UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

COMPARAÇÃO DE ESQUEMAS XML NA EVOLUÇÃO DAS FONTES EM UM AMBIENTE DE INTEGRAÇÃO DE DADOS

TRABALHO DE GRADUAÇÃO

Aluno: Lucas Freire Melo (lfm2@cin.ufpe.br)
Orientador: Ana Carolina Salgado (acs@cin.ufpe.br)

26 de setembro de 2006

Resumo

O sistema de integração de dados Integra foi proposto por Bernadette Farias Lóscio em sua tese de doutorado pelo Centro de Informática da Universidade Federal de Pernambuco. Como todo sistema de integração de dados, o Integra tem por objetivo fornecer uma interface uniforme para fontes de dados distribuídas, heterogêneas e autônomas, de forma a responder a consultas que necessitam de dados integrados dessas diversas fontes.

O tratamento de fontes de dados autônomas pelo sistema de integração é uma tarefa complexa, pois tais fontes podem sofrer alterações em seus esquemas e é essencial que essas alterações sejam refletidas no restante do sistema, especialmente no esquema global e nos mapeamentos entre o esquema global e os esquemas das fontes. Essas alterações nos esquemas podem ser inserções, remoções ou alterações de seus elementos.

Dessa forma, nesse trabalho será desenvolvido um módulo de comparação de esquemas XML, que irá comparar o esquema antigo e o novo de uma mesma base de dados e retornar como resultado as diferenças existentes entre eles, de forma a permitir a evolução dos esquemas das fontes e do esquema global.

Palavras-chave: Integração de dados, evolução de esquemas, comparação de esquemas XML.

Agradecimentos

Algumas pessoas consideram que essa seção é apenas para agradecer às pessoas que ajudaram no processo de desenvolvimento deste mesmo trabalho.

Eu penso diferente. Eu penso que esse trabalho simboliza a conclusão do período de graduação e, portanto, nada mais justo do que agradecer a todos que, de alguma forma, me ajudaram durante a realização do meu curso.

Primeiramente, e acima de tudo, agradeço a Deus pela oportunidade que Ele me deu de vir para Recife fazer o curso de Computação e também por sempre me proteger e às pessoas que são importantes para mim.

Agradeço imensamente a meus pais, que fizeram tudo o que era possível — e, às vezes, o que parecia impossível — para que eu pudesse vir morar em Recife. Apesar de todas as dificuldades, eles nunca desistiram e sempre acreditaram que nós, juntos, íamos conseguir isso.

Agradeço também a minha irmãzinha, sempre me apoiando e me aconselhando quando eu mais precisava.

Agradeço a minha Vanessa por todos os momentos felizes que ela me deu — e continua dando — e por estar sempre ao meu lado. Além de tudo, ainda foi ela que me levou para fazer a IC junto com ela e foi essa IC que deu origem a esse trabalho.

Agradeço também a todos os meus familiares que, de uma forma ou de outra, sempre me deram força para vir para Recife.

Agradeço a todos os amigos e colegas de faculdade que eu fiz, alguns mais próximos, outros menos, mas todos legais. Em particular, eu agradeço a Guilherme e Victor, que me ajudaram muito mesmo nos meus primeiros meses em Recife (apesar do açúcar para baixar minha pressão!). E, é claro, não posso deixar de agradecer ao grande Cristiano por todas as caronas concedidas.

E, finalmente, agradeço a professora Carol pela oportunidade que ela me deu de fazer a IC e por me orientar nesse trabalho, e também a todos dos projetos GridVida e Integra, ótimas pessoas com quem aprendi bastante.

Bom, eu sei que, para a maioria dessas pessoas, esses agradecimentos vão simplesmente passar em branco, já que elas não vão ler esse trabalho. Mas, de qualquer forma, fica aqui o meu agradecimento a todas essas pessoas, por quem eu tenho tanto carinho e respeito. Obrigado por tudo e que Deus ilumine vocês.

Índice

Índice de Figuras

Índice de Quadros

1. Introdução

A integração de dados tem como principal objetivo fornecer ao usuário uma visão integrada de várias fontes de dados heterogêneas e, muitas vezes, distribuídas e autônomas. Quando se trata de um ambiente dinâmico, como a Web, por exemplo, o sistema de integração de dados precisa se preocupar com fatores cruciais, tais como: fontes com elevado grau de autonomia e altamente evolutivas, pouca representação dos metadados e poucas informações sobre as características das fontes de dados.

A principal vantagem de um sistema de integração de dados é que o usuário precisa apenas especificar o que ele deseja, sem se preocupar em como nem onde a resposta será obtida. Para isso, o usuário só precisa conhecer o esquema do sistema de integração, que representa uma combinação dos esquemas das fontes integradas. Esse esquema do sistema de integração é chamado de esquema global ou esquema de mediação. É o esquema global que provê ao usuário uma interface uniforme para a submissão de suas consultas.

Como normalmente as fontes de dados de um sistema de integração são autônomas, elas podem modificar seus esquemas sem notificar ao sistema de integração. Quando essas modificações acontecem, elas precisam ser propagadas para o sistema, especificamente para o esquema global e suas assertivas de correspondência (forma como o sistema mapeia o esquema global nos esquemas das fontes).

Sendo assim, o sistema de integração precisa constantemente verificar se houve alguma evolução nos esquemas das fontes. Para isso, ele precisa comparar, para cada base de dados, o esquema que o sistema tem armazenado (considerado como o esquema antigo) e o esquema que ele obteve diretamente da fonte (o esquema novo).

Portanto, esse trabalho irá desenvolver um Comparador de Esquemas de Fontes de Dados, que irá comparar o esquema antigo e o novo de uma mesma base de dados e apresentar como resultado as diferenças existentes entre eles, de forma a permitir a evolução dos esquemas das fontes e, por conseguinte, do esquema global.

O restante deste trabalho está organizado da seguinte forma:

- O Capítulo 2 apresenta os principais fundamentos que é preciso saber para entender a proposta deste trabalho. Por isso, será dada uma breve introdução sobre XML, XML Schema e sobre o sistema de integração de dados Integra;
- O Capítulo 3 descreve os conceitos clássicos relacionados à questão de *Matching* de esquemas, assim como uma classificação dos sistemas de *Matching* e também as principais características de três desses sistemas;
- O Capítulo 4 expõe a solução proposta nesse trabalho para resolver o problema de comparação de esquemas XML na evolução das fontes de dados no sistema Integra;
- O Capítulo 5 mostra as conclusões obtidas com esse trabalho e também algumas sugestões para trabalhos futuros.

2. Fundamentos

Esse capítulo apresenta os fundamentos necessários para que se possa compreender os conteúdos que serão vistos nos capítulos seguintes. Tais fundamentos são os seguintes:

- **XML**: linguagem de marcadores utilizada mundialmente e adotada como padrão interno pelo sistema Integra;
- **XML Schema**: linguagem XML criada para validar um documento XML e usada pelo sistema Integra como base para a definição do modelo conceitual X-Entity;
- **Integra**: sistema de integração de dados que oferece uma visão única dos dados distribuídos em diversas fontes de dados autônomas e heterogêneas.

Cada um desses fundamentos será descrito em maiores detalhes nas seções a seguir.

2.1 XML

Desenvolvido pela W3C (*World Wide Web Consortium*), o XML (*eXtensible Markup Language*) [WWWC 2004a] é uma linguagem de marcadores de propósito geral, criada para descrever diferentes tipos de dados e servir como padrão para representação e troca de dados entre diferentes sistemas, especialmente sistemas conectados via Internet.

A sintaxe básica de um elemento XML é:

```
<nome atributo="valor">conteúdo</nome>
```

Um elemento consiste de duas *tags*: a *tag* inicial (por exemplo, <aluno>) e a *tag* final (por exemplo, </aluno>). Entre as *tags*, encontra-se o conteúdo do elemento, que pode tanto ser um texto como outros elementos aninhados. Além disso, um elemento pode conter atributos, que são pares nome-valor incluídos dentro da *tag* inicial, após o nome do elemento [Wikipedia 2006]. O documento XML a seguir ilustra esses conceitos:

```
<pessoa sexo="masculino">
  <nome>Ronaldinho</nome>
  <sobrenome>Gaúcho</sobrenome>
  <email>ronaldinhogaucho@gmail.com</email>
</pessoa>
```

Nesse exemplo, "pessoa", "nome", "sobrenome" e "email" são elementos. O elemento "pessoa" é o único que contém um atributo, no caso "sexo", e outros elementos aninhados. Enquanto isso, os elementos "nome", "sobrenome" e "email" possuem como conteúdo apenas texto.

Para um documento XML ser considerado correto, ele precisa ser bem-formado e válido. Um documento XML é bem-formado se ele está de acordo com as regras de sintaxe descritas na especificação XML [WWWC 2004a]. Diz-se que um documento XML é válido se seus dados satisfazem um conjunto de regras definidas por um esquema XML. Esse esquema pode ser representado por diversas linguagens, entre elas: DTD (*Document Type Definition*) [WWWC 1999a], XML Schema [WWWC 2004b] e RELAX NG [OASIS 2001].

2.2 XML Schema

A W3C criou o *XML Schema Definition Language* (ou apenas XML Schema) [WWWC 2004b] como uma linguagem XML para descrever, validar e restringir um documento XML. Um documento XML cujo conteúdo estiver de acordo com um XML Schema é chamado de "documento-instância".

Em um XML Schema, define-se a estrutura dos documentos-instância, especificando: os elementos que podem aparecer, os atributos que podem estar associados com esses elementos, quais elementos estão aninhados com outros (nesse caso, diz-se que um elemento é filho de outro elemento), a ordem em que os elementos filhos podem aparecer, entre outros [TheScarms 2000].

Como foi dito, XML Schema define os elementos que estarão no documento-instância. Tais elementos podem ser simples ou complexos. Elementos simples são aqueles que contêm apenas texto, ou seja, não contêm atributos nem outros elementos aninhados. Já os elementos complexos podem conter, além do texto, atributos e elementos aninhados [W3Schools 2006].

Um elemento simples pode ser definido usando o elemento "xs:element", de acordo com a seguinte sintaxe:

```
<xs:element name="zzz" type="yyy"/>
```

onde "zzz" é o nome do elemento e "yyy" é o tipo de dado do elemento. Esse tipo de dado pode ser um tipo definido no próprio esquema (como será visto mais adiante) ou um dos muitos tipos básicos definidos pelo XML Schema, como [Tech-Know-Ware 2006]:

- "xs:boolean": um valor booleano (verdadeiro ou falso);
- "xs:string": uma seqüência de caracteres;
- "xs:int": um número de 32 bits;
- "xs:time": uma hora;
- "xs:date": uma data;
- entre outros.

Em alguns casos, deseja-se que um elemento possa aparecer várias vezes dentro do documento XML. Isso pode ser feito através dos atributos "minOccurs" e "maxOcurrs" dentro do elemento "xs:element". O atributo "minOccurs" especifica o número mínimo de vezes que um elemento pode

aparecer e pode receber qualquer valor inteiro não-negativo (0, 1, 2, 3, ...). Já o atributo "maxOccurs" especifica o número máximo de vezes que um elemento pode aparecer e também pode receber qualquer valor inteiro não-negativo ou o valor constante "unbounded", representando um valor ilimitado. Por exemplo, a declaração a seguir especifica que o elemento deve aparecer pelo menos 2 vezes e no máximo 5 vezes:

```
<xs:element name="MeuElemento" type="tipo"
       minOccurs="2" maxOccurs="5"/>
```

Para se definir um atributo, usa-se o elemento "xs:attribute" da seguinte forma:

```
<xs:attribute name="zzz" type="yyy"
      use="definição de uso"/>
```

onde "zzz" é o nome do atributo e "yyy" é o tipo de dado do atributo. O atributo "use" é usado para especificar se o atributo é opcional ("optional"), obrigatório ("required") ou proibido ("prohibited").

Em alguns casos, é preciso impor restrições aos valores que um elemento simples ou um atributo podem ter. Essas restrições podem ser impostas com a declaração de um tipo simples dentro do esquema, através do elemento "xs:simpleType". Suponha que o conteúdo de um determinado elemento é do tipo inteiro, mas é preciso que o dado valor esteja entre 1 e 100. Isso pode ser feito da seguinte maneira:

```
<xs:element name="zzz">
  <xs:simpleType>
    <xs:restriction base="xs:int">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Como foi dito anteriormente, os elementos complexos são aqueles que podem conter atributos e outros elementos aninhados, além do próprio texto. Um elemento complexo é definido associando o elemento a um tipo complexo. Esse tipo complexo pode ser criado a partir do elemento "xs:complexType". Veja o seguinte exemplo:

```
<xs:element name="xxx" type="MeuTipo"/>

<xs:complexType name="MeuTipo">
  <xs:simpleContent>
    <xs:extension base="xs:int">
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Nesse exemplo, o elemento "xxx" é associado ao tipo "MeuTipo", declarado em seguida, cujo conteúdo é do tipo "xs:int" (especificado pelo atributo "base" do elemento "xs:extension").

É possível também criar tipos complexos cujos elementos contenham outros elementos aninhados: os elementos filhos. Veja o exemplo a seguir:

```
<xs:complexType name="MeuTipo2">
  <xs:sequence>
    <xs:element name="Filho1" type="xs:int"/>
    <xs:element name="Filho2" type="xs:string"/>
    <xs:element name="Filho3" type="xs:boolean"/>
  </xs:sequence>
  <xs:attribute name="Atributo1" type="xs:time"/>
  <xs:attribute name="Atributo2" type="xs:float"/>
</xs:complexType>
```

Nesse exemplo, o tipo complexo "MeuTipo2" é definido com três elementos filhos e dois atributos. O indicador "xs:sequence" especifica que os elementos filhos podem aparecer todos juntos em uma dada instância do tipo.

Talvez seja necessário que os elementos que são filhos não apareçam juntos, mas sim apenas um em cada instância do tipo. Para isso, existe o indicador "xs:choice", que pode ser usado como no exemplo abaixo:

```
<xs:complexType name="MeuTipo3">
  <xs:choice>
    <xs:element name="Filho1" type="xs:int"/>
    <xs:element name="Filho2" type="xs:string"/>
    <xs:element name="Filho3" type="xs:boolean"/>
  </xs:choice>
</xs:choice>
```

Nesse exemplo, o elemento que for associado ao tipo "MeuTipo3" terá apenas um elemento filho, cujo nome deve ser "Filho1", "Filho2" ou "Filho3".

XML Schema apresenta também restrições de unicidade, ou seja, permite que elementos e atributos sejam declarados unicamente em um determinado escopo e não recebam valores nulos. Uma das formas de se conseguir isso é através do elemento "xs:unique". Considere um esquema em que há um elemento "livro", dentro dele um elemento "personagem" e dentro desse um elemento "nome". Inserindo-se a declaração "xs:unique" dentro do elemento "livro", pode-se especificar que o nome do personagem tem que ser único dentro do contexto de cada livro [XML.com 2001], como mostra o exemplo a seguir:

```
<xs:unique name="nomePersonagem">
  <xs:selector xpath="livro/personagem">
  <xs:field xpath="nome">
</xs:unique>
```

Os dois elementos filhos de "xs:unique" definem a restrição de unicidade relativa ao contexto em que se encontra o elemento "xs:unique". Os valores desses dois elementos são dados na linguagem XPath [WWWC 1999b] (linguagem XML usada para navegar em documentos XML). O elemento "xs:selector" especifica o escopo em que a restrição de unicidade deve ser válida. Já o elemento "xs:field" especifica o elemento ou atributo que deve conter a unicidade, ou seja, que deve ser único dentro do escopo especificado pelo elemento "xs:selector".

A outra forma de se impor restrições de unicidade é através do elemento "xs:key", que é semelhante ao "xs:unique", exceto que o valor, além

de único, não pode ser nulo. Veja o seguinte exemplo, semelhante ao exemplo anterior:

```
<xs:key name="nomePersonagem">
  <xs:selector xpath="livro/personagem">
  <xs:field xpath="nome">
</xs:key>
```

A diferença desse exemplo para o anterior é que o nome do personagem tem que ser único dentro do contexto de cada livro e não pode receber um valor nulo.

É possível fazer referência a um elemento "xs:unique" ou "xs:key" através do elemento "xs:keyref". Ainda de acordo com o esquema do livro mostrado acima, considere que há um elemento "nomePersonagemPrincipal" dentro do elemento "livro". Colocando-se a declaração a seguir no mesmo nível em que foi definida a restrição de unicidade acima, indica-se que o nome do personagem principal precisa se referir ao nome de algum personagem do mesmo livro:

```
<xs:keyref name="nomePersRef" refer="nomePersonagem">
  <xs:selector xpath="livro">
  <xs:field xpath="nomePersonagemPrincipal">
</xs:key>
```

2.3 Sistema Integra

O sistema de integração de dados Integra foi proposto por Bernadette Farias Lóscio [Lóscio 2003] em sua tese de doutorado pelo Centro de Informática da Universidade Federal de Pernambuco. Como todo sistema de integração de dados, o Integra tem por objetivo fornecer uma interface uniforme para fontes de dados distribuídas, heterogêneas e autônomas, de forma a responder a consultas que necessitam de dados integrados dessas várias fontes. O maior diferencial do Integra em relação aos outros sistemas de integração é o fato de que ele apresenta soluções para a geração de

consultas de mediação (consulta que representa um conjunto de relacionamentos entre o esquema de mediação e os esquemas das fontes) e a manutenção do esquema de mediação em ambientes dinâmicos, como a Web [Costa 2005].

A arquitetura do Integra é baseada em uma abordagem híbrida, unindo um mediador e um *data warehouse*, de forma a dar suporte a consultas virtuais e materializadas, respectivamente [Batista 2003]. Assim, o Integra é capaz de fornecer informações integradas sempre atualizadas (uma vantagem dos mediadores) e, ao mesmo tempo, aumenta a disponibilidade dos dados e a velocidade nas respostas às consultas (vantagens do *data warehouse*). Além disso, ele faz uso de uma estrutura de *cache* para armazenar os resultados das consultas submetidas mais freqüentemente ao sistema.

2.3.1 Arquitetura

A arquitetura geral do sistema Integra é mostrada na Figura 2.1:

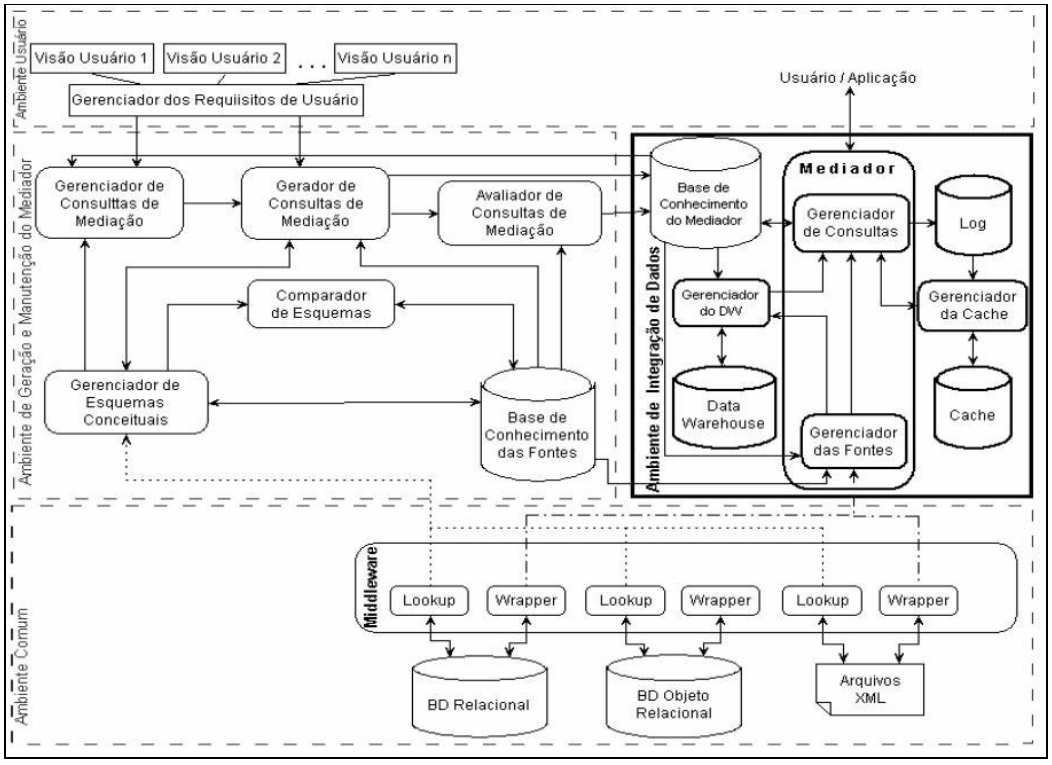

Figura 2.1 - Arquitetura do sistema Integra [Costa 2005]

Como pode ser visto na Figura 2.1, a arquitetura do Integra está dividida em quatro ambientes: Ambiente Comum, Ambiente de Integração de Dados, Ambiente de Geração e Manutenção do Mediador e Ambiente do Usuário.

Ambiente Comum

O Ambiente Comum fornece ao sistema de integração informações sobre os esquemas das fontes de dados, recebe as subconsultas direcionadas para as fontes e retorna as respostas de cada uma delas. Esse ambiente é composto pelas fontes de dados e pelo *Middleware*.

As fontes de dados são as que terão seus dados disponibilizados para o sistema de integração. Elas podem ser heterogêneas (de diversos tipos), dinâmicas (em que os esquemas sofrem modificações freqüentes) e autônomas (que funcionam independentemente do sistema de integração). As fontes podem ser adicionadas ao Ambiente Comum, removidas do ambiente ou sofrer alterações em seu esquema. Em qualquer um desses casos, as camadas superiores do sistema precisam ser notificadas das mudanças ocorridas.

O *Middleware* é uma camada intermediária entre as fontes de dados e o sistema de integração e é responsável por traduzir consultas e dados e extrair os esquemas das fontes. Essas tarefas são realizadas pelos módulos *Lookup* e *Wrapper*.

- Módulo *Lookup*: extrai o esquema de uma dada fonte de dados, mantendo-o atualizado na Base de Conhecimento das Fontes, e traduz esse esquema para o formato comum do sistema (no caso, XML Schema);
- Módulo *Wrapper*: recebe uma subconsulta e a traduz da linguagem usada no sistema de integração para a linguagem nativa da fonte de dados; ele também traduz os dados retornados pela fonte para o padrão XML, o formato comum do sistema.

Ambiente de Integração de Dados

Esse ambiente recebe as consultas dos usuários, verifica se os resultados estão na *cache*, faz a decomposição da consulta original, submete as subconsultas para as fontes e/ou para o *data warehouse* e integra os resultados obtidos. Seus módulos serão detalhados a seguir.

O Mediador é o principal módulo do Ambiente de Integração de Dados. Ele se decompõe nos seguintes submódulos:

- Gerenciador de Consultas: é o responsável por identificar onde estão armazenados os dados relevantes para responder a uma consulta de usuário (usando, para isso, a Base de Conhecimento do Mediador), por decompor a consulta original, identificar as subconsultas (uma para cada fonte relevante) e integrar os resultados das fontes;
- Gerenciador das Fontes: interage com os *Wrappers*, enviando as subconsultas e recebendo os resultados, e também monitora as fontes para selecionar os dados mais adequados a serem materializados no *data warehouse*.

O Gerenciador da Cache retorna resultados de consultas armazenadas na *cache* e efetua a manutenção da *cache*, ou seja, analisa o espaço disponível, as políticas de substituição das consultas armazenadas e a atualização do conteúdo.

O Gerenciador do *Data Warehouse* tem como principal atividade a manutenção do *data warehouse* e a análise de critérios de materialização dos dados, de forma a mantê-los atualizados.

Por último, a Base de Conhecimento do Mediador armazena o esquema de mediação e as assertivas de correspondência [Lóscio 2003].

Ambiente de Geração e Manutenção do Mediador

O Ambiente de Geração e Manutenção do Mediador tem como principais objetivos a geração e a manutenção do mediador e das consultas de

mediação. Os módulos que fazem parte desse ambiente serão explicados a seguir.

A Base de Conhecimento das Fontes armazena os esquemas exportados pelas fontes de dados e as correspondências entre os esquemas.

O módulo Gerenciador de Esquemas Conceituais é o principal módulo para esse trabalho, visto que nele se faz a comparação dos esquemas das fontes para identificar a evolução de um esquema. Mais detalhes sobre a comparação serão dados no Capítulo 4. Esse módulo extrai, através do *Lookup*, o esquema atual (em XML Schema) da fonte sendo analisada; em seguida, traduz esse esquema para X-Entity (modelo conceitual que será explicado mais adiante); consulta a Base de Conhecimento das Fontes para obter o esquema antigo (em X-Entity) da fonte; compara os dois esquemas e retorna o resultado da comparação.

O Comparador de Esquemas faz uma análise sintática e semântica para comparar dois esquemas de fontes diferentes e verificar quais elementos de um esquema são semanticamente correspondentes aos elementos do outro esquema.

O Gerador de Consultas de Mediação define o conjunto de consultas de mediação, em um processo que envolve três etapas: (i) seleção das fontes relevantes para a entidade de mediação; (ii) identificação de possíveis operadores para aplicar entre as fontes; e (iii) geração de todas as consultas de mediação possíveis com o conjunto de fontes e o conjunto de operadores [Lóscio 2003].

O módulo Gerenciador de Consultas de Mediação propaga no esquema de mediação as adições e remoções de fontes, as modificações nos esquemas das fontes e a evolução dos requisitos dos usuários.

Por fim, o Avaliador de Consultas de Mediação avalia como as mudanças ocorridas nos requisitos dos usuários e nos esquemas das fontes podem afetar a qualidade do esquema de mediação. Ele também avalia o impacto da propagação dessas mudanças para o sistema de integração. Esse

impacto é avaliado através de métricas como a disponibilidade das fontes e o seu tempo de resposta.

Ambiente do Usuário

Nesse ambiente são tratadas as questões que se referem às especificações de requisitos do usuário. Modificações nesses requisitos são propagadas para o esquema de mediação. Ele é composto pelos módulos:

- Visão do Usuário: define as visões dos diversos usuários;
- Gerenciador dos Requisitos do Usuário: permite ao usuário ver as informações que as fontes podem oferecer e definir um esquema de mediação de acordo com os seus requisitos.

2.3.2 X-Entity

O X-Entity é um modelo conceitual, proposto por Bernadette Farias Lóscio [Lóscio 2003], com o objetivo de ser uma extensão do modelo ER [Chen 1976] para esquemas XML, de forma a fornecer um maior nível de abstração para as informações contidas em um documento XML Schema. No modelo X-Entity, existem três estruturas principais: entidades, relacionamentos do tipo *contém* e relacionamentos do tipo *referencia* [Costa 2005].

- **Entidades**: o conceito de entidades é o mais importante no modelo X-Entity, visto que as entidades representam os elementos XML, que por sua vez são compostos por atributos e outros elementos (chamados de subelementos). Uma entidade é denotada por $E(\{A_1, ..., A_n\}, \{R_1, ..., R_m\})$, onde E é o nome da entidade, $\{A_1, ..., A_n\}$ é o conjunto de atributos e subelementos (desde que não sejam compostos por outros elementos ou atributos), e $\{R_1, ..., R_m\}$ é o conjunto de relacionamentos;
- **Relacionamentos do tipo *contém***: um relacionamento do tipo *contém* entre duas entidades E_1 e E_2 especifica que cada instância de E_1 contém instâncias de E_2. Um relacionamento desse tipo é denotado por $R(E_1, E_2, (min, max))$, onde R é o nome do relacionamento e (min, max) define

os números mínimo e máximo de instâncias de E_2 que podem ser associadas a uma instância de E_1;

- **Relacionamento do tipo *referencia***: um relacionamento do tipo *referencia* entre duas entidades E_1 e E_2 especifica que a entidade E_1 referencia a entidade E_2. Esse tipo de relacionamento é denotado por $R(E_1, E_2, \{A_{11}, ..., A_{1n}\}, \{A_{21}, ..., A_{2n}\})$, onde R é o nome do relacionamento e $\{A_{11}, ..., A_{1n}\}$ e $\{A_{21}, ..., A_{2n}\}$ representam os atributos de referência entre as entidades E_1 e E_2, tal que o valor de A_{1i} em qualquer entidade E_1 deve ter o mesmo valor de A_{2i} em qualquer entidade E_2, $1 \leq i \leq n$.

A Figura 2.2 ilustra um exemplo de um diagrama X-Entity, onde a entidade "Livro" *contém* uma ou mais entidades "Capítulo" e *referencia* a entidade "Editora".

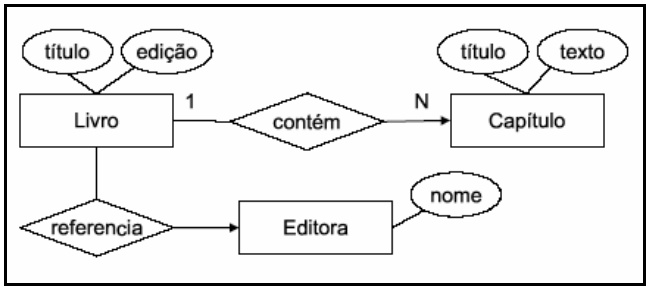

Figura 2.2 – Exemplo de um diagrama X-Entity [Costa 2005]

2.4 Considerações Finais

Neste capítulo foram abordados os principais fundamentos que irão servir de base para o restante deste trabalho: o padrão XML (linguagem utilizada como padrão interno pelo sistema Integra), a linguagem XML Schema (usada pelo Integra como base para a definição do modelo conceitual X-Entity) e o Integra (sistema de integração de dados para o qual será desenvolvido o Comparador de Esquemas de Fontes de Dados, de modo a permitir a evolução dos esquemas das fontes).

3. O Problema do *Matching* de Esquemas

Matching é a tarefa de comparar dois esquemas e retornar um mapeamento que identifique os elementos que são semanticamente correspondentes nos dois esquemas. Essa tarefa costumava ser feita manualmente, geralmente com o auxílio de uma ferramenta gráfica. Porém, quanto maiores ficavam os esquemas, mais caro, lento e propenso a erros ficava o processo de *matching*. Por ser uma atividade importante em diversas aplicações, como integração de dados, *data warehouse*, evolução de esquemas, design de banco de dados, comércio eletrônico e tantas outras, começaram a surgir diferentes propostas de sistemas para automatizar o *matching* de esquemas, fazendo com que todo o processo pudesse ser feito de forma mais rápida, simples e menos trabalhosa. Chama-se a esses sistemas de *matchers* [Do 2001].

3.1 Visão Geral

Suponha que uma empresa de comércio eletrônico adquiriu uma outra empresa e precisa integrar os bancos de dados das duas companhias. Os esquemas desses bancos são os esquemas XML S1 e S2 da Figura 3.1. O primeiro passo no processo de integração é a identificação, nos dois esquemas, dos elementos que são correspondentes, ou seja, é preciso fazer um *matching* desses dois esquemas. Por exemplo, os elementos "Escritório" nos dois esquemas são candidatos a serem relacionados entre si, assim como os elementos "NKN" e "Nikon" [Shvaiko 2005].

Figura 3.1 - Esquemas XML S1 e S2 [Shvaiko 2005]

Um esquema pode ser definido como um conjunto de elementos conectados por alguma estrutura. Existem diversas formas de se representar um esquema, como: o modelo entidade-relacionamento (ER), o modelo orientado a objetos (OO), XML ou grafos [Rahm 2001].

O mapeamento, que é o resultado do *matching*, é um conjunto de elementos de mapeamento, onde cada um desses indica quais elementos do esquema S1 são mapeados em quais elementos do esquema S2. Além disso, cada elemento de mapeamento contém uma expressão de mapeamento que especifica como os elementos de S1 e S2 estão relacionados. Uma expressão de mapeamento pode ser: uma relação entre escalares (por exemplo, '=', '<', '>'), uma função (por exemplo, adição ou concatenação), um relacionamento entre conjuntos (por exemplo, contém ou está-contido), ou qualquer outra forma de expressão [Rahm 2001].

Por exemplo, veja o Quadro 3.1, que mostra dois esquemas E1 e E2. Alguns elementos de mapeamento entre E1 e E2 poderiam ser "Cli.C# = Cliente.CliID" e "Concatenar(Cli.PrimeiroNome, Cli.ÚltimoNome) = Cliente.Contato".

E1	E2
Cli	Cliente
C#	CliID
CNome	Empresa
PrimeiroNome	Contato
ÚltimoNome	Telefone

Quadro 3.1 - Esquemas E1 e E2 [Rahm 2001]

Porém, nem sempre se consegue capturar a semântica dos dados através apenas dos esquemas, pois o *matching* de esquemas é inerentemente subjetivo. Por isso, outras entradas, além dos esquemas, são importantes para se conseguir um melhor resultado. Essas entradas poderiam ser um mapeamento inicial, um dicionário de sinônimos, uma lista de mapeamentos conhecidos, entre outras.

3.1.1 Classificação

Rahm [Rahm 2001] classifica as diversas propostas de *matching* de esquemas de acordo com diferentes critérios. Primeiro, o autor divide as propostas de acordo com a quantidade de algoritmos envolvidos: os *matchers* individuais, que utilizam um único algoritmo e um único critério para realizar o *matching*; e os *matchers* combinados, que utilizam vários algoritmos. Os *matchers* combinados se dividem em dois subgrupos: os híbridos, que utilizam múltiplos critérios de avaliação (por exemplo, o nome dos elementos e seu tipo); e os compostos, que aplicam vários algoritmos independentes sobre os dois esquemas e combinam os resultados.

Quanto aos *matchers* individuais, os critérios considerados são os seguintes:

- Baseado em instância ou esquema: os *matchers* baseados em esquema consideram apenas informações dos esquemas (como nomes,

descrições, relacionamentos, restrições), enquanto os baseados em instância levam em consideração também o conteúdo dos dados e estatísticas sobre eles;

- Granularidade no nível dos elementos ou da estrutura: técnicas no nível de elemento analisam os elementos isoladamente, enquanto as técnicas no nível de estrutura analisam como os elementos aparecem no esquema e como eles se relacionam entre si;
- Baseado em lingüística ou em restrição: os *matchers* baseados em lingüística utilizam nomes e descrições textuais dos elementos (como sinônimos, hiperônimos, abreviações), enquanto os baseados em restrições utilizam as restrições dos esquemas (como chaves, tipos de dados, cardinalidade, unicidade);
- Cardinalidade do resultado: o *matcher* pode produzir como resultado mapeamentos de um ou vários elementos de um esquema para um ou vários elementos do outro esquema, ou seja, a cardinalidade do resultado pode ser de um para um, um para vários, vários para um ou vários para vários;
- Informações auxiliares: como foi dito, alguns *matchers* utilizam não só os esquemas como entrada, mas também outras informações, tais como dicionários, mapeamentos antigos e entradas do usuário.

3.2 Trabalhos Relacionados

Dentre as diversas propostas feitas para realizar a tarefa de *matching* de esquemas, algumas delas foram feitas para resolver problemas específicos, enquanto outras foram desenvolvidas para ser um componente independente. Neste segundo caso, diz-se que tais propostas são genéricas, pois podem ser aplicadas em diferentes modelos de representação de esquemas e diferentes domínios de aplicação. Em meio a tantas propostas genéricas que existem, algumas delas se destacam por serem mais conhecidas e apresentarem

melhores resultados [Yatskevich 2003]. Sendo assim, os sistemas de *matching* COMA [Do 2001], Cupid [Madhavan 2001] e Similarity Flooding [Melnik 2002] serão descritos a seguir.

3.2.1 COMA

O sistema COMA é um *matcher* composto, ou seja, ele se propõe a combinar, de forma flexível, diversos algoritmos de *matching*. O COMA é um sistema genérico que suporta diferentes tipos de esquemas, como XML e esquemas relacionais, e provê uma vasta biblioteca de algoritmos de *matching*. Além disso, o COMA permite a inclusão de novos algoritmos de *matching* e suporta diferentes formas de combinar os resultados de cada algoritmo. Uma inovação desse sistema é que ele permite o reuso de resultados de *matching* obtidos anteriormente, visto que muitos esquemas que serão comparados são bastante similares a esquemas comparados antes. O principal foco do COMA está na geração de resultados com cardinalidade de um para um.

Ao receber os esquemas a serem mapeados, o COMA transforma-os para um formato interno, no qual os algoritmos irão operar. Esse formato interno é um grafo acíclico direcionado, com um elemento sendo a raiz. Os elementos do esquema são representados pelos nós do grafo, enquanto as arestas que conectam os nós representam os relacionamentos entre os elementos. A Figura 3.2 mostra dois esquemas, um relacional e o outro XML, e a representação interna de cada um.

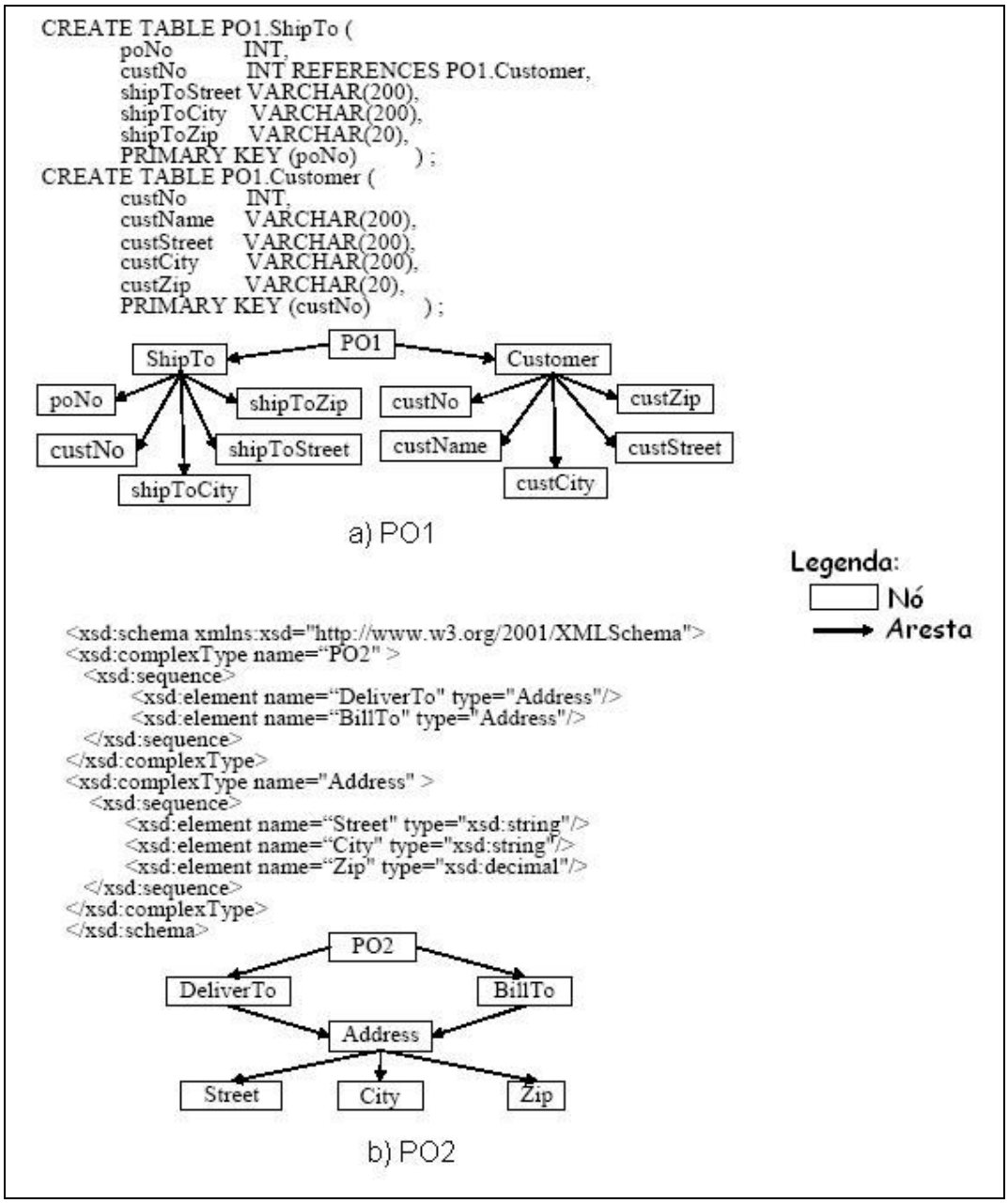

Figura 3.2 - Esquemas PO1 e PO2 [Do 2001]

O processamento dos algoritmos no COMA pode acontecer de duas formas: automática ou interativa. No modo automático, o processamento consiste de uma única iteração, na qual é aplicada a estratégia (ou seja, a escolha dos algoritmos e a forma de combinar os resultados) definida nos parâmetros de entrada. No modo interativo, o usuário interage com o sistema a cada iteração, podendo modificar a estratégia escolhida e aceitar ou rejeitar

elementos de mapeamento propostos na iteração anterior. A Figura 3.3 ilustra o processamento do *matching* realizado pelo COMA.

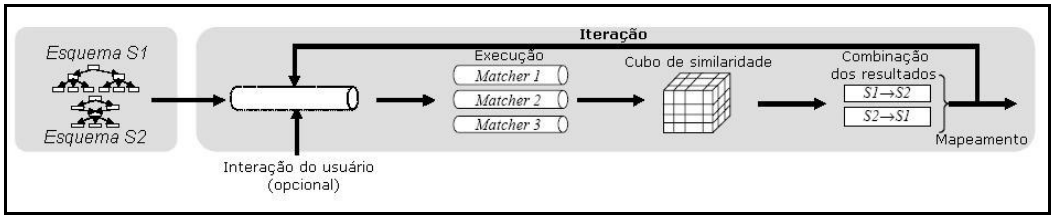

Figura 3.3 - Processamento do COMA [Do 2001]

O processamento do COMA é feito através da realização de diversos passos. Primeiramente, os esquemas a serem comparados são convertidos para o formato de representação interna do sistema. Em seguida, percorre-se cada esquema para determinar seus elementos. Os elementos são representados pelos seus caminhos dentro do esquema, ou seja, pela seqüência de nós da raiz até o nó correspondente. Por exemplo, na Figura 3.2a, o elemento "custCity" é representado pelo caminho *PO1.Customer.custCity*.

Com a interação do usuário, que é um passo opcional, o COMA assegura que os elementos de mapeamento já aceitos ou rejeitados pelo usuário se mantenham inalterados pelos algoritmos de *matching* em iterações posteriores. Essa interação do usuário influencia no processamento dos elementos relacionados aos elementos aceitos ou rejeitados, melhorando, assim, a precisão dos *matchers*.

O principal passo de cada iteração é a execução dos *matchers* independentes que foram escolhidos da biblioteca. Cada *matcher* determina um valor entre 0 (pouca semelhança) e 1 (muita semelhança) para cada combinação de elementos dos esquemas S1 e S2. O resultado do processamento de todos os *matchers* é armazenado num cubo de similaridade, um repositório para facilitar o passo seguinte de combinação dos resultados. O Quadro 3.2 mostra uma parte do cubo de similaridade para os esquemas da Figura 3.2.

Matcher	Elementos do PO1	Elementos do PO2	Similaridade
Nome do Tipo	*PO1.ShipTo.shipToCity*	*PO2.DeliverTo.Address.City*	0,65
	PO1.ShipTo.shipToStreet		0,3
	PO1.Customer.custCity		0,80
Caminho do Nome	*PO1.ShipTo.shipToCity*	*PO2.DeliverTo.Address.City*	0,78
	PO1.ShipTo.shipToStreet		0,73
	PO1.Customer.custCity		0,53

Quadro 3.2 - Similaridade entre os esquemas PO1 e PO2 [Do 2001]

O passo final de uma iteração é combinar os resultados de cada *matcher* individual armazenados no cubo de similaridade. Essa combinação é feita em duas etapas: agregação dos resultados específicos e seleção dos candidatos. Na primeira, para cada resultado de um *matcher* de cada combinação de elementos dos esquemas é feita uma média ou pego o maior valor. O Quadro 3.3 mostra o resultado dessa etapa para o exemplo do Quadro 3.2, usando a estratégia da média. Na segunda etapa, aplica-se uma estratégia de seleção para escolher os melhores candidatos para o elemento de mapeamento. Essa seleção pode ser, por exemplo, escolher os elementos com melhor valor de similaridade combinada. Para o exemplo do Quadro 3.3, o candidato a formar o elemento de mapeamento do elemento *PO2.DeliverTo.Address.City* seria o *PO1.ShipTo.shipToCity*.

Elementos do PO1	Elementos do PO2	Similaridade Combinada
PO1.ShipTo.shipToCity	*PO2.DeliverTo.Address.City*	0,72
PO1.Customer.custCity		0,67
PO1.ShipTo.shipToStreet		0,52

Quadro 3.3 - Similaridade combinada do Quadro 3.2 [Do 2001]

3.2.2 Cupid

O Cupid é um *matcher* individual, ou seja, ele é composto de um único algoritmo genérico. Esse algoritmo se propõe a ser completo, pois ele dispõe de granularidade tanto no nível de elemento quanto no de estrutura, é baseado tanto na lingüística quanto nas restrições e gera resultados com cardinalidade de um para um ou um para vários. Além disso, o algoritmo é baseado em esquema e, assim como o COMA, determina um valor entre 0 e 1 para cada par de elementos dos esquemas comparados. O Cupid pode modelar os esquemas como uma árvore hierárquica ou, de uma forma mais genérica, como um grafo. Neste trabalho, será mostrado apenas o caso dos esquemas modelados como árvore, visto que é mais simples e de fácil compreensão.

O processamento do algoritmo do Cupid se dá em três fases: o *matching* lingüístico, o *matching* estrutural e a geração do mapeamento.

Na primeira fase, o *matching* lingüístico, os elementos dos esquemas são comparados individualmente baseando-se nos seus nomes, tipos de dados e o domínio relacionado. Para cada par de elementos, é dado um valor chamado de *coeficiente de similaridade lingüística* (*lsim*), no intervalo de 0 a 1. Essa fase é subdividida em três etapas:

- Normalização: alguns nomes de elementos podem ser semanticamente equivalentes, mas sintaticamente diferentes, pois podem conter abreviações, acrônimos, pontuação, símbolos especiais, entre outros. É preciso, então, que eles sejam normalizados para um conjunto de nomes, para que eles possam ser comparados (por exemplo, os elementos “cli_PE” e “clientes_de_Pernambuco”);
- Categorização: os elementos dos esquemas são agrupados em categorias, para que apenas os elementos que pertençam a categorias compatíveis sejam comparados. Dessa forma, o número de comparações de elementos é bastante reduzido. Além disso, um elemento pode pertencer a várias categorias;

- Comparação: a similaridade lingüística é calculada para cada par de elementos de categorias compatíveis. Como resultado, é gerada uma tabela com os coeficientes de similaridade lingüística entre os elementos dos dois esquemas.

A segunda fase é o *matching* estrutural, baseado na similaridade do contexto e da vizinhança dos elementos dos esquemas. Por exemplo, na Figura 3.4, "Posição" pode ser mapeado em "Número", caso já se saiba que os dois elementos "Item" são mapeados um no outro e que "Qtd" é mapeado em "Quantidade".

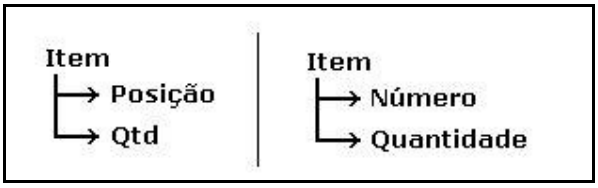

Figura 3.4 - Exemplo de *matching* estrutural [Madhavan 2001]

Assim como na primeira fase, a segunda gera, para cada par de elementos, o *coeficiente de similaridade estrutural* (*ssim*), no intervalo de 0 a 1. Com esses dois coeficientes, calcula-se a *similaridade ponderada* (*wsim*), para cada par de elementos, como: $wsim = w_{struct} \times ssim + (1-w_{struct}) \times lsim$, onde w_{struct}, um valor entre 0 e 1, indica o quanto cada coeficiente irá contribuir para o resultado final. O cálculo do *coeficiente de similaridade estrutural* e da *similaridade ponderada* é feito durante a execução do algoritmo *TreeMatch*. Esse algoritmo percorre as árvores a partir das folhas em direção à raiz e é baseado em três premissas:

- Os elementos atômicos (ou seja, as folhas) das duas árvores são similares se eles são individualmente similares (de acordo com a lingüística e o tipo de dado) e se os elementos da vizinhança (no mesmo nível e nos níveis acima) são similares;
- Dois elementos não-atômicos são similares se eles são lingüisticamente similares e se suas subárvores são similares;

- Dois elementos não-atômicos são estruturalmente similares se o conjunto de suas folhas são altamente similares, mesmo que seus filhos imediatos não o sejam. Isso se dá porque são as folhas que representam os dados que o esquema descreve.

Partindo dessa terceira premissa, quando se compara dois elementos não-atômicos e se descobre que eles são estruturalmente similares, o *coeficiente de similaridade estrutural* de suas folhas é aumentado. Da mesma forma, se eles não forem estruturalmente similares, o coeficiente das folhas é reduzido.

A terceira e última fase é a geração do mapeamento. Os elementos de mapeamento são gerados de acordo com as similaridades lingüística e estrutural calculadas anteriormente. O mapeamento pode ser feito só para os elementos nas folhas das árvores ou para todos os elementos. No caso do mapeamento só para as folhas, para cada elemento folha *t* do esquema de destino, se um elemento folha *s* do esquema de origem é aceitável (ou seja, se a *similaridade ponderada* entre *s* e *t* excede um determinado limiar), então é gerado um elemento de mapeamento de *s* para *t*. No caso do mapeamento para todos os elementos, é preciso executar o algoritmo *TreeMatch* mais uma vez para recalcular as similaridades dos elementos não-atômicos. Isso se dá porque, durante a execução do algoritmo, a similaridade das folhas é constantemente atualizada e isso pode afetar as similaridades dos elementos não-atômicos que já haviam sido calculadas antes das atualizações.

3.2.3 Similarity Flooding

Assim como o Cupid, o Similarity Flooding é um *matcher* individual, composto de um único algoritmo genérico e que determina um valor entre 0 e 1 para cada par de elementos dos esquemas comparados. O primeiro passo de execução do Similarity Flooding é a conversão de cada esquema em um grafo de rótulos direcionado. Por exemplo, para os esquemas mostrados na Figura 3.5, a Figura 3.6 mostra uma parte do grafo que representa o esquema S1. Os

retângulos e os ovais são os nós do grafo. Os rótulos dentro dos retângulos representam valores, enquanto os rótulos dentro dos ovais denotam os identificadores e os tipos dos nós.

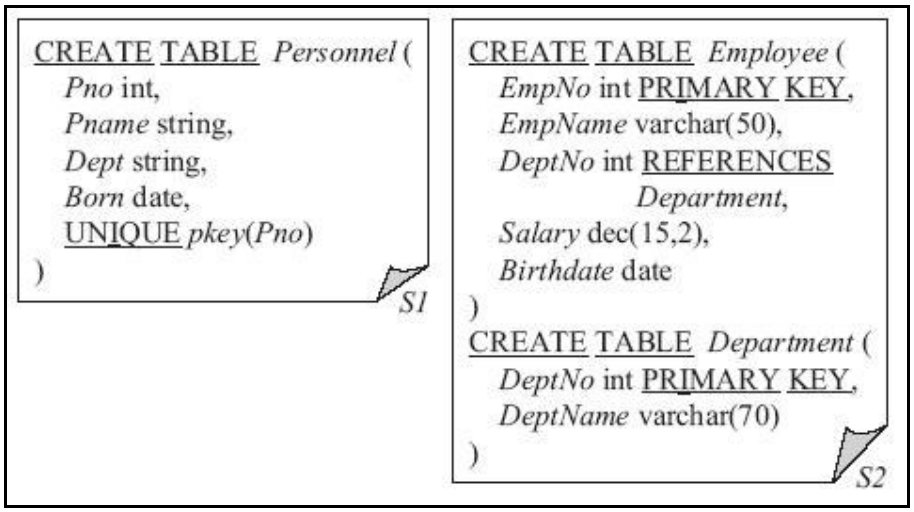

Figura 3.5 - Esquemas S1 e S2 [Melnik 2002]

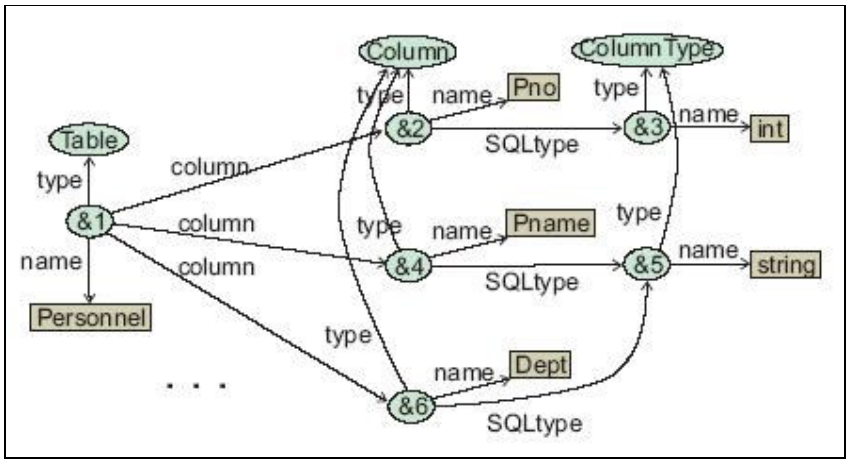

Figura 3.6 - Parte da representação do esquema S1 [Melnik 2002]

Estando os dois esquemas convertidos em grafos, o próximo passo é fazer um mapeamento inicial, que consiste em submeter os esquemas a um algoritmo simples de comparação de strings. É gerado um valor entre 0 e 1 para cada par de nós entre os dois esquemas. Através do Quadro 3.4, que mostra uma porção do resultado dessa etapa aplicada aos esquemas S1 e S2, nota-se que essa parte do processo ainda é bastante imprecisa, visto que são

feitas comparações de nomes de colunas com nomes de tabelas e nomes de tipos.

Similaridade	Nó de S1	Nó de S2
1.0	Column	Column
0.66	ColumnType	Column
0.66	'Dept'	'DeptNo'
0.66	'Dept'	'DeptName'
0.5	UniqueKey	PrimaryKey
0.26	'Pname'	'DeptName'
0.26	'Pname'	'EmpName'
0.22	'date'	'Birthdate'
0.11	'Dept'	'Department'
0.06	'int'	'Department'

Quadro 3.4 - Porção do mapeamento inicial de S1 e S2 [Melnik 2002]

O mapeamento inicial resultado da segunda etapa será usado como ponto de partida para o algoritmo principal do Similarity Flooding, que consiste na terceira etapa do processo. Esse algoritmo, que não leva em consideração a semântica dos nós nem das arestas dos grafos, irá gerar um mapeamento mais refinado do que o mapeamento inicial. Ele é baseado na seguinte idéia: sempre que dois elementos nos esquemas comparados forem considerados similares, a similaridade dos elementos adjacentes a eles aumenta. Dessa forma, à medida que são feitas sucessivas iterações, a similaridade dos dois nós é propagada através dos grafos. O algoritmo termina quando a similaridades dos elementos se estabiliza. Por exemplo, considerando os esquemas S1 e S2, na primeira iteração a similaridade dos elementos *S1.Personnel.Pname* e *S2.Employee.EmpName* é propagada para os tipos *S1.string* e *S2.varchar*. Dessa forma, é visto que a similaridade entre *S1.Personnel.Dept* e *S2.Department.DeptName* deve ser maior do que entre *S1.Personnel.Dept* e *S2.Department.DeptNo*, e com isso a similaridade do primeiro par aumenta e a do segundo diminui.

A última etapa consiste em submeter o mapeamento resultado da etapa anterior a um operador de filtro, que irá escolher um subconjunto dos pares de nós gerados de acordo com os objetivos do *matching*.

3.3 Considerações Finais

Neste capítulo, foram vistos os principais conceitos relacionados à comparação de esquemas, utilizada para identificar os elementos que são semanticamente correspondentes em dois esquemas. Foram definidos: o conceito de *matching* de esquemas, onde ele é aplicado, uma classificação para os *matchers* (sistemas que fazem o *matching*) e como funcionam três dos sistemas mais em destaque atualmente: COMA, Cupid e Similarity Flooding. Dentre esses três, o Cupid servirá como base para o desenvolvimento do módulo Comparador dos Esquemas das Fontes.

4. Comparador dos Esquemas das Fontes

Como foi dito anteriormente, o módulo Gerenciador de Esquemas Conceituais do sistema Integra é o principal módulo para esse trabalho, pois é nele que se faz a comparação dos esquemas das fontes para identificar a evolução de um esquema. Ele é responsável por extrair, através do módulo *Lookup*, o esquema atual (em XML Schema) da fonte sendo analisada; traduzir esse esquema para o modelo X-Entity; consultar a Base de Conhecimento das Fontes para obter o esquema antigo (em X-Entity) da mesma fonte; comparar os dois esquemas; e, por fim, retornar o resultado da comparação.

Porém, a tarefa de comparar os esquemas para avaliar a evolução da fonte ainda não é feita pela atual versão do sistema Integra. Como essa é uma funcionalidade essencial para o sistema, o objetivo deste trabalho é desenvolver o submódulo Comparador dos Esquemas das Fontes para o módulo Gerenciador de Esquemas Conceituais.

Durante o processo de desenvolvimento do submódulo, foi implementada uma primeira versão, que satisfazia o objetivo desejado de comparar os esquemas antigo e novo de uma mesma fonte de dados e retornar um resultado mostrando sua evolução. Porém, percebeu-se que essa era uma abordagem muito simples, que apenas mostrava o que havia sido modificado e o que não havia sido modificado de um esquema para o outro. Com isso, deixava-se para quem fosse avaliar o resultado da comparação a tarefa de identificar quais, dentre os elementos que haviam sido modificados, poderiam representar o mesmo elemento semanticamente.

Daí surgiu a necessidade de se desenvolver uma segunda versão, como uma extensão da primeira, que buscasse mostrar em seu resultado quais os elementos que, apesar de terem sido modificados, provavelmente correspondiam ao mesmo elemento.

4.1 Primeira versão

Esse primeiro algoritmo de comparação parte de uma única premissa: dois elementos (atômicos ou não) serão considerados similares se eles forem lingüisticamente similares, ou seja, se tiverem o mesmo nome. A partir disso, os elementos do esquema antigo e do esquema novo serão comparados e, durante a execução do algoritmo, todos os elementos serão marcados da seguinte forma:

- Se um elemento aparece no esquema antigo, mas não aparece no esquema novo, ele é marcado como "*deleted*" (removido);
- Se um elemento aparece no esquema novo, mas não aparece no esquema antigo, ele é marcado como "*added*" (adicionado);
- Se um elemento atômico aparece no esquema antigo e também no esquema novo, ele é marcado como "*unchanged*" (não-modificado);
- Se um elemento não-atômico aparece no esquema antigo e também no esquema novo, com a mesma estrutura (ou seja, com os mesmos filhos), ele é marcado como "*unchanged*" (não-modificado);
- Se um elemento não-atômico aparece no esquema antigo e também no esquema novo, porém com uma estrutura diferente, ele é marcado como "*modified*" (modificado);

Os esquemas das fontes são visto como uma árvore que possui três níveis de elementos: no primeiro nível encontra-se a raiz, que é o elemento que representa o esquema como um todo; no segundo nível encontram-se os filhos do esquema, que são as entidades; e no terceiro nível encontram-se os filhos das entidades, que são os seus atributos.

4.1.1 Algoritmo de comparação

O algoritmo de comparação começa recebendo os dois esquemas (o antigo e o novo) da fonte de dados, em formato X-Entity. A primeira etapa

começa por verificar, para cada entidade do esquema antigo, se existe uma entidade correspondente no novo esquema, ou seja, se aquela entidade foi mantida na evolução do esquema. Duas entidades são consideradas correspondentes se elas tiverem o mesmo nome. Se não existir, a entidade é marcada como "*deleted*", assim como todos os seus atributos. Se for encontrada uma entidade no esquema novo com o mesmo nome da entidade do antigo, então é verificado, para cada atributo da entidade antiga, se existe um atributo correspondente na entidade nova. Da mesma forma que para as entidades, dois atributos são considerados correspondentes se eles tiverem o mesmo nome. Se existir, o atributo é marcado como "*unchanged*". Se não existir, o atributo é marcado como "*deleted*". Após verificar todos os atributos, se apareceu algum "*deleted*", então a entidade é marcada como "*modified*". Caso contrário, a entidade é marcada como "*unchanged*". Após verificar todas as entidades, se alguma delas foi "*deleted*" ou "*modified*", então o esquema antigo é marcado como "*modified*". Caso contrário, o esquema antigo é marcado como "*unchanged*".

A segunda etapa é análoga à primeira, com a diferença de que a comparação é feita a partir do esquema novo. Ela começa por verificar, para cada entidade do esquema novo, se existe uma entidade correspondente (que tenha o mesmo nome) no esquema antigo. Se não existir, a entidade é marcada como "*added*", assim como todos os seus atributos. Se existir, então é verificado, para cada atributo da entidade nova, se existe um atributo correspondente (com o mesmo nome) na entidade antiga. Se existir, o atributo é marcado como "*unchanged*". Se não existir, o atributo é marcado como "*added*". Se houve pelo menos um atributo "*added*", então a entidade é marcada como "*modified*". Caso contrário, a entidade é marcada como "*unchanged*". Após verificar todas as entidades, se alguma delas foi "*added*" ou "*modified*", então o esquema novo é marcado como "*modified*". Caso contrário, o esquema novo é marcado como "*unchanged*".

4.1.2 Representação dos Resultados

O formato do resultado da comparação feita pelo algoritmo anterior é baseado no formato apresentado por [Fontaine 2001] para a representação de mudanças em documentos XML [WWWC 2004a].

Considere os dois esquemas a seguir da base de dados "escola". A Figura 4.1 mostra o esquema antigo dessa base de dados.

```
<XENTITY_SCHEMA>
  <ENTITY name="aluno">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE name="idade"/>
  </ENTITY>
  <ENTITY name="professor">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE name="cpf"/>
    <ATTRIBUTE name="telefone"/>
  </ENTITY>
  <ENTITY name="livro">
    <ATTRIBUTE name="titulo"/>
    <ATTRIBUTE name="editora"/>
  </ENTITY>
</XENTITY_SCHEMA>
```

Figura 4.1 - Esquema antigo da base de dados "escola"

A Figura 4.2 mostra o novo esquema dessa mesma base de dados.

```
<XENTITY_SCHEMA>
  <ENTITY name="aluno">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE name="idade"/>
  </ENTITY>
  <ENTITY name="professor">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE name="cpf"/>
    <ATTRIBUTE name="endereco"/>
    <ATTRIBUTE name="email"/>
  </ENTITY>
  <ENTITY name="sala_de_aula">
    <ATTRIBUTE name="numero"/>
    <ATTRIBUTE name="andar"/>
    <ATTRIBUTE name="capacidade"/>
  </ENTITY>
</XENTITY_SCHEMA>
```

Figura 4.2 - Esquema novo da base de dados "escola"

Vê-se, pelas duas figuras, que houve uma evolução no esquema da base de dados "escola". A entidade "aluno" e seus atributos não sofreram modificações. Porém, a entidade "professor" foi modificada, pois o atributo "telefone" foi removido e os atributos "endereco" e "email" foram adicionados. Além disso, a entidade "livro", juntamente com seus atributos, foi removida do esquema, enquanto a entidade "sala_de_aula" foi adicionada ao esquema. O resultado da execução do algoritmo anterior para esses dois esquemas pode ser visto na Figura 4.3.

```
<MATCHING_RESULT status="modified">
  <ENTITY name="aluno" status="unchanged">
    <ATTRIBUTE name="nome" status="unchanged"\>
    <ATTRIBUTE name="idade" status="unchanged"\>
  </ENTITY>
  <ENTITY name="professor" status="modified">
    <ATTRIBUTE name="nome" status="unchanged"\>
    <ATTRIBUTE name="cpf" status="unchanged"\>
    <ATTRIBUTE name="telefone" status="deleted"\>
    <ATTRIBUTE name="endereco" status="added"\>
    <ATTRIBUTE name="email" status="added"\>
  </ENTITY>
  <ENTITY name="livro" status="deleted">
    <ATTRIBUTE name="titulo" status="deleted"\>
    <ATTRIBUTE name="editora" status="deleted"\>
  </ENTITY>
  <ENTITY name="sala_de_aula" status="added">
    <ATTRIBUTE name="numero" status="added"\>
    <ATTRIBUTE name="andar" status="added"\>
    <ATTRIBUTE name="capacidade" status="added"\>
  </ENTITY>
</MATCHING_RESULT>
```

Figura 4.3 - Resultado da evolução da base de dados "escola"

4.2 Segunda versão

Como dito anteriormente, foi necessário o desenvolvimento de uma segunda versão do Comparador dos Esquemas das Fontes que mostrasse em seu resultado quais elementos que sofreram modificações e que poderiam corresponder ao mesmo elemento semanticamente. Esse novo algoritmo de comparação, além de ser uma extensão do algoritmo anterior, tomou como

base o algoritmo de comparação do *matcher* Cupid [Madhavan 2001], descrito no capítulo anterior.

Durante a execução desse novo algoritmo, alguns coeficientes serão calculados. Esses coeficientes, que são valores entre 0 e 1, foram baseados nos coeficientes do sistema Cupid. Quanto mais próximo de 1, maior a similaridade entre os elementos comparados. São eles:

- Coeficiente de similaridade lingüística (*lsim*) ⇒ é o resultado da comparação de dois nomes através do algoritmo de comparação de strings desenvolvido por [Gondim 2006];
- Coeficiente de similaridade estrutural (*ssim*) ⇒ quando se compara dois elementos s′ e s′′, esse coeficiente corresponde à fração do número de elementos filhos de s′ e s′′ que possuem um elemento correspondente aceitável (esse conceito será explicado adiante) dividido pelo total de elementos filhos de s′ e s′′;
- Coeficiente de similaridade ponderada (*wsim*) ⇒ é resultado de uma média ponderada entre o coeficiente de similaridade lingüística (cujo peso é o valor do limiar lingüístico, que será visto a seguir) e o coeficiente de similaridade estrutural (cujo peso é o complemento do valor do limiar lingüístico para 1).

Além desses coeficientes, o algoritmo utiliza dois valores chamados de limiares. São eles:

- Limiar lingüístico (*linguistic threshold*) ⇒ é o valor mínimo para que uma comparação lingüística seja considerada aceitável. Se um valor *lsim* de uma comparação for maior ou igual ao limiar lingüístico, essa comparação é dita aceitável. Esse limiar também corresponde ao peso do coeficiente de similaridade lingüística no cálculo do coeficiente de similaridade ponderada;
- Limiar de *matching* (*matching threshold*) ⇒ é o valor mínimo para que uma comparação ponderada seja considerada aceitável. Se um valor

wsim de uma comparação for maior ou igual ao limiar de *matching*, essa comparação é dita aceitável.

O segundo algoritmo de comparação de esquemas tem como base, além da premissa do primeiro algoritmo, algumas outras herdadas do sistema Cupid. As premissas do novo algoritmo são:

- Dois elementos (atômicos ou não) serão considerados similares se eles forem lingüisticamente similares, ou seja, se o resultado do *lsim* entre eles for igual a 1;
- Dois elementos atômicos serão considerados potencialmente similares (ou aceitáveis) se o resultado do *lsim* entre eles for maior ou igual ao limiar lingüístico;
- Dois elementos não-atômicos serão considerados potencialmente similares (ou aceitáveis) se o resultado do *wsim* entre eles for maior ou igual ao limiar de *matching*;

4.2.1 Algoritmo de comparação

Da mesma forma que o primeiro algoritmo, o novo algoritmo de comparação também começa recebendo os dois esquemas (o antigo e o novo) da fonte de dados, em formato X-Entity.

A primeira etapa é verificar, para cada entidade do esquema antigo, se existe uma entidade que lhe seja similar no novo esquema. Se existir, então é verificado, para cada atributo das duas entidades, se existe um atributo que lhe seja similar na outra entidade. Os que tiverem um atributo similar encontrado, não sofreram alterações.

Entre os atributos que não tiveram um similar encontrado, serão procurados os que são potencialmente similares. Para isso, eles serão comparados lingüisticamente entre si. Ou seja, cada atributo de uma entidade será comparado com cada atributo da outra entidade, e cada comparação gerará um valor *lsim*. Após todas as comparações, será escolhido o maior

valor de *lsim* que seja aceitável (maior ou igual ao limiar lingüístico), significando que os dois atributos que geraram esse *lsim* são considerados potencialmente similares.

Entre as comparações de atributo restantes, mais uma vez será escolhido o maior valor de *lsim* que seja aceitável, e assim sucessivamente, até que não haja mais nenhum valor de *lsim* que seja maior ou igual ao limiar lingüístico. Para os atributos da entidade antiga que sobraram (ou seja, que não tiveram um atributo potencialmente similar), significa que estes foram removidos do novo esquema. E para os atributos da nova entidade que sobraram, significa que estes foram adicionados ao novo esquema.

Essa primeira etapa tratou das entidades que tinham uma entidade similar no outro esquema. A segunda etapa irá tratar das entidades que sobraram, ou seja, que não tiveram um similar encontrado.

Dentre estas entidades, elas serão comparadas entre si para procurar as que são potencialmente similares. Ou seja, cada entidade de um esquema será comparada com cada entidade do outro esquema, e cada comparação gerará um valor de *wsim*. Após todas as comparações, será escolhido o maior valor de *wsim* que seja aceitável (maior ou igual ao limiar de *matching*), significando que as duas entidades que geraram esse *wsim* são consideradas potencialmente similares. Com essas duas entidades, repete-se o processo de comparação de atributos da primeira etapa, para verificar quais atributos possuem um similar, quais são potencialmente similares e quais foram adicionados ou removidos.

Entre as comparações de entidades restantes, será escolhido mais uma vez o maior valor de *wsim* que seja aceitável, e assim sucessivamente, até que não haja mais nenhum valor de *wsim* que seja maior ou igual ao limiar de *matching*. Para as entidades do esquema antigo que sobraram, significa que elas foram removidas do novo esquema. E para as entidades do novo esquema que sobraram, significa que elas foram adicionadas ao novo esquema.

4.2.2 Representação dos Resultados

Verificou-se que o formato do resultado da comparação do primeiro algoritmo não satisfazia a representação dos novos resultados. Por isso, foi definido um novo formato, que pudesse mostrar os novos resultados gerados e que fosse de fácil visualização.

Nesse novo formato, o elemento raiz "MATCHING_RESULT" contém dois atributos: "linguistic_threshold", que contém o valor do limiar lingüístico, e "matching_threshold", que contém o valor do limiar de *matching*.

As modificações nos atributos são representadas pelo elemento "ATTRIBUTE_CHANGE". Nos casos em que forem encontrados dois atributos potencialmente similares, esse elemento possui o atributo "similarity", que mostra o resultado da comparação lingüística entre os atributos em questão. Além disso, esse elemento contém os elementos filhos "OLD_ATTRIBUTE", para representar os atributos removidos, e "NEW_ATTRIBUTE", para representar os atributos adicionados.

As modificações nas entidades são mostradas de forma semelhante pelo elemento "ENTITY_CHANGE". Quando são encontradas duas entidades potencialmente similares, esse elemento possui o atributo "similarity", para mostrar o valor da similaridade ponderada entre as entidades. Esse elemento também contém dois elementos filhos: "OLD_ENTITY", para representar as entidades removidas, e "NEW_ENTITY", para representar as entidades adicionadas.

Considere os dois esquemas a seguir da base de dados "loja". A Figura 4.4 mostra o esquema antigo dessa base de dados.

```
<XENTITY_SCHEMA>
  <ENTITY name="cliente">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE name="cpf"/>
    <ATTRIBUTE name="telefone"/>
    <ATTRIBUTE name="endereco"/>
  </ENTITY>
  <ENTITY name="vendedor">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE name="endereco"/>
    <ATTRIBUTE name="telefones"/>
    <ATTRIBUTE name="rg"/>
  </ENTITY>
  <ENTITY name="entrega">
    <ATTRIBUTE name="destinatario"/>
    <ATTRIBUTE name="endereco"/>
    <ATTRIBUTE name="cep"/>
  </ENTITY>
</XENTITY_SCHEMA>
```

Figura 4.4 - Esquema antigo da base de dados "loja"

A Figura 4.5 mostra o novo esquema dessa mesma base de dados.

```
<XENTITY_SCHEMA>
  <ENTITY name="cliente">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE name="rg"/>
    <ATTRIBUTE name="telefones"/>
  </ENTITY>
  <ENTITY name="vendedora">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE name="endereco"/>
    <ATTRIBUTE name="telefone"/>
    <ATTRIBUTE name="cpf"/>
    <ATTRIBUTE name="email"/>
  </ENTITY>
  <ENTITY name="produto">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE name="codigo"/>
    <ATTRIBUTE name="fabricante"/>
  </ENTITY>
</XENTITY_SCHEMA>
```

Figura 4.5 - Esquema novo da base de dados "loja"

Vê-se que houve uma evolução no esquema da base de dados "loja". A entidade "cliente" permaneceu, mas alguns de seus atributos foram

modificados. Apenas o atributo "nome" apareceu sem modificações. Os atributos do esquema antigo "telefone", "cpf" e "endereco" não estão no esquema novo, enquanto os atributos do novo esquema "telefones" e "rg" não estão no esquema antigo. Com isso, o algoritmo comparou esses atributos entre si e verificou que a maior semelhança era entre os atributos "telefone" e "telefones", com similaridade lingüística igual a "0,94". Assim, eles foram considerados como potencialmente similares. Como os outros valores de *lsim* não ultrapassaram o limiar lingüístico (linguistic_threshold = "0,5"), os atributos antigos "cpf" e "endereco" foram considerados como removidos, e o atributo novo "rg" foi considerado como adicionado.

Além disso, a entidade "cliente" foi a única que teve sua similar encontrada. As outras entidades, "vendedor" e "entrega" no esquema antigo e "vendedora" e "produto" no esquema novo, foram comparadas entre si e a maior semelhança se deu entre as entidades "vendedor" e "vendedora", com similaridade ponderada igual a "0,80". Portanto, elas foram consideradas como potencialmente similares. Além da mudança de nome entre essas entidades, houve também uma mudança nos atributos. Verificou-se que os atributos "nome" e "endereco" mantiveram-se iguais. Do mesmo modo, "telefone" do esquema antigo e "telefones" do esquema novo foram considerados potencialmente similares. Porém, os demais atributos não tiveram um par correspondente e, assim, concluiu-se que o atributo "rg" foi removido do esquema antigo, enquanto os atributos "cpf" e "email" foram adicionados ao esquema novo.

Como as duas entidades restantes, "entrega" do esquema antigo e "produto" do novo esquema, não tiveram um *wsim* aceitável (matching_threshold = "0,5"), considerou-se que a entidade "entrega" foi removida do esquema antigo e a entidade "produto" foi adicionada ao novo esquema.

O resultado da execução do algoritmo anterior para esses dois esquemas pode ser visto na Figura 4.6.

```
<MATCHING_RESULT linguistic_threshold="0.5" matching_threshold="0.5">
  <ENTITY name="cliente">
    <ATTRIBUTE name="nome"/>
    <ATTRIBUTE_CHANGE similarity="0.9411765">
      <OLD_ATTRIBUTE name="telefone"/>
      <NEW_ATTRIBUTE name="telefones"/>
    </ATTRIBUTE_CHANGE>
    <ATTRIBUTE_CHANGE>
      <OLD_ATTRIBUTE name="cpf"/>
    </ATTRIBUTE_CHANGE>
    <ATTRIBUTE_CHANGE>
      <OLD_ATTRIBUTE name="endereco"/>
    </ATTRIBUTE_CHANGE>
    <ATTRIBUTE_CHANGE>
      <NEW_ATTRIBUTE name="rg"/>
    </ATTRIBUTE_CHANGE>
  </ENTITY>
  <ENTITY_CHANGE similarity="0.8039216">
    <OLD_ENTITY name="vendedor"/>
    <NEW_ENTITY name="vendedora">
      <ATTRIBUTE name="nome"/>
      <ATTRIBUTE name="endereco"/>
      <ATTRIBUTE_CHANGE similarity="0.9411765">
        <OLD_ATTRIBUTE name="telefones"/>
        <NEW_ATTRIBUTE name="telefone"/>
      </ATTRIBUTE_CHANGE>
      <ATTRIBUTE_CHANGE>
        <OLD_ATTRIBUTE name="rg"/>
      </ATTRIBUTE_CHANGE>
      <ATTRIBUTE_CHANGE>
        <NEW_ATTRIBUTE name="cpf"/>
      </ATTRIBUTE_CHANGE>
      <ATTRIBUTE_CHANGE>
        <NEW_ATTRIBUTE name="email"/>
      </ATTRIBUTE_CHANGE>
    </NEW_ENTITY>
  </ENTITY_CHANGE>
  <ENTITY_CHANGE>
    <OLD_ENTITY name="entrega">
      <ATTRIBUTE name="destinatario"/>
      <ATTRIBUTE name="endereco"/>
      <ATTRIBUTE name="cep"/>
    </OLD_ENTITY>
  </ENTITY_CHANGE>
  <ENTITY_CHANGE>
    <NEW_ENTITY name="produto">
      <ATTRIBUTE name="nome"/>
      <ATTRIBUTE name="codigo"/>
      <ATTRIBUTE name="fabricante"/>
    </NEW_ENTITY>
  </ENTITY_CHANGE>
</MATCHING_RESULT>
```

Figura 4.6 - Resultado da evolução da base de dados "loja"

4.3 Considerações Finais

Neste capítulo, foi apresentado como foi desenvolvido o Comparador dos Esquemas das Fontes para o sistema Integra. Para as duas versões implementadas, foram explicados: alguns conceitos que seriam importantes para os algoritmos, os algoritmos desenvolvidos e como os resultados gerados seriam representados. Foram mostrados também alguns exemplos para melhor ilustrar os algoritmos e seus resultados.

5. Conclusões e Trabalhos Futuros

Com a grande evolução em relação às diversidades de sistemas de informação e, principalmente, com o advento da Web, é cada vez maior a quantidade de aplicações que precisam ter acesso a esses sistemas de forma integrada. Como já foi discutido, um sistema de integração de dados tem por objetivo fornecer uma interface uniforme para fontes de dados distribuídas, heterogêneas e autônomas. Contudo, o tratamento de fontes autônomas é uma tarefa complicada, pois elas podem sofrer alterações em seus esquemas e é essencial que essas alterações sejam refletidas no restante do sistema de integração.

Apesar da existência de vários comparadores de esquemas atualmente, verificou-se que a utilização destes dentro do sistema de integração de dados Integra não seria adequado. Essa constatação se deu, principalmente, devido ao fato de que o modelo interno para representação de esquemas (X-Entity) foi desenvolvido especificamente para o sistema e, portanto, o desenvolvimento de um comparador específico para esse modelo resolveria a tarefa de comparação de forma mais eficiente. Em virtude disso, esse trabalho desenvolveu, para o sistema Integra, um módulo para comparar o esquema antigo e o novo de uma mesma base de dados e retornar como resultado as diferenças existentes entre eles, de forma a permitir a evolução dos esquemas das fontes.

Baseados nos resultados aqui apresentados, alguns pontos que poderiam servir como trabalhos futuros, de forma a engrandecer esse trabalho, seriam:

- O aperfeiçoamento do comparador para o estado de *matcher* genérico, de forma que pudessem ser aplicados diferentes modelos de representação de esquemas;

- O passo seguinte na evolução de esquemas do sistema Integra, após a comparação dos esquemas, que seria mostrar ao usuário o resultado da evolução e permitir que ele fizesse as alterações, quando necessárias, no esquema global e nos mapeamentos entre os esquemas.

Referências Bibliográficas

[Batista 2003] Batista, M. C. M., Otimização de Acesso em um Sistema de Integração de Dados Através do Uso de Caching e Materialização de Dados, Dissertação de Mestrado em Ciência da Computação, Centro de Informática, UFPE, 2003

[Chen 1976] Chen, P. P., The Entity-Relationship Model: Toward a unified view of data, ACM Transactions on Database Systems, 1-36, 1976.

[Costa 2005] Costa, T. A., O Gerenciador de Consultas de um Sistema de Integração de Dados, Dissertação de Mestrado em Ciência da Computação, Centro de Informática, UFPE, 2005.

[Do 2001] Do, H. H., Rahm, E., COMA – A system for flexible combination of schema matching approaches, Proceedings of the Very Large Data Bases Conference (VLDB), 2001.

[Fontaine 2001] Fontaine, R., A DELTA Format for XML: Identifying Changes in XML Files and Representing the Changes in XML, Proceedings of XML Europe, 2001.

[Gondim 2006] Gondim, F. M., Algoritmo de Comparação de Strings para Integração de Esquemas de Dados, Trabalho de Graduação em Ciência da Computação, Centro de Informática, UFPE, 2006.

[Lóscio 2003] Lóscio, B. F., Managing the Evolution of XML-Based Mediation Queries, Tese de Doutorado em Ciência da Computação, Centro de Informática, UFPE, 2003.

[Madhavan 2001] Madhavan, J., Bernstein, P., Rahm, E., Generic schema matching with Cupid. Proceedings of the Very Large Data Bases Conference (VLDB), 2001.

[Melnik 2002] Melnik, S., Garcia-Molina, H., Rahm, E., Similarity flooding: A versatile graph matching algorithm. Proceedings of the International Conference on Data Engineering (ICDE), 2002.

[OASIS 2001] The Organization for the Advancement of Structured Information Standards, RELAX NG Specification, 2001. Disponível em: http://relaxng.org/spec-20011203.html. Último acesso em setembro de 2006.

[Rahm 2001] Rahm, E., Bernstein, P., A survey of approaches to automatic schema matching. The International Journal on Very Large Data Bases (VLDB), 2001.

[Shvaiko 2005] Shvaiko, P., Euzenat, J., A survey of schema-based matching approaches. Journal on Data Semantics, 2005.

[Tech-Know-Ware 2006] A Simple Overview of XML Schema, 2006. Disponível em: http://www.tech-know-ware.com/lmx/xsd-overview.html. Último acesso em setembro de 2006.

[TheScarms 2000] TheScarms: An XML Schema Tutorial, 2000. Disponível em: http://www.thescarms.com/XML/SchemaTutorial.asp. Último acesso em setembro de 2006.

[XML.com 2001] XML.com: Using W3C XML Schema, 2001. Disponível em: http://www.xml.com/pub/a/2000/11/29/schemas/part1.html. Último acesso em setembro de 2006.

[W3Schools 2006] W3Schools: XML Schema Tutorial, 2006. Disponível em: http://www.w3schools.com/schema/default.asp. Último acesso em setembro de 2006.

[Wikipedia 2006] Wikipedia, the Free Encyclopedia: XML, 2006. Disponível em: http://en.wikipedia.org/wiki/XML. Último acesso em setembro de 2006.

[WWWC 1999a] World Wide Web Consortium, HTML 4.01 Specification, 1999. Disponível em: http://www.w3.org/TR/html401. Último acesso em setembro de 2006.

[WWWC 1999b] World Wide Web Consortium, XML Path Language (XPath) Version 1.0, 1999. Disponível em: http://www.w3.org/TR/xpath. Último acesso em setembro de 2006.

[WWWC 2004a] World Wide Web Consortium, Extensible Markup Language (XML) 1.0 (Third Edition), 2004. Disponível em: http://www.w3.org/TR/REC-xml. Último acesso em setembro de 2006.

[WWWC 2004b] World Wide Web Consortium, XML Schema Part 0: Primer Second Edition, 2004. Disponível em: http://www.w3.org/TR/xmlschema-0/. Último acesso em setembro de 2006.

[Yatskevich 2003] Yatskevich, M., Preliminary Evaluation of Schema Matching Systems. University of Trento, 2003.