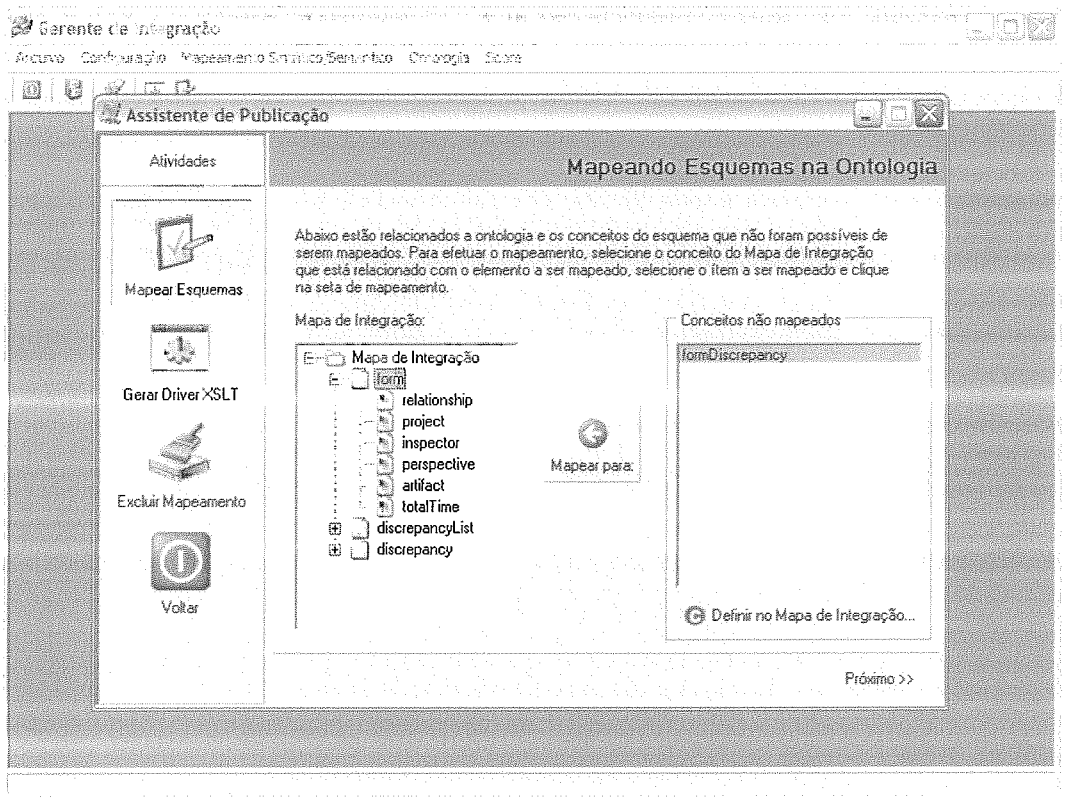


Figura 5.8 – Finalizando o mapeamento dos elementos complexos.

Caso não haja um conceito ou uma propriedade necessária na ontologia para efetuar o mapeamento, o Assistente de Publicação permite que eles sejam definidos para que as informações sejam mapeadas. Isto ocorreu, por exemplo, no mapeamento do esquema proveniente da ferramenta PBR *Tool* onde existem diversas propriedades que não possuiam correspondente no mapa de integração. Isto pode ser visualizado na Figura 5.9 onde são apresentadas um conjunto de propriedades que o algoritmo não conseguiu mapear automaticamente. Pode-se perceber também que as propriedades em questão (*line*, *PBRHelp* e *ToolHelp* dentre outras) são informações específicas da PBR *Tool* e por isso não estavam contempladas na semântica definida para um relatório de discrepância.

Para solucionar este impasse, o usuário deve clicar no botão *Definir no Mapa de Integração* e o componente Assistente de Publicação utilizará alguns serviços disponibilizados pelo componente OWLEditor para efetuar as alterações necessárias no mapa de integração tornando possível o mapeamento do conceito ou da propriedade.

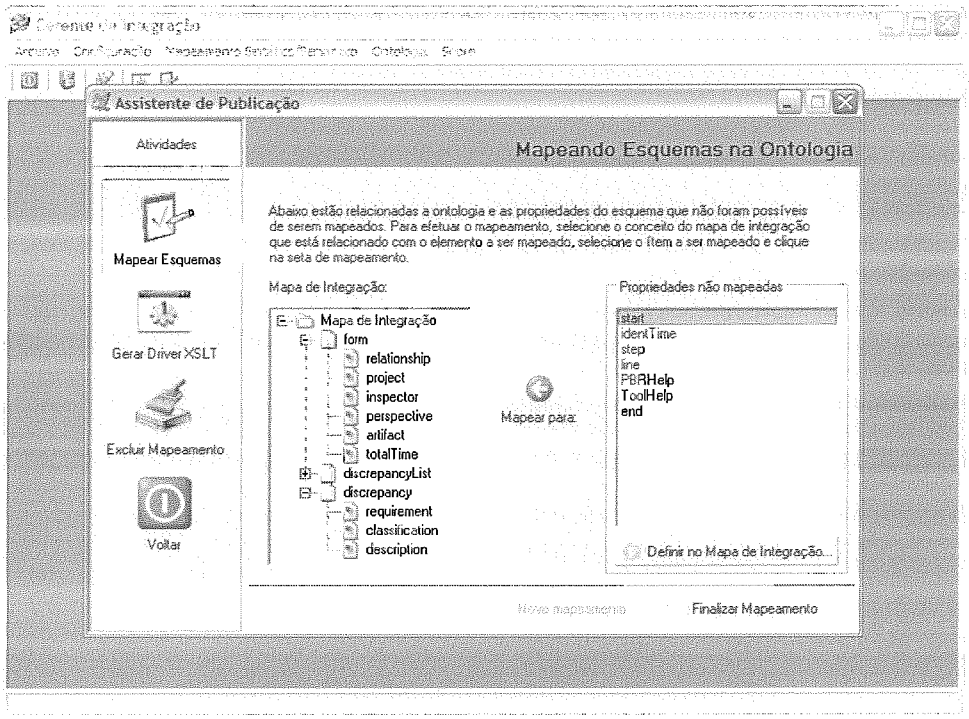


Figura 5.9 – Finalizando o mapeamento dos elementos de tipo nativo.

Para finalizar, a Figura 5.10 apresenta a janela do assistente onde é possível efetuar mapeamentos entre atributos e conceitos.

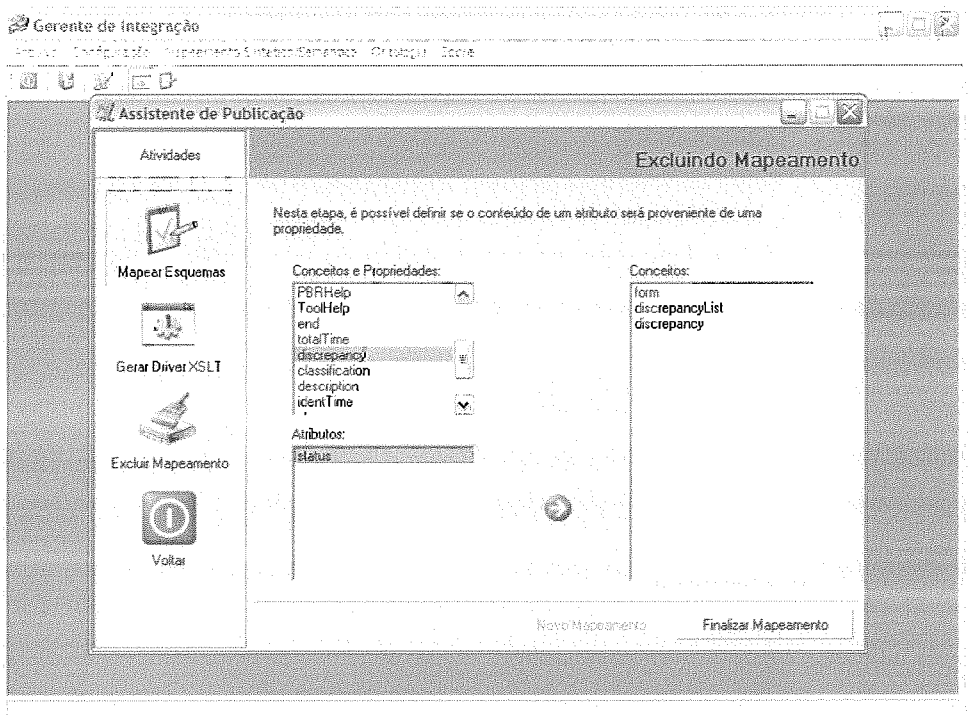


Figura 5.10 – Finalizando o mapeamento dos atributos.

Como neste exemplo não há necessidade de efetuar este tipo de ação, o mapeamento foi finalizado. A Figura 5.11 apresenta o trecho do mapa de integração com informações estruturais (em destaque) da PBR *Tool* mapeadas no conceito *form*.

```
<owl:Class rdf:ID="form">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:attribute>type</map:attribute>
            <map:name>formDiscrepancy</map:name>
            <map:sequence>
                <map:next moreOne="0">perspective</map:next>
                <map:next moreOne="0">project</map:next>
                <map:next moreOne="0">inspector</map:next>
                <map:next moreOne="0">start</map:next>
                <map:next moreOne="0">artifact</map:next>
                <map:next moreOne="0">discrepancyList</map:next>
                <map:next moreOne="0">PBRHelp</map:next>
                <map:next moreOne="0">ToolHelp</map:next>
                <map:next moreOne="0">end</map:next>
                <map:next moreOne="0">totalTime</map:next>
            </map:sequence>
        </map:schema>
    </map:mapFile>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#form_type"/>
        <owl:cardinality rdf:datatype="#NonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
    ...
</owl:Class>
```

Figura 5.11 – Trecho do Mapa de Integração.

Ao final destes passos, teremos as informações estruturais e semânticas necessárias para a geração dos conversores centralizadas no mapa de integração. O mapa de integração atualizado com o mapeamento dos dois esquemas pode ser visto no Anexo D.4.

5.3.5 – Apoio à Geração do Conversor

Efetuados os mapeamentos necessários, pode-se iniciar a geração dos conversores. O módulo responsável por esta geração foi implementado no componente Assistente de Publicação. Para efetuar a geração do conversor, deve ser inicialmente selecionado o mapa de integração onde estão informações de mapeamento. Feito isto, são apresentadas as possibilidades de geração de conversores. Basta selecionar os esquemas, dentre os mapeados, que representam os artefatos de origem e alvo (ver Figura 5.12).

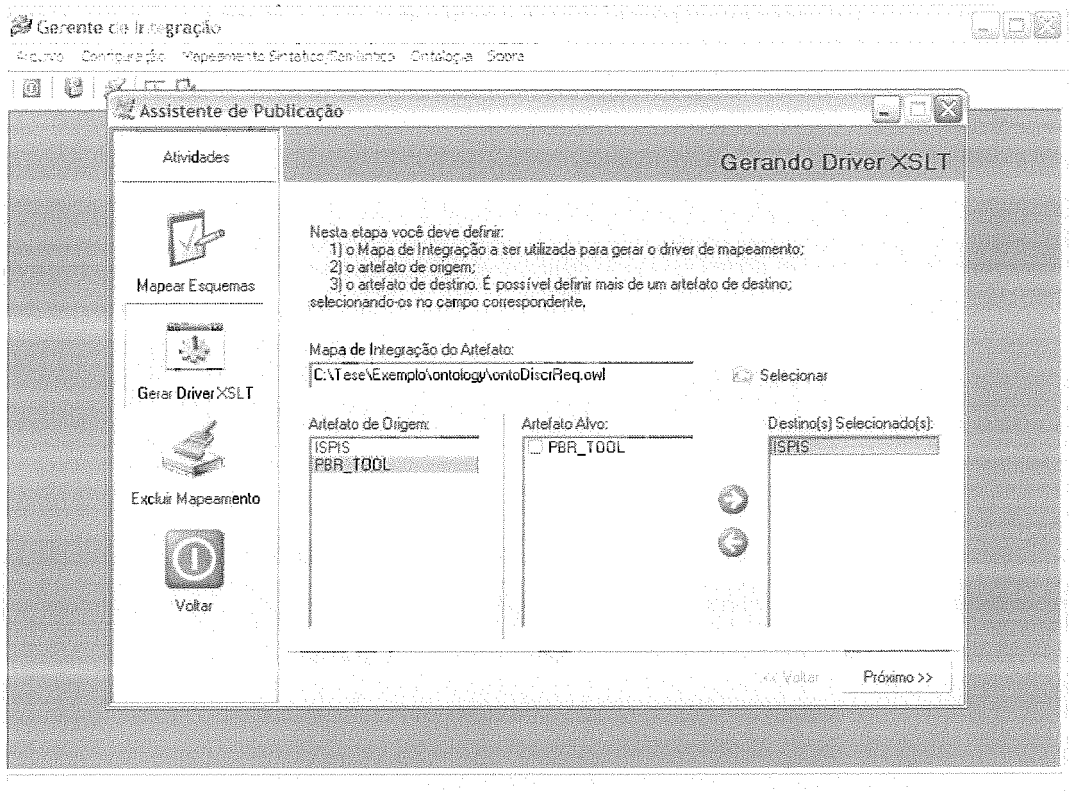


Figura 5.12 – Selecionando o conversor a ser criado.

Neste exemplo, será definido como artefato de origem o proveniente da PBR *Tool* e como artefato alvo, o documento a ser importado por ISPIS. Com estas informações configuradas, é possível então dar início ao processamento do mapa de integração para geração do conversor através do algoritmo apresentado na seção 4.4.1.3.2 (ver Figura 5.13).

Feito isto, é gerado o conversor descrito em XSLT. O conversor gerado para a transformação da ferramenta de PBR para ISPIS pode ser visto no Anexo D.5.

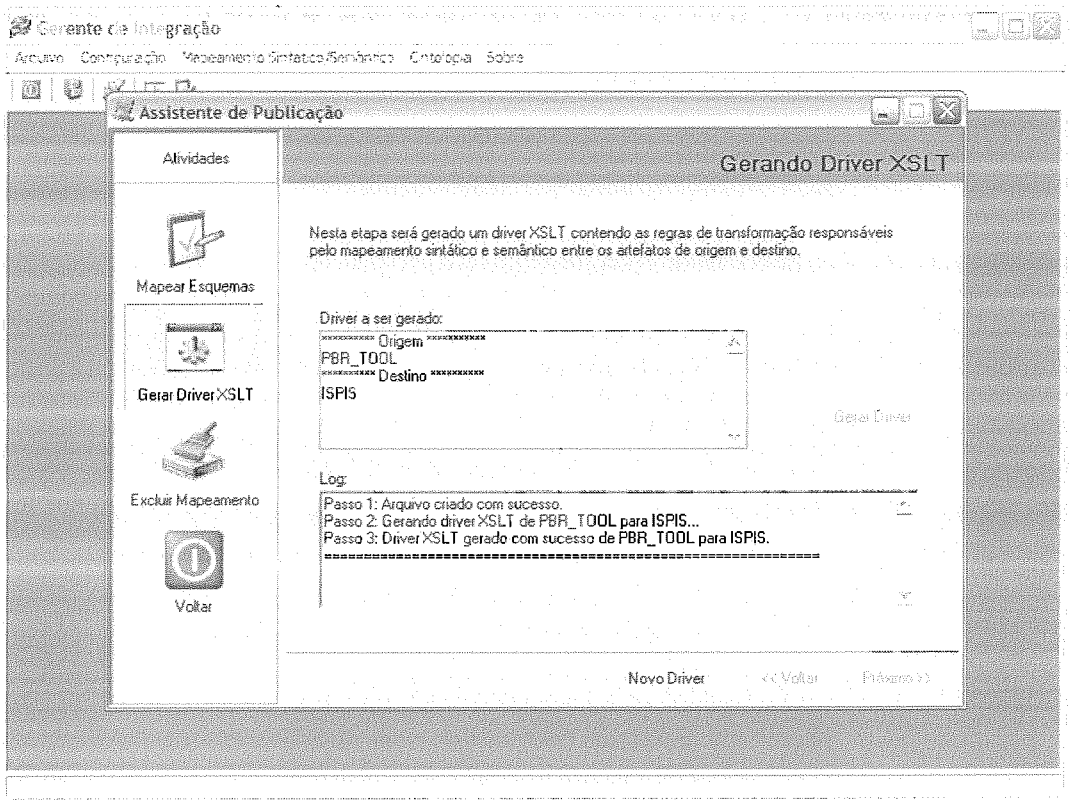


Figura 5.13 – Criando o conversor.

5.3.6 – Apoio à Transformação de Artefatos Definidos em XML

Por fim, com o conversor gerado, é possível integrar as duas ferramentas. Para dar apoio à atividade de transformação necessária nos artefatos para a geração do artefato alvo é utilizado o componente Transformação (ver Figura 5.14). Nele são definidos: o artefato descrito em XML que sofrerá as transformações e o conversor. Com estas informações especificadas, o novo artefato poderá ser gerado. O documento de origem e o gerado podem ser vistos no Anexo D.1 e D.6, respectivamente.

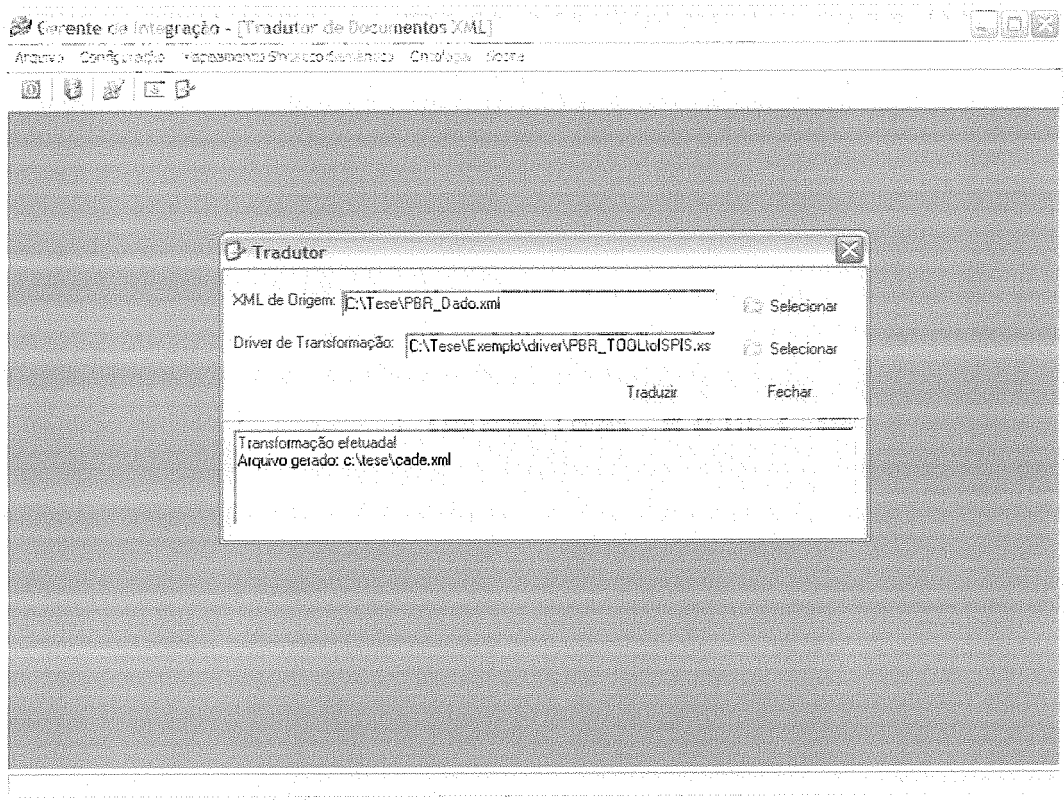


Figura 5.14 – O Componente Transformação.

5.3.7 – Apoio à Remoção de Informações Mapeadas

A Figura 4.16 apresenta a janela do Assistente de Publicação que provê esta funcionalidade. Nela, primeiro é escolhida o mapa de integração onde as informações tenham sido mapeadas. Então são listadas as legendas de todos os esquemas mapeados. Basta então selecionar e clicar no botão Remover Esquema. Todas as informações inseridas no mapa de integração cujo identificador refira-se ao esquema escolhido serão eliminadas.

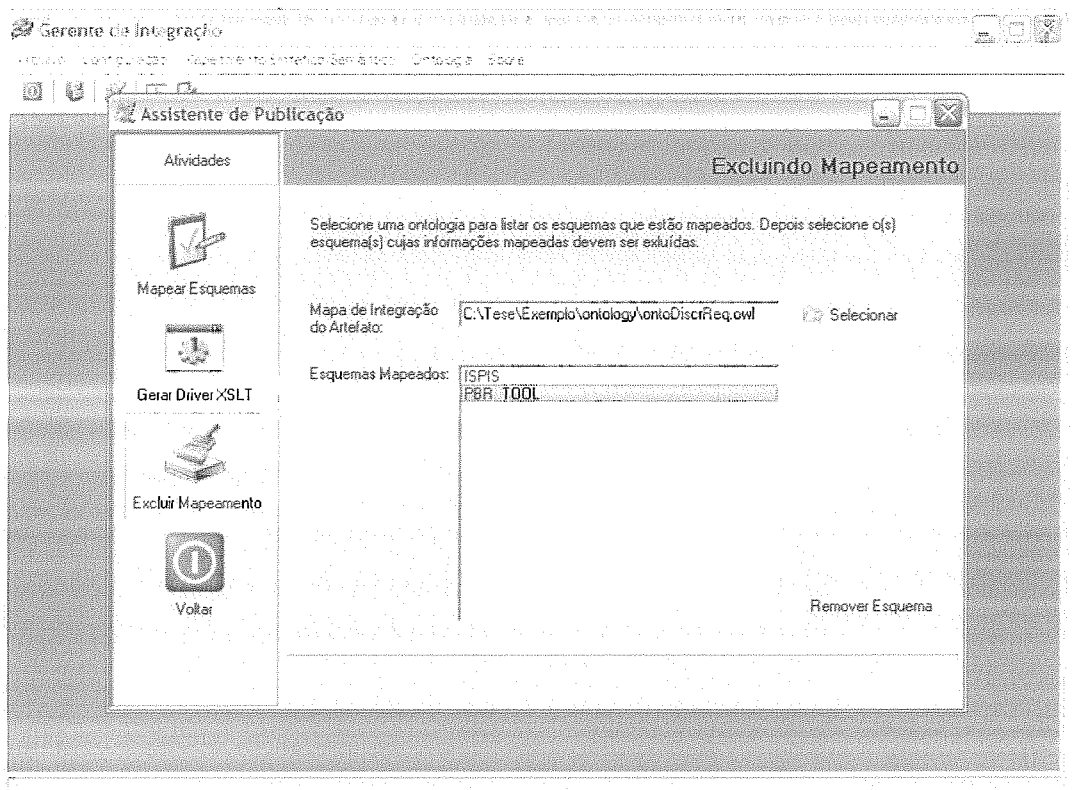


Figura 4.16 – Removendo esquemas mapeados na ontologia.

5.4 – Considerações Finais

Este capítulo apresentou a ferramenta implementada para apoiar a abordagem de integração apresentada no capítulo 4 e como ela está inserida no processo de integração descrito na seção 4.5. A importância deste processo está principalmente no fato de prover ao usuário uma noção de como o xMapper funciona e quais os passos devem ser seguidos para sua correta utilização.

Foi visto também que a ferramenta não contempla todos os componentes definidos na abordagem de integração pelo fato de parte deles não estarem relacionados diretamente com o problema da integração. Embora não tenha sido implementado, a incorporação destes componentes à ferramenta xMapper não é difícil. Isto por que eles estão fracamente acoplados ao restante da infra-estrutura. Neste caso, há a necessidade de comunicação apenas entre o componente Comunicação do Gerente de Integração e os componentes Gerente de Metadado e Transformação visto que o objetivo da ferramenta cliente junto aos componentes

responsáveis pela comunicação é tornar possível o acesso, por um usuário remoto, aos conversores e mecanismo de transformação.

Por fim, vale ressaltar que a implementação do mecanismo de mapeamento tenta automatizar ao máximo as tarefas objetivando fazer com que a interferência do usuário no processo de mapeamento seja minimizado.

No próximo capítulo será apresentado o estudo de caso realizado para avaliar a solução de integração descrita aqui e no capítulo 4.

Capítulo 6

Estudo de Caso

Este capítulo apresenta uma breve introdução à Engenharia de Software Experimental e o Estudo de Caso efetuado para avaliar a abordagem de integração de ferramentas apresentada no capítulo 4 – Requisitos para uma Infra-Estrutura de Integração de Ferramentas CASE com XML.

6.1 – Introdução

Experimentação é o centro do processo científico. Ela oferece o modo sistemático, disciplinado, computável e controlado para avaliação de novos métodos, técnicas, linguagens e ferramentas. Estes não deveriam ser apenas sugeridos, publicados ou apresentados para venda sem experimentação e avaliação (AMARAL, 2003). Segundo WOHLIN *et al.* (2000) os métodos experimentais são classificados em:

- **Pesquisa de campo:** é, geralmente, uma investigação executada em retrospecto quando, por exemplo, uma ferramenta ou técnica tem sido utilizada em uma empresa e pretende-se avaliá-la sob algum aspecto (AMARAL, 2003). Algumas formas para coleta de dados como entrevistas e questionários são bastante conhecidas e utilizadas.
- **Estudos de caso:** são usados para monitorar projetos, atividades ou exercícios objetivando rastrear um atributo específico ou estabelecer relacionamentos entre diferentes atributos sem um controle muito formal sobre as atividades relacionadas ao método experimental.
- **Experimentos:** é uma atividade com o propósito de descobrir algo desconhecido ou de testar uma hipótese envolvendo uma investigação de coleta de dados e de execução de uma análise para determinar o significado dos dados (BASILI *et al.*, 1999). São apropriados para confirmar teorias, o conhecimento convencional, explorar os relacionamentos, avaliar

a predição dos modelos ou validar as medidas. As principais características dos experimentos estão no controle total sobre o processo e variáveis e na possibilidade de ser repetido.

A escolha de qual método de pesquisa experimental utilizar depende dos: pré-requisitos da investigação, propósito, disponibilidade de recursos e de como se pretende analisar os dados coletados (WOHLIN *et al.*, 2000).

No campo da Engenharia de Software, a utilização de métodos experimentais pode trazer alguns benefícios (TICHY, 1998) (TRAVASSOS *et al.*, 2001):

- Ajudar a construir uma base confiável de conhecimento e desta forma reduzir a incerteza sobre quais teorias, métodos e ferramentas são adequados;
- Poder levar a novas compreensões sobre áreas atualmente pesquisadas;
- Conduzir a áreas desconhecidas, onde a engenharia de software progride lentamente;
- Acelerar o progresso através da rápida eliminação de abordagens não fundamentadas, suposições errôneas e modismos, ajudando a orientar a engenharia e a teoria para direções promissoras;
- Ajudar no processo de formação da Engenharia de Software como ciência, através do crescimento do número de trabalhos científicos com uma avaliação experimental significativa.

Estes pontos estimularam a definição e planejamento de um estudo de caso para avaliar a proposta de integração apresentada no capítulo 4.

6.2 – Planejamento do Estudo

6.2.1 – Objetivo Global

Caracterizar a viabilidade de utilização do mecanismo para integração de ferramentas proposto.

6.2.2 – Objetivo da Medição

Tendo como base os problemas e os requisitos identificados para integração de dados, caracterizar:

- Quais requisitos são contemplados no mecanismo de integração proposto;
- Se sua utilização diminui o esforço de integração de ferramentas;
- Se o mecanismo causa alguma perda de informação (semântica, estrutural ou de conteúdo) considerando a integração de duas ferramentas.

6.2.3 – Objetivo do Estudo

Analisar a abordagem para integração de artefatos de um mesmo domínio descritos em XML
Com o propósito de caracterizar
Com respeito à aderência da abordagem aos requisitos para integração de dados identificados e as facilidades para integração de ferramentas obtidas
Do ponto de vista do desenvolvedor de ferramenta
No contexto da utilização da abordagem integrando uma ferramenta de apoio à aplicação de PBR e: (1) uma de apoio ao Processo de Inspeção de Software e, (2) uma de apoio à construção de casos de uso.

6.2.4 – Questões

Q1: A abordagem de integração de artefatos de um mesmo domínio descritos em XML contempla os requisitos para integração de dados identificados?
Métrica: A lista dos critérios identificados para integração de dados e a lista dos critérios para integração de dados solucionados com a utilização da abordagem proposta.
Q2: A utilização da abordagem de integração de artefatos de um mesmo domínio descritos em XML reduz esforço de integração de ferramentas?
Métrica: Esforço (tempo) necessário para integração de duas ferramentas utilizando a abordagem de integração proposta.
Q3: A utilização da abordagem para integração de artefatos de um mesmo domínio descritos em XML causa alguma perda de informação durante a integração?

Métrica: Perda entre dados considerando o intercâmbio de dados entre duas ferramentas. Tendo identificado a perda, será necessário classificá-la quanto ao seu tipo: semântica, estrutural ou de conteúdo.

6.2.5 – Definição das Hipóteses

Hipótese nula (H0): a abordagem para integração de dados proposta não altera a integração de ferramentas que exportem e importem dados definidos em XML em vista dos requisitos para integração identificados na literatura.

$R_{Literatura}$: requisitos para integração de dados identificados;

$R_{XMapper}$: requisitos para integração de dados solucionados com a abordagem proposta;

H0: $R_{Literatura} \cap R_{XMapper} = \varnothing$

Hipótese alternativa (H1): a abordagem de integração proposta lida com os requisitos para integração de dados identificados na literatura.

$R_{Literatura}$: requisitos para integração de dados identificados;

R_{XML}: requisitos para integração de dados solucionados com a abordagem proposta;

H1: $R_{Literatura} \cap R_{XMapper} = \varnothing$

Hipótese alternativa (H2): a utilização da abordagem de integração proposta reduz esforço de integração de ferramentas.

E_{Manual}: Esforço medido em tempo para integração manual de duas ferramentas que exportem dados em XML;

E_{XML}: Esforço medido em tempo para integração de duas ferramentas utilizando a abordagem de integração proposta;

H2: $E_{XML} < E_{Manual}$

Hipótese alternativa (H3): A perda de informação (de acordo com os critérios abaixo) causada pela utilização da abordagem de integração de artefatos proposta não compromete a utilização dos dados, considerando a integração de duas ferramentas. Esta perda de informação é classificada de acordo com os seguintes critérios:

- Estrutural:
    - Grave: houve perda estrutural e ela compromete a utilização do dado;

- Baixa importância: houve perda estrutural mas não compromete a utilização do dado;

- Semântica:
  - Grave: houve perda semântica e ela compromete a utilização e entendimento do dado;
  - Baixa importância: houve perda semântica mas ela não compromete e utilização do dado;
- Perda de Conteúdo:
  - Grave: houve perda de conteúdo e ela compromete a utilização do dado;
  - Baixa importância: houve perda de conteúdo mas ela não compromete a utilização do dado;

$D_{Importado}$: dado a ser importado pela ferramenta de destino;
$D_{Exportado}$: dado exportado pela ferramenta de origem;
$P_{semântica}$: perda semântica do $D_{Importado}$ em relação ao $D_{Exportado}$;
$P_{estrutural}$: perda estrutural;
$P_{conteudo}$: perda de conteúdo;

H3: $(((P_{semântica} = \text{Sem Perda}) \vee (P_{semântica} = \text{Baixa Importância})) \wedge ((P_{estrutural} = \text{Sem Perda}) \vee (P_{estruturalG} = \text{Baixa Importância})) \wedge ((P_{conteudo} = \text{Sem Perda}) \vee (P_{conteudo} = \text{Baixa Importância})))$ = Não compromete a utilização do dado

6.2.6 – Descrição da Instrumentação

Para este estudo de caso, serão necessários os seguintes instrumentos:

- Mecanismo para Integração de Ferramentas proposto.
- Ferramenta de apoio ao processo de inspeção – ISPIS (KALINOWSKI *et al.*, 2004).
- Artefato gerado por ISPIS;
- Ferramenta de apoio à aplicação de PBR (SILVA *et al.*, 2004);
- Artefato gerado pela ferramenta de apoio à aplicação de PBR;
- Ferramenta de apoio à construção de casos de uso, *Use Case Tool* (CHAPETTA, 2004);

- Artefato gerado pela ferramenta de apoio à construção de casos de uso.

6.2.7 – Seleção do contexto

Este estudo está inserido no seguinte contexto:

- O processo: on-line. Isto por que medidas serão extraídas durante a atuação do desenvolvedor ao integrar as duas ferramentas;
- Os participantes: desenvolvedores de ferramentas da equipe de Engenharia de Software Experimental da COPPE/UFRJ;
- Realidade: problema modelado.
- Generalidade: específico. A questão da integração de ferramentas está sendo analisada no escopo de ferramentas de apoio à inspeção de software.

6.2.8 – Seleção dos indivíduos

A seleção dos participantes para este estudo será limitada aos desenvolvedores de ferramentas da equipe de Engenharia de Software Experimental da COPPE/UFRJ. Desta forma, a escolha dos indivíduos será baseada em princípios não probabilísticos onde a população será determinada por conveniência.

6.2.9 – Variáveis

6.2.9.1 – Variáveis Independentes

1. A lista dos critérios identificados para integração de dados e a lista dos critérios para integração de dados contemplados na abordagem de integração proposta.
2. Mecanismo para Integração de ferramentas proposto.
3. Ferramenta de apoio ao processo de inspeção.
4. Ferramenta de apoio à aplicação de PBR.
5. Ferramenta de apoio à construção de casos de uso.

6.2.9.2 – Variáveis Dependentes

1. A lista dos critérios identificados para integração de dados especificando os que foram solucionados com a utilização da abordagem para integração proposta.
2. O esforço (tempo) necessário na integração de duas ferramentas de forma manual e utilizando a abordagem para integração de ferramentas proposta.
3. Perda de dados considerando a integração de duas ferramentas.

6.2.10 – Projeto do Estudo de Caso

Para as hipóteses alternativas formuladas na seção 5.2.5, serão utilizados os seguintes padrões de projeto.

- **H1**
  - **Fator:** aderência aos requisitos para integração de dados identificados;
  - **Tratamento:** Abordagem para Integração de Dados XML proposta;
  - Esta tarefa não será efetuada pelos participantes. As informações para este fator serão capturadas através do formulário "Aderência da Abordagem de Integração aos requisitos de integração".
- **H2**
  - **Fator:** Esforço;
  - **Tratamento:** Utilizando a Abordagem para Integração de Dados XML proposta e a integração feita de forma manual;
  - Cada indivíduo estará envolvido nos dois tratamentos para o fator esforço.
- **H3**
  - **Fator:** Integridade do dado;
  - **Tratamento:** apenas um tratamento;
  - Comparação dos dados pelos indivíduos envolvidos na integração de forma a determinar se houve perda de informação (semântica, estrutural ou de conteúdo) durante quatro casos de integração. Estes quatro casos estão organizados da seguinte forma:
    - Caso 1: integração da ferramenta de apoio à aplicação de PBR e de ISPIS pelo desenvolvedor do mecanismo de

integração com objetivo de observar o comportamento do protótipo implementado.

- Caso 2: integração da ferramenta de apoio à aplicação de PBR e de ISPIS pelo desenvolvedor de ISPIS com objetivo de caracterizar a viabilidade do mecanismo de integração.
- Caso 3: integração da ferramenta de apoio à aplicação de PBR e da de apoio à construção de casos de uso pelo desenvolvedor do mecanismo de integração com objetivo o de observar o comportamento da implementação feita.
- Caso 4: integração da ferramenta de apoio à aplicação de PBR e da de apoio à construção de casos de uso pelo desenvolvedor da ferramenta de apoio à construção de casos de uso com objetivo de caracterizar a viabilidade do mecanismo de integração.

6.3 – Operação do Estudo

Nesta seção serão descritos: (1) a aderência da abordagem de integração implementada no xMapper frente aos requisitos de integração especificados e, (2) os quatros casos de integração realizados e as conclusões obtidas em cada um deles.

Antes de iniciar a discussão sobre a execução do estudo, vale destacar que a hipótese dois (H2), **H2: $E_{XML} < E_{Manual}$**, não foi avaliada diretamente pelo fato da integração manual ser trabalhosa o que implica, naturalmente, na confirmação da hipótese. Para se chegar a essa conclusão, foi tomado como parâmetro a experiência do integrador no desenvolvimento manual de conversores em XSLT. Por exemplo, o conversor apresentado no Anexo D.5 também foi desenvolvido manualmente durante o desenvolvimento do xMapper e o tempo necessário para sua criação, considerando o conhecimento prévio da XSLT, foi de aproximadamente 35 minutos. Este valor cresceria bastante para um desenvolvedor sem experiência na manipulação da linguagem. Este mesmo conversor foi criado pelo desenvolvedor do xMapper utilizando-o em aproximadamente 5 minutos. Como será visto na seção 6.3.3, um desenvolvedor sem experiência no uso do xMapper precisou de 17 minutos para criação de um conversor semelhante. Em ambos os casos, com ou sem experiência no uso do xMapper, o tempo para desenvolvimento do conversor foi reduzido significativamente. Além disso, deve-se considerar também a qualidade do conversor

construído de forma manual ou automatizada. O esforço exigido durante a construção manual de um conversor XSLT onde as regras estejam definidas corretamente é alto. Sua construção manual exige muitas vezes que ele seja construído utilizando uma abordagem de tentativa e erro implicando, muitas vezes, em retrabalho.

6.3.1 – Aderência da Abordagem de Integração frente aos Requisitos de Integração

Durante o desenvolvimento da abordagem de integração proposta nesta tese foram identificados um conjunto de requisitos para integração de ferramentas na literatura. Cada um deles foi discutido no capítulo 4. A tabela 1 faz uma compilação destas informações indicando quais requisitos foram contemplados na abordagem de integração.

Requisitos de Integração	Abordagem de Integração
Permitir a integração de ferramentas externas.	Este é o objetivo da abordagem. Neste caso, as ferramentas a serem integradas estão restritas àquelas que possam exportar seus dados no formato XML. Além disso, para duas ferramentas trocarem informações é preciso que os artefatos a serem integrados sejam de um mesmo domínio.
Tornar possível a abstração de dados.	Permite a abstração de dados através da integração de artefatos desde que estes possuam seu domínio descrito em uma ontologia e estejam descritos em XML.
Considerar a evolução semântica.	A semântica considerada neste caso é a definição dos significados dos termos presentes em artefatos e como eles se relacionam. A tecnologia adotada, OWL, permite que novos conceitos sejam acrescentados e/ou modificados.
Considerar a semântica de marcações.	A semântica das marcações é considerada desde que seu contexto seja suficiente para especificar seu significado e ela esteja descrita em uma ontologia. Por exemplo, um conceito com dois significados distintos dentro de um mesmo domínio não é considerado pela abordagem.
Considerar a integridade semântica.	Este requisito é contemplado parcialmente pela abordagem de integração. Como a abordagem é semi-automatizada, há a possibilidade do usuário quebrar a integridade durante o processo de mapeamento.
Considerar o contexto da informação.	A abordagem considera o contexto da informação através da definição de domínios de artefatos e sua descrição em ontologias. Desta forma, só podem ser integradas ferramentas cujos domínios dos artefatos tenham sido previamente definidos.
Permitir o mapeamento de informações estruturais e semânticas.	Este requisito está presente na abordagem e é efetuado de forma semi-automatizada. Entretanto, sua abrangência para mapeamento está limitada ao conjunto de marcações consideradas das linguagens OWL e XML *Schema* explicitado nos anexos A e B, respectivamente.

Tornar transparente a comunicação entre o usuário e o mecanismo de integração.	Este requisito está previsto na abordagem de integração embora não tenha sido implementado no xMapper.
Uso de um formalismo comum para estruturação de informação.	O formalismo utilizado é a XML. Para ferramentas que não exportem seus dados neste formato é necessária a criação de um dispositivo para converter o dado do formato legado para XML. A criação deste dispositivo não é contemplado na abordagem de integração.
Possuir mecanismo de transformação dos dados.	A abordagem faz uso de um processador externo responsável pela transformação nos documentos XML.
Gerar conversores de forma automatizada.	Os conversores são gerados de forma automatizada a partir das informações presentes no mapa de integração. Entretanto, só é possível a geração dos conversores para os artefatos cujos esquemas e ontologias tenham sido definidos utilizando o subconjunto das linguagens OWL e XML *Schema* definido nos anexos A e B, respectivamente.

Tabela 6.1 – Aderência da Abordagem de Integração aos Requisitos de Integração

A partir da Tabela 6.1 pode-se perceber que a hipótese nula (H0) foi refutada uma vez que a abordagem de integração contempla vários dos requisitos identificados na literatura. Por conseqüência, a hipótese alternativa 1 (H1) na qual temos que **H1:** $R_{Literatura} \cap R_{XMapper} = \varnothing$ foi confirmada.

6.3.2 – Caso 1

Para este caso, foram utilizados os seguintes instrumentos:

- Ferramenta de apoio à aplicação de PBR, PBR *Tool*;
- Ferramenta de apoio ao Processo de Inspeção, ISPIS;
- Artefato gerado pela ferramenta PBR *Tool* referente ao relatório de discrepância.

Ambas ferramentas lidam com informações relacionadas à inspeção de software. Entretanto, o artefato é estruturado de forma diferente e cada ferramenta possui necessidades específicas de informação. Parte dos artefatos a serem integrados está apresentada nas **Figuras 6.1** e **6.2** (os artefatos completos podem ser vistos nos anexos D.1 e D.6). Neste caso, o artefato a ser compartilhado é o relatório de discrepância e o sentido do intercâmbio da informação é da ferramenta de apoio a PBR para ISPIS.

```
<formDiscrepancy type="pbr">
    <perspective>USER</perspective>
    <project>ESE</project>
    <inspector>103137506</inspector>
    <start>7/9/2003 09:40:56</start>
    <artifact>C:\Program Files\PBRTool\SRS_Documents\ABC_Video_System.rtf</artifact>
    <discrepancyList>
        <discrepancy status="updated">
            <classification>Omissão</classification>
            <description>Como calcular a taxa de aluguel? Que outras taxas existem?</description>
            <identTime>7/9/2003 10:44:44</identTime>
            <step>Participants - View Document</step>
            <line>300</line>
            <requirement>Functional Requirement 8</requirement>
        </discrepancy>
    </discrepancyList>
    <PBR_help>0</PBR_help>
    <Tool_help>0</Tool_help>
    <end>7/9/2003 12:13:47</end>
    <totalTime>02:32:50</totalTime>
</formDiscrepancy>
```

Figura 6.1 – Fragmento do artefato gerado pela ferramenta de apoio à aplicação de PBR.

```
<formDiscrepancy type="pbr">
    <discrepancyList type="pbr">
        <inspectionStart>7/9/2003 09:40:56</inspectionStart>
        <discrepancy>
            <location>Functional Requirement 8 - line: 300</location>
            <classification>Omissão</classification>
            <severity/>
            <description>Como calcular a taxa de aluguel? Que outras taxas existem?</description>
        </discrepancy>
        <inspectionEnd>7/9/2003 12:13:47</inspectionEnd>
    <discrepancyList>
</formDiscrepancy>
```

Figura 6.2 – Fragmento do artefato utilizado por ISPIS.

Para estes artefatos, há a necessidade de considerar as seguintes alterações: (1) omitir marcações como é o caso de *perspective, project* e *artifact*, dentre outras; (2) modificar o nome de algumas como *start* e *end*, dentre outras, e; (3) acrescentar a marcação *severity*. Estas tarefas foram realizadas com sucesso pelo integrador.

Durante este primeiro estudo de caso, o integrador gerou o conversor para transformação dos dados. Apesar do artefato ter sido transformado sem perda de estrutura e semântica, houve uma perda de informação. O mecanismo de integração não possibilitava o mapeamento que envolvesse a relação de dois ou mais conceitos no artefato de origem para um conceito no artefato destino. Neste caso, o conteúdo proveniente das marcações *<line>* e *<requirement>* do artefato da ferramenta de PBR deveriam ser mapeados para o marcador *<location>* do documento importado por ISPIS.

Assim, para a hipótese 3 (H3), obteve-se o seguinte resultado:

$P_{semântica}$: perda semântica do **$D_{Importado}$** em relação ao **$D_{Exportado}$** = ∅;

$P_{estrutural}$: perda estrutural = ∅

$P_{conteudo}$: perda de conteúdo = perda do conteúdo do marcador *line*.

E pela hipótese formulada:

H3: (((**$P_{semântica}$ = Sem Perda**) v ($P_{semântica}$ = Baixa Importância)) ^ ((**$P_{estrutural}$ = Sem Perda**) v ($P_{estruturalG}$ = Baixa Importância)) ^ (($P_{conteudo}$ = Sem Perda) v ($P_{conteudo}$ = Baixa Importância))) = Não compromete a utilização do dado

Percebe-se que a última condição para a perda não comprometer a utilização do dado (($P_{conteudo}$ = Sem Perda) v ($P_{conteudo}$ = Baixa Importância)) não foi contemplada. Assim, a perda de conteúdo comprometia sua utilização.

6.3.2.1 – Lições Aprendidas

A partir deste primeiro estudo de viabilidade, foi possível perceber que a infra-estrutura de integração torna possível a geração de conversores para transformação dos dados de forma que a necessidade de interferência do usuário durante o processo de integração seja reduzido. Além disso, as atividades do processo de integração descritas na seção 4.5 foram contempladas no xMapper e o apoio fornecido foi satisfatório.

Quanto ao mapeamento efetuado, apesar do artefato ter sido transformado sem perda de estrutura e semântica, houve uma perda de informação pelo fato da solução não possuir mecanismos que permitissem o mapeamento que envolvesse a relação de dois ou mais conceitos no artefato de origem para um conceito no artefato destino. Isto implicou na necessidade de efetuar alguns ajustes no xMapper a avaliá-lo em um segundo estudo.

6.3.3 – Caso 2

Este segundo estudo objetivou analisar a evolução do mecanismo de integração frente às dificuldades encontradas no primeiro caso. Desta forma, foram considerados os mesmos instrumentos e a mesma situação de integração. A diferença estava no indivíduo para executar o estudo, neste caso, o próprio desenvolvedor de ISPIS.

Durante este estudo, percebeu-se que o problema encontrado durante o caso 1 foi solucionado uma vez que não houve perdas estruturais, semânticas e de conteúdo. Notou-se também que o esforço exigido para integração é baixo. Embora o usuário do integrador não tivesse utilizado a ferramenta anteriormente, o tempo utilizado durante a integração das ferramentas foi de aproximadamente 17 minutos.

Assim, para a hipótese 3 (H3), obteve-se o seguinte resultado:

$\mathbf{P_{semântica}}$: perda semântica do $\mathbf{D_{Importado}}$ em relação ao $\mathbf{D_{Exportado}}$ = ∅;

$\mathbf{P_{estrutural}}$: perda estrutural = ∅;

$\mathbf{P_{conteudo}}$: perda de conteúdo = ∅.

E pela hipótese formulada:

H3: (((**$P_{semântica}$ = Sem Perda**) v ($P_{semântica}$ = Baixa Importância)) ^ ((**$P_{estrutural}$ = Sem Perda**) v ($P_{estruturalG}$ = Baixa Importância)) ^ ((**$P_{conteudo}$ = Sem Perda**) v ($P_{conteudo}$ = Baixa Importância))) = Não compromete a utilização do dado

Percebe-se que a última condição, não contemplada no Caso 1, foi considerada o que implicou no não comprometimento de utilização do dado.

6.3.3.1 – Lições Aprendidas

Este segundo estudo permitiu avaliar a evolução do xMapper frente às dificuldades encontradas no caso 1. Observou-se também que, embora o desenvolvedor envolvido na integração não tivesse passado por um treinamento no xMapper, ele conseguiu realizar as atividades envolvidas no mapeamento.

Este caso, fechou o primeiro ciclo para avaliação do xMapper. Sua execução permitiu o aperfeiçoamento do mecanismo de integração frente às dificuldades não consideradas durante seu projeto inicial. Para dar continuidade então à avaliação da abordagem, foi executado mais um ciclo de avaliação considerando outras ferramentas e necessidades de integração.

6.3.4 – Caso 3

Para este caso, foram utilizados os seguintes instrumentos:

- a ferramenta PBR *Tool*;
- a ferramenta de apoio à construção de casos de uso;

- Artefato gerado pela ferramenta PBR *Tool* referente ao esboço dos casos de uso.

Ambas ferramentas lidam com informações relacionadas à criação dos casos de uso. Enquanto a ferramenta para PBR permite seu esboço, a de apoio à construção de casos de uso permite evoluir o esboço inicial até chegar a uma especificação mais completa.

O artefato a ser compartilhado é estruturado de forma diferente, pois cada ferramenta possui necessidades específicas de informação. Parte dos artefatos a serem integrados está apresentada nas **Figuras 6.3** e **6.4** (os artefatos completos podem ser vistos nos anexos D.1 e D.6). Neste caso, o artefato a ser compartilhado é diagrama de casos de uso e o sentido do intercâmbio é ferramenta de apoio a PBR para a ferramenta de apoio à construção de casos de uso.

```
<model>
    <participantList>
        <participant id="0">Cliente</participant>
        <participant id="2">Gerente</participant>
        <participant id="3">Empregado</participant>
    </participantList>
    <functionalityList>
        <functionality id="0">Ler código fita</functionality>
        <functionality id="1">Informar código cliente</functionality>
        <functionality id="2">Informar novo aluguel</functionality>
        <functionality id="3">Calcular total aluguel</functionality>
        ...
        <functionality id="27">Alterar dados fita</functionality>
    </functionalityList>
    <useCaseList>
        <useCase name="Aluguel de fita">
            <start>Cliente dirigi-se ao balcão com fitas selecionadas</start>
            <description>- Cliente solicita empréstimo - Empregado informa novo aluguel - Informar
        código cliente (código barra ou teclado) - O sistema verifica atrasos do cliente (mostra valor
        calculado dos atrasos por cliente) - Enquanto houver fitas (crítica: máximo de 20 fitas): . ler
        código fita. calcular data devolução de fita. atualizar valor do aluguel - Empregado informa final
        de aluguel - Calcular total de aluguel ( valor do aluguel + atrasos ) - Cliente confirma aluguel -
        Empregado informa pagamento (cheque, dinheiro ou cartão) - Armazenar aluguel (data, cliente,
        fitas) - Armazenar empréstimo fita com a data de devolução - Armazenar histórico do cliente -
        Empregado imprime formulário de aluguel</description>
                <listUCFunctionality>
                    <relationf functionality="0" />
                    <relationf functionality="1" />
                    <relationf functionality="2" />
                    ...
                    <relationf functionality="19" />
                </listUCFunctionality>
                <listUCParticipant>
                    <relationp participant="3" />
                </listUCParticipant>
        </useCase>
        ...
    </useCaseList>
</model>
```

Figura 6.3 – Fragmento do artefato gerado pela PBR *Tool*.

```
<UseCaseModel>
    <UseCases>
        <UseCase name="Aluguel de fita" id="" severity="">
            <start>Cliente dirigi-se ao balcão com fitas selecionadas</start>
        </UseCase>
        ...
    </UseCases>
    <Actors>
        <Actor name="Cliente" id="0">
            <description/>
        </Actor>
        <Actor name="Gerente" id="2">
            <description/>
        </Actor>
        <Actor name="Empregado" id="3">
            <description/>
        </Actor>
    </Actors>
    <Relationship/>
    <Diagrams/>
    <Funcionalities>
        <Funcionality userID="Ler código fita" severity="" id="0"/>
        <Funcionality userID="Informar código cliente" severity="" id="1"/>
        <Funcionality userID="Informar novo aluguel" severity="" id="2"/>
        <Funcionality userID="Calcular total aluguel" severity="" id="3"/>
        ...
        <Funcionality userID="Alterar dados fita" severity="" id="27"/>
    </Funcionalities>
</UseCaseModel>
```

Figura 6.4 – Fragmento do artefato utilizado pela *Use Case Tool*.

Para estes artefatos, o que se deseja compartilhar são as listas de: atores, funcionalidades e casos de uso. Essas informações serão utilizadas pela *Use Case Tool* para concluir o detalhamento dos casos de uso. Para efetuar este mapeamento houve então necessidade de omitir algumas marcações (por exemplo, *description* do caso de uso), modificar o nome de algumas (*participant*, *useCaseList*, dentre outras) e, acrescentar outras como *Diagrams* e *Relationships*. Além destas ações, surgiu uma nova necessidade durante o processo de integração, efetuar mapeamento entre atributos e conceitos. Neste caso, o mapeamento a ser efetuado é do atributo *userId* do marcador *Functionality* do artefato da *Use Case Tool* para o conteúdo do conceito *functionality* do artefato da PBR *Tool*.

Durante este estudo de caso, o integrador gerou o conversor. Apesar do artefato ter sido transformado sem perda de estrutura e semântica, houve perda de informação. O mecanismo de integração não possibilitava que o dado contido em um determinado conceito fosse mapeado para um atributo.

Assim, para a hipótese 3 (H3), obteve-se o seguinte resultado:

$P_{semântica}$: perda semântica do $D_{Importado}$ em relação ao $D_{Exportado}$ = ∅;

$P_{estrutural}$: perda estrutural = ∅

$P_{conteudo}$: perda de conteúdo = perda do conteúdo do atributo *userID*.

E pela hipótese formulada:

H3: (((**$P_{semântica}$ = Sem Perda**) v ($P_{semântica}$ = Baixa Importância)) ^ ((**$P_{estrutural}$ = Sem Perda**) v ($P_{estruturalG}$ = Baixa Importância)) ^ (($P_{conteudo}$ = Sem Perda) v ($P_{conteudo}$ = Baixa Importância))) = Não compromete a utilização do dado

Percebe-se que a última condição para a perda não comprometer a utilização do dado (($P_{conteudo}$ = Sem Perda) v ($P_{conteudo}$ = Baixa Importância)) não foi contemplada. Assim, a perda de conteúdo comprometia sua utilização.

6.3.4.1 – Lições Aprendidas

A partir deste terceiro estudo, foi identificado outro ponto não solucionado pela infra-estrutura de integração. Não era possível mapear um dado contido em um determinado conceito para um atributo. Isto implicou na necessidade de efetuar alguns ajustes no xMapper a avaliá-lo em um quarto estudo.

6.3.5 – Caso 4

Este estudo objetivou analisar a evolução do mecanismo de integração frente às dificuldades encontradas no terceiro caso. Para isto, foram considerados os mesmos artefatos a serem integrados mas o indivíduo a executar o estudo foi o desenvolvedor da ferramenta de apoio à construção de casos de uso.

Durante este estudo, percebeu-se que o problema encontrado durante o caso 3 foi solucionado uma vez que a perda de conteúdo deixou de existir. Notou-se também que o esforço exigido para integração é baixo. Embora o usuário do integrador não tivesse utilizado a ferramenta anteriormente, o tempo utilizado durante a integração das ferramentas foi de aproximadamente 21 minutos.

Assim, para a hipótese 3 (H3), obteve-se o seguinte resultado:

$P_{semântica}$: perda semântica do $D_{Importado}$ em relação ao $D_{Exportado}$ = ∅;

$P_{estrutural}$: perda estrutural = ∅;

$P_{conteudo}$: perda de conteúdo = ∅.

E pela hipótese formulada:

H3: (((**$P_{semântica}$ = Sem Perda**) v ($P_{semântica}$ = Baixa Importância)) ^ ((**$P_{estrutural}$ = Sem Perda**) v ($P_{estruturalG}$ = Baixa Importância)) ^ ((**$P_{conteudo}$ = Sem Perda**) v ($P_{conteudo}$ = Baixa Importância))) = Não compromete a utilização do dado

Percebe-se que a última condição, não contemplada no Caso 3, foi considerada o que implicou no não comprometimento de utilização do dado.

6.3.5.1 – Lições Aprendidas

Este quarto estudo permitiu avaliar a evolução do xMapper frente às dificuldades encontradas no caso 3. O comportamento do individuo utilizado para realizar a integração foi semelhante ao do caso 2, ou seja, embora ele não tivesse passado por um treinamento no xMapper, conseguiu realizar as atividades envolvidas no mapeamento.

Este caso, encerrou o segundo ciclo para avaliação do xMapper. Sua execução permitiu o aperfeiçoamento do mecanismo de integração frente às dificuldades não consideradas durante seu projeto inicial e a evolução alcançada durante os estudos anteriores.

6.4 – Considerações Finais

Este capítulo fez uma breve introdução à experimentação destacando alguns de seus métodos – pesquisa de campo, estudo de caso e estudo experimental – e a importância de sua utilização na engenharia de software.

Feito isto, foi apresentado o estudo de caso definido e planejado para caracterizar a viabilidade da abordagem de integração apresentada no capítulo 4. Através de sua execução e da análise dos resultados, foi possível identificar gradualmente os pontos de melhoria da abordagem. A percepção se deu principalmente nos casos 1 e 3 onde a xMapper encontrou necessidades de mapeamento não previstas durante sua concepção. Já os casos 2 e 4 serviram para avaliar as modificações implementadas no xMapper frente às dificuldades encontradas.

O estudo como um todo foi útil também na percepção de que a infra-estrutura de integração torna possível a geração de conversores de forma a minimizar a necessidade de interferência do usuário durante o processo de integração. Além disso,

foi verificado também que as atividades do processo de integração descritas na seção 4.5 foram contempladas no xMapper e o apoio fornecido mostrou-se satisfatório.

Frente ao esforço de integração exigido, notou-se que o xMapper foi utilizado corretamente pelos indivíduos envolvidos nos estudos 2 e 4 e, o tempo médio gasto para efetuar o mapeamento e transformação nos artefatos foi de 19 minutos. Este valor pode ser considerado baixo se comparado ao desenvolvimento manual de conversores semelhantes como discutido na seção 6.3. Entretanto, nestes estudos não foi considerado o tempo gasto para preparação dos artefatos para integração (atividade analisar as ferramentas a serem integradas). Esta preparação envolve a definição da estutura dos artefatos a serem integrados em esquemas e a semântica do tipo de artefato a ser integrado em uma ontologia. O esforço necessário para esta atividade foi de aproximadamente 3 horas para os casos 1 e 2 e, 5 horas para os casos 3 e 4.

Considerando os problemas encontrados durante a realização dos estudos, percebeu-se que, no pior caso, havia perda de informação. Assim, a semântica e a estrutura do artefato sempre foi mantida e isto possibilitava sempre a utilização do artefato integrado pela ferramenta de destino, embora parte do seu conteúdo tivesse sido perdido. Ainda assim, esses problemas foram considerados e solucionados na evolução da infra-estrutura de integração.

O estudo, como um todo, demonstrou que a utilização prática da abordagem de integração é viável. Um outro indicador desta viabilidade é o fato da xMapper ser atualmente utilizada para prover a integração de ferramentas externas a ISPIS, uma infra-estrutura de apoio ao processo de inspeção de software que está disponível e tem sido utilizada em inspeções reais na COPPE/UFRJ. Neste contexto, a xMapper tem se mostrado uma solução adequada para a integração de ferramentas CASE tanto no processo de construção do mapa de integração para cada um das ferramentas externas quanto na geração e disponibilização dos conversores.

Por fim, vale ressaltar que, embora os estudos tenham demonstrado que a abordagem de integração elaborada é viável, é necessário que outros estudos sejam executados com objetivo de analisar a abrangência do xMapper frente à integração de outras ferramentas CASE cujos artefatos possuam particularidades de estruturação não contempladas nos estudos executados. Assim, o estudo efetuado é de abrangência específica não sendo possível verificar sua validade em termos estatísticos.

No próximo capítulo serão discutidas algumas considerações finais e direções futuras para dar continuidade a este trabalho.

Capítulo 7

Considerações Finais

Este capítulo apresenta as conclusões, contribuições, limitações, e as perspectivas futuras de trabalhos que poderão ser realizados a partir desta tese.

7.1 – Conclusões

Lidar com os problemas de heterogeneidade semântica e estrutural na integração de ferramentas CASE, como discutido durante esta tese, implica na necessidade de transformações entre diferentes perspectivas de utilização de um mesmo artefato. O ideal seria que essas transformações fossem efetuadas de forma automatizada, entretanto, criar mecanismos que infiram automaticamente os mapeamentos necessários para efetuar estas transformações nem sempre é possível.

Na engenharia de software, abordagens de integração têm sido desenvolvidas, em particular, na área de ambientes de desenvolvimento de software. Existem duas abordagens principais de integração: a baseada em um modelo canônico e a fundamentada na integração de aplicações aos pares. Os principais problemas destas abordagens estão na dificuldade de integração de ferramentas externas e na manutenção dos mapeamentos entre aplicações, respectivamente.

Neste contexto, esta tese apresentou uma abordagem para integração de artefatos de um mesmo domínio descritos em XML utilizando informações estruturais provenientes de esquemas e semânticas provenientes de ontologias. Ela lida com os problemas enfrentados pela abordagem do modelo canônico provendo uma solução menos acoplada ao mesmo tempo em que reduz o esforço de manutenção exigido para integração de ferramenta aos pares

Para isto, a abordagem possibilita que o mapeamento entre os diferentes artefatos ocorra de forma semi-automatizada e a geração das regras de transformação nos documentos ocorra de forma automatizada. O mapeamento é semi-automatizado por que embora ontologias permitam uma boa especificação dos significados das

informações, ainda assim elas nem sempre capturam toda a semântica dos artefatos. Um dos motivos disto são as necessidades específicas de cada ferramenta frente a um mesmo tipo de informação. Assim, é necessária a intervenção humana no processo de identificação de correspondências entre diferentes artefatos de um mesmo domínio.

A abordagem foi implementada na ferramenta xMapper. Um estudo de caso permitiu avaliar a viabilidade de sua utilização. Além disto, a ferramenta é atualmente utilizada para prover a integração de ferramentas externas a ISPIS, uma infra-estrutura de apoio ao processo de inspeção de software que está disponível e tem sido utilizada em inspeções reais na COPPE/UFRJ. Neste contexto, o xMapper tem se mostrado uma solução adequada para a integração de ferramentas CASE tanto no processo de construção do mapa de integração para cada um das ferramentas externas quanto na geração e disponibilização dos conversores.

Entretanto, vale lembrar que, embora os estudos realizados tenham demonstrado a viabilidade da abordagem de integração, é necessário que outros estudos sejam executados para avaliar a efetividade do xMapper na integração de ferramentas CASE cujos artefatos possuam particularidades semânticas e estruturais não contempladas nos estudos executados.

A partir de agora serão descritas as contribuições deste trabalho, suas limitações e por último, as perspectivas futuras.

7.2 – Contribuições

Dentre as principais contribuições deste trabalho, destacamos:

1. A pesquisa feita na literatura para identificação de requisitos para integração de dados;
2. A definição de uma abordagem para integração de dados de um mesmo domínio descritos em XML utilizando esquemas e ontologias:
    a. Construção de um algoritmo para mapeamento semi-automatizado de esquemas em ontologias;
    b. Construção de um algoritmo para geração automatizada de *drivers* de transformação descritos em XSLT a partir de informações contidas no mapa de integração;
3. Consideração da semântica e da estrutura dos dados no processo de integração;

4. Definição de um processo de integração para dar apoio às atividades de integração definidas na abordagem proposta;
5. A construção da ferramenta xMapper que permite a integração de artefatos de um mesmo domínio descritos em XML de forma semi-automatizada;
6. Definição e execução de um estudo de caso para caracterizar a viabilidade da abordagem de integração proposta;
7. A utilização da abordagem de integração proposta e de um protótipo na definição e construção de uma infra-estrutura de apoio ao processo de inspeção de software tornando possível a utilização de ferramentas externas para apoiar diferentes tipos de detecção de defeitos em ISPIS.

A abordagem proposta introduz o apoio à integração de ferramentas e representa um passo importante na construção de ambientes para apoio às diversas atividades do desenvolvimento de software.

7.3 – Limitações

Algumas limitações foram consideradas durante o desenvolvimento da abordagem proposta nesta tese. Primeiro, foi delimitado o escopo dos artefatos a serem integrados como sendo do mesmo domínio e descritos em XML. Em decorrência da utilização de XML e suas linguagens correlatas, considerou-se também a necessidade de limitar o poder de expressão considerado nestas tecnologias correlatas. Isto por que a complexidade decorrente dos variados tipos de marcações presentes e das diversas possibilidades de combinação estenderiam em muito o tempo de desenvolvimento da ferramenta xMapper. As limitações referentes ao poder de expressão das tecnologias utilizadas estão melhor descritas nos Anexos A e B.

7.4 – Perspectivas Futuras

Buscando-se melhorar e expandir a abordagem de integração proposta, algumas perspectivas de trabalhos futuros são destacadas.

1. Implementar a Ferramenta Cliente e o componente Comunicação do Gerente de Integração;

2. Aperfeiçoar o algoritmo de mapeamento para que ele contemple mecanismos que possibilitem a transformação nos dados considerando, por exemplo: conversões de moeda e unidades de medida;
3. Considerar maiores possibilidades de combinação e definição entre as marcações das linguagens para definição de esquema (XML *Schema*) e ontologias (OWL);
4. Realizar mais alguns estudos de caso para verificar outras possíveis melhorias na abordagem de integração;
5. Utilizar a abordagem de integração na definição e construção de um ambiente para apoiar a experimentação em engenharia de software;
6. Finalizar a implementação do protótipo para que ele contemple todas as características especificadas pela abordagem de integração proposta;
7. Pesquisar possibilidades para lidar com o problema da garantia da integridade semântica dos dados uma vez que durante o mapeamento, o usuário do xMapper pode definir relacionamentos entre os dados que são inconsistentes com a semântica do artefato descrito, e;
8. Definir heurísticas para avaliar a qualidade de documentos XML em termos semânticos e estruturais.

Por fim, este trabalho mostrou, além das contribuições que trouxe e dos trabalhos futuros que proporciona, o nível de complexidade em lidar com problemas de integração. É um campo muito vasto de pesquisa e por isso, as soluções devem possuir um contexto bem definido para que sejam viáveis.

REFERÊNCIAS BIBLIOGRÁFICAS

ADAMS, T., DULLEAM J., CLARK, P., SRIPADA, S., BARRETT, T., 2000, Semantic Integration of Heterogeneous Information. Sources Using a Knowledge-Based System. In Proc 5th Int. Conf. on CS and Informatics (CS&I'2000).

AMARAL, E.A.G., 2003, Empacotamento de Experimentos em Engenharia de Software, Tese de Mestrado, COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Rio de Janeiro.

APPARAO, V., BYRNE, S., CHAMPION, M., *et al.*, 1998, Document Object Model (DOM) Level 1 Specification, W3C Recommendation. Disponível em http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/. Último acesso em 16/11/2003.

BARLOW, S., 2000, Data Integration. University of Passau.

BASILI, V.R., SHULL, F., LANUBILE, F., 1999, "Building Knowledge Through Families of Experiments". IEEE Transactions on Software Engineering, Vol. 25, No. 4, July/August.

BERNERS-LEE, T., HENDLER, J., LASSILA, O., 2001, The Semantic Web, Scientific American 248, 5, pp. 34 – 43.

BIRD L.J., GOODCHILD, A., BEALE, T., 2000a, Integrating Health Care Information Using XML-Based Metadata. HIC'2000.

BIRD, L., GOODCHILD, A., SUE, H., 2000b, "Describing Electronic Health Records Using XML-Schema", XML Asia Pacific'2000.

BIRD L.J., GOODCHILD, A., TUN, Z., 2003, Experiences with a Two-Level Modelling Approach to Eletronic Health Records. Journal of Research and Practice in Information Technology, vol. 35, nº 2, p.p. 121 – 138.

BOMPANI, L., CIANCARINI, P., VITALI, F., 2000, "Software Engineering and the Internet: a Roadmap". Pp. 261-277, ACM Press.

BOURRET, R., 2001, Mapping W3C Schemas to Object Schemas to Relational Schemas. Disponível em http://www.rpbourret.com/xml/SchemaMap.htm. Último acesso em 22/11/2003.

BRYAN,M. 1992. An introduction to the standard generalization markup language (SGML). Disponível em http://www.personal.u-net.com/sgml/sgml.htm. Último acesso em 15/11/2003.

CHAMPION, M., 2001, Storing XML in Databases. EAI Journal, October, pp. 53-55.

CHAPETTA, W.A., 2004, "Ferramenta para Construção de Modelos de Casos de Uso", Projeto Final de Curso, Departamento de Ciência da Computação/Universidade Federal do Rio de Janeiro.

CHAUDHRI, A.B., RASHID, A., ZICARI, R., 2003, XML Data Management: Native XML and XML-Enabled Database Systems. Addison-Wesley Pub Co, 1st edition.

COLLINS, S.R., NAVATHE, S., MARK, L., 2002, "XML Schema mappings for heterogeneous database access". Information and Software Technologies, v. 44, pp. 251-257.

ConceptBase, 1998. Disponível em www-i5.informatik.rwth-aachen.de/Cbdoc. Último acesso em 28/01/2004.

COLLINS, S.R., NAVATHE, S., MARK, L., 2002, "XML Schema mappings for heterogeneous database access". Information and Software Technologies, v. 44, pp. 251-257.

COWAN, J., *et al.*, 2003, Extensible Markup Language (XML) 1.1. W3C Proposed Recommendation. Disponível em http://www.w3.org/TR/xml11/. Último acesso em 14/11/2003.

CUI, Z., COX, M.D.J., JONES, D.M., 2001a, An Environment for Managing Enterprise Domain Ontologies. M. Rossi and K. Siau (eds.) *Information Modelling in the New Millennium,* Idea Group Publishing, London.

CUI, Z., JONES, D., O'BRIEN, P., 2001b, Issues in Ontology-based Information Integration. IJCAI – Seattle, USA.

ELMASRI, R., WU, Y., HOJABRI, B., *et al.*, 2002, Conceptual Modeling for Customized XML Schemas. In: Lecture Notes in Computer Science, Springer-Verlag Heidelberg, vol. 2503, p.p. 429 – 443.

EMBLEY, D.W., MOK, W.Y., 2001, "Developing XML Documents with Guaranteed "Good" Properties", In: Lecture Notes in Computer Science. Springer-Verlag Heidelberg, vol. 2224.

EMMERICH, W., 2000, Software Engineering and Middleware: A Roadmap. In A. Finkelstein, editor, *Future of Software Engineering*, pages 117–129. ACM Press.

EMMERICH, W., et al., 2001, TIGRA – An Architectural Style for Enterprise Application Integration. In Proc. of the 23rd Int. Conference on Software Engineering, Toronto, Canada. pp. 567-576. ACM Press.

FALBO, R.A., 1998, *Integração de Conhecimento em um Ambiente de Desenvolvimento de Software*, Tese de Doutorado, COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Rio de Janeiro, Dez.

FALBO, R.A., MENEZES, C.S., ROCHA, A.R.C., A Systematic Approach for Building Ontologies, In: Proceedings of the IBERAMIA'98, Lisboa, Portugal, 1998.

FALLSIDE, D.C., 2001, XML Schema Part 0: Primer. W3C Recommendation. Disponível em http://www.w3c.org/TR/xmlschema-0/. Último acesso em 13/11/2003.

FENG, L., CHANG, E., DILLON, T., 2002, A Semantic Network-Based Design Methodology for XML Documents, ACM Transactions on Information Systems, Vol. 20, No. 4, October 2002, Pages 390–421.

FpML, 1999, Introducing FpML: A New Standard for ecommerce. Disponível em: http://www.fpml.org. Último acesso em 18/01/2004.

GUARINO, N., 1998, "Formal Ontology and Information Systems". In: *Proceedings of the First International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, June.

GRUBER, T.R., 1995, "Towards principles for the design of ontologies used for knowledge sharing", *Int. J. Human-Computer Studies*, v. 43, n. 5/6.

GRUNINGER, M., LEE, J., 2002, Ontology Applications and Design. Communications of the ACM, p.p. 39 - 41, vol. 45, nº. 2.

HA, S., KIM, K., 2001, Mapping XML Documents to the Object-Relational Form, Proceedings of The 2001 International Symposium on Industrial Electronics, June, IEEE.

HARRISON, W., OSSHER, H., TARR, P., 2000, *The Future of Software Engineering , Software Engineering Tools and Environments: a Roadmap.* Pp. 261-277, ACM Press.

HARMELEN, O., HENDLER, J., HORROCKS, I., MCGUINNESS, D.L., PATEL-CHNEIDER, P.F., STEIN, L.A., 2003. OWL Web Ontology Language Reference. Working Draft, World Wide Web Consortium. Disponível em http://www.w3.org/TR/owl-ref/. Último acesso em 14/11/2003.

HARTMANN, J., HUANG, H., TILLEY, S., 2001, Documenting software systems with views II: an integrated approach based on XML. Proceedings of the 19th annual international conference on Computer documentation, p.p. 237 – 246.

HASSELBRING, W., 2000, Information System Integration. Communications of the ACM, vol. 43, nº 6, pp. 33 – 38.

HEFLIN, J., 2003, OWL Web Ontology Language Use Cases and Requirements. W3C Candidate Recommendation. Disponível em http://www.w3.org/TR/webont-req/. Último acesso em 02/12/2003.

HINA, 2000, A Health Information Network for Austrália. Disponível em: http://www.health.gov.au/healthonline/ehr_rep.htm. Último acesso em 15/01/2004.

HORS, A.L., HÉGARET, P.L., WOOD, L., NICOL, G., ROBIE, J., CHAMPION, M., BYRNE, S., 2003, Document Object Model (DOM) Level 3 Core Specification – Version 1.0. W3C Candidate Recommendation. Disponível em http://www.w3c.org/TR/2003/CR-DOM-Level-3-Core-20031107/. Último acesso em 14/01/2004.

HSIAO, H., HUI J., LI, N., TIJARE, P., 2002, "Integrated XML Document Management". First VLDB Workshop on Efficiency and Effectiveness of XML Tools, and Techniques (EEXTT2002), August 20 – 23, Hong Kong, China.

IEEE Std 830-1998, 1998, IEEE Recommended Practice for Software Requirements Specifications.

KALINOWSKI, M., SPINOLA, R. O., TRAVASSOS, G.H., 2004, "Infra-Estrutura Computacional para Apoio ao Processo de Inspeção de Software", Simpósio Brasileiro de Qualidade de Software.

KAPPEL, G., KAPSAMMER, E., RETSCHITZEGGER, W., 2001, XML and Relational Database Systems – A Comparison of Concepts. International Conference on Internet Computing (IC'2001) Las Vegas, USA, June.

KARSAI, G., 2000, "Design Tool Integration: An exercise in Semantic Interoperability". In 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems April 03 - 07, Edinburgh, Scotland.

KARSAI, G., GRAY, J., 2000, Component Generation Technology for Semantic Tool Integration. IEEE.

KAY, M., 2003, XSL Transformations (XSLT) Version 2.0. W3C Working Draft. Disponível em http://www.w3.org/TR/xslt20/. Último acesso em 20/11/2003.

LASSILA, O., VAN HARMELEN, F., HORROCKSM, I., HENDLER, J., MCGUINNESS, D. L., 2000, The Semantic Web and its Languages, In: IEEE Intelligent Systems, p. 67-73, November/December.

MADNICK, S., 2001, "The Misguided Silver Bullet: What XML will and will NOT do to help Information Integration," *Proceedings of the Third International Conference on Information Integration and Web-based Applications and Services*, IIWAS2001, Linz, Austria, September: 1-72.

MAEDCHE, A., STAAB, S., 2001, Ontology Learning for the Semantic Web, IEEE Intelligent Systems. March/April, p.p. 72-79.

MARCHAL, B., 2000, *XML By Example*. QUE.

MASCOLO, C., EMMERICH, W., FINKELSTEIN, A., 2001, *XML technologies and software engineering*. Proceedings of the 23rd international conference on Software engineering, ICSE2001, Canada, July, p.p.: 775 – 776.

MEDEIROS, S., SOUZA, J.M., STRAUCH, J.C.M., PINTO, G.R.B., 2000, SPeCS - A spatial decision support collaborative system for environment design. In *Proceedings of the Fifth International Workshop in CSCW in Design*. Hong Kong, Nov.

MYLOPOULOS, J., BORGIDA, A., JARKE, M., KOUBARAKIS, M., 1990, Telos: Representing Knowledge about Information Systems. In ACM Transactions on Information Systems, 8(4):325-362.

O´LEARY, D. E., 1998, "Enterprise Knowledge Management", *IEEE Intelligent Systems*, v.13, n.3, pp.54-61.

OLIVEIRA, K.M, 1999, "Modelo para Construção de Ambientes de Desenvolvimento Orientados a Domínio", Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

PFLEEGER, S.L. Software Engineering: Theory and Parctice. 2 ed. Prentice Hall, 2001.

PINTO, G.R.B., MEDEIROS, S.P.J., SOUZA, J.M., STRAUCH, J.C.M., MARQUES, C.R.F., 2003, "X-Arc spatial data integration in the SPeCS collaborative design framework". Communications of the ACM, v. 46, n 3, pp 86-90.

RAGGETT, D., HORS, A.L., JACOBS, I., 1999, HTML 4.01 Specification. W3C Recommendation. Disponível em http://www.w3.org/TR/html4/. Último acesso em 15/11/2003.

ROCHA, A.R.C, AGUIAR T. C., SOUZA, J. M., 1990, *"TABA: A Heuristic Workstation for Software Development"*, In: Proceedings of COMPEURO'90, Tel Aviv, Israel, Maio.

RODRÍGUEZ-GIANOLLI, P., 2001, Data Integration for XML based on Schematic Knowledge. Master's Thesis, Department of Electrical and Computer Engineering, University of Toronto.

RODRÍGUEZ-GIANOLLI, P., MYLOPOULOS, J., 2001, A Semantic Approach to XML-based Data Integration. Proceedings of the 20th. International Conference on Conceptual Modelling (ER) , Yokohama, Japan.

SATTLER, K., 1997, A Framework for Component-Oriented Tool Integration, In: Proc. of 4th Int. Conference on Object-Oriented Information Systems (OOIS'97), Brisbane, Australia, Nov 10-12, S. 455-465, Springer-Verlag.

SELIGMAN, L., ROSENTHAL, A., 2001, "XML´s impact on databases and data sharing". Computer, June 2001, pp. 59-67.

SHANMUGASUNDARAM, J., et. Al., 1999, Relational Databases for Querying XML Documents: Limitations and Opportunities. Proceedings of the 25th VLDB Conference, Edinburgh, Scotland.

SHARON, D., BELL, R., 1995, Tools that Bind: Creating Integrated Environments. IEEE Software, 12(2):76--85, Mar.

SHEKHAR, S., VATSAVAI, R.R., SAHAY, N., *et al.*, 2001, WMS and GML based Interoperable Web Mapping System. Proceedings of the Ninth ACM International lymposium on Advances in Geographic Information Systems, pp. 106-111, USA.

SHETH, A.P., 1998, Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics. M.F. Goodchild, M.J. Egenhofer, R. Fegeas, e C.A. Kottman (eds.) *Interoperating Geographic Information Systems*, Kluwer.

SILVA, L.F.S., CHAPETTA, W.A., TRAVASSOS, G.H., 2004, "Inspeções de Requisitos de Software Utilizando PBR e Apoio Ferramental", Simpósio Brasileiro de Qualidade de Software.

SPINOLA, R.O., TRAVASSOS, G.H., 2003, Uma Abordagem para Integração de Ferramentas, *In: Anais do WTES´2003 (SBES) - Workshop de Teses em Engenharia de Software*, p.59-64, Manaus.

SUCIU, D., 2001. "On Database Theory and XML". SIGMOD Record, Vol. 30, No. 3, September, pp 39-45.

TIAN, F., DEWITT, D., CHEN, J., ZHANG, C., 2002, The Design and Performance Evaluation of Alternative XML Storage Strategies. SIGMOD Record, Vol 31, n 1, March, pp 5-10.

TICHY, W.F., 1998, "Should Computer Scientists Experiment More?". IEEE Computer, 31(5), pp. 32-39.

TRAVASSOS, G. H., 1994, O Modelo de Integração de Ferramentas da Estação TABA, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

TRAVASSOS, G.H.; GUROV, D.; AMARAL, E.A.G.G., 2001, Introdução à Engenharia de Software Experimental. In: Relatório Técnico ES-590/02-Abril, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, disponível em http://www.cos.ufrj.br/publicacoes.

USCHOLD, M.E., 1998. Knowledge level modelling: Concepts and Terminology. Knowledge Engineerign Review, 12(1). Also available as AIAI-T R-196 from AIAI, The Uuniversity of Edinburgh.

USCHOLD, M.E., JASPER, R., 1999, A Framework for Understanding and Classifying Ontology Applications. In Proceedings of the IJCAI99 Workshop on Ontologies and Problem-Solving Methods(KRR5), Stockholm, Sweden, August 1999.

VAN HEIJST, G. VAN, SCHREIBER, A TH., WIELENGA, B. J., 1997, "Using Explicit Ontologies in KBS Development", *International Journal of Human-Computer Studies*, vol 45, No 2/3; pp 183-292.

VARLAMIS, I., VAZIRGIANNIS, M., 2001, Bridging XML-Schema and relational databases. A system for generating and manipulating relational databases using valid XML documents.

VILLELA, K., 2004, "Ambientes de Desenvolvimento de Software Orientados à Organização", Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

VISSER, P.R.S., JONES, D.M., BENCH-CAPON, T.J.M., SHAVE, M.J.R., 1998, Assessing Heterogeneity by Classifying Ontology Mismatches. In N. Guarino, editor, Proceedings of 1st International Conference on Formal Ontologies in Information Systems, FOIS'98, Trento, Italy, pages 148--162. IOS Press.

YAKIMOVICH, D.; TRAVASSOS, G.; BASILI, V., 1999, A Classification of Software components Incompatibilities for COTS Integration, Software Engineering

Laboratory Workshop, NASA/Goddard Space Flight Center, Greenbelt, Maryland, December.

WASSERMAN, A.I., 1990, Tool Integration in Software Engineering Environments. In F. Long, editor, Software Engineering Environments: Proc. Int. Workshop on Environments, LNCS 467, p.p. 137-149. Springer Verlag.

WIEDERHOLD, G., 1995, Mediation in information systems. *ACM Computing Surveys* 27, 2.

WÖB, W., PÜHRETMAIR, F., 2001, "XML-Based Integration of GIS and Heterogeneous Information". The 13th Conference on Advanced Information Systems Engineering, [ISSN 0302-9743] [ISBN 3-540-42215-3], Springer Verlag, Proceedings, pp.346-358, LNCS 2068.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.C.; REGNELL, B.; WESSLÉN, A., 2000. Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers, Massachusetts.

Anexo A – Escopo Definido da OWL

Este anexo apresenta o escopo da OWL a ser considerado em sua manipulação.

A complexidade decorrente dos variados tipos de marcações presentes nesta linguagem e nas diversas possibilidades de combinação entre elas tornou necessária a definição de seu escopo. Para isto, tentou-se limitar o poder de expressão da OWL com o cuidado de selecionar um conjunto de marcações que fosse abrangente e conseguisse lidar com a maioria dos problemas envolvendo a definição da semântica dos artefatos.

A figura A.1 apresenta o meta-modelo definido com base na especificação da OWL (HARMELEN *et al.*, 2003) a ser considerado nesta proposta.

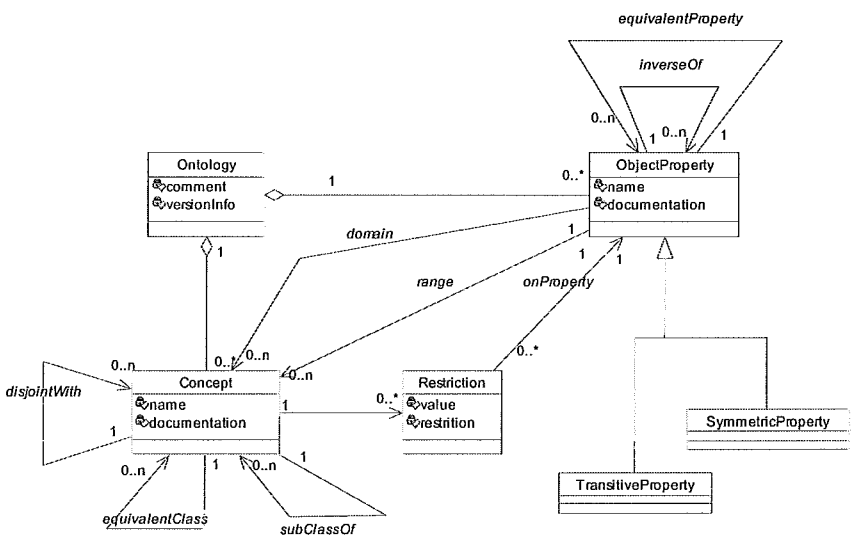


Figura A.1 Meta-modelo de Ontologias definidas em OWL considerada.

Assim, na solução proposta nesta tese, esta se considerando a definição de conceitos, propriedades e:

- **Restrições:** *allValuesFrom, someValuesFrom, hasValue, maxCardinality, minCardinality, cardinality, domain, range.*
- **Axiomas:** *subClassOf, equivalentClass, disjointWith, equivalentProperty, inverseOf, TransitiveProperty, SymmetricProperty, subPropertyOf.*

Anexo B – Escopo Definido do XML *Schema*

Este anexo apresenta o escopo do XML Schema a ser considerado em sua manipulação.

Assim como a OWL, o XML *Schema* também teve seu escopo limitado. Este, por sua vez, necessitou mais atenção pelo fato da quantidade de combinações possíveis entre suas marcações ser muito grande. Assim, definiu-se primeiro um sub-conjunto das marcações existentes – o que já reduziu bastante as possibilidades de combinação – e depois, o modo como suas combinações seriam consideradas.

Embora estas definições limitem o poder de expressão da XML *Schema*, atentou-se para o fato do conjunto de marcações e combinações serem representativos contemplando, desta forma, os elementos necessários para uma especificação abrangente da estruturação de artefatos XML. A definição do escopo da XML *Schema* fundamentou-se na seguinte constatação: documentos XML são, em sua maioria, compostos por elementos simples e complexos. Cada um destes elementos aparece em uma determinada seqüência podendo ocorrer uma ou mais vezes dentro de um mesmo documento. Além disso, é muito comum também a presença de atributos nos elementos complexos.

A Figura B.1 apresenta o sub-conjunto de marcações e como elas podem estar combinadas. Neste caso, temos as seguintes limitações:

- Esquemas devem ser representados por elementos e estes devem ser definidos como nativo ou como complexo;
- Caso seja nativo, um elemento é caracterizado pelas marcações *name*, *minOccurs, maxOccurs* e o tipo propriamente dito;
- Caso seja complexo, o elemento poderá conter atributos e uma seqüência de outros elementos que pertencem ao tipo complexo.

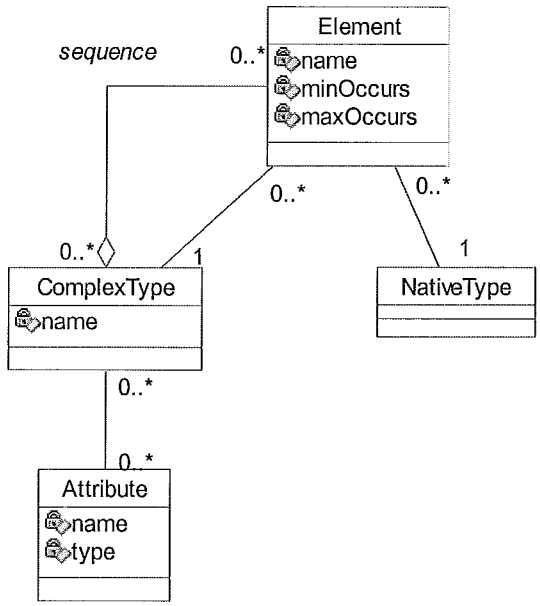
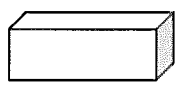
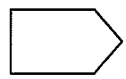


Figura B.1 Meta-modelo de Esquemas considerados.

Anexo C – Linguagem para Modelagem de Processos

Este anexo apresenta a notação utilizada para representar o Processo de Configuração da Infra-Estrutura de Integração descrito no capítulo 4 – Requisitos para uma Infra-estrutura de Integração de Ferramentas com XML.

A linguagem proposta para modelagem de processos em (VILELA, 2004) é composta de elementos gráficos que podem ser do tipo área, objeto ou ligação, onde uma ligação estabelece uma relação entre dois objetos e uma área agrupa objetos, definindo um contexto para os mesmos. Objetos ainda permitem adornos, utilizados para representar explicitamente características dos objetos. A seguir, cada elemento da linguagem é brevemente apresentado.

Objeto	Notação	Definição
Processo		Objeto referente ao conceito de mesmo nome definido na ontologia de organização (seção 5.7). *Atributos Especiais:* Origem (Interno, Externo)
Evento		Objeto que representa um acontecimento no ambiente que provoca o início ou fim de um processo. A notação é proveniente do produto comercial de *workflow* ARIS *ToolSet*.
Ator		Objeto que representa um pessoa, agente ou unidade organizacional. Estes conceitos encontram-se definidos na ontologia de organização. A notação foi utilizada por KRUCHTEN [179] para representação dos *workflows* básicos do *Rational Unified Process*.
Atividade		Objeto referente ao conceito de mesmo nome definido na ontologia de organização. A notação foi utilizada por KRUCHTEN [179] para representação dos *workflows* básicos do *Rational Unified Process*. *Atributos Especiais:* Origem (Interna, Externa) Granularidade (Elementar ou Composta)
Estado Inicial		Objeto puramente notacional, proveniente dos diagramas de estado e que indica onde é iniciado o fluxo de atividades que definem um processo ou uma atividade composta
Estado Final		Objeto puramente notacional, proveniente dos diagramas de estado e que indica onde é encerrado o fluxo de atividades que definem um processo ou uma atividade composta

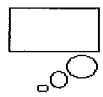
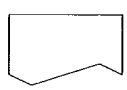
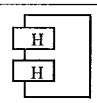
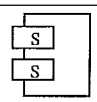
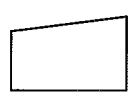
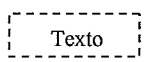
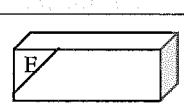
Conhecimento Explícito		Objeto que representa um conhecimento que pode ser expresso em palavras e números e ser facilmente transmitido e compartilhado. A notação foi proposta por ALLWEYER [10].
Conhecimento Implícito		Objeto que representa um conhecimento que é altamente pessoal e difícil de formalizar, o que o torna também difícil de ser compartilhado. A notação foi proposta por ALLWEYER [10].
Comunicação		Objeto que representa a comunicação de dados ou informações a partir da, ou para a, execução de uma atividade. A comunicação pode ser verbal ou escrita e exemplos são e-mail e fax.
Repositório (Meio Magnético)		Objeto que representa um meio magnético para o armazenamento de dados e informações. A notação é proveniente do produto comercial de *workflow* ARIS *ToolSet*.
Arquivo (Local Físico)		Objeto que representa um local físico para armazenamento de documentos e comunicações escritas.
Documento		Objeto referente ao conceito de mesmo nome definido na ontologia de organização. A notação é proveniente do produto comercial de *workflow* ARIS *ToolSet*.
Componente de Hardware		Objeto referente ao conceito de mesmo nome definido na ontologia de organização. A notação é baseada na notação de componente da UML.
Componente de Software		Objeto referente ao conceito de mesmo nome definido na ontologia de organização. A notação é baseada na notação de componente da UML.
Peça		Objeto referente ao conceito de mesmo nome definido na ontologia de organização.
Matéria-Prima		Objeto referente ao conceito de mesmo nome definido na ontologia de organização.
Bem		Objeto referente ao conceito de mesmo nome definido na ontologia de organização. A notação fornecida pode ser substituída por uma mais significativa para o objeto específico do modelo como, por exemplo, o logotipo do software. *Atributos Especiais:* Tipo (Usufruto, Software, Hardware e Equipamento de Produção)
Nota Explicativa		Objeto que permite que notas explicativas sejam adicionadas ao modelo. *Atributos Especiais:* Texto

Tabela C.1 – Definição e Notação dos Objetos

Objeto	Notação com Adornos	Definição dos Adornos
Processo		Adorno que indica que o processo é externo, ou seja, que é executado por outra organização.

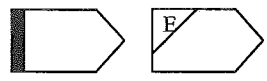
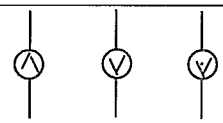
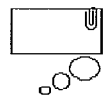
Atividade	 (a) (b)	(a) Adorno que indica que a atividade é composta, o que significa que ela pode ser decomposta em sub-atividades; (b) Adorno que indica que a atividade é externa, ou seja, que é executada por outra organização.
Operação Lógica	 (a) (b) (c)	(a) Adorno que indica a operação lógica E; (b) Adorno que indica a operação lógica OU; (c) Adorno que indica a operação lógica OU Exclusivo.
Conhecimento Explícito		Adorno que indica que foi especificado um caminho para acesso ao conhecimento disponível em meio magnético. Este adorno só deve ser utilizado se a visualização do modelo for apoiada por uma ferramenta de software que permita o acesso ao conhecimento. *Atributos Especiais:* Localização do Arquivo

Tabela C.2 – Definição e Notação dos Adornos

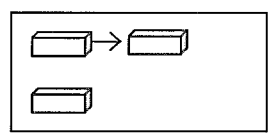
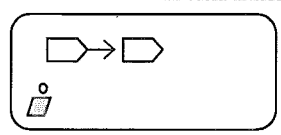
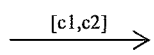
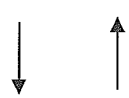
Objeto	Notação	Definição
Grupo de Processos		Área que agrupa processos relacionados.
Área de Ator		Área que agrupa atividades executadas por um ator ou grupo de atores. O ator ou o grupo de atores também precisa estar contido na área.

Tabela C.3 – Definição e Notação das Áreas

Objeto	Notação	Definição
Fluxo de Controle		Ligação que indica a passagem de controle do objeto origem para o objeto destino. O c1 e o c2 indicados na notação são os rótulos das condições estabelecidas para que a passagem de controle ocorra. *Atributo Especial:* Condição, formada por rótulo e descrição
Fluxo de Entrada/Saída		Ligação que estabelece um insumo (se o fluxo é de entrada) ou um produto de uma atividade (se o fluxo é de saída). Quando o objeto de origem ou destino é um armazenador (repositório ou arquivo), a notação pode incluir os rótulos das informações trafegadas, existindo, então, um atributo especial. *Atributo Especial:* Informação, formada por rótulo e descrição
Ligação Não Direcionada		Ligação que não indica passagem de controle nem estabelece insumos e produtos para uma atividade, sendo utilizada para conectar bens de produção (software, hardware e equipamentos) utilizados como recursos para execução das atividades e para conectar eventos que atuam sobre processos, provocando o seu início ou fim. No segundo caso, um atributo especial é definido. *Atributo Especial:* Papel do Evento (Iniciador, Terminador)

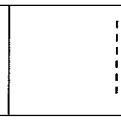
Ligação para Nota Explicativa		Ligação que estabelece que uma nota explicativa é referente a um elemento do modelo.

Tabela C.4 – Definição e Notação das Ligações

Anexo D – Artefatos Gerados e Utilizados pelo xMapper

Este anexo apresenta os artefatos gerados e utilizados durante o exemplo de uso da ferramenta xMapper apresentado no capítulo 4 – Requisitos para uma Infra-estrutura de Integração de Ferramentas com XML.

D.1 – Artefato de Origem

O documento XML apresentado neste apêndice representa o artefato gerado pela ferramenta de apoio à aplicação de PBR.

```
<?xml version="1.0"?>
<formDiscrepancy type="pbr">
    <perspective>USER</perspective>
    <project>ESE</project>
    <inspector>103137506</inspector>
    <start>7/9/2003 09:40:56</start>
    <artifact>C:\Program Files\PBRTool\SRS_Documents\ABC_Video_System.rtf</artifact>
    <discrepancyList>
        <discrepancy status="new">
            <classification>Informação Estranha</classification>
            <description>O cliente não tem interação com o sistema. Não precisa trazer o cartão pois o código
pode ser entrado pelo teclado. O cliente precisa saber seu código? O sistema pode auxiliar.</description>
            <identTime>7/9/2003 10:40:28</identTime>
            <step>Use Cases - Hints</step>
            <line>Características do Usuário</line>
        </discrepancy>
        <discrepancy status="updated">
            <classification>Omissão</classification>
            <description>Como calcular a taxa de aluguel? Que outras taxas existem?</description>
            <identTime>7/9/2003 10:44:44</identTime>
            <step>Participants - View Document</step>
            <line>300</line>
            <requirement>Functional Requirement 8</requirement>
        </discrepancy>
        <discrepancy status="updated">
            <classification>Omissão</classification>
            <description>Onde estão as funcionalidades do gerente no menu?</description>
            <identTime>7/9/2003 10:51:38</identTime>
            <step>Participants - View Document</step>
            <line>224</line>
            <requirement>Functional Requirement 1</requirement>
        </discrepancy>
        <discrepancy status="updated">
            <classification>Omissão</classification>
            <description>De que modo: teclado e/ou leitor de código de barra?</description>
            <identTime>7/9/2003 10:54:33</identTime>
            <step>Participants - View Document</step>
            <line>344</line>
```

```
            <requirement>Functional Requirement 12</requirement>
        </discrepancy>
        <discrepancy status="updated">
            <classification>Omissão</classification>
            <description>Como o cliente pode saber o total de atrasos daquelas fitas devolvidas?</description>
            <identTime>7/9/2003 11:16:38</identTime>
            <step>Participants - View Document</step>
            <line>364</line>
            <requirement>Functional Requirement 14</requirement>
        </discrepancy>
        <discrepancy status="updated">
            <classification>Omissão</classification>
            <description>Como verificar se o cliente é realmente novo? Poderia ser um cliente com pagamentos
pendentes querendo burlar o sistema. Solução : cadastrar por CPF e verificar na inclusão se CPF já
existe.</description>
            <identTime>7/9/2003 11:21:21</identTime>
            <step>Participants - View Document</step>
            <line>377</line>
            <requirement>Functional Requirement 15</requirement>
        </discrepancy>
        <discrepancy status="updated">
            <classification>Om!issão</classification>
            <description>Crítica dos dados de entrada: o que é obrigatótio e o que seria opcional?</description>
            <identTime>7/9/2003 11:24:34</identTime>
            <step>Participants - View Document</step>
            <line>378</line>
            <requirement>Functional Requirement 15</requirement>
        </discrepancy>
        <discrepancy status="new">
            <requirement>Functional Requirement 16</requirement>
            <classification>Omissão</classification>
            <description>Como seria o ID da fita?</description>
            <identTime>7/9/2003 11:31:51</identTime>
            <step>Participants - View Document</step>
            <line>391</line>
        </discrepancy>
        <discrepancy status="new">
            <requirement>Functional Requirement 18</requirement>
            <classification>Inconsistência</classification>
            <description>Deveria pedir neste momento senha de gerência para permitir
funcionalidade</description>
            <identTime>7/9/2003 11:44:36</identTime>
            <step>System Functionalities - View Document</step>
            <line>414</line>
        </discrepancy>
        <discrepancy status="new">
            <requirement>Functional Requirement 18</requirement>
            <classification>Omissão</classification>
            <description>Críticas omissas: o cliente não poderia estar com fita alugada e nem em
atraso.</description>
            <identTime>7/9/2003 11:45:42</identTime>
            <step>System Functionalities - View Document</step>
            <line>418</line>
        </discrepancy>
        <discrepancy status="new">
            <requirement>Functional Requirement 18</requirement>
            <classification>Omissão</classification>
            <description>Crítica: verificar se fita está alugada</description>
            <identTime>7/9/2003 11:48:01</identTime>
            <step>System Functionalities - View Document</step>
            <line>418</line>
        </discrepancy>
        <discrepancy status="new">
            <requirement>Functional Requirement 17</requirement>
            <classification>Fato Incorreto</classification>
            <description>Fita e cliente só poderiam ser realmente apagados do sistema no caso de nunca terem
sidos emprestado e feito empréstimo
respectivamente. Nestes casos deveria ser desabilitados (?) </description>
```

<identTime>7/9/2003 11:51:24</identTime>
<step>System Functionalities - View Document</step>
<line>417</line>
</discrepancy>
<discrepancy status="new">
<requirement>Functional Requirement 17</requirement>
<classification>Omissão</classification>
<description>Não diz nada sobre alteração de código.</description>
<identTime>7/9/2003 11:57:02</identTime>
<step>System Functionalities - View Document</step>
<line>402</line>
</discrepancy>
<discrepancy status="new">
<requirement>Functional Requirement 17</requirement>
<classification>Omissão</classification>
<description>Crítica de inexistência dos códigos com msg de erro.</description>
<identTime>7/9/2003 11:57:40</identTime>
<step>System Functionalities - View Document</step>
<line>402</line>
</discrepancy>
<discrepancy status="new">
<classification>Omissão</classification>
<description>Não esta definido o que seria o relatório diário do gerente (informações e formato)</description>
<identTime>7/9/2003 12:01:26</identTime>
<step>Participants - View Document</step>
<line>427</line>
</discrepancy>
<discrepancy status="new">
<classification>Inconsistência</classification>
<description>No cadastramento de cliente, não está armazenado a data de sua inclusão, como saber clientes novos no período?</description>
<identTime>7/9/2003 12:03:07</identTime>
<step>Participants - View Document</step>
<line>428</line>
</discrepancy>
<discrepancy status="new">
<classification>Omissão</classification>
<description>Definir mal cliente. De que período em atraso ?</description>
<identTime>7/9/2003 12:04:22</identTime>
<step>Participants - View Document</step>
<line>429</line>
</discrepancy>
<discrepancy status="new">
<classification>Omissão</classification>
<description>Quais são os status da fita?</description>
<identTime>7/9/2003 12:06:20</identTime>
<step>Participants - View Document</step>
<line>431</line>
</discrepancy>
<discrepancy status="new">
<requirement>Functional Requirement 19</requirement>
<classification>Inconsistência</classification>
<description>Nos dados da fita não estava definido: cópia, título .....</description>
<identTime>7/9/2003 12:07:39</identTime>
<step>Participants - View Document</step>
<line>433</line>
</discrepancy>
<discrepancy status="new">
<classification>Ambigüidade</classification>
<description>O que seria amigável?</description>
<identTime>7/9/2003 12:08:37</identTime>
<step>Participants - View Document</step>
<line>450</line>
</discrepancy>
<discrepancy status="new">
<classification>NÃO CLASSIFICÁVEL</classification>
<description>5 MINUTOS (?)</description>

```
            <identTime>7/9/2003 12:10:24</identTime>
            <step>Participants - View Document</step>
            <line>454</line>
        </discrepancy>
        <discrepancy status="new">
            <classification>Omissão</classification>
            <description>Back-up, nunca falado antes ?????</description>
            <identTime>7/9/2003 12:10:56</identTime>
            <step>Participants - View Document</step>
        </discrepancy>
    </discrepancyList>
    <PBR_help>0</PBR_help>
    <Tool_help>0</Tool_help>
    <end>7/9/2003 12:13:47</end>
    <totalTime>02:32:50</totalTime>
</formDiscrepancy>
```

D.2 – Ontologia

A ontologia apresentada abaixo na notação OWL descreve a semântica de um formulário de discrepância.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF xml:lang="en" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:map="http://www.rodrigospinola.com">
    <owl:Ontology rdf:about="http://www.cos.ufrj.br/~ros/OntoDiscReq">
        <rdfs:comment>Ontologia em OWL descrevendo a semântica de um relatório de
discrepância.</rdfs:comment>
        <owl:versionInfo>1.0</owl:versionInfo>
    </owl:Ontology>
    <owl:Class rdf:ID="form">
        <rdfs:comment>Um formulário de discrepância é um artefato que descreve as discrepâncias
encontradas por inspetores durante a inspeção de artefatos de software.</rdfs:comment>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#form_type"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#project"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#inspector"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#perspective"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#artifact"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#relationship"/>
```

```
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#totalTime"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
    </owl:Class>
    <owl:Class rdf:ID="discrepancyList">
        <rdfs:comment>Uma lista de discrepância é ...</rdfs:comment>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#include"/>
            <owl:minCardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">0</owl:minCardinality>
            <owl:maxCardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">unbounded
            </owl:maxCardinality>
        </owl:Restriction>
    </owl:Class>
    <owl:Class rdf:ID="discrepancy">
        <owl:Restriction>
            <owl:onProperty rdf:resource="#requirement"/>
            <owl:minCardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:minCardinality>
            <owl:maxCardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">unbounded
            </owl:maxCardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#classification"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#description"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
    </owl:Class>
    <owl:ObjectProperty rdf:ID="relationship">
        <rdfs:domain rdf:resource="#Form_Discrepancy"/>
        <rdfs:range rdf:resource="#discrepancyList"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="include">
        <rdfs:domain rdf:resource="#discrepancyList"/>
        <rdfs:range rdf:resource="#discrepancy"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="project">
        <rdfs:domain rdf:resource="#Form_Discrepancy"/>
        <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="inspector">
        <rdfs:comment>Inspector é o responsável por inspecionar um artefato ...</rdfs:comment>
        <rdfs:domain rdf:resource="#Form_Discrepancy"/>
        <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="perspective">
        <rdfs:domain rdf:resource="#Form_Discrepancy"/>
        <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="artifact">
        <rdfs:domain rdf:resource="#Form_Discrepancy"/>
        <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="totalTime">
        <rdfs:domain rdf:resource="#Form_Discrepancy"/>
        <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
```

```
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="requirement">
        <rdfs:domain rdf:resource="#discrepancy"/>
        <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="classification">
        <rdfs:domain rdf:resource="#discrepancy"/>
        <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="description">
        <rdfs:domain rdf:resource="#discrepancy"/>
        <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
    </owl:ObjectProperty>
</rdf:RDF>
```

D.3 – Esquemas

O esquema abaixo especifica a estrutura do artefato definido em XML utilizado pela ferramenta de apoio à aplicação de PBR.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="formDiscrepancy" type="form"/>
    <xs:complexType name="form">
        <xs:sequence>
            <xs:element name="perspective" type="xs:string"/>
            <xs:element name="project" type="xs:string"/>
            <xs:element name="inspector" type="xs:string"/>
            <xs:element name="start" type="xs:string"/>
            <xs:element name="artifact" type="xs:string"/>
            <xs:element name="discrepancyList" type="discrepancyListType"/>
            <xs:element name="PBRHelp" type="xs:integer"/>
            <xs:element name="ToolHelp" type="xs:integer"/>
            <xs:element name="end" type="xs:string"/>
            <xs:element name="totalTime" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="type" type="xs:string" default="pbr"/>
    </xs:complexType>
    <xs:complexType name="discrepancyListType">
        <xs:sequence>
            <xs:element name="discrepancy" type="discrepancyType" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="discrepancyType">
        <xs:sequence>
            <xs:element name="classification" type="xs:string"/>
            <xs:element name="description" type="xs:string"/>
            <xs:element name="identTime" type="xs:string"/>
            <xs:element name="step" type="xs:string"/>
            <xs:element name="line" type="xs:string"/>
            <xs:element name="requirement" type="xs:string" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="status" type="xs:string"/>
    </xs:complexType>
</xs:schema>
```

O esquema abaixo especifica a estrutura do artefato definido em XML utilizado pela ferramenta de apoio ao processo de inspeção ISPIS.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="formDiscrepancy" type="form"/>
    <xs:complexType name="form">
        <xs:sequence>
            <xs:element name="discrepancyList" type="discrepancyListType"/>
        </xs:sequence>
        <xs:attribute name="type" type="xs:string" default="pbr"/>
    </xs:complexType>
    <xs:complexType name="discrepancyListType">
        <xs:sequence>
            <xs:element name="inspectionStart" type="xs:string"/>
            <xs:element name="discrepancy" type="discrepancyType" minOccurs="0"
maxOccurs="unbounded"/>
            <xs:element name="inspectionEnd" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="type" type="xs:string" default="pbr"/>
    </xs:complexType>
    <xs:complexType name="discrepancyType">
        <xs:sequence>
            <xs:element name="location" type="xs:string" maxOccurs="unbounded"/>
            <xs:element name="classification" type="xs:string"/>
            <xs:element name="severity" type="xs:string"/>
            <xs:element name="description" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

D.4 – Mapa de Integração

O mapa de integração apresentado abaixo representa a ontologia apresentada no apêndice D.2 acrescida de marcações (apresentadas na seção 4.4.1.3.1) que descrevem informações estruturais provenientes dos esquemas apresentados no apêndice D.3.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF xml:lang="en" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:map="http://www.rodrigospinola.com">
    <owl:Ontology rdf:about="http://www.cos.ufrj.br/~ros/OntoDiscReq">
        <rdfs:comment>Ontologia em OWL descrevendo a semântica de um relatório de
discrepância.</rdfs:comment>
        <owl:versionInfo>1.0</owl:versionInfo>
    </owl:Ontology>
    <map:mappedFiles>
        <map:fileName fileDescription="ISPIS">C:\Tese\Exemplo\schema\schema_ISPIS.xsd</map:fileName>
        <map:fileName
fileDescription="PBRTool">C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd</map:fileName>
    </map:mappedFiles>
    <owl:Class rdf:ID="form">
        <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
            <map:schema>
                <map:attribute>type</map:attribute>
```

```
                <map:name>formDiscrepancy</map:name>
                <map:sequence>
                    <map:next moreOne="0">perspective</map:next>
                    <map:next moreOne="0">project</map:next>
                    <map:next moreOne="0">inspector</map:next>
                    <map:next moreOne="0">start</map:next>
                    <map:next moreOne="0">artifact</map:next>
                    <map:next moreOne="0">discrepancyList</map:next>
                    <map:next moreOne="0">PBRHelp</map:next>
                    <map:next moreOne="0">ToolHelp</map:next>
                    <map:next moreOne="0">end</map:next>
                    <map:next moreOne="0">totalTime</map:next>
                </map:sequence>
            </map:schema>
        </map:mapFile>
        <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_ISPIS.xsd">
            <map:schema>
                <map:attribute>type</map:attribute>
                <map:name>formDiscrepancy</map:name>
                <map:sequence>
                    <map:next moreOne="0">discrepancyList</map:next>
                </map:sequence>
            </map:schema>
        </map:mapFile>
        <rdfs:comment>Um formulário de discrepância é um artefato que descreve as discrepâncias
encontradas por inspetores durante a inspeção de artefatos de software.</rdfs:comment>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#form_type"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#project"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#inspector"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#artifact"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#relationship"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#totalTime"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
    </owl:Class>
    <owl:Class rdf:ID="discrepancyList">
        <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
            <map:schema>
                <map:name>discrepancyList</map:name>
                <map:sequence>
                    <map:next moreOne="1">discrepancy</map:next>
                </map:sequence>
            </map:schema>
        </map:mapFile>
        <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_ISPIS.xsd">
            <map:schema>
```

```
                <map:attribute>type</map:attribute>
                <map:name>discrepancyList</map:name>
                <map:sequence>
                    <map:next moreOne="0">start</map:next>
                    <map:next moreOne="1">discrepancy</map:next>
                    <map:next moreOne="0">end</map:next>
                </map:sequence>
            </map:schema>
        </map:mapFile>
        <rdfs:comment>Uma lista de discrepância é ...</rdfs:comment>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#include"/>
            <owl:minCardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">0</owl:minCardinality>
            <owl:maxCardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">unbounded
            </owl:maxCardinality>
        </owl:Restriction>
    </owl:Class>
    <owl:Class rdf:ID="discrepancy">
        <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
            <map:schema>
                <map:attribute>status</map:attribute>
                <map:name>discrepancy</map:name>
                <map:sequence>
                    <map:next moreOne="0">classification</map:next>
                    <map:next moreOne="0">description</map:next>
                    <map:next moreOne="0">identTime</map:next>
                    <map:next moreOne="0">step</map:next>
                    <map:next moreOne="0">line</map:next>
                    <map:next moreOne="1">requirement</map:next>
                </map:sequence>
            </map:schema>
        </map:mapFile>
        <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_ISPIS.xsd">
            <map:schema>
                <map:name>discrepancy</map:name>
                <map:sequence>
                    <map:next moreOne="1">requirement</map:next>
                    <map:next moreOne="0">classification</map:next>
                    <map:next moreOne="0">severity</map:next>
                    <map:next moreOne="0">description</map:next>
                </map:sequence>
            </map:schema>
        </map:mapFile>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#requirement"/>
            <owl:minCardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:minCardinality>
            <owl:maxCardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">unbounded
            </owl:maxCardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#classification"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#description"/>
            <owl:cardinality
rdf:datatype="http://www.w3.org/2000/10/XMLSchema#NonNegativeInteger">1</owl:cardinality>
        </owl:Restriction>
    </owl:Class>
    <owl:ObjectProperty rdf:ID="relationship">
        <rdfs:domain rdf:resource="#Form_Discrepancy"/>
        <rdfs:range rdf:resource="#discrepancyList"/>
    </owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="include">
    <rdfs:domain rdf:resource="#discrepancyList"/>
    <rdfs:range rdf:resource="#discrepancy"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="project">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>project</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#Form_Discrepancy"/>
    <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="inspector">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>inspector</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:comment>Inspetor é o responsável por inspecionar um artefato ...</rdfs:comment>
    <rdfs:domain rdf:resource="#Form_Discrepancy"/>
    <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="artifact">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>artifact</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#Form_Discrepancy"/>
    <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="totalTime">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>totalTime</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#Form_Discrepancy"/>
    <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="requirement">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>requirement</map:name>
            <map:name>line</map:name>
        </map:schema>
    </map:mapFile>
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_ISPIS.xsd">
        <map:schema>
            <map:name>location</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#discrepancy"/>
    <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="classification">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>classification</map:name>
        </map:schema>
    </map:mapFile>
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_ISPIS.xsd">
        <map:schema>
            <map:name>classification</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#discrepancy"/>
```

```
    <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="description">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>description</map:name>
        </map:schema>
    </map:mapFile>
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_ISPIS.xsd">
        <map:schema>
            <map:name>description</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#discrepancy"/>
    <rdfs:range rdf:resource="#http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="start">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>start</map:name>
        </map:schema>
    </map:mapFile>
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_ISPIS.xsd">
        <map:schema>
            <map:name>inspectionStart</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#form"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="end">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>end</map:name>
        </map:schema>
    </map:mapFile>
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_ISPIS.xsd">
        <map:schema>
            <map:name>inspectionEnd</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#form"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="severity">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_ISPIS.xsd">
        <map:schema>
            <map:name>severity</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#discrepancy"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="line">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>line</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#discrepancy"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="step">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
        <map:schema>
            <map:name>step</map:name>
        </map:schema>
    </map:mapFile>
    <rdfs:domain rdf:resource="#discrepancy"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="perspective">
    <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
```

```
            <map:schema>
                <map:name>perspective</map:name>
            </map:schema>
        </map:mapFile>
        <rdfs:domain rdf:resource="#form"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="identTime">
        <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
            <map:schema>
                <map:name>identTime</map:name>
            </map:schema>
        </map:mapFile>
        <rdfs:domain rdf:resource="#discrepancy"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="PBRHelp">
        <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
            <map:schema>
                <map:name>PBRHelp</map:name>
            </map:schema>
        </map:mapFile>
        <rdfs:domain rdf:resource="#form"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="ToolHelp">
        <map:mapFile map:id="C:\Tese\Exemplo\schema\schema_PBR_Tool.xsd">
            <map:schema>
                <map:name>ToolHelp</map:name>
            </map:schema>
        </map:mapFile>
        <rdfs:domain rdf:resource="#form"/>
    </owl:ObjectProperty>
</rdf:RDF>
```

D.5 – Conversor definido em XSLT

O conversor apresentado abaixo define em XSLT as regras para as transformações a serem efetuadas no artefato de origem mostrado no Apêndice D.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template name="form" match="formDiscrepancy">
        <formDiscrepancy>
            <xsl:attribute name="type">
                <xsl:value-of select="./@type"/>
            </xsl:attribute>
            <xsl:call-template name="discrepancyList"/>
        </formDiscrepancy>
    </xsl:template>
    <xsl:template name="discrepancyList" match="discrepancyList">
        <discrepancyList>
            <xsl:attribute name="type">
                <xsl:value-of select="./@type"/>
            </xsl:attribute>
            <xsl:call-template name="start"/>
            <xsl:for-each select="discrepancyList/discrepancy">
                <xsl:call-template name="discrepancy"/>
            </xsl:for-each>
            <xsl:call-template name="end"/>
      </discrepancyList>
    </xsl:template>
    <xsl:template name="start" match="start">
```

```
        <inspectionStart>
            <xsl:value-of select="start"/>
        </inspectionStart>
    </xsl:template>
    <xsl:template name="end" match="end">
        <inspectionEnd>
            <xsl:value-of select="end"/>
        </inspectionEnd>
    </xsl:template>
    <xsl:template name="discrepancy" match="discrepancy">
        <discrepancy>
            <xsl:for-each select="requirement">
                <xsl:call-template name="requirement"/>
            </xsl:for-each>
            <xsl:call-template name="classification"/>
            <xsl:call-template name="severity"/>
            <xsl:call-template name="description"/>
        </discrepancy>
    </xsl:template>
    <xsl:template name="requirement" match="requirement">
        <location>
            <xsl:value-of select="."/>
            <xsl:text> - line: </xsl:text>
            <xsl:value-of select="./../line"/>
        </location>
    </xsl:template>
    <xsl:template name="classification" match="classification">
        <classification>
            <xsl:value-of select="classification"/>
        </classification>
    </xsl:template>
    <xsl:template name="severity">
        <severity/>
    </xsl:template>
    <xsl:template name="description" match="description">
        <description>
            <xsl:value-of select="description"/>
        </description>
    </xsl:template>
</xsl:stylesheet>
```

D.6 – Artefato Alvo

O documento XML apresentado neste apêndice representa o artefato gerado a partir da aplicação das regras definidas no contemplado no Apêndice D.5 sobre o documento XML apresentado no Apêndice D.1. O artefato gerado é importado e utilizado pela ferramenta de apoio ao processo de inspeção ISPIS.

```
<?xml version="1.0" encoding="UTF-8"?>
<formDiscrepancy type="pbr">
    <discrepancyList type="pbr">
        <inspectionStart>7/9/2003 09:40:56</inspectionStart>
        <discrepancy>
            <classification>Informação Estranha</classification>
            <severity/>
            <description>O cliente não tem interação com o sistema. Não precisa trazer o cartão pois o código
pode ser entrado pelo teclado. O cliente precisa saber seu código? O sistema pode auxiliar.</description>
        </discrepancy>
        <discrepancy>
            <location>Functional Requirement 8 - line: 300</location>
```

<classification>Omissão</classification>
<severity/>
<description>Como calcular a taxa de aluguel? Que outras taxas existem?</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 1 - line: 224</location>
<classification>Omissão</classification>
<severity/>
<description>Onde estão as funcionalidades do gerente no menu?</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 12 - line: 344</location>
<classification>Omissão</classification>
<severity/>
<description>De que modo: teclado e/ou leitor de código de barra?</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 14 - line: 364</location>
<classification>Omissão</classification>
<severity/>
<description>Como o cliente pode saber o total de atrasos daquelas fitas devolvidas?</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 15 - line: 377</location>
<classification>Omissão</classification>
<severity/>
<description>Como verificar se o cliente é realmente novo? Poderia ser um cliente com pagamentos pendentes querendo burlar o sistema. Solução : cadastrar por CPF e verificar na inclusão se CPF já existe.</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 15 - line: 378</location>
<classification>Om!issão</classification>
<severity/>
<description>Crítica dos dados de entrada: o que é obrigatório e o que seria opcional?</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 16 - line: 391</location>
<classification>Omissão</classification>
<severity/>
<description>Como seria o ID da fita?</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 18 - line: 414</location>
<classification>Inconsistência</classification>
<severity/>
<description>Deveria pedir neste momento senha de gerência para permitir funcionalidade</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 18 - line: 418</location>
<classification>Omissão</classification>
<severity/>
<description>Críticas omissas: o cliente não poderia estar com fita alugada e nem em atraso.</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 18 - line: 418</location>
<classification>Omissão</classification>
<severity/>
<description>Crítica: verificar se fita está alugada</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 17 - line: 417</location>
<classification>Fato Incorreto</classification>
<severity/>
<description>Fita e cliente só poderiam ser realmente apagados do sistema no caso de nunca terem sidos emprestado e feito empréstimo

respectivamente. Nestes casos deveria ser desabilitados (?) </description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 17 - line: 402</location>
<classification>Omissão</classification>
<severity/>
<description>Não diz nada sobre alteração de código.</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 17 - line: 402</location>
<classification>Omissão</classification>
<severity/>
<description>Crítica de inexistência dos códigos com msg de erro.</description>
</discrepancy>
<discrepancy>
<classification>Omissão</classification>
<severity/>
<description>Não esta definido o que seria o relatório diário do gerente (informações e formato)</description>
</discrepancy>
<discrepancy>
<classification>Inconsistência</classification>
<severity/>
<description>No cadastramento de cliente, não está armazenado a data de sua inclusão, como saber clientes novos no período?</description>
</discrepancy>
<discrepancy>
<classification>Omissão</classification>
<severity/>
<description>Definir mal cliente. De que período em atraso ?</description>
</discrepancy>
<discrepancy>
<classification>Omissão</classification>
<severity/>
<description>Quais são os status da fita?</description>
</discrepancy>
<discrepancy>
<location>Functional Requirement 19 - line: 433</location>
<classification>Inconsistência</classification>
<severity/>
<description>Nos dados da fita não estava definido: cópia, título .....</description>
</discrepancy>
<discrepancy>
<classification>Ambigüidade</classification>
<severity/>
<description>O que seria amigável?</description>
</discrepancy>
<discrepancy>
<classification>NÃO CLASSIFICÁVEL</classification>
<severity/>
<description>5 MINUTOS (?)</description>
</discrepancy>
<discrepancy>
<classification>Omissão</classification>
<severity/>
<description>Back-up, nunca falado antes ?????</description>
</discrepancy>
<inspectionEnd>7/9/2003 12:13:47</inspectionEnd>
</discrepancyList>
</formDiscrepancy>