I2I: Transformação Automatizada de Instâncias XML

Marcos de Barros

Instituto de Matemática - IM
Núcleo de Computação Eletrônica - NCE
Universidade Federal do Rio de Janeiro – UFRJ
Mestre em Informática

Prof. Eber Assis Schmitz, Ph.D.
NCE/UFRJ

Profa. Priscila Machado Vieira Lima, Ph.D.
NCE-UFRJ

Rio de Janeiro, RJ – Brasil
Dezembro - 2004

I2I: Transformação Automatizada de Instâncias XML

Marcos de Barros

Tese submetida ao corpo docente do Instituto de Matemática / Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro – UFRJ - como parte dos requisitos necessários à obtenção do grau de Mestre em Informática.

Aprovada por:

_____

Prof. Eber Assis Schmitz, Ph.D.
DCC-IM/NCE-UFRJ (Orientador)

_____

Profa. Priscila Machado Vieira Lima, Ph.D.
DCC-IM/NCE-UFRJ (Co-Orientadora)

_____

Prof. Fabio Protti, D.Sc
DCC-IM/NCE-UFRJ

_____

Prof. Ivan Mathias Filho, D.Sc.
DI / PUC-RIO

Rio de Janeiro, RJ – Brasil

Dezembro - 2004

Agradecimentos

Ao professor Eber Schmitz, por estar sempre aberto para novas idéias de trabalho, pelo apoio, dedicação e constante confiança na conclusão deste projeto.

A professora Priscila Lima, pela dedicação e contribuição dada.

Ao professor Fabio Protti, pela ajuda que deu na definição formal e prova do algoritmo.

A IClass Consultoria, pela oportunidade em desenvolver este trabalho que sempre foi dada desde o início do curso.

A todos os meus amigos e pessoas que sempre demonstraram um grande interesse e confiança no término do mestrado.

A Ivana, que sempre me apoiou e deu força nas horas em que ela estava acabando.

A minha família, especialmente pai e mãe, que nunca passaram sequer uma semana sem perguntar sobre o status desta dissertação, demonstrando toda a preocupação e confiança no término.

Sumário

Lista de Siglas e Abreviaturas

CIO – **C**hief **I**nformation **O**fficers

TI – **T**ecnologia da **I**nformação

B2B – **B**usiness to **B**usiness

ERP – **E**nterprise **R**esource **P**lanning

BI – **B**usiness **I**ntelligence

EAI – **E**nterprise **A**pplication **I**ntegration

CRM – **C**ustomer **R**elationship **M**anagement

SGBD – **S**istema de **G**erenciamento de **B**anco de **D**ados

API – **A**pplication **P**rogram **I**nterfaces

XML – E**x**tensible **M**arkup **L**anguage

XSLT – E**x**tensible **S**tylesheet **L**anguage for **T**ransformations

DOM – **D**ocument **O**bject **M**odel

SAX – **S**imple **A**PI for **X**ML

DTD – **D**ocument **T**ype **D**efinition

SGML – **S**tandard **G**eneralized **M**arkup **L**anguage

GML – **G**eography **M**arkup **L**anguage

W3C – **W**orld Wide Web **C**onsortium

HTML – **H**yper**t**ext **M**arkup **L**anguage

UML – **U**nified **M**odeling **L**anguage

JAXB – **J**ava **A**rchiteture for **X**ML **B**inding

PDF – **P**ortable **D**ocument **F**ormat

cXML – **C**ommerce **XML** Resources

xCBL – **X**ML **C**ommon **B**usiness **L**ibrary

Índice de Figuras

Pág.

Índice de Quadros

Pág.

RESUMO

BARROS, MARCOS DE. **I2I: Transformação Automatizada de Instâncias XML**; Orientador: Eber Assis Schmitz. Rio de Janeiro : UFRJ/NCE; novembro/2003. Dissertação (Mestrado em Informática).

A necessidade de automação da transformação entre instâncias XML, onde cada uma está relacionada a um diferente esquema (*schema*), aumentou significativamente devido ao número elevado de sistemas que utilizam o XML como padrão de representação de dados em suas interfaces de integração.

Os métodos tradicionais são baseados na geração de programas em XSLT, que necessitam de um especialista além de constituírem uma atividade relativamente custosa. Alternativamente, as propostas envolvendo a geração automatizada de programas em XSLT apresentam os seguintes pontos negativos: os resultados são difíceis de serem verificados ou são inconclusivos a respeito da melhor solução a ser adotada.

Este trabalho propõe um algoritmo de automatização da transformação entre uma instância XML de origem e um esquema XML (*XML schema*) de destino. O trabalho difere dos anteriores nos seguintes aspectos: i) não requer um especialista; ii) não necessita do XML Schema de origem; iii) resulta em uma solução única e iv) a prova de correção é apresentada.

Abstract

BARROS, MARCOS DE. **I2I: Transformação Automatizada de Instâncias XML**; Orientador: Eber Assis Schmitz. Rio de Janeiro : UFRJ/NCE; novembro/2003. Dissertação (Mestrado em Informática).

The need for automated transformation between pairs of XML instances, where each instance may belong to a different schema, has been increasing due to the fact that numerous software systems make use of the XML standard in their integration interfaces.

The conventional methods for this conversion are based on the generation of XSLT programs, which not only turns out to be costly but also requires in most cases the assistance of a specialist. Alternatively, the proposals involving the automatic generation of XSLT programs present the following drawbacks: they are either very difficult to verify or inconclusive about the best solution among the several candidate solutions.

This work proposes a method for the automatic transformation between a source XML instance and a target XML instance, given a target XML schema. Our proposal differs from the preceding ones in the following aspects: i) it does not require the aid of a specialist; ii) it does not need a source schema; iii) it results in a single solution and iv) a proof of correctness is presented.

1. Introdução

1.1 – Apresentação

Pesquisas com *Chief Information Officers* (CIOs) e demais executivos das principais empresas de Tecnologia da Informação (TI) do mundo indicam que os investimentos em TI planejados para os próximos anos incluem o comércio *Business to Business* (B2B), auto-atendimento de clientes, expansão e integração dos sistemas de *Enterprise Resource Planning* (ERP), fusões e aquisições, adequação às mudanças legais, iniciativas de *B*

usiness Intelligence (BI), integração da "*supply chain*" e "*Mobile computing*".

Praticamente todos estes investimentos demandam a necessidade de integração de dados entre os sistemas. Tal integração pode se dar através de iniciativas isoladas, à medida que os processos sejam implantados ou através do planejamento e implantação de uma arquitetura chamada de *Enterprise Application Integration* (EAI).

A arquitetura EAI proporciona uma abordagem sistemática ao problema de integração de sistemas, a partir de soluções que possibilitam a interconexão de transações complexas, percorrendo todo o processo de negócio através de diferentes sistemas e plataformas. O dinamismo do mundo dos negócios requer soluções altamente flexíveis que permitam a integração de dois ambientes (conjunto de sistemas), na maioria das vezes, completamente distintos, tanto no que diz respeito às regras de negócios quanto às plataformas adotadas.

Podemos afirmar que o XML (*Extensible Markup Language*) se tornou um padrão consolidado para troca de dados nas arquiteturas EAI. O padrão XML (XML, 2004) vem sendo amplamente adotado por ser um formato simples de representação de informações. Trata-se de um padrão baseado em texto marcado, que pode ser utilizado em diversas plataformas para integração de sistemas de diferentes corporações (JENSEN, 2003).

1.2 – Motivação

Atualmente, a maioria dos softwares adotados nas empresas (ERPs, *Customer Relationship Management* - CRMs, *Billings*, EAIs) expõe interfaces que permitem a exportação de informações utilizando o padrão XML. As últimas versões dos principais SGBDs (Sistemas de Gerenciamento de Banco de Dados) também permitem a representação dos dados das tabelas em formato XML e as linguagens de programação mais utilizadas (Java, .Net, Delphi) expõe APIs (*Application Program Interfaces*) que manipulam de uma forma prática e rápida qualquer arquivo XML.

Toda essa infra-estrutura para "lidar" com o padrão XML aliada à facilidade de manipulação intrínseca do formato permitiu a ampla adoção do padrão nos sistemas das empresas. Entretanto, cada sistema expõe suas informações em arquivos XML de formatos diferentes. A instância XML com os dados referentes aos filmes mais procurados de um site de vendas *online* pode ter um formato bem diferente da instância XML referente aos dados de bilheteria dos filmes de um determinado cinema em um *shopping center*. A natureza da informação exposta nas duas instâncias é a mesma, entretanto o formato pode ser completamente distinto. Surge então uma grande dificuldade quando o site de vendas online quer integrar de alguma forma o seu negócio com o sistema de bilheteria do cinema. Os dois sistemas não permitem uma integração rápida e direta, apesar de ambos os sistemas terem a capacidade de expor dados no formato XML e importar dados de instâncias XML. Torna-se necessário transformar uma instância XML do sistema de origem dos dados para o formato da instância XML do sistema de destino.

1.3 – Objetivo

O problema de transformação de instâncias XML pode ser dividido basicamente em dois sub-problemas: identificação das correspondências semânticas entre os schemas e transformação estrutural de instâncias de diferentes formatos. Como solução da transformação estrutural de instâncias

XML, vemos atualmente arquiteturas muito acopladas que realizam a transformação de diferentes formatos de instâncias XML pontualmente, ou seja, para cada par de schemas XML é gerado um programa em linguagem de alto nível ou um programa em XSLT (*Extensible Stylesheet Language for Transformations*) para realizar a transformação (XSLT, 2004). Como conseqüência, observamos um grande custo para realizar a manutenção das estruturas de integração construídas, além de um grande trabalho para novas integrações.

O objetivo deste trabalho é propor um algoritmo que automatize a transformação estrutural de instâncias XML de diferentes formatos, permitindo a simplificação do esforço que existe para realizar a integração da informação entre diferentes sistemas.

1.4 – Escopo do trabalho

Este trabalho contempla as transformações estruturais de instâncias XML. Não são tratados no escopo do algoritmo alguns pontos também envolvidos na transformação de documentos XML, tais como o mapeamento semântico entre os elementos, os tipos de dado dos elementos das instâncias XML, os possíveis valores de atributos ou elementos e as operações nos dados mapeados. O mapeamento semântico indica qual informação dos dados de origem da integração corresponde à informação dos dados de destino. Os tipos de dado restringem o conteúdo de um determinado nó ou atributo, de modo que o mesmo só possa conter um valor do tipo numérico, por exemplo. As operações nos dados permitem que um valor numérico possa ser somado com outro em um determinado elemento ou atributo.

O escopo do algoritmo está definido com detalhes no capítulo 3 que apresenta a definição formal, além da sua prova de correção e complexidade.

1.5 – Contribuição do trabalho

A contribuição deste trabalho é um algoritmo correto, que permite automatizar de uma forma rápida e prática, parte do processo de integração dos sistemas de uma ou mais empresas. Algumas ferramentas automatizam este processo, mas a teoria envolvida na transformação é obscura e reservada aos seus respectivos fabricantes. Por outro lado, os artigos publicados de temas semelhantes não expõem um detalhamento como o mostrado neste trabalho.

1.6 – Estrutura da Dissertação

O segundo capítulo deste trabalho ilustra a utilização do padrão XML, os principais *parsers* definidos para manipulação das instâncias XML, o problema da transformação de duas instâncias XML de formatos distintos e algumas soluções para o problema. O terceiro capítulo mostra o algoritmo de transformação I2I, além da prova formal de correção do algoritmo e sua complexidade. No quarto capítulo, uma implementação do algoritmo é apresentada e suas principais funções são listadas. O quinto capítulo avalia a partir de alguns casos reais a eficácia do algoritmo. No sexto capítulo, uma discussão sobre os pontos positivos e negativos é apresentada e finalmente no sétimo são mostrados a conclusão e trabalhos futuros.

2. XML e XML Schemas

Este capítulo detalha o padrão XML e os assuntos relacionados ao mesmo. São mostradas as duas principais especificações de programas de manipulação das instâncias XML: *Document Object Model* (DOM) e *Simple API for XML* (SAX). Outro ponto abordado é a forma de definir um schema de uma determinada instância, que pode ser realizado através de DTDs (*Document Type Definition*) ou XML Schemas. Por último, serão mostradas duas alternativas de representar os dados de uma instância XML em programas procedurais ou orientados a objetos.

2.1 – XML

O XML é um formato de texto simples, derivado do SGML (*Standard Generalized Markup Language*) que foi criado em 1980 pelo pesquisador Charles Goldfarb, e adotado por algumas organizações americanas dentre elas o Departamento de Defesa dos Estados Unidos. O SGML por sua vez é baseado no padrão GML (*Geography Markup Language*) que introduziu pela primeira vez o conceito de formação de tags explicitamente aninhadas (SGML HISTORY, 1990).

O XML foi originalmente desenvolvido para atender à demanda de troca de informações em larga escala, entretanto vem sendo amplamente utilizado para troca de informações de qualquer natureza (XML, 2004). O padrão foi submetido por Bray, Paoli, e Sperberg-McQueen para o *World Wide Web Consortium* (W3C) em 1998 e aprovado no mesmo ano. Inicialmente, a adoção do XML foi pequena por causa da comparação errônea entre o XML e o HTML (*Hypertext Markup Language*). Entretanto, a comunidade da Internet logo percebeu que o XML se aplicava a situações que transcendiam o mundo das *Home Pages*. A adoção do padrão por grandes empresas como IBM, Microsoft, Sun Microsystems, SAP e Software AG, consolidou definitivamente o padrão (DAUM, 2003).

Os arquivos XML são também chamados de instâncias XML e seguem uma regra de formação que tem algumas características presentes em qualquer documento XML. Uma instância XML é composta por um ou mais elementos, que são identificados por um nome declarado entre duas tags, uma inicial e outra final (XML TUTORIAL, DATA). Uma tag corresponde a um nome identificador delimitado pelos caracteres “<” e “>”. O quadro 2.1 exemplifica elementos de um possível documento XML.

```
<?xml version="1.0" encoding="ISO-8859-1‟ ?>
<raiz>
      <elemento atributo=”valor atributo”>valor elemento</elemento>
      <elemento_unico atributo=”valor atributo”/>
</raiz>
```

Quadro 2.1 - Exemplo de DTD

A lista abaixo mostra algumas regras básicas que uma instância XML deve conter para que sua formação esteja correta:

- Uma instância XML deve conter um ou mais elementos.

- Há exatamente um elemento, denominado raiz que não pode ser filho de nenhum outro elemento.

- O nome do elemento contido em uma tag final deve ser exatamente igual ao elemento de uma tag inicial. Os nomes de tags são *case-sensitive*, ou seja, não é permitido uma tag inicial com letras maiúsculas e uma tag final com letras minúsculas.

- Se uma tag inicial estiver contida em outro elemento, a tag final também deve estar contida no mesmo elemento. Os elementos, limitados pelas tags iniciais e finais devem estar aninhados corretamente.

- O final da cada elemento deve ser marcado com a tag final compatível com a tag inicial, que contém exatamente o mesmo nome da tag inicial.

O texto contido entre as tags iniciais e finais é denominado de conteúdo do elemento. Caso o elemento não contenha nenhum conteúdo, pode assumir a seguinte forma: <elemento/>. A barra anterior ao caracter “>” substitui a tag final.

- O nome de um determinado elemento pode conter letras, dígitos, hífens, underscores. O caracter dois pontos pode ser usado somente para designar *namespaces*. *Namespaces* definem um domínio de uma determinada tag.

- Um elemento pode ter nenhum, um ou diversos atributos. Os caracteres permitidos são os mesmos que para nomes dos elementos. O nome do atributo é separado de seu valor por =. O valor do atributo deve ser dado dentro das apóstrofes '... ' ou aspas "...". O mesmo delimitador deve ser usado para início e fim da definição do valor do atributo. Não é possível iniciar um valor de atributo com apóstrofe e terminar com aspas nem vice-versa.

- Alguns caracteres são reservados e devem ser substituídos por expressões definidas.

Caracter	**Expressão de Substituição**
&	&amp
“	&quot
‘	&após
<	<
>	>

Tabela 1 - Caracteres reservados de uma instância XML

- Comentários podem aparecer em qualquer lugar fora das tags dos elementos. Um processador XML pode disponibilizar para uma aplicação o texto dos comentários. O texto "--" (duplo-hífen) não deve ocorrer dentro dos comentários.

- As instâncias XML podem, e devem, começar com uma declaração em XML que especifique a versão de XML que está sendo usada.

A figura 2.1 ilustra um arquivo XML visualizado no *browser* Internet Explorer.

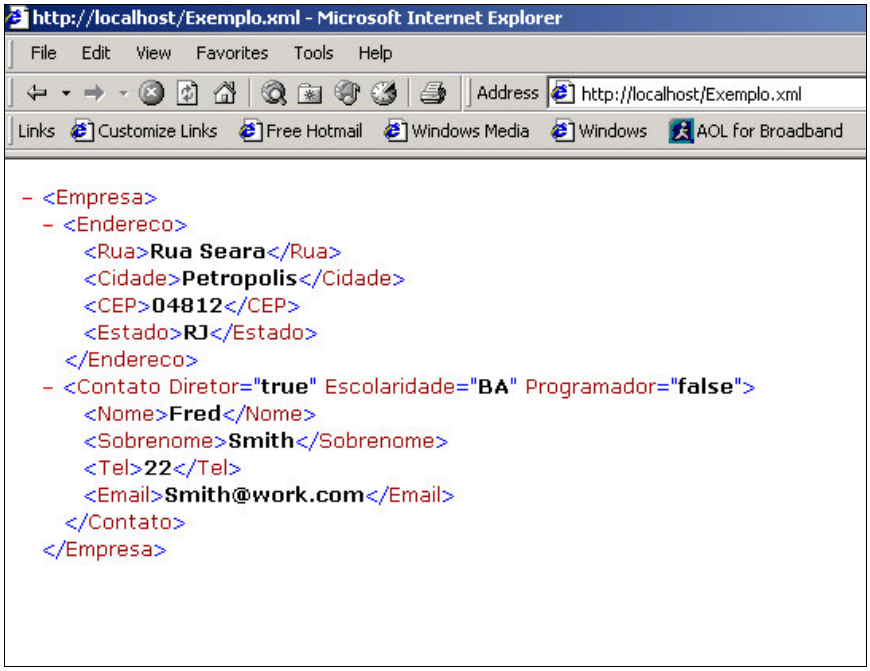

Figura 2.1- Exemplo de arquivo XML visualizado no Internet Explorer

Existem basicamente duas formas de representar informações em uma instância XML: as informações podem estar representadas como conteúdo ou em atributos do elemento. A representação da informação como conteúdo apresenta a restrição de que somente as folhas da árvore podem ter alguma informação e cada elemento representa apenas um dado. Qualquer elemento que contenha pelo menos um filho só pode ter alguma informação se representá-la através de atributos. Assim sendo, a relação entre representação de uma informação como conteúdo ou como um determinado atributo é a mesma para os elementos que são folhas. Porém, a representação em atributos não requer que a informação esteja armazenada necessariamente em folhas, tornando-se assim mais abrangente do que a representada como

conteúdo. Portanto, consideramos neste trabalho apenas as representações feitas em atributos.

2.2 – DTDs e XML Schemas

As linguagens de programação definem tipos e operadores que os programadores podem utilizar com o intuito de garantir uma razoável qualidade no código produzido como, por exemplo, a declaração de uma variável do tipo numérica. A principal vantagem destas definições é a identificação de erros antes da execução dos programas. Os compiladores conseguem identificar erros em tempo de desenvolvimento (SKONNARD, 2003).

O XML é um padrão livre que não define tipo ou operadores. Esta abertura torna o padrão bastante flexível, mas pode ser muito custoso para os sistemas avaliarem o conteúdo das instâncias XML. Portanto, torna-se necessária à definição do formato das instâncias XML, que podemos chamar de *schemas*.

Algumas propostas foram criadas com o intuito de definir schemas para instâncias XML. Duas delas, DTD e XML Schema, ainda são amplamente utilizadas. Este trabalho utiliza a segunda proposta como forma de representar os schemas das instâncias XML, por ser mais atual e abrangente.

2.2.1 – DTD

O DTD foi a primeira proposta amplamente utilizada nos softwares que manipulam instâncias XML. Trata-se, na maioria das vezes, de um arquivo texto definido fora da instância XML e permite a definição basicamente dos elementos e dos atributos existentes na instância (DAUM, 2003).

Podemos identificar alguns pontos negativos da especificação do DTD como:

- Usa outra sintaxe diferente do XML
- Contém apenas um único tipo atômico (PCDATA)

- Não tem tipos compatíveis com o que vemos em linguagens de programação ou databases, como por exemplo: integer, float, data, etc.
- Não permite declarar restrições de domínio.
    - Exemplo: Os valores para o elemento <altura> devem variar entre 0,5 e 3,0.

```
<?xml version="1.0" encoding="ISO-8859-1“ ?>

<!ELEMENT veiculo (tipo,modelo,opcional*)>
<!ATTLIST veiculo
      motorizacao CDATA #REQUIRED
      cor CDATA #REQUIRED
      fabricacao CDATA #REQUIRED
      revisao CDATA #REQUIRED
>
<!ELEMENT tipo (#PCDATA)>
<!ATTLIST tipo fabricante CDATA #REQUIRED>
<!ELEMENT modelo (#PCDATA)>
<!ELEMENT opcional (#PCDATA)>
```

Quadro 2.2 - Exemplo de DTD

2.2.2 – XML Schema

O XML Schema é uma especificação criada e mantida pelo W3C (XML SCHEMA, 2004). A especificação, diferentemente do DTD, segue o padrão XML e permite uma definição maior das informações contidas nas instâncias XML. As vantagens da utilização do XML Schema incluem a sintaxe da especificação que é também em XML, definição de domínios de nomes (*namespaces*), universo de tipos de dados mais amplo, etc.

A especificação permite a definição de tipos simples e complexos (*simple types* e *complex types*). Os tipos simples definem o tipo dos valores que podem ser aplicados para o conteúdo dos elementos ou para determinados atributos. Tipos complexos permitem a combinação de tipos simples em estruturas. Isto torna o XML Schema muito expressivo e capaz de definir schemas de

instâncias XML que são amplamente utilizados tanto nos sistemas EAI quanto nos sistemas que acessam e utilizam Web Services (SKONNARD, 2003).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
      <xs:element name="ano" type="xs:short"/>
      <xs:element name="chassi" type="xs:int"/>
      <xs:element name="cor" type="xs:string"/>
      <xs:element name="marca" type="xs:string"/>
      <xs:element name="modelo">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ano"/>
                        <xs:element ref="marca"/>
                        <xs:element ref="cor"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="veiculo">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="chassi"/>
                        <xs:element ref="modelo"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Quadro 2.3 – Exemplo de XML Schema

Alguns autores abordam o mapeamento de XML Schemas para modelos de classes definidos pela *Unified Modeling Language* (UML). Estes trabalhos fazem a comparação dos tipos e relacionamentos existentes no XML Schema com os tipos contidos na especificação da UML.

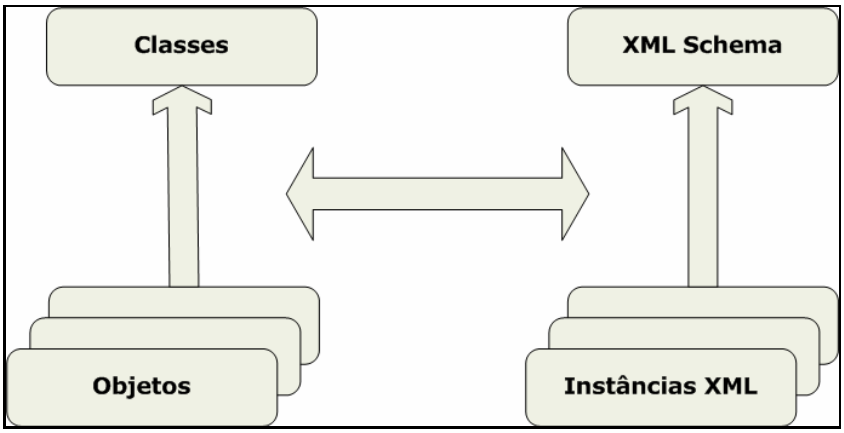

Figura 2.2 - Analogia de XML Schemas com Classes

2.3 – Especificações para Manipulação de XML

Várias especificações foram definidas para manipular instâncias XML. As duas mais utilizadas são o DOM (DOM, 2004) e o SAX (SAX, 2002).

A especificação DOM trata as instâncias XML em forma de objetos e permite o acesso à informação em forma de árvore de nós. Já a especificação SAX trata a instância XML em forma de uma seqüência de eventos (DOM VS SAX, 2003). Consideremos a instância XML definida no quadro 2.4:

```
<?xml version="1.0"?>
<Employees>
      <Employee>
            <EmpID>1</EmpID>
            <EmpName>Ravi</EmpName>
            <Telephone>
                  <House>080-6660666</House>
                  <Office>080-5550555</Office>
            </Telephone>
      </Employee>
      <Employee>
            <EmpID>2</EmpID>
            <EmpName>kiran</EmpName>
            <Telephone>
                  <House>080-3330333</House>
                  <Office>080-4440444</Office>
            </Telephone>
      </Employee>
</Employee>
```

Quadro 2.4 – Manipulação da Instância XML

Um *parser* baseado na especificação DOM geraria a seguinte árvore de objetos no início do processamento:

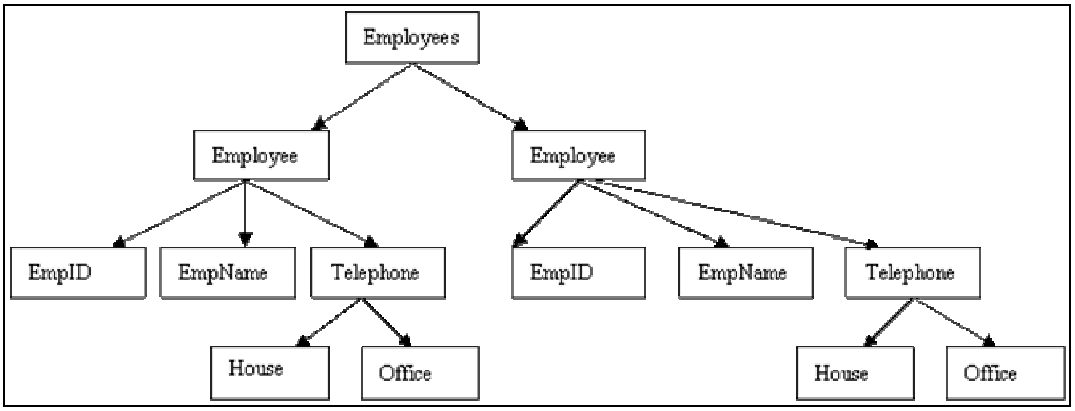

Figura 2.3- Árvore de Objetos de um *parser* DOM

Esta abordagem tem um custo alto de memória para a máquina em que está sendo executado. Este problema não acontece no processamento da instância XML por um *parser* baseado em SAX. Não é necessário ter toda a árvore do

XML em memória, já que a especificação define eventos para o processamento. A figura 2.4 ilustra o processamento de um *parser* SAX.

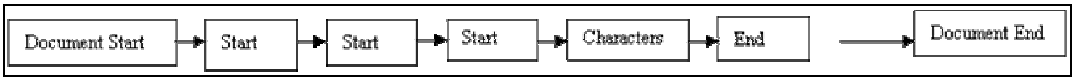

Figura 2.4 – Eventos ocorridos no processamento de um *parser* SAX

2.4 – Representação interna de instâncias XML

Bert Bos sugere que o XML seja tratado como uma gramática livre de contexto sem especificar nenhum terminal, já que o XML já define os terminais intrinsecamente (BOS, 1999). O XML retira a ambigüidade do seu conteúdo ao inserir o nome dos elementos não terminais entre os caracteres “<” e “>”. O autor também compara a sintaxe de uma instância XML com estruturas de dados, e afirma que o XML pode ser tratado diretamente como uma. O quadro 2.5 ilustra um arquivo XML e sua respectiva estrutura definida na linguagem C.

```
<S>
  <node>
    <val>aap</val>
    <type>7</val>
    <left>
      <node>
        <val>noot</val>
        <type>8</type>
      </node>
    </left>
  </node>
</S>

struct node {
    char *val;
    int type;
    struct node *left;
    struct node *right;
}
```

Quadro 2.5 – Estrutura de dados formada a partir de uma instância XML

A Sun Microsystems desenvolveu em 2003 uma arquitetura chamada JAXB (*Java Architeture for XML Binding*). A especificação tem como objetivo básico mapear as informações de instâncias XML para instâncias de objetos (JAXB, 2004).

A arquitetura atende basicamente a quatro conceitos gerais:

- Geração de classes automaticamente.
- Mapeamento dos dados das instâncias XML para o modelo de classes criado.
- Mapeamento dos dados do modelo de classes para instâncias XML.
- Consideração de regras adicionais para criação do modelo de classes.

A API permite que desenvolvedores que manipulam instâncias XML façam o mapeamento de forma transparente para o modelo de classes gerado a partir do XML Schema de um determinado documento. Esta abordagem facilita a manipulação dos dados das instâncias XML e inserção destas informações nos sistemas existentes dentro das companhias. A arquitetura não engloba a questão de transformação de diferentes instâncias XML.

A utilização crescente do JAXB pode eliminar a necessidade de utilização de *parsers* como o DOM e o SAX (MCLAUGHLIN, 2002). A arquitetura já realiza automaticamente a conversão de instâncias XML para o conjunto de classes criado a partir do XML Schema ou do DTD, ou seja, em muitos casos não será necessário manipular as informações das instâncias XML. Estas estarão disponíveis automaticamente nos objetos criados a partir da arquitetura JAXB.

A figura 2.5 apresenta o esquema da arquitetura desenvolvida.

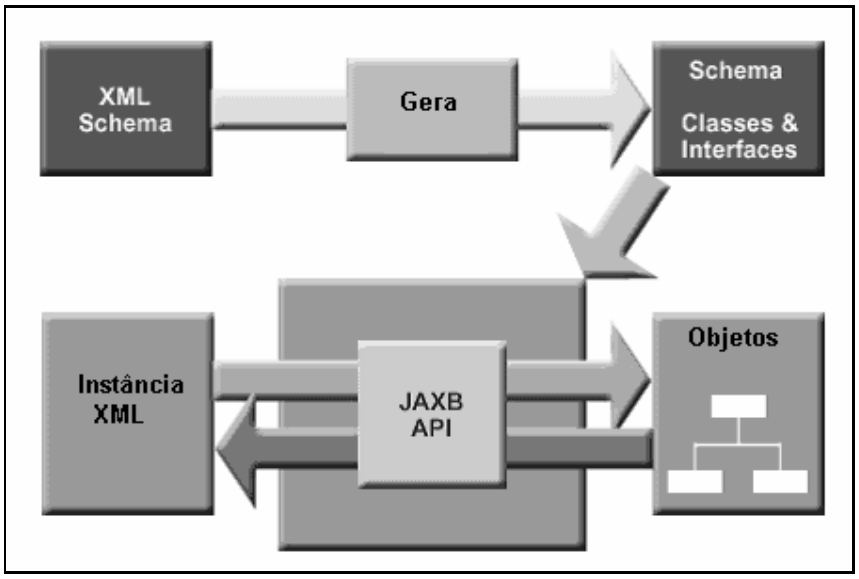

Figura 2.5 - Arquitetura do JAXB

2.5 – O problema da transformação de diferentes Schemas

O problema da transformação de instâncias XML pode ser dividido em duas áreas distintas: identificação das correspondências semânticas entre os schemas e transformação estrutural de instâncias de diferentes formatos.

A identificação das correspondências semânticas entre diferentes *schemas* utiliza geralmente thesaurus e ontologias que auxiliam na identificação de sinônimos ou elementos que têm significados semelhantes dentro de um certo domínio. Alguns trabalhos aliam o retorno do resultado de thesaurus com a análise estrutural dos schemas que estão sendo avaliados. Assim, consegue-se chegar a um resultado mais expressivo com uma maior probabilidade de acerto dos mapeamentos.

A maioria destes trabalhos, que realizam a análise semântica e estrutural dos elementos de dois determinados schemas, gera também um coeficiente que tem como objetivo quantificar o resultado obtido da análise. Este coeficiente é geralmente um número no intervalo de 0 a 1 e resulta muitas vezes em mais de uma solução para a análise. Isto ocorre principalmente quando existem nós opcionais no schema destino. A figura 2.6 ilustra um caso em que mais de um resultado pode ser gerado em uma abordagem do tipo.

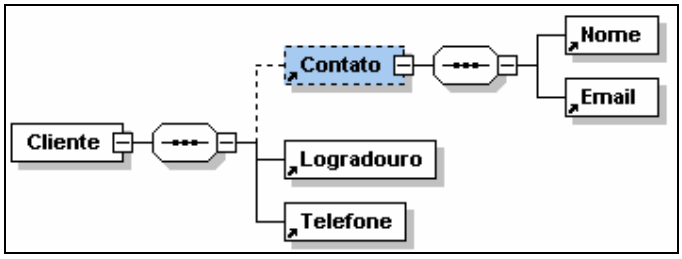

Figura 2.6- XML Schema que pode gerar mais de uma instância como resultado

No XML Schema ilustrado, o elemento `contato` é opcional. Os trabalhos que avaliam um determinado custo para cada operação podem chegar a dois possíveis resultados: o primeiro leva em consideração a criação do nó contato e de seus dois filhos, `nome` e `email`. Um segundo resultado seria a não criação do nó `contato`. Ambas as instâncias estariam em conformidade com o XML Schema destino, porém são diferentes em seu conteúdo. O trabalho aqui proposto aborda o problema de uma forma distinta. O objetivo principal é manter, da melhor maneira possível, a informação de forma consistente na instância destino gerando apenas 1 (um) resultado em todas as transformações. A instância gerada contém o maior número de nós que podem ser transformados. Neste caso, iriam ser gerados o nó contato e seus filhos `nome` e `email`. Acreditamos que o custo de transformação tem um peso muito menor para a integração de dois sistemas do que a informação perdida, mesmo que esta esteja definida como opcional no schema destino.

2.6 – Algumas soluções para o problema

Existem algumas abordagens para solucionar o problema de transformação de instâncias XML. Esta seção apresenta as soluções não automatizadas, os trabalhos publicados na literatura relacionados a este assunto e as ferramentas comerciais que abordam o tema.

2.6.1 – XSLT

O XSLT é um padrão mantido pelo W3C que possibilita a transformação de instâncias XML em vários formatos de arquivo como HTML, outra instância XML, arquivos PDF (*Portable Document Format*) ou praticamente qualquer outro formato desejado (TIDWELL, 2002). Um script XSLT também segue o padrão XML e é composto por regras de transformação denominadas *templates* que são interpretadas por um processador XSLT da seguinte forma: quando uma determinada regra é encontrada na instância XML, uma ação de conversão é tomada pelo interpretador XSLT.

Atualmente o XSLT é amplamente utilizado como solução para a transformação de instâncias XML (HOLMAN, 2000). Além disso, a maioria das últimas versões dos *browsers* já traz interpretadores XSLT de forma embutida. Sendo assim, é possível fazer a transformação de arquivos XML de forma dinâmica e automática através de um *browser*.

Apesar desta ampla adoção do XSLT, a tecnologia não automatiza o processo de transformação de instâncias XML. Para cada par de instâncias XML referentes a XML Schemas distintos, é necessário a ação de um especialista na tecnologia para desenvolver e manter os scripts de transformação. Isto pode gerar uma carga de trabalho muito grande nas empresas que mantém um grande número de sistemas integrados.

O quadro 2.7 mostra um exemplo de programa escrito em XSLT. O script transforma a instância XML do quadro 2.6 em um código HTML que pode ser visualizado em um *browser* padrão (quadro 2.8).

```
<?xml version="1.0"?>
<greeting>
Hello, World!
</greeting>
```

Quadro 2.6 – Instância XML que será transformada

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:apply-templates select="greeting"/>
  </xsl:template>
  <xsl:template match="greeting">
    <html>
    <body>
    <h1>
      <xsl:value-of select="."/>
    </h1>
    </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Quadro 2.7 – Programa em XSLT para transformação de esquemas do Quadro 2.6

```
<html>
<body>
  <h1>
    Hello, World!
  </h1>
</body>
</html>
```

Quadro 2.8 – HTML gerado a partir da transformação do Quadro 2.6

2.6.2 - Trabalhos Acadêmicos

O problema de transformação estrutural de arquivos XML vem sendo estudado por diversos autores. Podemos visualizar tais trabalhos da seguinte forma:

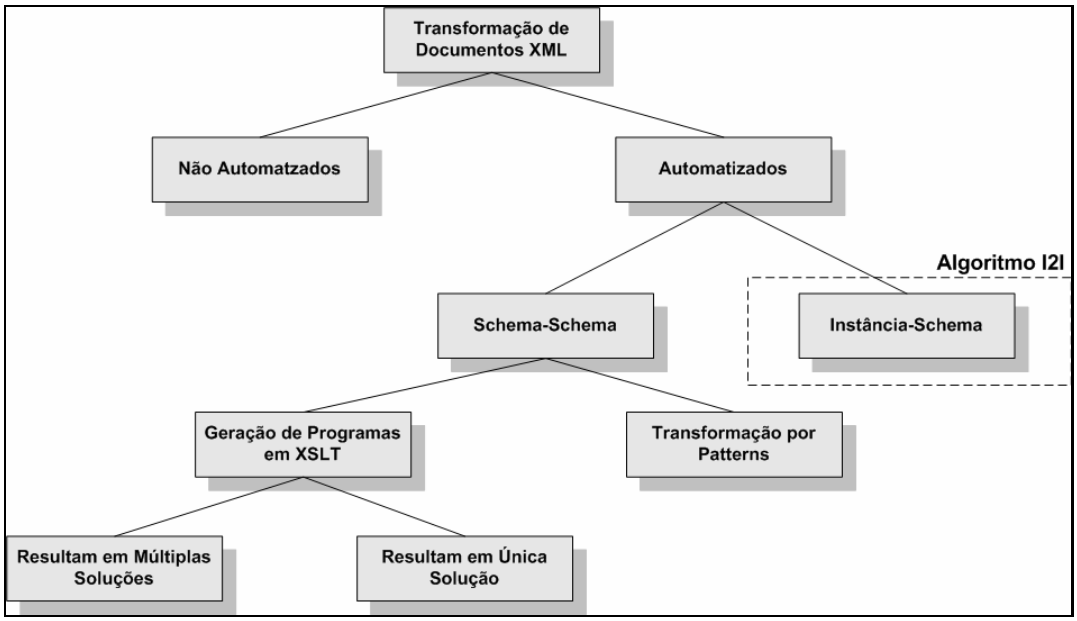

Figura 2.7 - Categorização dos Trabalhos Correlatos

Su Hong descreve um conjunto de transformações que podem ser realizadas entre diferentes DTDs e um algoritmo que gera scripts XSLT de transformação. O artigo define um modelo de custos para cada operação mapeada. O script escolhido para realizar a transformação é aquele que tem o menor custo e permite a transformação das instâncias XML referentes aos DTDs origem e destino. O artigo não aborda o problema semântico da transformação (SU, 2001).

Murata Makoto propõe um método de transformação utilizando "*patterns"* e "*contextual conditions"* a partir dos DTDs origem e destino que são transformados em um autômato de árvore. *Patterns* são regras aplicadas a nós subordinados, enquanto *contextual conditions* são regras aplicadas a ancestrais, irmãos, etc. O artigo cita ainda, a funcionalidade de converter diferentes DTDs (MURATA, 1997).

Na categoria de solução única, Aida Boukottaya sugere a criação de duas camadas ("*semantic layer*" e "*schema layer*") para representar as informações contidas nos schemas. O artigo considera a utilização do XML Schema e define um conjunto de operações entre elementos. O resultado do processo são

alguns mapeamentos entre nós que são utilizados para realizar a criação de scripts XSLT de transformação dos dois XML Schemas definidos (BOUKOTTAYA, 2004).

Bendeck Fawsy sugere um repositório de regras, criadas a partir de um conjunto de exemplos de transformação dos schemas origem e destino. A transformação utiliza scripts XSLT que são refinados e aprimorados com o aumento da base de exemplos. O autor apresenta a prova formal de que o repositório pode ser implementado. Há o uso de algoritmos genéticos para realizar a transformação (BENDECK, 2001).

Elisa Bertino propõe um algoritmo de classificação de instâncias XML, segundo o qual é possível quantificar a aderência de uma instância XML a determinados DTDs (BERTINO, 2004) . Algumas aplicações do algoritmo são listadas: classificação de documentos, evolução da estrutura de DTDs, *queries* estruturadas, filtro personalizado de documentos XML e segurança.

Erhard Rahm define o algoritmo chamado Cupid, que descobre mapeamentos entre elementos de diferentes schemas, baseado nos nomes dos elementos, tipos, constraints, etc (MADHAVAN, 2001). O resultado do algoritmo é uma lista de elementos correspondentes dos DTDs origem e destino. Um estudo comparativo é mostrado no final do artigo com alguns resultados que comprovam a eficiência do trabalho.

2.6.3 – Softwares Comerciais

Alguns softwares desenvolvidos pelas maiores empresas de TI do mundo tratam do problema de integração e implementam algum tipo de transformação envolvendo arquivos XML. O software XML for Tables (XML FOR TABLES, 2003) oferece uma visão do banco de dados como se fosse um documento XML. Desta forma é possível usar a linguagem XQuery (XQUERY, 2004) para ler uma tabela do SGBD. O XQuery é uma especificação do W3C que permite realizar buscas em um arquivo XML.

Um outro software mais complexo e completo que permite a integração entre diferentes sistemas é o BizTalk Server 2004 (BIZTALK, 2004). Nele é possível criar e definir as regras necessárias para a transformação de duas instâncias a partir do schema de cada documento. As regras são definidas no BizTalk Server Mapper de forma visual a partir da árvore hierárquica dos schemas origem e destino. O resultado do mapeamento pode ser um documento XML de outro formato ou até mesmo um relatório. O mapeamento permite o processamento de algum nó que é realizado através de um link a uma função implementada no próprio software. Desta forma, é possível manipular a transformação através de conversões, *loops*, etc.

Cada conjunto de mapeamentos é armazenado no BizTalk Server como um projeto e pode fazer parte de um processo de integração composto por várias outras operações. O projeto pode ser utilizado em mais de um processo e pode ser modificado a qualquer momento, permitindo a manutenção de eventuais mudanças na integração. O resultado do mapeamento são scripts XSLT que realizam a transformação definida no BizTalk Server Mapper.

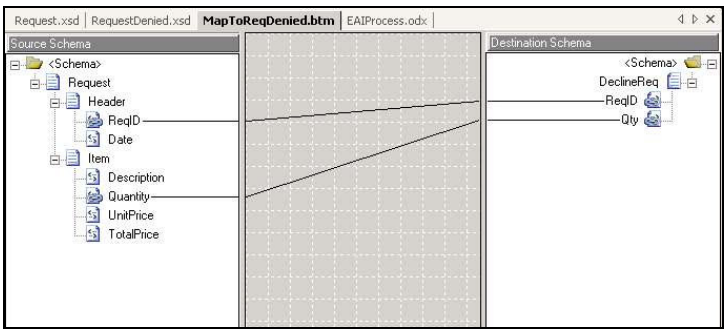

Figura 2.8 - Transformação de instâncias XML do BizTalk Server 2004

Outra ferramenta que implementa a transformação de instâncias XML, a partir dos schemas de origem e destino é o Mapforce (MAPFORCE, 2004). Da mesma forma que o BizTalk Server 2004, o mapforce permite a definição da transformação de instâncias XML de diferentes schemas a partir de um ambiente visual formado a partir dos XML Schemas origem e destino. É possível processar algum elemento a partir de funções definidas pelo usuário.

Como saída do mapeamento gráfico, é possível na versão Enterprise gerar o código de transformação em XSLT, Java, C# ou C++.

O preço destes softwares mais poderosos (BizTalk Server 2004 e Mapforce) que permitem a transformação de instâncias XML é muito elevado (uma licença do BizTalk Server 2004 custa U$ 25.000,00) e o conceito teórico utilizado na geração dos scripts de transformação é obscuro e não está disponível. É difícil definir o escopo de atuação da cada produto.

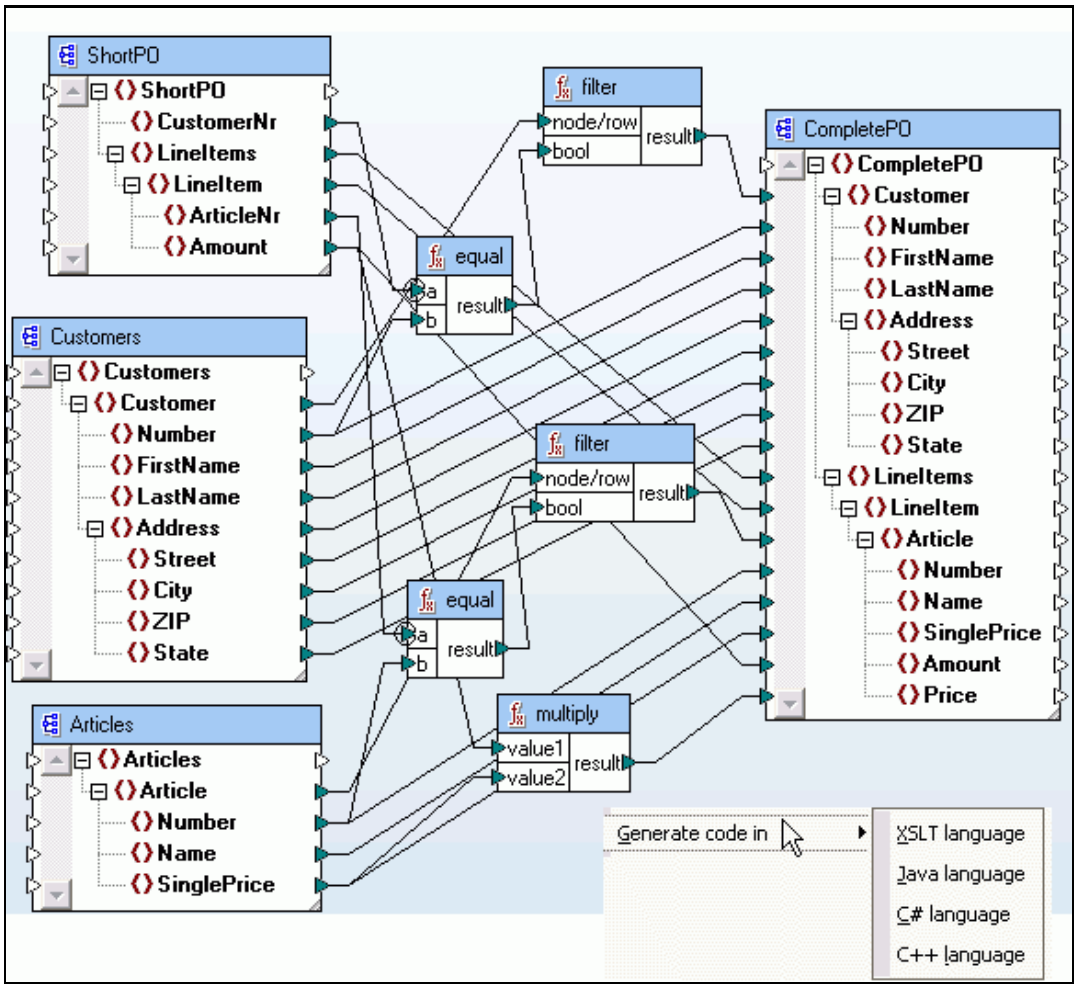

Figura 2.9 - Exemplo de mapeamento do MapForce 2004

3. I2I - Um algoritmo para transformação de instâncias XML

3.1 – Introdução

Este capítulo apresenta a definição formal do algoritmo I2I. O objetivo do algoritmo proposto é automatizar o processo de transformação no que diz respeito à estrutura das instâncias. Não consideramos os problemas semânticos da transformação, que já foram discutidos na literatura como sugerem (MADHAVAN, 2001), (KURGAN, 2002) e (DOAN, 2001). O estudo aborda as relações de parentescos entre nós e os problemas encontrados nas diferentes formas de representar a mesma informação em instâncias XML distintas.

3.2 – Definições

O algoritmo I2I necessita como informações de entrada a *instância XML origem* (*source instance* ou SI) e o XML schema de destino (*target schema* ou TS) e tem como saída a instância XML destino (*target instance* ou TI), que segue as características definidas em TS. Algumas definições são necessárias para definir as características de SI, TS e TI.

SI (ou TI) é descrito em termos de uma árvore com as seguintes propriedades:

- Cada nó da árvore é um elemento da forma

  < nome_elemento valor = v >

  onde valor é o nome do único atributo do elemento e v é o texto contendo um valor plausível para o atributo.

- Todos os elementos identificados pelo mesmo nome ocorrem no mesmo nível da árvore.

- Filhos de elementos identificados pelo mesmo nome ocorrem como irmãos consecutivos.

Podemos dizer que dois nós da árvore *T* contém uma *Referencia Forte* (“SR”) se um deles é ancestral do outro em *T*.

Agora, podemos descrever TS em termos da *árvore de aceitação* com as seguintes propriedades:

1. Todo o nó de TS é identificado por algum elemento de nome ***e*** que ocorre em SI.

2. Dois nós de TS não podem ser identificados pelo mesmo nome de elemento.

3. Nem todo o elemento de SI é necessariamente um nó de TS.

4. A raiz de TS é identificada pelo nome da raiz de SI.

5. Dois nós ***e*** e ***e’*** de TS satisfazem a *Condição de Referência Forte* (“SRC”):

- Se ***e*** e ***e’*** são SR, então qualquer par de elementos *E*, *E’* de SI com o nome respectivo de ***e***, ***e’*** são SR.

- Se ***e*** e ***e’*** não são SR, então qualquer par de elementos *E*, *E’* de SI com o nome respectivo de ***e***, ***e’*** não são SR.

Descreveremos abaixo o algoritmo I2I.

3.3 – Descrição do Algoritmo I2I

Entrada: **SI, TS**
Saída: **TI**

Passo 1 Gera todos os caminhos da forma $P = E_1\ E_2\ ...\ E_{|P|}$ onde E_1 é a raiz da árvore SI, E_{i+1} é o filho de E_i em SI e $E_{|P|}$ é uma folha de SI (a notação $|P|$ é referente ao tamanho de *P*).

Passo 2 Gera todos os caminhos da forma $Q = \boldsymbol{e}_1\ \boldsymbol{e}_2\ ...\ \boldsymbol{e}_{|Q|}$ onde $\boldsymbol{e}_1$ é a raiz de TS, $\boldsymbol{e}_{i+1}$ é o filho de $\boldsymbol{e}_i$ em TS, e $\boldsymbol{e}_{|Q|}$ éuma folha de TS.

Passo 3 Para cada par $P = E_1\ E_2\ ...\ E_{|P|}$, $Q = \boldsymbol{e}_1\ \boldsymbol{e}_2\ ...\ \boldsymbol{e}_{|Q|}$ de caminhos gerados nos Passos 1 e 2, execute as seguintes operações:

Se para todo $\boldsymbol{e}_i$ em *Q* existir um elemento E_k em *P* referenciado por $\boldsymbol{e}_i$ então

Construa o caminho $R = E^1\ E^2\ ...\ E^{|Q|}$ tal que $E^j \in \{ E_1, E_2, ..., E_{|P|} \}$ e E^j é referenciado por $\boldsymbol{e}_j$ para todos os valores $j = 1, 2, ..., |Q|$

Passo 4 Para todos os caminhos $R = E^1\ E^2\ ...\ E^{|R|}$ construídos no Passo 3, gere TI da seguinte forma:

- Como todos os caminhos construídos no Passo 3 começam pelo mesmo elemento E^1 defina E^1 como raiz de TI.

- Tendo colocado um elemento *E* em TI, defina os filhos da seguinte forma:
  - Para cada caminho $R = E^1\ E^2\ ...\ E^{|R|}$ construído no Passo 3:
    Se o caminho corrente de E^1 até *E* em TI corresponde ao sub-caminho $R^j = E^1\ E^2\ ...\ E^j$ de *R*, onde $E = E^j$, então defina E^{j+1} como filho de *E* em TI.
  - Remova filhos *E'* que sejam duplicados, se existir algum.
  - Reordene os filhos de *E'* de uma forma que:
    Se o elemento **e** ocorrer a esquerda de um elemento irmão cujo o nome é **e'** em TS, então todos os filhos com o nome **e** devem ocorrer a esquerda de todos os filhos de nome **e'**.

Fim do Algoritmo

3.4 – Prova de Correção do algoritmo

A transformação de SI para TI é correta quando TI e SI são *compatíveis*, isto é, dado dois elementos *E* e *E'* de TI, a seguinte propriedade é verdadeira:

E e *E'* são SR em SI se e somente se *E* e *E'* são SR em TI.

A corretude do algoritmo I2I é garantida pelo seguinte teorema:

Teorema 1. Após a execução do algoritmo I2I, TI e SI são compatíveis.

Prova: Suponha que *E* e *E'* são dois elementos de TI, e **e** e **e'** são seus respectivos nomes. Assuma primeiramente que *E* e *E'* são SR em SI. Sendo assim, existe um caminho $P = E_1\ E_2\ ...\ E_{|P|}$ gerado no Passo 1 contendo *E* e *E'*. Como **e** e **e'** necessariamente ocorrem em TS, existe um caminho $Q = \boldsymbol{e}_1\ \boldsymbol{e}_2\ ...\ \boldsymbol{e}_{|Q|}$ gerado no Passo 2 contendo ambos. Então, existe um caminho $R = E^1\ E^2\ ...\ E^{|R|}$ construído no Passo 3 contendo *E* e *E'*. Por último, pela construção de TI no Passo 4, existe um caminho descendente em TI contendo *E* e *E'*, isto é, um deles é ancestral do outro. Logo, *E* e *E'* são SR em TI.

Assuma agora que *E* e *E'* são SR em TI. Sendo assim existe um caminho descendente P em TI de, por exemplo, *E* até *E'*. Na construção de TI no Passo 4, existe um caminho $R = E^1\ E^2\ ...\ E^{|R|}$ construído no Passo 3 tal que $E = E_i$ e $E' = E_j$ para $i, j \in \{ 1, 2, ..., |R| \}$ and $i < j$. Entretanto note que, no Passo 3, R foi obtido através do caminho $P = E_1\ E_2\ ...\ E_{|P|}$ gerado no Passo 2. Além disso, P contém $E_i = E$ e $E_j = E'$. Isto significa que existe um caminho ascendente ou descendente de *E* para *E'* em SI, isto é, um deles é ancestral do outro em SI. Logo, *E* e *E'* são SR em SI.

3.5 – Escopo do Algoritmo

O escopo do algoritmo engloba somente instâncias e XML Schemas bem formados que sigam todas as regras de formação definidas por suas respectivas especificações.

O algoritmo considera apenas as informações contidas no atributo nomeado **valor**. A relação entre o nome de um atributo da instância XML de origem e o nome de outro atributo do XML Schema destino deve ser tratado pela análise semântica que não é considerada neste trabalho. O fato de não considerarmos a representação da informação como conteúdo do elemento também não restringe a atuação do algoritmo dado que uma simples transformação resolve esta questão como sugerem os quadros 3.1 e 3.2:

```
<alimentos>
    <frutas>
        <fruta>laranja</fruta>
        <fruta>maça</fruta>
        <fruta>melancia</fruta>
    </frutas>
    <legumes>
        <legume>cenoura</legume>
        <legume>abóbora</legume>
    </legumes>
</alimentos>
```

Quadro 3.1 - Instância XML com informações representadas como conteúdo

```
<alimentos>
    <frutas>
        <fruta valor="laranja"/>
        <fruta valor="maça"/>
        <fruta valor="melancia"/>
    </frutas>
    <legumes>
        <legume valor="cenoura"/>
        <legume valor="abóbora"/>
    </legumes>
</alimentos>
```

Quadro 3.2 - Instância transformada para representar informações em atributos

Uma outra restrição do algoritmo I2I é a representação de apenas um atributo por elemento. Algumas instâncias XML representam mais de um atributo por

elemento, entretanto de forma análoga à representação da informação por conteúdo, é possível desmembrar a instância XML que tenha algum elemento com mais de um atributo em elementos que tenham um atributo apenas. Os quadros 3.3 e 3.4 ilustram o desmembramento sugerido.

```
<alimentos>
    <frutas>
        <fruta nome=”laranja” tipo=”amarga” época=”verão”/>
        <fruta nome=”laranja” tipo=”doce” época=”inverno”/>
        <fruta nome=”melancia” tipo=”doce” época=”primavera”/>
    </frutas>
    <legumes>
        <legume nome=”cenoura” época=”verão”/>
        <legume nome=”abóbora” época=”inverno”/>
    </legumes>
</alimentos>
```

Quadro 3.3 - Instância XML com mais de um atributo por elemento

```
<alimentos>
    <frutas>
        <fruta>
            <nome valor=”laranja”/>
            <tipo valor=”amarga”/>
            <época valor=”verão”/>
        </fruta>
        <fruta>
            <nome valor=”laranja”/>
            <tipo=”doce”/>
            <época=”inverno”/>
        </fruta>
        <fruta>
            <nome valor=”melancia”/>
            <tipo=”doce”/>
            <época=”primavera”/>
        </fruta>
    </frutas>
    <legumes>
        <legume>
            <nome=”cenoura”/>
            <época=”verão”/>
        </legume>
        <legume>
            <nome=”abóbora”/>
            <época=”inverno”/>
        </legume>
    </legumes>
</alimentos>
```

Quadro 3.4 - Instância transformada representando informações em apenas um atributo

Alguns cuidados também são tomados na formação dos parentescos dos elementos, para que não seja gerada uma instância destino aceita pelo seu respectivo XML schema, porém inconsistente em seu conteúdo. Sendo assim, o escopo do algoritmo engloba apenas as transformações entre nós que têm uma referência forte entre si. Um elemento é criado na instância destino se existir pelo menos uma referência forte a todos os seus nós ancestrais na instância origem.

Acreditamos que esta restrição não limita a atuação do algoritmo. O exemplo abaixo mostra o tipo de inconsistência que pode ser montada se o algoritmo não checasse a consistência de toda a árvore de ascendentes.

Suponha que o sistema da universidade A exponha a informação de disciplina e professores de acordo com a instância apresentada no quadro 3.5:

```
<universidade valor="ufrj">
    <ano valor="2002">
        <periodo valor="1">
            <disciplina valor="DISCIPLINA 1">
                  <professor valor="PROFESSOR 1"/>
            </disciplina>
        </periodo>
        <periodo valor="2">
            <disciplina valor="DISCIPLINA 2">
                <professor valor ="PROFESSOR 2"/>
            </disciplina>
        </periodo>
    </ano>
    <ano valor="2003">
        <periodo valor="1">
            <disciplina valor="DISCIPLINA 3">
                <professor valor="PROFESSOR 3"/>
            </disciplina>
        </periodo>
    </ano>
</universidade>
```

Quadro 3.5 - Exemplo de Instância XML de Origem

Na universidade B, os dados são expostos seguindo o schema da figura 3.1:

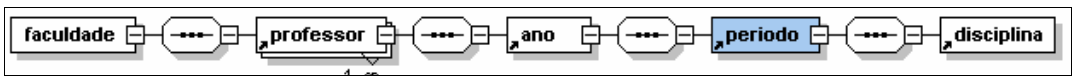

Figura 3.1 - Schema Destino da Universidade B

Caso a verificação da referência forte não fosse realizada, a instância destino montada seguiria o schema destino em sua forma, mas estaria inconsistente

em seu conteúdo. No quadro 3.6 é mostrado o início da árvore destino já inconsistente.

```
<faculdade valor="ufrj">
    <professor valor=”PROFESSOR 1”>
        <ano valor=”2002”>
            <periodo valor=”1”>
                <disciplina valor=”DISCIPLINA 1”/>
            </periodo>
            <periodo valor=”2”>
```

Quadro 3.6 – Instância XML de Destino Inconsistente

3.6 – Complexidade do algoritmo

A complexidade do algoritmo é de $O(n^2)$, onde *n* corresponde ao número de nós que estão sendo transformados. A tabela 2 ilustra a complexidade de cada passo do algoritmo separadamente.

Passo	Descrição	Complexidade
1	O passo 1 percorre todos os caminhos em SI de forma a gerar todos os caminhos da instância origem.	**O(n)**
2	O passo 2 percorre todos os caminhos em TS de forma a gerar todos os caminhos do schema destino.	**O(n)**
3	O passo 3 checa para cada caminho $P = E_1 E_2 \ldots E_{\lvert P \rvert}$ se há correspondência nos caminhos $R = E^1 E^2 \ldots E^{\lvert R \rvert}$.	**$O(p * q) = O(n^2)$** , onde: p - número de folhas da árvore de origem. q – número de folhas da árvore de destino. n - máximo do número de folhas da origem e destino.
4	O passo 4 define todos os filhos de cada nó e reordena a árvore criada.	**O(n)**

Tabela 2- Passos do Algoritmo de transformação

4. Uma implementação

4.1 – Introdução

A implementação do algoritmo recebe como parâmetros de entrada: o XML Schema do arquivo de destino denominado *schema destino*, a instância XML que deve ser transformada denominada *instância origem* e um conjunto de informações sobre os mapeamentos entre os nós denominado *mapeamento dos nós*. O resultado do processo é a instância XML denominada *instância destino*, que atende às especificações do schema destino.

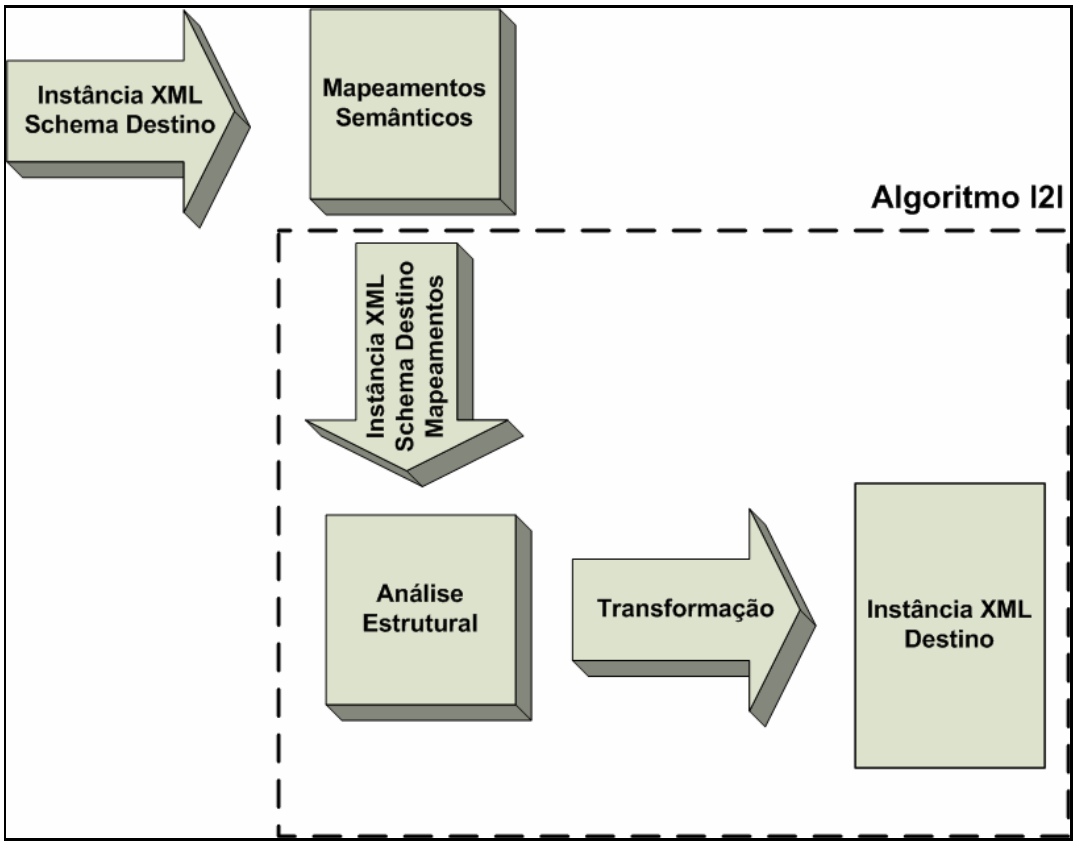

Figura 4.1- Implementação do algoritmo I2I

O arquivo de mapeamentos é simples e apresenta apenas informações a respeito dos elementos correspondentes entre a instância origem e o schema destino. Esta solução poderia facilmente ser combinada com outro trabalho que tenha como objetivo realizar o mapeamento semântico de arquivos XML. Desta forma, os nós não precisam ter necessariamente o mesmo nome.

A transformação é realizada com o auxílio de um banco de dados relacional. A partir da instância origem, uma tabela desnormalizada é criada no banco de dados, onde cada coluna da tabela corresponde a uma ocorrência dos nós distintos da instância XML. A partir da tabela gerada, é possível inserir e armazenar todo o conteúdo da instância origem. No final deste processamento a tabela terá exatamente n registros, onde n corresponde ao número de percursos da raiz do arquivo XML às folhas da árvore. Todos os nós criados obedecem ao conceito de referência forte.

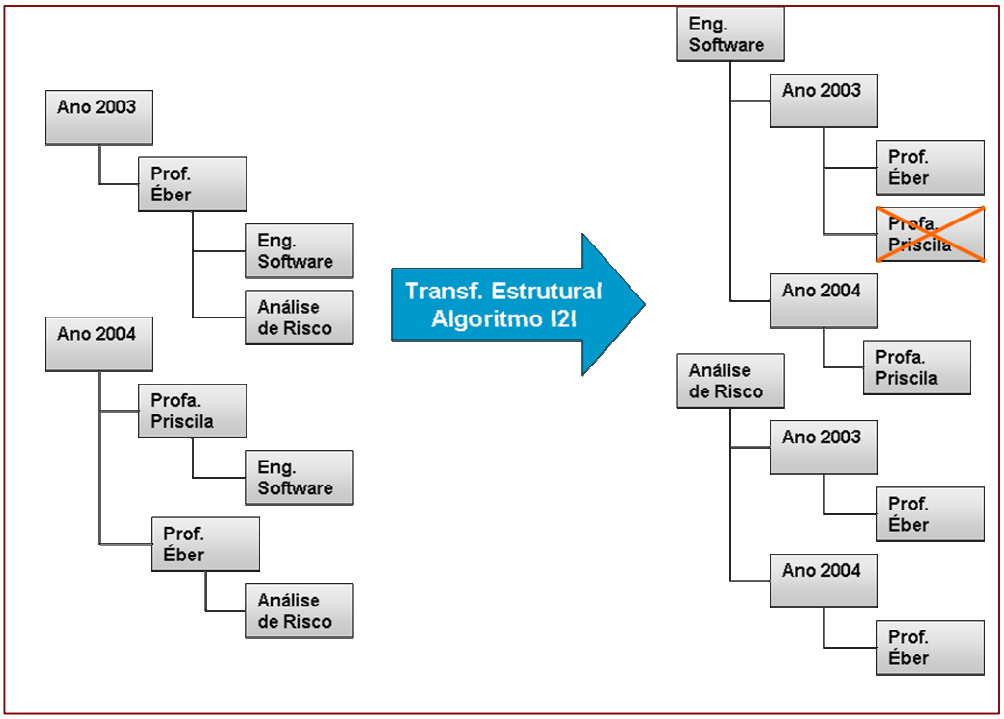

Figura 4.2- Ilustração do conceito de Referência Forte

Para o schema destino é executado um processo distinto. É necessário descobrir a partir do XML Schema fornecido, o que chamamos de "Árvore de Aceitação". Esta árvore de aceitação garante a consistência da instância que será gerada com o seu schema correspondente. Esta estrutura contém todos os nós e parentescos que estão definidos no XML Schema. A representação da árvore de aceitação informa apenas a estrutura do documento.

4.2 – Escopo do Algoritmo Extendido

Algumas funcionalidades foram inseridas na implementação do algoritmo de forma que este consiga contemplar a transformação de um número maior de instâncias XML.

A implementação considera o que denominamos de nós de categorização. É muito comum existir nas instâncias XML elementos que não representam nenhuma informação na árvore hierárquica da instância. Estes nós desempenham um papel organizacional dentro da estrutura formada e são uma espécie de categorização de seus filhos.

Um elemento A é considerado um nó de categorização na instância destino se e somente se não existir nenhum atributo definido para este elemento no schema destino.

Nestes casos, o algoritmo não realiza a checagem de referência forte, já que o elemento não representa nenhuma informação.

Por último, a implementação também considera na formação da instância destino os elementos que são definidos em seus respectivos schemas como opcionais. A checagem de referência forte ocorre apenas no caso do elemento avaliado ter que existir pelo menos uma vez na instância destino (elemento obrigatório – cardinalidade 1:N). Caso o elemento seja opcional, o algoritmo não é abortado se a referência não existir na instância origem. Neste caso toda a sub-árvore deste elemento opcional não existente na instância origem é descartada.

4.3 – Passos do Algoritmo

O algoritmo de transformação pode ser dividido, basicamente, em sete passos que são apresentados e detalhados na tabela 3:

Passo do Algoritmo	Passo da Implementação	Descrição
1	1.1	Retorna os elementos da instância origem.
1	1.2	Cria tabela origem no Banco de Dados.
1	1.3	Insere dados da instância recebida na tabela origem.
2	2	Retorna os elementos do XML Schema destino.
3	3.1	Cria tabela destino no Banco de Dados.
3	3.2	Insere dados da tabela origem na tabela destino.
4	4	Cria instância destino, a partir da tabela destino.

Tabela 3- Passos do Algoritmo de transformação

Retorna os elementos da instância origem

Percorre a instância origem e para a primeira ocorrência de cada nó distinto encontrado, armazena o nome do nó em um array.

```
Função colunasTabela(instancia)
  conjuntoNós := instancia.primeiraOcorrenciaNós()
  i := 0
  Para nó ∈ conjuntoNós faça
    nomeNó := nó.retornaNome()
    arrayNomes[i] := nomeNó
    i := i + 1
  colunasTabela := arrayNomes
```

Quadro 4.1 - Função colunasTabela

Criar tabela origem no Banco de Dados

A partir do array de nós gerados no passo 1.1, é possível criar a tabela que irá conter todas as informações contidas na instância XML de origem

```
Procedimento criaTabela(arrayNomes, nomeTabela)
  scriptTabela := "CREATE TABLE (" + nomeTabela +  "
  Para nomeColuna ∈ arrayNomes faça
    scriptTabela := scriptTabela + nomeColuna + " VARCHAR,"
  scriptTabela := scriptTabela.subtexto(1, tam(scriptTabela))
  scriptTabela := scriptTabela + ")"
  executaSql(scriptTabela)
```

Quadro 4.2 - Procedimento criaTabela

Insere dados da instância recebida na tabela origem

Este passo percorre a árvore origem utilizando um algoritmo de busca em profundidade. Toda a vez que o algoritmo encontra uma folha um *insert* contendo todas as informações coletadas desde o nó raiz até a folha é gerado e executado no banco de dados.

```
procedimento insereDadoDatabase(elemento, coluna, valor)
  atributo := elemento.atributo(CONSTANTE_ATRIBUTO)
  se (atributo != nulo)
    coluna = coluna + elemento.retornaColuna() + ","
    valorTmp = atributo.retornaValor()
    se (valorTmp = nulo)
      valorTmp = "NULL"
    senão
      valorTmp = "'" + valorTmp + "'"
    valor = valor + valorTmp + ","
  lista = elemento.retornaFilhos()
  flagFolha = true
  Para elementos ∈ lista faça
    flagFolha = false
    insereDadoDatabase(elemento, coluna, valor)
  se (flagFolha)
    coluna = coluna.substring(0, coluna.length() - 1)
    valor = valor.substring(0, valor.length() - 1)
    listaColunas.adiciona(coluna)
    listaValores.adiciona(valor)
```

Quadro 4.3 - Procedimento insereDadoDatabase

Ao final do procedimento, as variáveis `listaColunas` e `listaValores` contém todas as informações necessárias para inserir os dados na tabela criada. Um *loop* é executado nas duas listas e um *insert* é gerado a cada iteração.

Retorna os elementos do XML Schema destino

Neste passo, é criada a árvore de aceitação do schema destino, onde todos os possíveis nós da instância XML estão relacionados com seus descendentes e ascendentes. Este passo está dividido em dois sub-passos:

O primeiro sub-passo consiste na leitura do schema destino e armazenamento das suas principais informações em tabelas de dispersão. O procedimento mostrado no quadro 4.4 guarda em uma tabela de dispersão nomeada elementos o elemento que está sendo lido e em outra tabela de dispersão nomeada metadadosNos os seu respectivos metadados. Em ambas as tabelas de dispersão, a chave utilizada é o nome do elemento. O procedimento também armazena em um vetor nomeado nósClassificação os elementos que são classificatórios e em um array nomeado arrayNomes o nome de cada elemento.

```
Procedimento arvoreAceitacao(schema)
  conjuntoNos := schema.retornaNos()
  i := 0
  Para nó ∈ conjuntoNos faça
    nome := nó.retornaNome()
    metadado := nó.retornaMetaDado()
    elementos.insere(nome, nó)
    metadadosNos.insere(nome, metadado)
    se (metadado.noClassificacao())
      nósClassificação.adiciona(nome)
    arrayNomes[i] := nome
    i := i + 1
```

Quadro 4.4 - Procedimento arvoreAceitacao

O segundo sub-passo consiste em percorrer todos os elementos de `metadadosNos` e avaliar o metadado de cada elemento. Todas as relações de paternidade são montadas neste passo. Para cada relação, o algoritmo obtém o nó pai e o nó filho da tabela de dispersão e inclui a relação no próprio objeto. Além disso, a função armazena na tabela de dispersão `cardinalidades` a cardinalidade de cada elemento.

Ao final deste passo, a árvore de aceitação do schema destino foi gerada com todas as relações de parentesco definidas nos próprios objetos.

```
Função relacionaNós(elementos, metadadosNos)
  Para nome ∈ elementos faça
    metadado := metadadosNos.retorna(nome)
    cardinalidades.insere(nome, metadado.retornaCardinalidade())
    nó := elementos.retorna(nome)
    listaNomeFilhos := metadado.retornaFilhos()
    Para nomeFilho ∈ listaNomeFilhos faça
      nóFilho := elementos.retorna(nomeFilho)
      nó.adicionaFilho(nóFilho)
```

Quadro 4.5 - Função relacionaNós

Cria tabela destino no Banco de Dados

Os mesmos passos listados no item 1.2 são executados. A diferença é que a variável `arrayNomes` no item 1 refere-se à instância origem. Neste caso, refere-se ao schema destino.

Insere dados da tabela origem na tabela destino

Neste passo, o algoritmo contém todas as informações referentes às colunas criadas na tabela `origem` e na tabela `destino`. A partir da consulta aos mapeamentos dos elementos fornecidos como parâmetro de entrada do algoritmo é possível montar um comando SQL para inserir dados da tabela origem na tabela destino. A tabela destino é preenchida a partir da execução da seguinte query:

```
INSERT INTO DESTINO([CAMPOS DO SCHEMA DESTINO])
SELECT [CAMPOS DO SCHEMA ORIGEM] FROM ORIGEM
```

Quadro 4.6 - Comando SQL que insere dados no database

Cria instância destino, a partir da tabela destino

A instância destino é gerada a partir da execução de uma série de consultas à tabela `destino`. A instância XML é criada seguindo a árvore de aceitação gerada no item 4. A variável `nóAceitação` é referente à árvore de aceitação, enquanto `nóInstancia` é a variável relativa a instância que está sendo criada (na primeira chamada essa variável é nula).

A função `retornaCondição` monta a condição que permite a criação da árvore seguindo o conceito de referência forte. Além disso, é nesta função que os nós classificatórios são analisados. Caso sejam identificados, estes não entram na cláusula da consulta que é executada na tabela `destino`.

A função `checaAncestralOpcional` é chamada para os casos em que um elemento retornado da árvore de aceitação não pode ser montado por não ter uma referência forte aos seus ancestrais. Neste caso, o algoritmo verifica se o próprio elemento ou se algum ancestral é opcional. O elemento ancestral opcional mais próximo é retirado da árvore. Caso contrário, um erro é gerado indicando que não foi possível montar o elemento em questão.

```
Procedimento criaInstancia(nóAceitação, nóInstancia)
  se (nóAceitação ∈ nósClassificação)
    nó = new Nó(nóAceitação.retornaNome())
    listaTmp = nóAceitação. retornaFilhos();
    Para cada nóTmp ∈ listaTmp faça
       criaInstancia(nóTmp, nó)
       retorna
  cond := " WHERE " + nóAceitação.retornaNome() + " IS NOT NULL"
  cond := retornaCondição(cond, nóInstancia)
  sql = " SELECT DISTINCT " + nóAceitação.retornaNome() +
        " FROM DESTINO " + cond
  lista := executaSql(sql)
  flag = verdadeiro
  Para linha ∈ listas faça
    flag = falso
    nó = new Nó(nóAceitação.getName())
    nó.insereAtributo("valor", linha.retornaValor());
    se (nóInstancia = null)
      listaDestino.add(nóInstancia);
    senão
      nóInstancia.adicionaFilho(nó);
    listaTmp = nóAceitação.retornaFilhos();
    Para nóTmp ∈ listaTmp faça
       criaInstancia(nóTmp, nó)
  se (flag)
    nomeTmp = nóAceitação.retornaNome()
    cardinalidade = cardinalidades.retorna(nomeTmp)
    se (cardinalidade != 0)
      elementoOpcional = checaAncestralOpcional(nóAceitação)
      se (elementoOpcional = nulo)
        retorna Erro("Não é possível criar o elemento" + nomeTmp )
      senão
        nóInstancia.retiraAncestral(elementoOpcional)
```

Quadro 4.7 - Procedimento criaInstancia

```
Função retornaCondição (cond, nóInstancia)
  se (nóInstancia = nulo) então
    retornaCondição := cond
  senão se (not nóInstancia ∈ nósClassificação)
    valor := nóInstancia.retornaAtt("valor").retornaValor()
    cond := cond + " AND " + nóInstancia.retornaNome() +
           " = '" + valor + "'"
  nóPaiInstancia := nóInstancia.retornaPai()
  se (nóPaiInstancia = nulo)então
    retornaCondição := cond
  senão
    retornaCondição := retornaCondição (cond, nóPaiInstancia)
```

Quadro 4.8 - Função retornaCondição

```
Função checaAncestralOpcional (nóAceitação)
  nóPai = nóAceitação.getPai()
  se (nóPai = nulo)
    checaAncestralOpcional := nulo
  cardinalidade = cardinalidades.retorna(nóPai.retornaNome())
  se (cardinalidade = 0)
    checaAncestralOpcional := nóPai
  senão
    checaAncestralOpcional := checaAncestralOpcional(nóPai)
```

Quadro 4.9 - Função checaAncestral

5. Avaliação experimental do algoritmo

5.1 – Introdução

A avaliação do algoritmo será feita através de 5 exemplos baseados em possíveis sistemas de grandes empresas e universidades.

5.2 – Metodologia

Cada caso é detalhado nos sete passos listados na implementação do algoritmo I2I que foi definido no capítulo 4.

Para criar os casos de teste do algoritmo montou-se uma parte do modelo de dados de sistemas que podem existir em empresas da área de TI (alguns casos foram retirados de sistemas existentes). O XML Schema destino então foi mapeado de acordo com o modelo criado e a instância origem seguiu sempre um padrão completamente distinto. O arquivo de mapeamentos semânticos também foi fornecido, dado que o algoritmo I2I não aborda esta questão. Cada passo do algoritmo foi executado para os modelos montados e o resultado de cada um é mostrado na próxima seção.

O caso da seção 5.3.2 foi retirado de um artigo publicado que aborda o problema da transformação de diferentes schemas em sistemas que integram ambientes de *Business to Business*.

O caso da seção 5.3.5 foi retirado de dois catálogos de instâncias XML focados em sistemas de negócios. Os dois sites *Commerce XML Resources* - cXML (CXML, 2004) e *XML Common Business Library* - xCBL (XCBL, 2004) sugerem modelos de arquivos para sistemas existentes nas empresas e são uma ótima fonte de exemplos para estudos baseados em XML Schemas e instâncias XML.

Como produto final de todos os casos temos a instância destino criada, a qual seria utilizada para importar os dados na interface de integração do sistema destino.

5.3 – Resultados

Os resultados de cada caso são detalhados em cada sub-seção de acordo com a metodologia adotada.

5.3.1 – Caso 1: Integração Empresa - Fornecedor

O primeiro caso se refere à integração entre empresa e fornecedor. Documentos de diferentes organizações geralmente contêm as mesmas informações dispostas de formas distintas (BELLASHENE, 2003). O sistema da empresa expõe as informações centralizadas por cliente e necessita automatizar o processo de pedidos de produtos com o fornecedor, que centraliza as informações por produto. Os dois XML Schemas estão exemplificados nas figuras 5.1 e 5.2. O XML Schema de origem não é fornecido como dado de entrada do algoritmo. Este é mostrado apenas para ilustrar o problema de uma forma mais detalhada.

‘

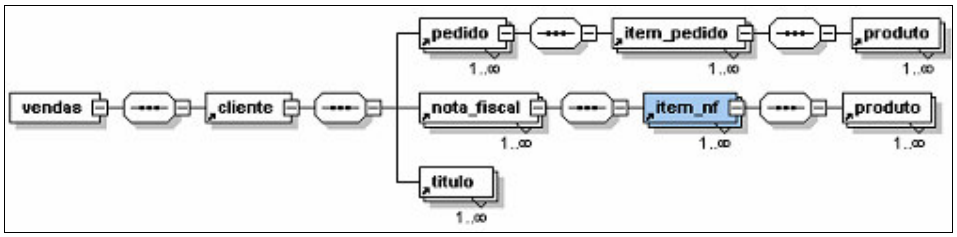

Figura 5.1 - Schema da Empresa (Schema Origem)

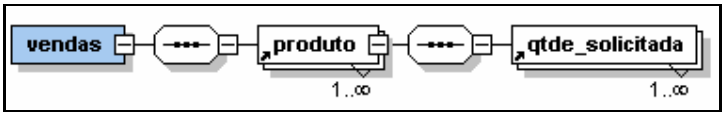

Figura 5.2 - Schema do Fornecedor (Schema Destino)

Certamente uma solução para o caso seria o uso de XSLT para realizar a transformação. Imaginemos que o cliente do exemplo queira automatizar todo o seu processo de troca de informações com seus fornecedores. Se o número de

fornecedores for superior a 30, seriam gerados pelo menos mais de 30 programas para realizar a integração. Utilizando o método proposto neste artigo, seriam gerados apenas 30 arquivos de mapeamentos para chegar ao mesmo resultado. O arquivo de mapeamentos utilizado no exemplo é mostrado no quadro 5.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
    <dados>
        <origem path="http://servidor/Vendas.xml" alias="mod1"/>
        <destino path="http://servidor/Destino.xsd" alias="mod2"/>
    </dados>
    <sinonimos>
        <sinonimo>
            <nodeOrigem source="mod1">vendas</nodeOrigem>
            <nodeDestino source="mod2">vendas</nodeDestino>
         </sinonimo>
        <sinonimo>
            <nodeOrigem source="mod1">produto</nodeOrigem>
            <nodeDestino source="mod2">produto</nodeDestino>
        </sinonimo>
         <sinonimo>
            <nodeOrigem source="mod1">item_pedido</nodeOrigem>
            <nodeDestino source="mod2">qtde_solicitada</nodeDestino>
        </sinonimo>
    </sinonimos>
</definition>
```

Quadro 5.1 - Mapeamentos dos nós do caso 1

O XML de entrada segue a estrutura do schema da empresa e é mostrado a seguir.

```
<?xml version="1.0" encoding="UTF-8"?>
<vendas valor="VENDAS">
    <cliente valor="CLIENTE 1">
        <pedido valor="PEDIDO 1">
            <item_pedido valor="5">
                 <produto valor="PRODUTO 1"/>
            </item_pedido>
            <item_pedido valor="10">
                <produto valor="PRODUTO 2"/>
            </item_pedido>
        </pedido>
        <pedido valor="PEDIDO 2">
            <item_pedido valor="20">
                <produto valor="PRODUTO 1"/>
            </item_pedido>
        </pedido>
        <nota_fiscal valor="NOTA FISCAL 1">
            <item_nf valor="ITEM NOTA FISCAL 1">
                <produto valor="PRODUTO 1"/>
            </item_nf>
        </nota_fiscal>
        <titulo valor="TITULO 1"/>
      </cliente>
</vendas>
```

Quadro 5.2 - Exemplo de Instância XML de Origem

Nos passos 1.1 e 1.2, o método de transformação lê as informações da instância origem e cria no banco de dados uma tabela nomeada ORIGEM.

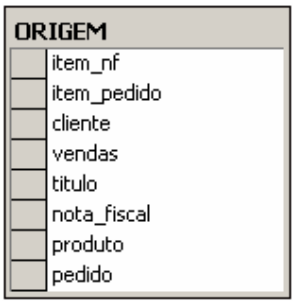

Figura 5.3 - Tabela Origem criada no Banco de Dados

A partir da tabela montada, é possível ler os dados da instância origem e inserir todas as informações na tabela. A função do passo 1.3 é chamada

recursivamente n vezes, onde n é o número de nós da instância origem. Todos os nós da instância origem são visitados neste passo. O primeiro insert gerado acontece quando o nó `produto` com o valor "`PRODUTO 1`" é encontrado. Neste ponto, o insert mostrado no quadro 5.3 é gerado e executado no banco de dados.

```
INSERT INTO ORIGEM (vendas, cliente, pedido, item_pedido, produto)
values('VENDAS', 'CLIENTE 1', 'PEDIDO 1', '5', 'PRODUTO 1')
```

Quadro 5.3 - Primeiro comando SQL gerado pelo algoritmo

O último nó visitado é o nó `título` e o insert do quadro 5.4 é executado finalizando o passo 1.3.

```
INSERT INTO ORIGEM (vendas,cliente,titulo)
values('VENDAS','CLIENTE 1','TITULO 1')
```

Quadro 5.4 - Último comando SQL gerado pelo algoritmo

A partir deste ponto, a tabela `ORIGEM` encontra-se populada. A figura 5.4 ilustra os dados em suas respectivas colunas.

item_nf	item_pedido	cliente	vendas	titulo	nota_fiscal	produto	pedido
<NULL>	5	CLIENTE 1	VENDAS	<NULL>	<NULL>	PRODUTO 1	PEDIDO 1
<NULL>	10	CLIENTE 1	VENDAS	<NULL>	<NULL>	PRODUTO 2	PEDIDO 1
<NULL>	20	CLIENTE 1	VENDAS	<NULL>	<NULL>	PRODUTO 1	PEDIDO 2
ITEM NOTA FISCAL	<NULL>	CLIENTE 1	VENDAS	<NULL>	NOTA FISCAL 1	PRODUTO 1	<NULL>
<NULL>	<NULL>	CLIENTE 1	VENDAS	TITULO 1	<NULL>	<NULL>	<NULL>

Figura 5.4 - Tabela Origem Populada

O passo 2 consiste em montar a árvore de aceitação, a partir do schema destino. As funções `arvoreAceitacao` e `relacionaNos` percorrem os nós definidos no schema destino uma vez cada, formando os relacionamentos definidos. O quadro 5.5 mostra a árvore de aceitação do caso real.

```
<vendas>
    <produto>
        <qtde_solicitada/>
    </produto>
</vendas>
```

Quadro 5.5 - Árvore de Aceitação do exemplo

Os passos 3.1 e 3.2 criam a tabela `destino` no banco de dados e a populam a partir da tabela `origem`. As figuras 5.5 e 5.6 mostram a tabela criada e populada.

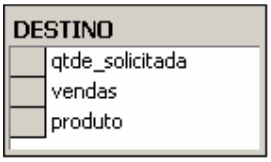

Figura 5.5 - Tabela Destino criada no Banco de Dados

	qtde_solicitada	vendas	produto
	10	VENDAS	PRODUTO 1
	20	VENDAS	PRODUTO 2
	30	VENDAS	PRODUTO 1
	<NULL>	VENDAS	PRODUTO 1
	<NULL>	VENDAS	PRODUTO 2
	<NULL>	VENDAS	<NULL>

Figura 5.6 - Tabela Destino Populada

O passo 4 monta a instância destino a partir da árvore de aceitação e da tabela `destino` populada. A instância destino é mostrada a seguir.

```
<?xml version="1.0" encoding="UTF-8"?>
<vendas valor="VENDAS">
    <produto valor="PRODUTO 1">
        <qtde_solicitada valor="20"/>
        <qtde_solicitada valor="5"/>
    </produto>
    <produto valor="PRODUTO 2">
        <qtde_solicitada valor="10"/>
    </produto>
</vendas>
```

Quadro 5.6 - Instância XML destino gerada pelo algoritmo

No estudo do caso 1 foi coletado o tempo de execução do algoritmo para mais de uma instância XML de entrada. Os tempos variam muito pouco com o aumento do número de nós transformados. Isto indica que o tempo efetivo de transformação do algoritmo é satisfatório até mesmo para um volume maior de dados, pois os pontos onde existe o maior consumo do tempo de processamento são a criação do banco de dados e das tabelas relativas a transformação. O micro utilizado foi um Athlon AMD 2.4 Ghz com 512 Mb de memória RAM com o sistema operacional Windows XP. O banco de dados utilizado foi o SQL Server 2000 e o tempo de resposta do aplicativo foi de 2,1 segundos para a instância listada no exemplo. Triplicando o número de nós da instância XML de entrada resultou em um tempo de 2,25 segundos.

5.3.2 – Caso 2: Catálogo de Produtos de Informática

Um segundo exemplo retirado do artigo “An Analysis of B2B Catalogue Integration Problems. Content and Document Integration” (OMELAYENKO, 2002) ilustra a automatização da transformação de duas instâncias XML relacionadas a dois catálogos de produtos distintos. Os dois fragmentos citados no artigo foram ligeiramente modificados (basicamente os valores dos atributos ou dos elementos foram migrados para a notação deste trabalho e estão contidos no atributo `valor` de cada elemento). O objetivo é transformar uma instância na outra.

O XML Schema destino foi criado a partir de uma das instâncias e um arquivo de mapeamentos foi gerado. Os três parâmetros de entrada do algoritmo estão exemplificados na figura 5.7 e nos quadros 5.7 e 5.8.

```
<?xml version="1.0" encoding="UTF-8"?>
<xCbl>
      <CatalogSchema>
            <SchemaVersion valor="1.0"/>
            <SchemaStandard valor="UNSPSC"/>
      </CatalogSchema>
      <Product_Type valor="good"/>
      <Product valor="C43171801">
            <ProductID valor="140141-002"/>
            <Manufacturer valor="Compaq"/>
            <CountryOfOrigin>
                  <Country>
                        <CountryCoded valor="US"/>
                  </Country>
            </CountryOfOrigin>
            <ShortDescription valor="Armada M700 PIII 500 12GB"/>
            <LongDescription valor="This light…"/>
            <ObjectAttribute>
                  <AttributeID valor="Processor Speed"/>
                  <AttributeValue valor="500MHZ"/>
            </ObjectAttribute>
            <ProductVendorDataPartnerRef valor="Acme_Laptops">
                  <VendorPartNumber valor="12345"/>
                  <ProductPrice>
                        <Amount valor="1000"/>
                        <Currency>
                              <CurrencyCoded valor="USD"/>
                        </Currency>
                  </ProductPrice>
            </ProductVendorDataPartnerRef>
      </Product>
</xCbl>
```

Quadro 5.7 - Exemplo de Instância XML de Origem do xCbl

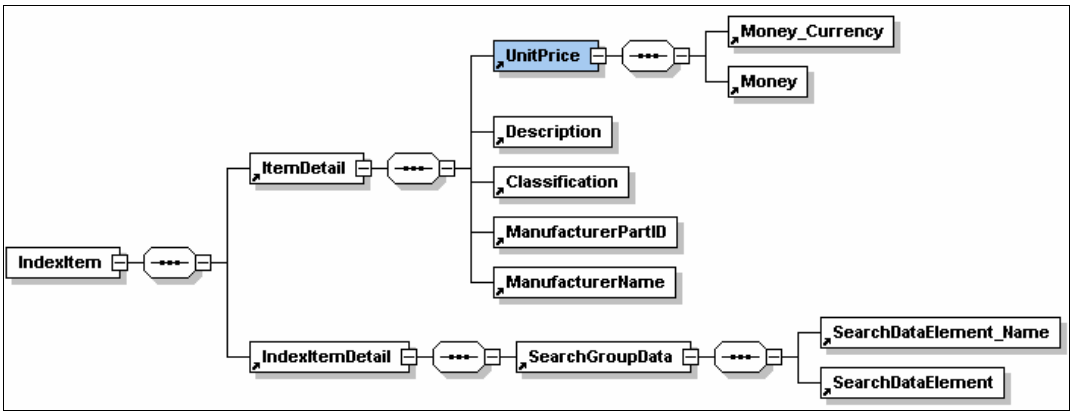

Figura 5.7 – XML Schema destino do arquivo cXml

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <dados>
    <origem path="http://servidor/xCbl_Origem.xml" alias="mod1"/>
    <destino path="http://servidor/cXml_Destino.xsd" alias="mod2"/>
  </dados>
  <sinonimos>
    <sinonimo>
      <nodeOrigem source="mod1">Amount</nodeOrigem>
      <nodeDestino source="mod2">Money</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">CurrencyCoded</nodeOrigem>
      <nodeDestino source="mod2">Money_Currency</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">ShortDescription</nodeOrigem>
      <nodeDestino source="mod2">Description</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">Product</nodeOrigem>
      <nodeDestino source="mod2">Classification</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">ProductID</nodeOrigem>
      <nodeDestino source="mod2">ManufacturerPartID</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">Manufacturer</nodeOrigem>
      <nodeDestino source="mod2">ManufacturerName</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">AttributeID</nodeOrigem>
      <nodeDestino source="mod2">SearchDataElement_Name</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">AttributeValue</nodeOrigem>
      <nodeDestino source="mod2">SearchDataElement</nodeDestino>
    </sinonimo>
  </sinonimos>
</definition>
```

Quadro 5.8 - Mapeamentos dos nós do caso 2

De forma análoga ao caso 1, os passos 1.1 e 1.2 criam a tabela `origem` no banco de dados.

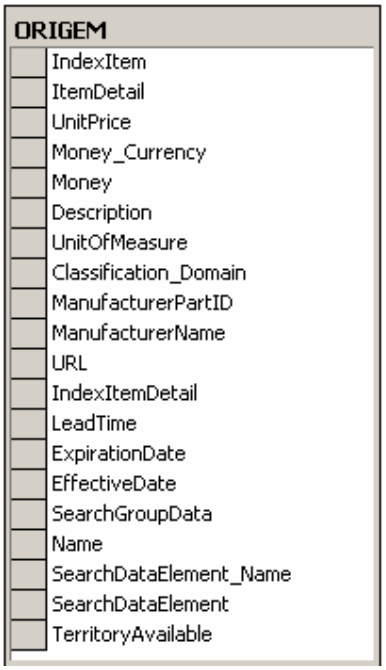

Figura 5.8 - Tabela de Origem do Caso 2

O passo 2 cria a árvore de aceitação que é ilustrada no quadro 5.9:

```
<IndexItem>
    <ItemDetail>
        <UnitPrice>
            <Money_Currency/>
            <Money/>
        </UnitPrice>
        <Description/>
        <Classification/>
        <ManufacturerPartID/>
        <ManufacturerName/>
    </ItemDetail>
    <IndexItemDetail>
        <SearchGroupData>
            <SearchDataElement_Name/>
            <SearchDataElement/>
        </SearchGroupData>
    </IndexItemDetail>
</IndexItem>
```

Quadro 5.9 - Árvore de aceitação do Caso 2

Os passos 3.1 e 3.2 criam a tabela `destino` no Banco de Dados e a populam com as informações da tabela `origem`.

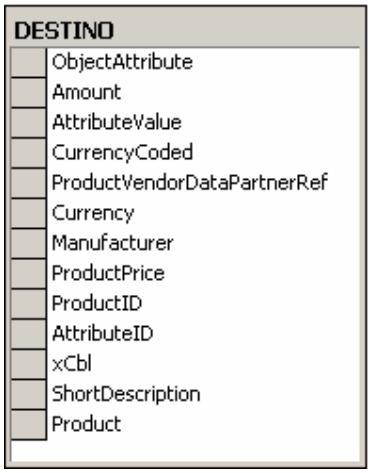

Figura 5.9 - Tabela Destino do Caso 2

Após a execução do algoritmo o arquivo exemplificado no quadro 5.10 é gerado.

```
<?xml version="1.0" encoding="UTF-8"?>
<IndexItem>
      <ItemDetail>
            <UnitPrice>
                  <Money_Currency valor="USD"/>
                  <Money valor="1000"/>
            </UnitPrice>
            <Description valor="Armada M700 PIII 500 12GB"/>
            <Classification valor=" C43171801"/>
            <ManufacturerPartID valor="140141-002"/>
            <ManufacturerName valor="Compaq"/>
      </ItemDetail>
      <IndexItemDetail>
            <SearchGroupData>
                  <SearchDataElement_Name valor="Processor Speed"/>
                  <SearchDataElement valor="500MHZ"/>
            </SearchGroupData>
      </IndexItemDetail>
</IndexItem>
```

Quadro 5.10 - Instância XML de Destino gerada

5.3.3 – Caso 3: Integração de informações de clientes e contatos

O terceiro caso refere-se à integração de informações de clientes e contatos, que ocorrem muitas vezes em sistemas distintos da mesma corporação. Os sistemas de Call Center, por exemplo, costumam ter a informação de clientes e contatos mais atualizada do que o ERP da empresa. Sendo assim, existem constantemente atualizações que podem ser realizadas através de instâncias XML entre os dois sistemas.

```
<?xml version="1.0" encoding="UTF-8"?>
<arquivo valor="arquivo">
      <cliente valor="2003">
            <cnpj valor="68639398001"/>
            <ie valor="12345"/>
            <razaosocial valor="Foco Informatica"/>
            <nomefantasia valor="Foco"/>
            <endereco valor="Rua Cde de Bonfim 32"/>
            <tel valor="2222222"/>
      </cliente>
</arquivo>
```

Quadro 5.11 - Exemplo de Instância XML de Origem do Caso 3

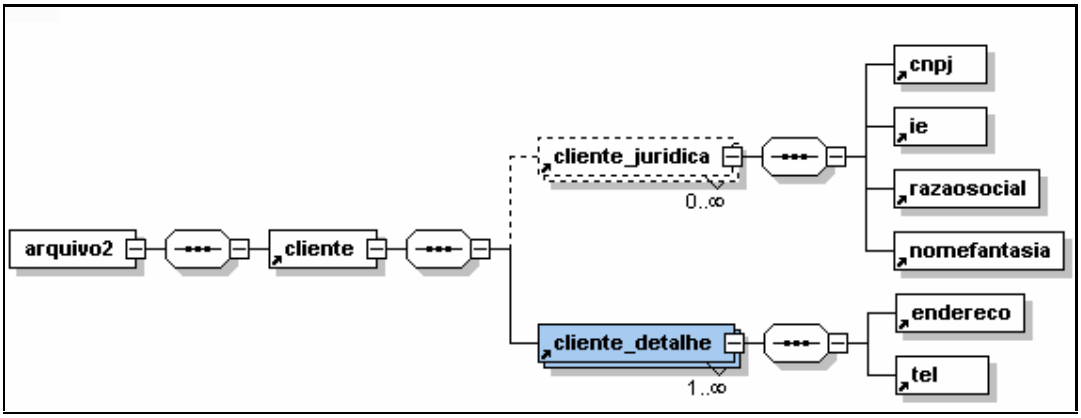

Figura 5.10 - Schema destino do arquivo do ERP

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <dados>
    <origem path="http://servidor/Cliente.xml" alias="mod1"/>
    <destino path="http://servidor/Cliente.xsd" alias="mod2"/>
  </dados>
  <sinonimos>
    <sinonimo>
      <nodeOrigem source="mod1">arquivo</nodeOrigem>
      <nodeDestino source="mod2">arquivo2</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">cliente</nodeOrigem>
      <nodeDestino source="mod2">cliente</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">cnpj</nodeOrigem>
      <nodeDestino source="mod2">cnpj</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">ie</nodeOrigem>
      <nodeDestino source="mod2">ie</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">razaosocial</nodeOrigem>
      <nodeDestino source="mod2">razaosocial</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">nomefantasia</nodeOrigem>
      <nodeDestino source="mod2">nomefantasia</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">endereco</nodeOrigem>
      <nodeDestino source="mod2">endereco</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">tel</nodeOrigem>
      <nodeDestino source="mod2">tel</nodeDestino>
    </sinonimo>
  </sinonimos>
</definition>
```

Quadro 5.12 - Mapeamentos dos nós do caso 3

De forma análoga ao caso 1, os passos 1.1 e 1.2 criam a tabela `origem` no Banco de Dados.

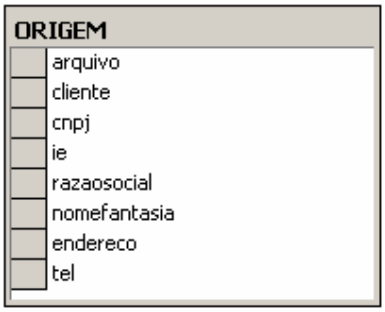

Figura 5.11 - Tabela de Origem do Caso 3

O passo 2 cria a árvore de aceitação que é ilustrada no quadro 5.13:

```
<arquivo2>
    <cliente>
        <cliente_juridica>
            <cnpj/>
            <ie/>
            <razaosocial/>
            <nomefantasia/>
        </cliente_juridica>
        <cliente_detalhe>
            <endereco/>
            <tel/>
        </cliente_detalhe>
    </cliente>
</arquivo2>
```

Quadro 5.13 - Árvore de aceitação do Caso 3

Os passos 3.1 e 3.2 criam a tabela `destino` no Banco de Dados e a populam com as informações da tabela `origem`.

Figura 5.12 - Tabela Destino do Caso 3

Após a execução do algoritmo o arquivo exemplificado no quadro 5.14 é gerado.

```
<?xml version="1.0" encoding="UTF-8"?>
<arquivo2 valor="arquivo">
    <cliente valor="2003">
        <cliente_juridica>
            <cnpj valor="68639398001"/>
            <ie valor="12345"/>
            <razaosocial valor="Foco Informatica"/>
            <nomefantasia valor="Foco"/>
        </cliente_juridica>
        <cliente_detalhe>
            <endereco valor="Rua Cde de Bonfim 32"/>
            <tel valor="2222222"/>
        </cliente_detalhe>
    </cliente>
</arquivo2>
```

Quadro 5.14 - Instância XML de Destino gerada

5.3.4 – Caso 4: Integração de informações de uma universidade

O quarto caso refere-se a instâncias XML de um sistema de universidade que representam a informação de forma distinta. A instância origem centraliza as informações por ano, enquanto o XML Schema destino apresenta as informações centralizadas por professor.

```
<?xml version="1.0" encoding="UTF-8"?>
<universidade valor="ufrj">
    <ano valor="2002">
        <periodo valor="1">
            <disciplina valor="TESI2">
                <professor valor="Maria Luiza"/>
            </disciplina>
        </periodo>
        <periodo valor="2">
            <disciplina valor="Engenharia de Marketing">
                <professor valor="Juarez"/>
            </disciplina>
        </periodo>
    </ano>
    <ano valor="2003">
        <periodo valor="1">
            <disciplina valor="Engenharia de Software">
                <professor valor="Eber"/>
            </disciplina>
        </periodo>
    </ano>
</universidade>
```

Quadro 5.15 - Exemplo de Instância XML de Origem do caso 4

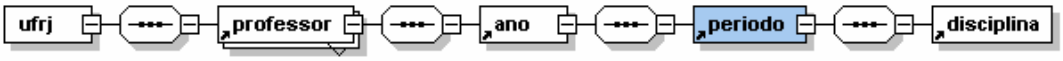

Figura 5.13 - Schema destino do arquivo da universidade

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <dados>
    <origem path="http://servidor/Disciplinas.xml" alias="mod1"/>
    <destino path="http://servidor/Disciplinas.xsd" alias="mod2"/>
  </dados>
  <sinonimos>
    <sinonimo>
      <nodeOrigem source="mod1">universidade</nodeOrigem>
      <nodeDestino source="mod2">ufrj</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">professor</nodeOrigem>
      <nodeDestino source="mod2">professor</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">disciplina</nodeOrigem>
      <nodeDestino source="mod2">disciplina</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">ano</nodeOrigem>
      <nodeDestino source="mod2">ano</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">periodo</nodeOrigem>
      <nodeDestino source="mod2">periodo</nodeDestino>
    </sinonimo>
  </sinonimos>
</definition>
<definition>
```

Quadro 5.16 - Mapeamentos dos nós do caso 4

De forma análoga ao caso 1, os passos 1.1 e 1.2 criam a tabela `origem` no Banco de Dados.

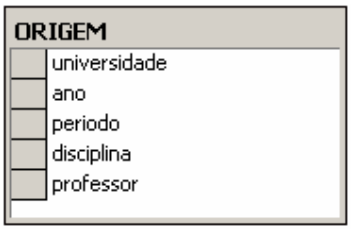

Figura 5.14 - Tabela de Origem do Caso 4

O passo 2 cria a árvore de aceitação que é ilustrada no quadro 5.17:

```
<ufrj>
    <professor>
        <ano>
            <periodo>
                <disciplina/>
            </periodo>
        </ano>
    </professor>
</ufrj>
```

Quadro 5.17 - Árvore de aceitação do Caso 4

Os passos 3.1 e 3.2 criam a tabela `destino` no Banco de Dados e a populam com as informações da tabela `origem`.

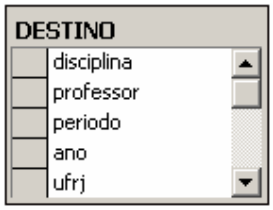

Figura 5.15 - Tabela Destino do Caso 4

Após a execução do algoritmo o arquivo exemplificado no quadro 5.18 é gerado.

```
<?xml version="1.0" encoding="UTF-8"?>
<ufrj>
  <professor valor="Eber">
    <ano valor="2003">
      <periodo valor="1">
        <disciplina valor="Engenharia de Software"/>
      </periodo>
    </ano>
  </professor>
  <professor valor="Juarez">
    <ano valor="2002">
      <periodo valor="2">
        <disciplina valor="Engenharia de Marketing">
      </periodo>
    </ano>
  </professor>
  <professor valor="Maria Luiza">
    <ano valor="2002">
      <periodo valor="1">
        <disciplina valor="TESI2"/>
      </periodo>
    </ano>
  </professor>
</ufrj>
```

Quadro 5.18 - Instância XML de Destino gerada

5.3.5 – Caso 5: Transformação de dois arquivos de Compras de formatos distintos

O quinto caso refere-se a instâncias XML de dois catálogos distintos que representam informações de compras. As instâncias origem e destino foram modificadas para armazenar seus valores em atributos com o nome valor. Além desta modificação que também foi realizada nos casos anteriores, foi necessário trocar alguns nomes de elementos das duas instâncias XML envolvidas. Isto ocorreu em virtude da restrição do algoritmo I2I em ter nomes de elementos iguais em níveis diferentes das árvores formadas. Nas duas instâncias deste caso algumas estruturas de sub-árvores se repetiam em diferentes níveis da instância XML. A estrutura das informações de contato, por

exemplo, se repete na sub-árvore de compra e na sub-árvore de cobrança. Todos estes casos tiveram que ser renomeados. Como exemplo desta operação, o nó `NameAddress` descendente de `BuyerParty` foi renomeado para `BuyerPartyNameAddress`, enquanto o nó `NameAddress` descendente de `OrderParty` foi renomeado para `OrderPartyNameAddress`.

Não foi possível mapear todos os nós envolvidos nas duas instâncias. Há uma diferença significativa no que diz respeito às suas estrutura e seus conteúdos. Sendo assim, o XML Schema destino foi criado de modo a definir como opcional todas as sub-árvores que não contém os nós correspondentes na instância origem. Com isso, o algoritmo I2I foi executado e uma instância XML destino consistente com o seu respectivo XML Schema foi gerada. O volume de informações deste exemplo é muito extenso, portanto os passos do algoritmo estão listados no apêndice 1.

6. Discussão

Existem quatro grandes diferenças entre o algoritmo aqui proposto e os trabalhos correlatos: A primeira diz respeito à dispensa de um especialista para realizar a transformação. A segunda está relacionada com a eliminação da necessidade dos dois schemas, origem e destino. Somente o XML Schema destino é imprescindível. Em muitos casos, esta diferença é significativa, principalmente nos cenários em que existe um grande volume de diferentes origens a serem transformadas. A terceira diferença é que o algoritmo produz uma única instância válida para o schema destino. A forma como o algoritmo foi projetado foca basicamente nesta garantia. Acreditamos que uma instância gerada de forma inconsistente pode trazer conseqüências drásticas para a maioria dos sistemas. Por último, a quarta diferença está na definição formal e prova do algoritmo. A nossa proposta está formalmente definida e provada.

Este capítulo apresenta uma discussão sobre os casos que foram estudados e os resultados obtidos em cada um deles.

6.1 – Pontos a serem observados

A automação da transformação de instâncias XML é sem dúvida nenhuma um assunto complexo que irá ser alvo de pesquisas tanto nas grandes empresas de desenvolvimento de software quanto no meio acadêmico. O principal ponto positivo do algoritmo deste trabalho é a forma transparente e automatizada que os dados podem ser trocados pelos sistemas distintos de diferentes corporações.

Este trabalho não resolve por completo o problema de integração de sistemas e nem a questão de transformação de instâncias XML que envolve muitos pontos que devem ser cuidadosamente estudados.

O algoritmo não gera a instância XML destino da melhor forma possível no caso do catálogo de produtos se a direção de transformação for oposta à que foi apresentada no capítulo anterior. Isto ocorre porque a relação de dois elementos na instância origem é de irmãos e no XML Schema destino os nós

equivalentes tem uma relação de paternidade. O conceito de referência forte indica que os nós podem ser criados caso exista uma relação onde os nós são ancestrais. O algoritmo não contempla a relação de irmandade por considerar que dois nós irmãos não podem ser transformados em filhos em 100% dos casos.

Outra questão identificada é que muitas vezes, os dados têm que ser manipulados na transformação. No exemplo do catálogo de produtos, a moeda listada nas instâncias pode ser de diferentes países. Na instância origem, o valor pode estar cotado em dólares enquanto que na instância destino, este valor tem que estar cotado em euros. Nesta transformação pode ser necessário consultar outro sistema ou algum Web Service para saber a cotação daquele dia das duas moedas. Mesmo neste caso, seria mais vantajoso utilizar o I2I. Uma simples manipulação das instâncias destino criadas poderia realizar a nova cotação do valor na moeda distinta.

6.2 – Uma comparação com outros métodos

Esta seção realiza uma comparação entre o algoritmo proposto neste trabalho e outras abordagens que foram detalhadas no Capítulo 2.

6.2.1 - MapForce

O MapForce foi escolhido na categoria dos softwares comerciais que realizam a transformação de instâncias XML. O Biz Talk Server (outro software detalhado na Seção 2.6) não contém nenhuma versão Trial que permita uma comparação com o algoritmo aqui proposto.

O primeiro caso apresentado no capítulo 5 foi escolhido inicialmente para avaliar o potencial da ferramenta. A interface do software é bem amigável e os dois XML Schemas (origem e destino) foram inseridos no novo projeto criado. A ferramenta não permite que seja feita uma transformação entre uma instância XML e o schema destino, conforme é realizada no algoritmo I2I.

O primeiro ponto negativo identificado foi que mesmo apresentando os dois schemas envolvidos na integração, é necessário incluir também no projeto a instância relacionada ao XML Schema origem para visualizar o script XSLT que realiza a transformação. Isto parece ser uma inconsistência porque se o XML Schema representa todas as informações, por que é necessário incluir também alguma instância correspondente?

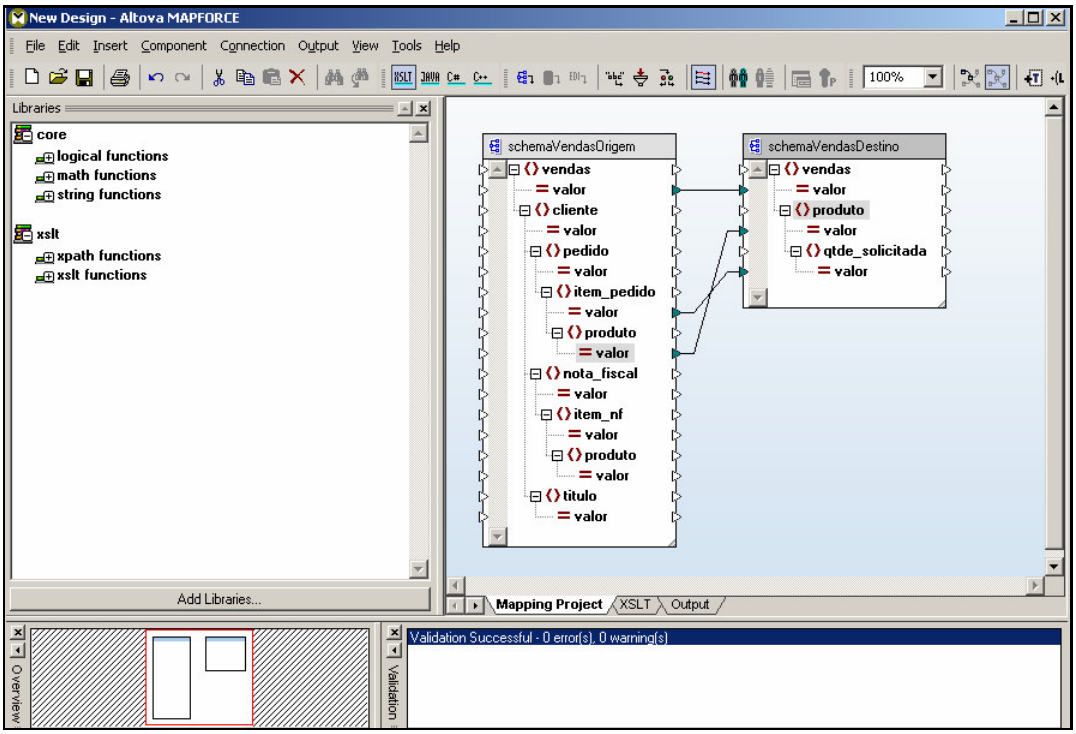

Figura 6.1 - Mapeamento dos XML Schemas origem e destino do Caso 1

Todos os mapeamentos foram inseridos e o script XSLT foi interpretado. O quadro 6.1 mostra a instância gerada.

```
<?xml version="1.0" encoding="UTF-8"?>
<vendas valor="VENDAS">
      <produto valor="PRODUTO 1">
            <qtde_solicitada valor="ITEM PEDIDO 2_1"/>
      </produto>
</vendas>
```

Quadro 6.1 – Primeira Instância XML Gerada no Software MapForce

O conteúdo da instância gerada está incompleto. Comparando com o resultado obtido com o algoritmo I2I, percebe-se que apenas um elemento foi gerado tanto no caso de produto quanto no caso de qtde_solicitada. Para que estas informações não fossem perdidas seria fundamental a intervenção de um especialista na linguagem XSLT para que, a partir do código já gerado, modificasse o script para atender as necessidades do caso. Talvez este esforço seja o mesmo que criar todo o script XSLT.

O segundo teste da ferramenta foi realizado para avaliar se o software mantém as informações da instância gerada de forma consistente. O script XSLT deve ser montado de modo que a instância resultante esteja conforme com relação à forma e consistente quanto ao conteúdo (Referência Forte). Sendo assim, foi criado um XML Schema destino diferente daquele listado no caso 1 deste trabalho. As informações de pedido, item de nota fiscal e nota fiscal também devem ser consideradas na instância gerada. A figura 6.2 e o quadro 6.2 ilustram o XML Schema destino modificado e a instância avaliada.

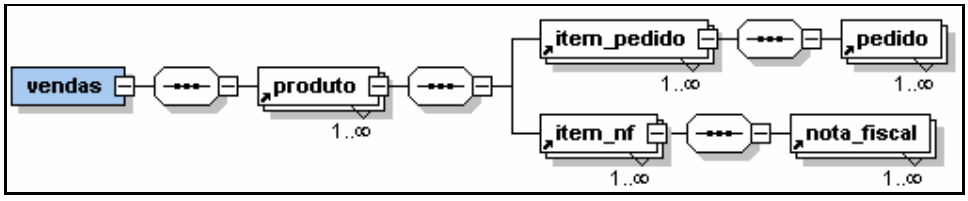

Figura 6.2 - Schema Destino Modificado para testar a consistência da instância gerada

```
<?xml version="1.0" encoding="UTF-8"?>
<vendas valor="VENDAS">
      <cliente valor="CLIENTE 1">
            <pedido valor="PEDIDO 1">
                  <item_pedido valor="ITEM PEDIDO 1_1">
                        <produto valor="PRODUTO 1"/>
                  </item_pedido>
                  <item_pedido valor="ITEM PEDIDO 1_2">
                        <produto valor="PRODUTO 2"/>
                  </item_pedido>
            </pedido>
            <pedido valor="PEDIDO 2">
                  <item_pedido valor="ITEM PEDIDO 2_1">
                        <produto valor="PRODUTO 1"/>
                  </item_pedido>
            </pedido>
            <nota_fiscal valor="NOTA FISCAL 1">
                  <item_nf valor="ITEM NOTA FISCAL 1">
                        <produto valor="PRODUTO 1"/>
                  </item_nf>
                  <item_nf valor="ITEM NOTA FISCAL 2">
                        <produto valor="PRODUTO 2"/>
                  </item_nf>
            </nota_fiscal>
            <titulo valor="TITULO 1"/>
      </cliente>
</vendas>
```

Quadro 6.2 - Segunda instância avaliada no software MapForce

Um novo mapeamento foi criado dentro da ferramenta conforme ilustra a figura 6.3:

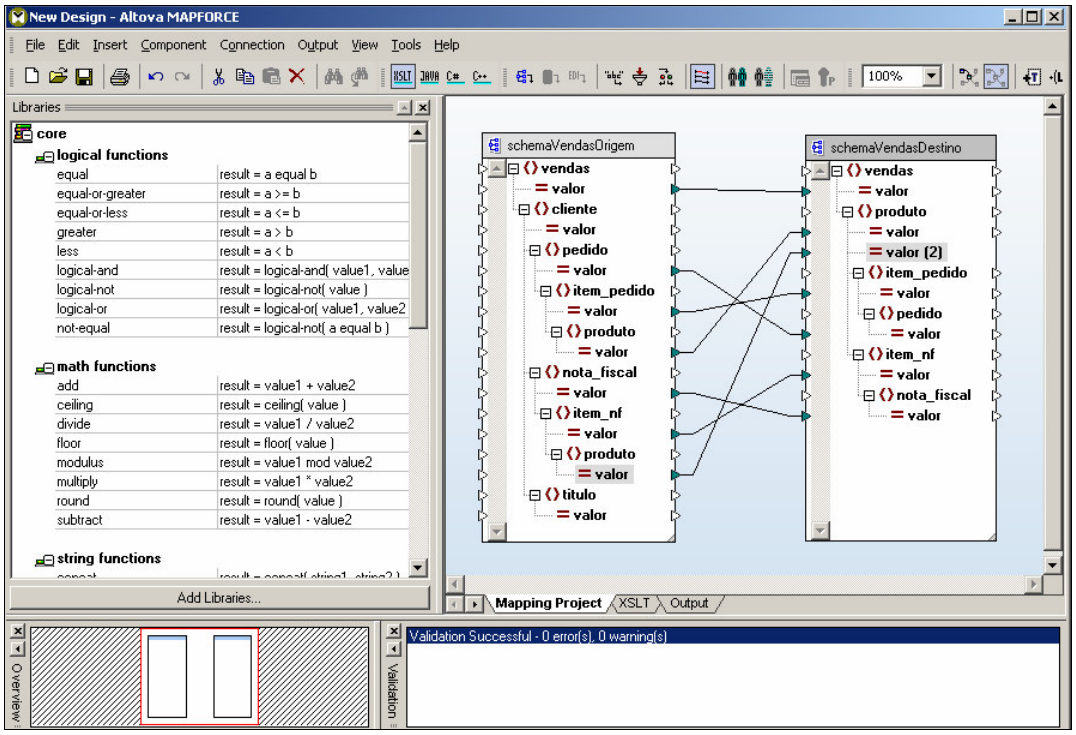

Figura 6.3 - Mapeamento dos XML Schemas origem e destino modificados

O elemento produto dos XML Schemas listados é o que tem a maior chance de estar inconsistente com relação ao conteúdo porque na árvore origem, esta informação encontra-se em folhas enquanto na árvore destino esta informação está próxima da raiz.

O MapForce não deixou que dois atributos do XML Schema de origem das informações fossem concatenados em apenas um atributo do XML Schema de destino. No momento em que o elo foi inserido, o software sugeriu duas ações a serem realizadas: a primeira seria substituir o elo anterior (atributo do elemento produto filho de item_pedido) pelo novo elo (atributo do elemento produto filho de item_nf). Esta opção não atende ao XML Schema destino porque as duas informações devem ser concatenadas para o mesmo elemento caso o valor dos dois atributos do elemento origem seja igual. A segunda opção sugerida foi duplicar graficamente o atributo valor do XML Schema destino.

Executou-se o script XSLT e o resultado não foi satisfatório. A instância criada está inconsistente com relação ao seu conteúdo. O quadro 6.3 ilustra a instância gerada.

```
<?xml version="1.0" encoding="UTF-8"?>
<vendas valor="VENDAS">
      <produto valor=PRODUTO 2">
            <item_pedido valor="ITEM PEDIDO 2_1">
                  <pedido valor="PEDIDO 2"/>
            </item_pedido>
            <item_nf valor="ITEM NOTA FISCAL 2">
                  <nota_fiscal valor="NOTA FISCAL 1"/>
            </item_nf>
      </produto>
</vendas>
```

Quadro 6.3 - Segunda Instância XML Gerada no Software MapForce

Se compararmos a instância gerada com a instância original, percebemos que o elemento com o valor "ITEM PEDIDO 2_1" está associado ao produto de valor "PRODUTO 1". No entanto, na instância gerada pela ferramenta o produto "PRODUTO 2" é que está associado a este item de pedido. Esta inconsistência iria trazer conseqüências drásticas para os sistemas em questão porque o pedido iria ser gerado com produtos trocados.

O terceiro teste foi baseado nos XML Schemas do segundo teste. Para o XML Schema destino foi modificado um pequeno detalhe em sua formação: o elemento produto se tornou opcional. Este teste visa avaliar se a ferramenta leva em consideração esta característica comum na definição de XML Schemas. A instância de origem também foi modificada, de forma a avaliar se a ferramenta iria gerar uma instância destino inconsistente ou incompleta. A instância origem e o XML Schema destino modificado estão ilustrados no quadro 6.4 e na figura 6.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<vendas valor="VENDAS">
      <cliente valor="CLIENTE 1">
            <pedido valor="PEDIDO 1">
                  <item_pedido valor="ITEM PEDIDO 1_1">
                        <produto valor="PRODUTO 1"/>
                  </item_pedido>
                  <item_pedido valor="ITEM PEDIDO 1_2">
                        <produto valor="PRODUTO 2"/>
                  </item_pedido>
            </pedido>
            <pedido valor="PEDIDO 2">
                  <item_pedido valor="ITEM PEDIDO 2_1">
                        <produto valor="PRODUTO 1"/>
                  </item_pedido>
            </pedido>
            <nota_fiscal valor="NOTA FISCAL 1">
                  <item_nf valor="ITEM NOTA FISCAL 1">
                        <produto valor="PRODUTO 1"/>
                  </item_nf>
            </nota_fiscal>
            <titulo valor="TITULO 1"/>
      </cliente>
</vendas>
```

Quadro 6.4 - Terceira Instância avaliada no Software MapForce

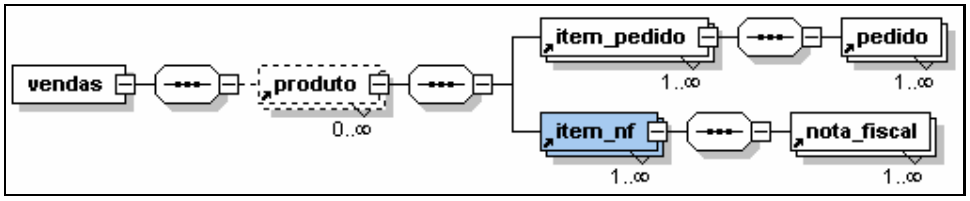

Figura 6.4 - Schema Destino Modificado com Elemento Opcional

O elemento "PRODUTO 2" não pode ser criado na instância destino porque o elemento "item_nf" correspondente não existe na instância origem. Os mapeamentos foram realizados na ferramenta da mesma forma que foram feitos no segundo teste. O quadro 6.5 mostra a instância destino gerada a partir da interpretação do script XSLT.

```
<?xml version="1.0" encoding="UTF-8"?>
<vendas valor="VENDAS">
      <produto valor="PRODUTO 1">
            <item_pedido valor="ITEM PEDIDO 2_1">
                  <pedido valor="PEDIDO 2"/>
            </item_pedido>
            <item_nf valor="ITEM NOTA FISCAL 1">
                  <nota_fiscal valor="NOTA FISCAL 1"/>
            </item_nf>
      </produto>
</vendas>
```

Quadro 6.5 - Terceira Instância XML Gerada no Software MapForce

O resultado acima se mostrou bastante interessante. Apesar de existir apenas duas diferenças entre o segundo e o terceiro teste (o elemento opcional e a instância origem), a instância gerada é totalmente diferente. Desta vez, a ferramenta gerou o elemento produto com o valor "PRODUTO 1" de forma consistente.

A forma como as ferramentas e algoritmos relacionados ao assunto se comportam ainda é obscuro o que dificulta a utilização das abordagens previamente apresentadas.

Os exemplos nativos da ferramenta também foram estudados. Todos eles parecem realizar apenas transformações de estruturas que são muito semelhantes. Entretanto, algumas funcionalidades poderiam ser úteis e consideradas no algoritmo I2I.

A primeira funcionalidade utilizada em um dos exemplos é a constante. O software permite que seja definido para algum elemento da instância destino, um valor constante que será utilizado em todos os elementos que forem criados na instância destino.

A segunda funcionalidade interessante da ferramenta é o número de funções disponíveis que podem ser utilizadas na transformação de um ou mais elementos da instância origem em um elemento da instância destino. É possível, por exemplo, multiplicar o valor de dois elementos, ou inserir uma condição para no caso do elemento da instância origem assumir um valor determinado o elemento da instância destino assume um valor distinto.

A terceira funcionalidade interessante está fora do escopo deste trabalho, mas ainda tem muita utilidade nas grandes companhias. As transformações não são feitas necessariamente entre instâncias XML ou XML Schemas. É possível, por exemplo, realizar a transformação de informações contidas em um banco de dados relacional para uma instância XML. Esta é uma funcionalidade que ainda tem grande relevância, principalmente nos ambientes mais antigos que utilizam plataformas obsoletas, que não expõe os dados em formato XML.

6.2.2 – Trabalhos Acadêmicos

Não é possível realizar uma comparação prática entre o algoritmo I2I e os trabalhos acadêmicos publicados. Todos os trabalhos relacionados ao mesmo assunto não apresentam uma definição detalhada e formal do algoritmo de uma forma que seja possível reproduzir o trabalho realizado e avaliá-lo para os casos aqui apresentados.

7. Conclusões

Um novo método de transformação de instâncias XML foi proposto neste artigo. O trabalho apresentado define um conjunto de transformações que podem ser realizadas de forma transparente e direta sem a necessidade dos dois schemas envolvidos na transformação. Além disso, a instância criada é consistente em seu conteúdo e válida para o schema definido. O escopo mostrado de forma detalhada do algoritmo permite que uma possível transformação de instâncias XML seja avaliada e, se possível, realizada sem a necessidade de envolvimento de um especialista na área de TI.

Como sugestão para novos estudos baseados no artigo em questão, iremos avaliar os casos em que algumas informações do schema destino não estão definidas na instância origem. A transformação para elementos com mais de um atributo ou com informações representadas como conteúdo dos elementos também serão avaliados como sugere o capítulo 6. Uma outra implementação que dispense a utilização de um banco de dados relacional será avaliada. Sugerimos também o uso de ontologias para realizar o mapeamento semântico dos dados. A ontologia pode trazer ao algoritmo conceitos relevantes de um determinado domínio, relacionamentos possíveis e derivações entre conceitos (ERDMANN, 1999), de modo a inferir um novo conhecimento na integração. Por último, iremos estudar a real aplicabilidade do algoritmo para a área de web services.

8. Referências

BELLASHENE, Z. Data integration over the web. **Data & knowledge Engineering**, Amsterdam, v.44, n. 3, p. 265-266, Mar. 2003. Guest Editorial.

BENDECK F. Automation of XML documents translators generation. In: IEEE WORKSHOPS ON ENABLING TECHNOLOGIES: INFRASTRUCTURE FOR COLLABORATIVE ENTERPRISES, 10., 2001, Cambridge. **Proceedings…** Cambridge: MIT, 2001. p. 37-38.

BERTINO, E.; GUERRINI, G; MESITI, M. A matching algorithm for measuring the structural between an XML document and a DTD and its application. **Information Systems.** V. 29 n.1 p. 23-46, Mar.2004.

BIZTALK. **BizTalk server 2004:building the smart, connected enterprise**. 2005. Disponivel em: http://www.microsoft.com/BizTalk. Acesso em: 2004.

BOS B. **A meta-grammar for describing XML-based formats.** 1999**.** Disponível em**:** http://www.w3.org/People/Bos/meta-bnf . Acesso em: 2005

BOUKOTTAYA, A; et al. Automating XML documents transformations**:** a conceptual modelling based approach. In: ASIA-PACIFIC CONFERENCE ON CONCEPTUAL MODELLING, 1., 2004, Dunedin. **Proceedings…** Dunedin: 2004. p. 18-22.

CXML - **Commerce XML resources.** 2004. Disponível em: http://cxml.org. Acesso em: 2004.

DAUM B.; MERTEN U. **System architecture with XML.** Morgan Kaufmann, 2003.

DOAN A.; DOMINGOS P.; HALEVY A. Reconciling schemas of disparate data sources: a machine-learning approach. In: SIGMOD CONFERENCE. 2001, Santa Barbara. **Proceedings…** Santa Bárbara**:** ACM, 2001

W3C . **Document Object Mode DOM.** 2004. Disponível em: http://www.w3.org/DOM, Acesso em :2004.

MUVVA, R. **DOM vs sax what is best?** 2003. Disponível em: http://www.code101.com/Code101/DisplayArticle.aspx?cid=37, Acesso em: 2003.

ERDMANN M.; STUDER R. Ontologies as conceptual models for XML documents. In: WORKSHOP ON KNOWLEDGE ACQUISITION, MODELING AND MANAGEMENT. 12., 1999, Banff. **Proceedings…** Banff: 1999.

HOLMAN G. **What is XSLT.** 2000. Disponível em : http://www.xml.com/pub/a/2000/08/holman/index.html, Acesso em: Agosto 2000.

JAVA architecture for XML Binding (JAXB). 2004. Disponível em: http://java.sun.com/xml/jaxb/index.jsp Acesso em: 2004.

JENSEN M.; MOLLER T.; PEDERSEN T. Converting XML DTDs to UML diagrams **for** conceptual da integration. **Data and Knowledge Engineering,** Amsterdam, North-Holland, v. 44, n. 3, p. 323-346, Mar. 2003.

KURGAN L.; SWIERCZ W.; CIOS K. Semantic mapping of XML tags using inductive machine learning. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND APPLICATIONS, 2002, Las Vegas. **Proceedings...** Las Vegas: 2002. p. 99-109.

MADHAVAN, J.; BERNSTEIN, P. A.; ERHARD, R. Generic schema matching with cupid. INTERNATIONAL CONFERENCE ON VERY LARGE DATABASE, 27., 2001, Roma. **Proceedings ...** Roma: Morgan Kaufmann, 2001.

ALTOVA. **Mapforce**. Disponível em: http://www.altova.com/products_mapforce.html, Acesso em: maio de 2004.

MCLAUGHLIN B. **Java and XML binding.** O'Reilly, 2002.

MURATA M. DTD transformation by patterns and contextual conditions**.** In: tado em SGML/XML´ 97 Conference, 1997. Washington DC. **Proceedings ...** Washington DC: 1997.

OMELAYENKO B.; FENSEL, D. An Analysis of B2B Catalogue Integration Problems. Content and Document Integration. In: Filipe, J.; Sharp, B.; Miranda, P. (eds.) **Enterprise Information systems III**, Dordrecht: Kluwer Academic Publishers, 2002. p. 270-277.

SAX. **About SAX**. Disponível em: http://www.saxproject.org. Acesso em: 2002.

SGML USERS´ GROUP. **A Brief historyof the developmentof SGML**. Disponível em: http://www.sgmlsource.com/history/sgmlhist.htm. Acesso em:1990.

SKONNARD A. **Designing XML schema libraries**. Microsoft Corporation, Maio 2003.

____. **Understanding XML schema**. Microsoft Corporation, Março 2003.

SU H.; KUNO H.; RUNDENSTEINER E. Automating the transformation of XML documents. In: INTERNATIONAL WORKSHOP ON WEB INFORMATION AND DATA MANAGEMENT, 3., 2001, Atlanta. **Proceedings ...** Atlanta: ACM, 2001. p. 68-75.

TIDWELL, D. **XSLT**. O'Reilly, 2002.

XCBL. **XML common business library.** Disponível em: http://www.xcbl.org. Acesso em: junho de 2004.

W3C. **Extensible markup language – XML**. Disponível em: http://www.w3.org/XML. Acesso em: 2004.

IBM. **XML for tables**. Disponível em: http://www.alphaworks.ibm.com/tech/xtable?Open&ca=daw-flts-041003. Acesso em: abril 2003.

W3C. **XML schema**. Disponível em: http://www.w3c.org/XML/Schema. Acesso em: 2004.

____. **XML QUERY (XQUERY).** Disponícel em: http://www.w3.org/XML/Query. Acesso em: 2004.

____. **The Extensible stylesheet language family (XSL),** Disponível em: http://www.w3c.org/Style/XSL. Acesso em: 2004.

Apêndice A

O apêndice 1 lista os passos e as informações utilizadas no quinto estudo de caso deste trabalho. A forma como os dados são mostrados diferem dos casos anteriores. Como a instância origem e o XML Schema Destino são muito extensos, não é possível mostrá-los graficamente de uma forma completa.

```
<?xml version="1.0" encoding="UTF-8"?>
<cXML>
  <Header>
    <From>
      <Credential valor="AribaNetworkUserId">
        <Identity valor="admin@acme.com"/>
      </Credential>
      <Credential_domain valor="AribaNetworkUserId">
        <Identity>bigadmin@marketplace.org</Identity>
      </Credential_domain>
      <Credential_type valor="marketplace"/>
    </From>
    <To>
      <Credential_domain valor="DUNS">
        <Identity valor="942888711"/>
      </Credential_domain>
    </To>
    <Sender>
      <Credential_domain valor="AribaNetworkUserId">
        <Identity valor="admin@acme.com"/>
        <SharedSecret valor="abracadabra"/>
      </Credential_domain>
      <UserAgent valor="Ariba.com Network V1.0"/>
    </Sender>
  </Header>
  <Request valor="test">
    <OrderRequest>
      <OrderRequestHeaderID valor="DO1234">
        <OrderRequestDate valor="1999-03-12"/>
        <OrderRequesttype valor="new"/>
        <Total>
          <MoneyCurrency valor="USD">
            <Money valor="2.68"/>
```

```
        </MoneyCurrency>
      </Total>
      <ShipTo>
        <Address>
          <Name valor="Acme"/>
          <PostalAddress valor="default">
            <DeliverTo valor="Joe Smith"/>
            <DeliverTo valor="Mailstop M-543"/>
            <Street valor="123 Anystreet"/>
            <City valor="Sunnyvale"/>
            <State valor="CA"/>
            <PostalCode valor="90489"/>
            <CountryCode valor="US"/>
            <Country valor="United States"/>
          </PostalAddress>
        </Address>
      </ShipTo>
      <BillTo>
        <Address>
          <NameBill valor="Acme"/>
          <PostalAddressBill valor="default">
            <StreetBill valor="123 Anystreet"/>
            <CityBill valor="Sunnyvale"/>
            <StateBill valor="CA"/>
            <PostalCodeBill valor="90489"/>
            <CountryCodeBill valor="US"/>
            <CountryBill valor="United States"/>
          </PostalAddressBill>
        </Address>
      </BillTo>
      <Tax>
        <MoneyCurrencyTax valor="USD"/>
        <MoneyTax valor="0.19"/>
        <Description valor="CA Sales Tax"/>
      </Tax>
      <Payment>
        <PCard valor="1234567890123456"/>
        <expiration valor="1999-03-12"/>
      </Payment>
      <Comments valor="Anything well formed in XML can go here."/>
    </OrderRequestHeaderID>
```

```
<ItemOut>
  <quantity valor="2"/>
  <requestedDeliveryDate valor="1999-03-12"/>
  <ItemID>
    <SupplierPartID valor="1233244"/>
  </ItemID>
  <ItemDetail>
    <UnitPrice>
      <MoneyCurrencyPrice valor="USD"/>
      <MoneyPrice valor="1.34"/>
    </UnitPrice>
    <ItemOutDescription valor="hello"/>
    <UnitOfMeasure valor="EA"/>
    <Classification valor="12345"/>
    <ManufacturerPartID valor="234"/>
    <ManufacturerName valor="foobar"/>
    <URL valor="www.foo.com"/>
  </ItemDetail>
  <ShippingTrackingId valor="1234567890">
    <TrackingDomain valor="FedEx"/>
    <MoneyCurrencyShipping valor="USD"/>
    <MoneyShipping valor="2.5"/>
    <Description valor="FedEx 2-day"/>
  </ShippingTrackingId>
  <Distribution>
    <Accounting valor="DistributionCharge">
      <Segment>
        <Type valor="G/L Account"/>
        <Id valor="23456"/>
        <Description valor="Entertainment"/>
      </Segment>
      <Segment>
        <Type valor="Cost Center"/>
        <Id valor="2323"/>
        <Description valor="Western Region Sales"/>
      </Segment>
    </Accounting>
    <Charge>
      <MoneyCurrencyCharge valor="USD"/>
      <MoneyCharge valor="0.34"/>
    </Charge>
```

```
        </Distribution>
        <Distribution>
          <Accounting name="DistributionCharge">
            <Segment>
              <Type valor="G/L Account"/>
              <Id valor="456"/>
              <Description valor="Travel"/>
            </Segment>
            <Segment>
              <Type valor="Cost Center"/>
              <Id valor="23"/>
              <Description valor="Europe Implementation"/>
            </Segment>
          </Accounting>
          <Charge>
            <MoneyCurrencyCharge valor="USD"/>
            <MoneyCharge valor="1"/>
          </Charge>
        </Distribution>
        <Comments valor="Anything well formed in XML can go here."/>
      </ItemOut>
    </OrderRequest>
  </Request>
</cXML>
```

Quadro A.1 - Instância XML de Origem do caso 5

O XML Schema foi dividido em várias figuras para que pudesse ser representado graficamente.

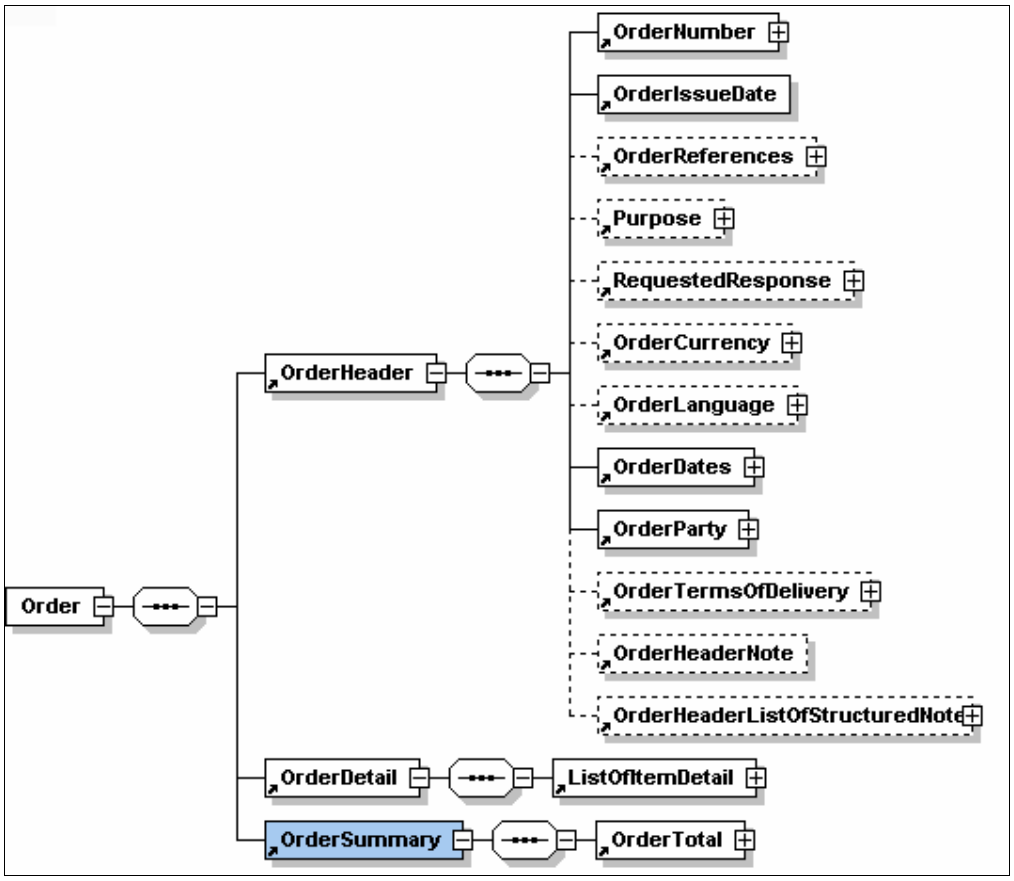

Figura A.1 - XML Schema destino do caso 5 (figura 1 / 10)

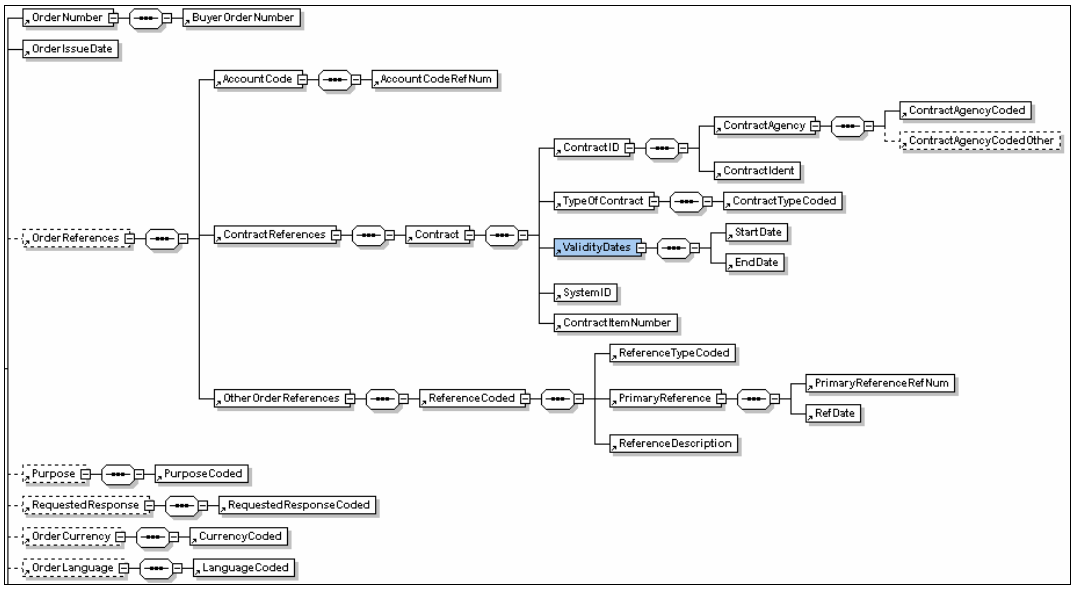

Figura A.2 - XML Schema destino do caso 5 (figura 2 / 10)

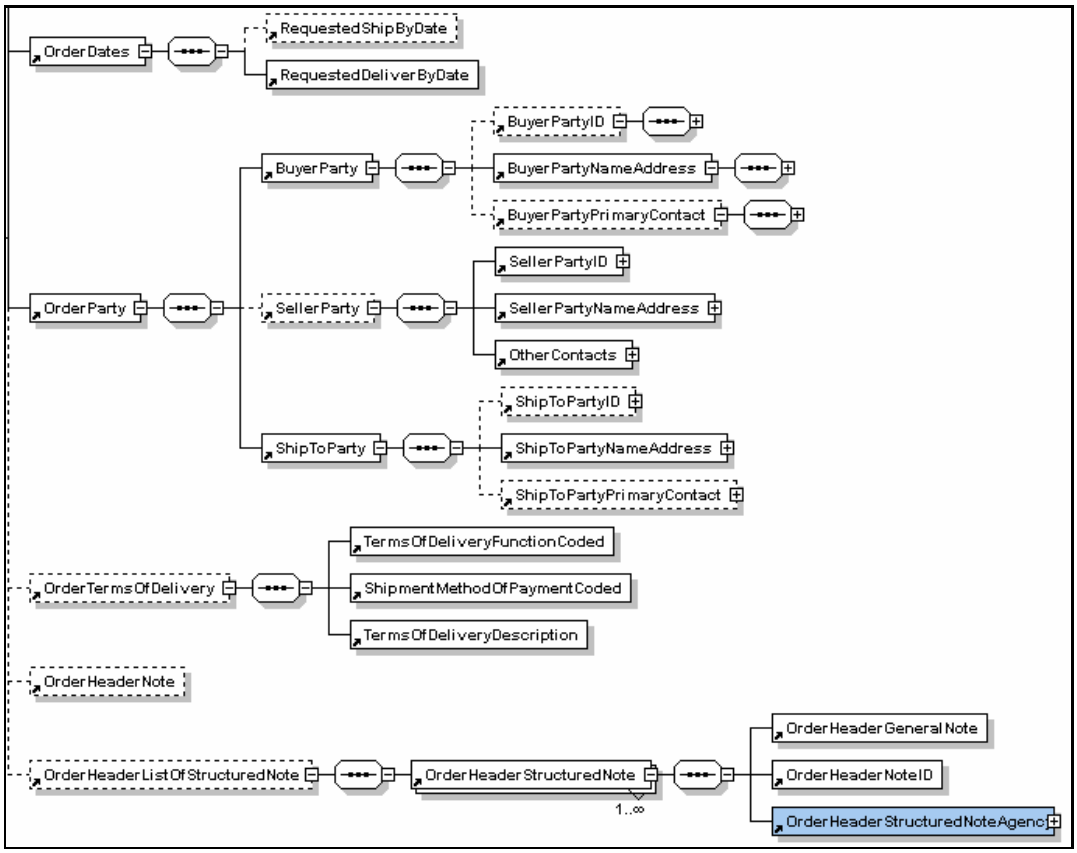

Figura A.3 - XML Schema destino do caso 5 (figura 3 / 10)

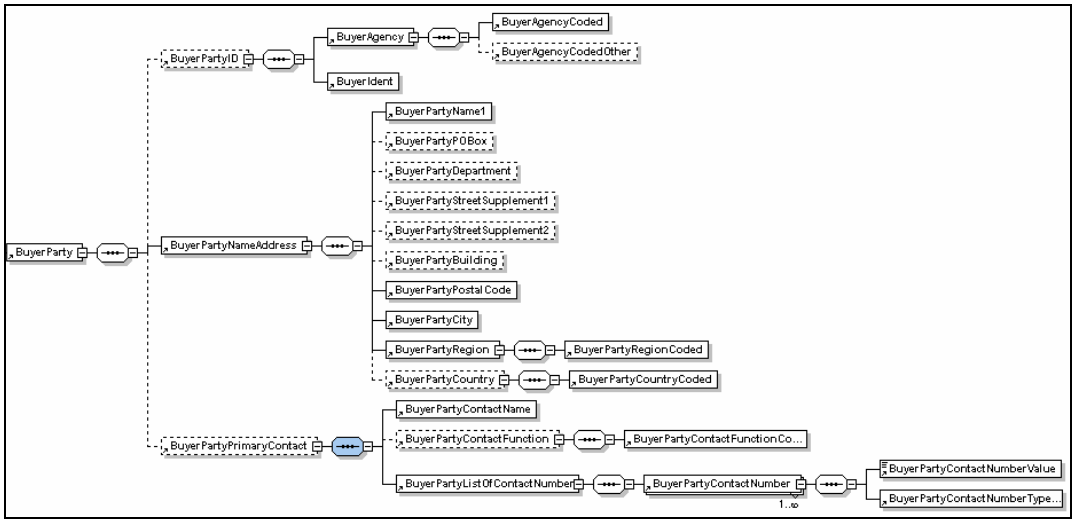

Figura A.4 - XML Schema destino do caso 5 (figura 4 / 10)

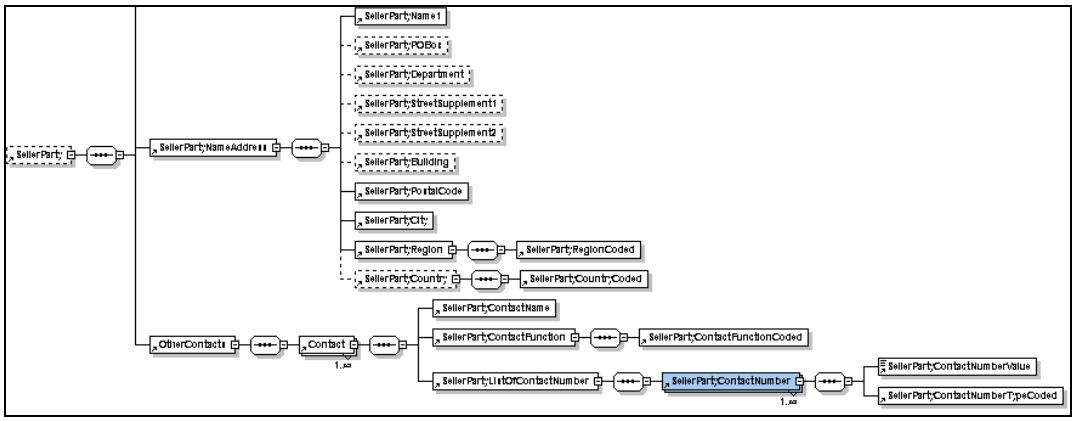

Figura A.5 - XML Schema destino do caso 5 (figura 5 / 10)

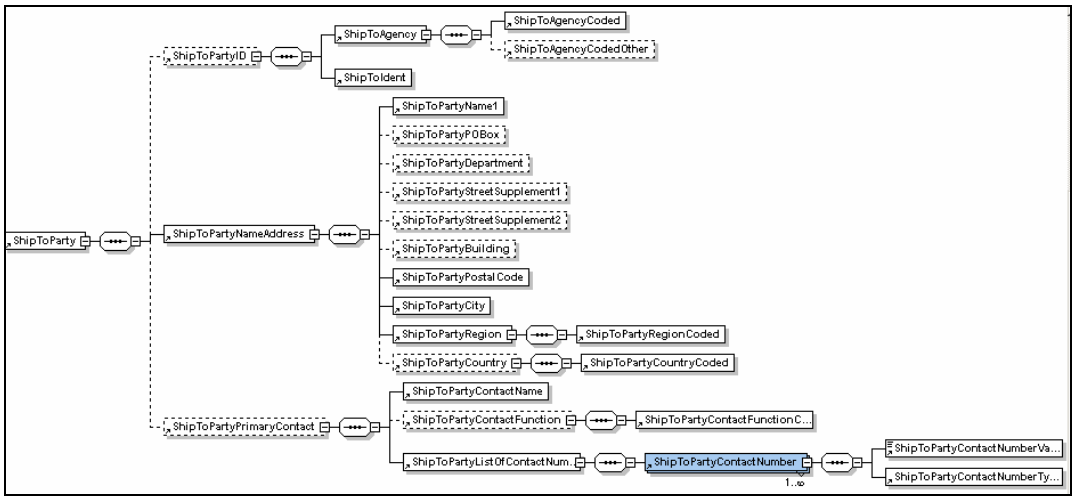

Figura A.6 - XML Schema destino do caso 5 (figura 6 / 10)

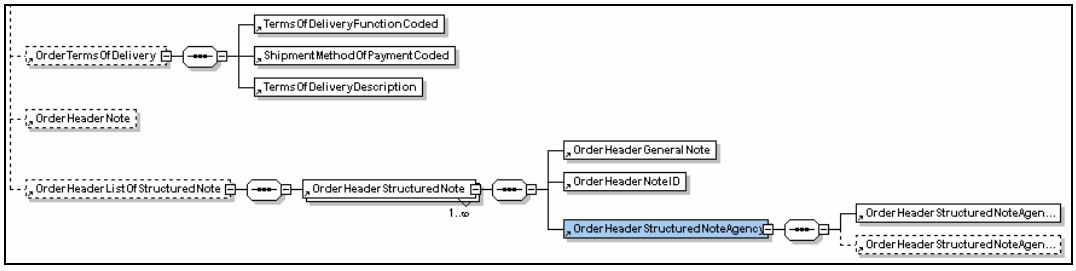

Figura A.7 - XML Schema destino do caso 5 (figura 7 / 10)

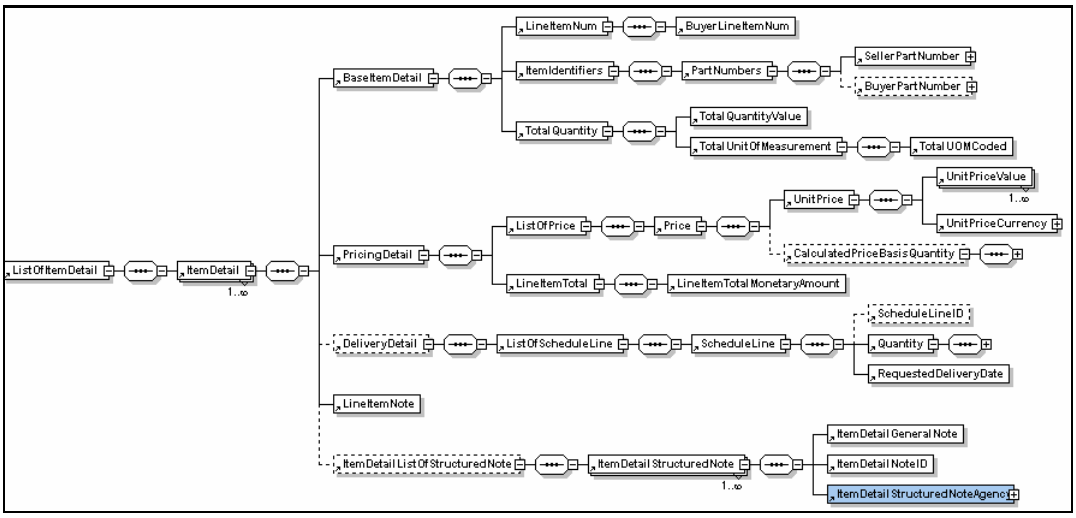

Figura A.8 - XML Schema destino do caso 5 (figura 8 / 10)

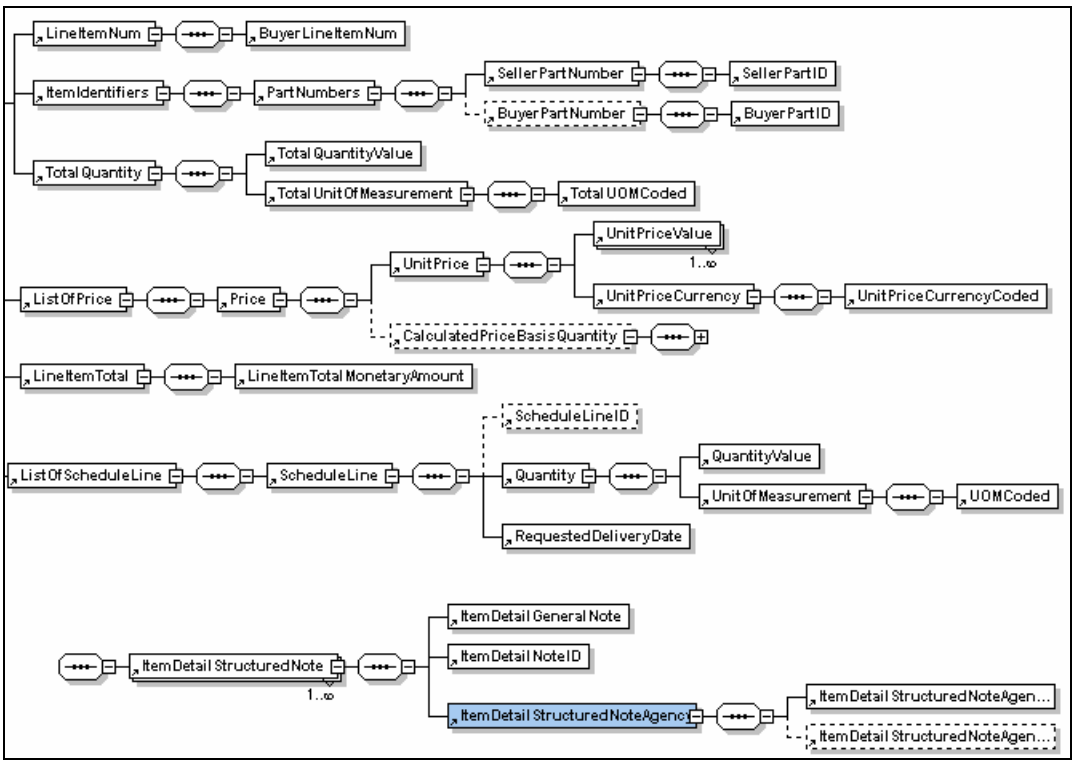

Figura A.9 - XML Schema destino do caso 5 (figura 9 / 10)

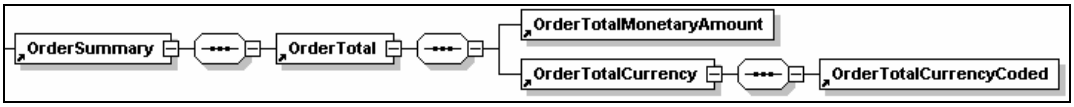

Figura A.10 - XML Schema destino do caso 5 (figura 10 / 10)

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  <dados>
    <origem path="E:\CVS\xml\ModelocXMLOrigem.xml" alias="mod1"/>
    <destino path="E:\CVS\xml\ModeloxCBLDestino.xml" alias="mod2"/>
  </dados>
  <sinonimos>
    <sinonimo>
            <nodeOrigem source="mod1">cXML</nodeOrigem>
            <nodeDestino source="mod2">order</nodeDestino>
    </sinonimo>
    <sinonimo>
            <nodeOrigem
source="mod1">OrderRequestHeaderID</nodeOrigem>
            <nodeDestino source="mod2">BuyerOrderNumber</nodeDestino>
    </sinonimo>
    <sinonimo>
            <nodeOrigem source="mod1">OrderRequestDate</nodeOrigem>
            <nodeDestino source="mod2">OrderIssueDate</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">MoneyCurrency</nodeOrigem>
      <nodeDestino source="mod2">CurrencyCoded</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">Money</nodeOrigem>
      <nodeDestino
source="mod2">OrderTotalMonetaryAmount</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">MoneyCurrency</nodeOrigem>
      <nodeDestino source="mod2">OrderTotalCurrencyCoded</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">Name</nodeOrigem>
      <nodeDestino source="mod2">ShipToPartyName1</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">Street</nodeOrigem>
      <nodeDestino
source="mod2">ShipToPartyStreetSupplement1</nodeDestino>
```

```
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">City</nodeOrigem>
      <nodeDestino source="mod2">ShipToPartyCity</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">State</nodeOrigem>
      <nodeDestino source="mod2">ShipToPartyRegionCoded</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">PostalCode</nodeOrigem>
      <nodeDestino source="mod2">ShipToPartyPostalCode</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">CountryCode</nodeOrigem>
      <nodeDestino source="mod2">ShipToPartyCountryCoded</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">NameBill</nodeOrigem>
      <nodeDestino source="mod2">BuyerPartyName1</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">CityBill</nodeOrigem>
      <nodeDestino source="mod2">BuyerPartyCity</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">StateBill</nodeOrigem>
      <nodeDestino source="mod2">BuyerPartyRegionCoded</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">PostalCodeBill</nodeOrigem>
      <nodeDestino source="mod2">BuyerPartyPostalCode</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">quantity</nodeOrigem>
      <nodeDestino source="mod2">TotalQuantityValue</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">requestedDeliveryDate</nodeOrigem>
      <nodeDestino source="mod2">RequestedDeliverByDate</nodeDestino>
    </sinonimo>
```

```
    <sinonimo>
      <nodeOrigem source="mod1">SupplierPartID</nodeOrigem>
      <nodeDestino source="mod2">SellerPartID</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">MoneyPrice</nodeOrigem>
      <nodeDestino
source="mod2">LineItemTotalMonetaryAmount</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">ItemOutDescription</nodeOrigem>
      <nodeDestino source="mod2">LineItemNote</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">UnitOfMeasure</nodeOrigem>
      <nodeDestino source="mod2">TotalUOMCoded</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">ManufacturerPartID</nodeOrigem>
      <nodeDestino source="mod2">BuyerLineItemNum</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">MoneyCurrencyCharge</nodeOrigem>
      <nodeDestino source="mod2">UnitPriceCurrencyCoded</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">MoneyCharge</nodeOrigem>
      <nodeDestino source="mod2">UnitPriceValue</nodeDestino>
    </sinonimo>
    <sinonimo>
      <nodeOrigem source="mod1">MoneyCharge</nodeOrigem>
      <nodeDestino source="mod2">UnitPriceCurrency</nodeDestino>
    </sinonimo>
  </sinonimos>
</definition>
```

Quadro A.2 - Mapeamentos dos nós do caso 5

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
```

```
      <xs:element name="Order">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="OrderHeader"/>
                        <xs:element ref="OrderDetail"/>
                        <xs:element ref="OrderSummary"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderCurrency">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="CurrencyCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="AccountCode">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="AccountCodeRefNum"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerAgency">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerAgencyCoded"/>
                        <xs:element ref="BuyerAgencyCodedOther"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerAgencyCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerAgencyCodedOther">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
```

```
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToAgency">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ShipToAgencyCoded"/>
                        <xs:element ref="ShipToAgencyCodedOther"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToAgencyCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToAgencyCodedOther">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerAgency">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerAgencyCoded"/>
                        <xs:element ref="SellerAgencyCodedOther"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerAgencyCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerAgencyCodedOther">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
```

```
      </xs:element>
      <xs:element name="ContractAgency">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ContractAgencyCoded"/>
                        <xs:element ref="ContractAgencyCodedOther"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ContractAgencyCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ContractAgencyCodedOther">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderHeaderStructuredNoteAgency">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="OrderHeaderStructuredNoteAgencyCoded"/>
                        <xs:element
ref="OrderHeaderStructuredNoteAgencyCodedOther" minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderHeaderStructuredNoteAgencyCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderHeaderStructuredNoteAgencyCodedOther">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
```

```
      </xs:element>
      <xs:element name="ItemDetailStructuredNoteAgency">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="ItemDetailStructuredNoteAgencyCoded"/>
                        <xs:element
ref="ItemDetailStructuredNoteAgencyCodedOther" minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ItemDetailStructuredNoteAgencyCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ItemDetailStructuredNoteAgencyCodedOther">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BaseItemDetail">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="LineItemNum"/>
                        <xs:element ref="ItemIdentifiers"/>
                        <xs:element ref="TotalQuantity"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyBuilding">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyBuilding">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
```

```
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyBuilding">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerLineItemNum">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerOrderNumber">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartNumber">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerPartID"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerParty">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerPartyID" minOccurs="0"/>
                        <xs:element ref="BuyerPartyNameAddress"/>
                        <xs:element ref="BuyerPartyPrimaryContact"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="CalculatedPriceBasisQuantity">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="CalculatedPriceQuantityValue"/>
```

```
                        <xs:element
ref="CalculatedPriceUnitOfMeasurement"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyCity">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyCity">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyCity">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="Contact">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerPartyContactName"/>
                        <xs:element ref="SellerPartyContactFunction"/>
                        <xs:element
ref="SellerPartyListOfContactNumber"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyContactFunction">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="SellerPartyContactFunctionCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
```

```
      <xs:element name="SellerPartyContactFunctionCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyContactFunction">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="ShipToPartyContactFunctionCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyContactFunctionCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyContactFunction">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="BuyerPartyContactFunctionCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyContactFunctionCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyContactName">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyContactName">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
```

```
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyContactName">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyContactNumber">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="BuyerPartyContactNumberValue"/>
                        <xs:element
ref="BuyerPartyContactNumberTypeCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyContactNumberTypeCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyContactNumberValue">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="valor"
type="xs:string"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyContactNumber">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="SellerPartyContactNumberValue"/>
                        <xs:element
ref="SellerPartyContactNumberTypeCoded"/>
                  </xs:sequence>
```

```
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyContactNumberTypeCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyContactNumberValue">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="valor"
type="xs:string"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyContactNumber">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="ShipToPartyContactNumberValue"/>
                        <xs:element
ref="ShipToPartyContactNumberTypeCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyContactNumberTypeCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyContactNumberValue">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute name="valor"
type="xs:string"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
```

```
      </xs:element>
      <xs:element name="Contract">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ContractID"/>
                        <xs:element ref="TypeOfContract"/>
                        <xs:element ref="ValidityDates"/>
                        <xs:element ref="SystemID"/>
                        <xs:element ref="ContractItemNumber"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ContractID">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ContractAgency"/>
                        <xs:element ref="ContractIdent"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ContractItemNumber">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ContractReferences">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="Contract"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ContractTypeCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyCountry">
            <xs:complexType>
```

```
                  <xs:sequence>
                        <xs:element ref="ShipToPartyCountryCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyCountry">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerPartyCountryCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyCountry">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerPartyCountryCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyCountryCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyCountryCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyCountryCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderTotalCurrency">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="OrderTotalCurrencyCoded"/>
```

```
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderTotalCurrencyCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="UnitPriceCurrency">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="UnitPriceCurrencyCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="UnitPriceCurrencyCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="CurrencyCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="DeliveryDetail">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ListOfScheduleLine"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyDepartment">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
```

```
      <xs:element name="BuyerPartyDepartment">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyDepartment">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="EndDate">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderHeaderGeneralNote">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ItemDetailGeneralNote">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerIdent">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToIdent">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerIdent">
```

```
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ContractIdent">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ItemDetail">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BaseItemDetail"/>
                        <xs:element ref="PricingDetail"/>
                        <xs:element ref="DeliveryDetail"
minOccurs="0"/>
                        <xs:element ref="LineItemNote"/>
                        <xs:element
ref="ItemDetailListOfStructuredNote" minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ItemIdentifiers">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="PartNumbers"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="LanguageCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="LineItemNote">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
```

```
      <xs:element name="LineItemNum">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerLineItemNum"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="LineItemTotal">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="LineItemTotalMonetaryAmount"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyListOfContactNumber">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerPartyContactNumber"
maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyListOfContactNumber">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ShipToPartyContactNumber"
maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyListOfContactNumber">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerPartyContactNumber"
maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ListOfItemDetail">
            <xs:complexType>
                  <xs:sequence>
```

```
                        <xs:element ref="ItemDetail"
maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ListOfPrice">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="Price"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ListOfScheduleLine">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ScheduleLine"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ItemDetailListOfStructuredNote">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ItemDetailStructuredNote"
maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderHeaderListOfStructuredNote">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="OrderHeaderStructuredNote"
maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="LineItemTotalMonetaryAmount">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderTotalMonetaryAmount">
```

```
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyName1">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyName1">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyName1">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyNameAddress">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ShipToPartyName1"/>
                        <xs:element ref="ShipToPartyPOBox"
minOccurs="0"/>
                        <xs:element ref="ShipToPartyDepartment"
minOccurs="0"/>
                        <xs:element ref="ShipToPartyStreetSupplement1"
minOccurs="0"/>
                        <xs:element ref="ShipToPartyStreetSupplement2"
minOccurs="0"/>
                        <xs:element ref="ShipToPartyBuilding"
minOccurs="0"/>
                        <xs:element ref="ShipToPartyPostalCode"/>
                        <xs:element ref="ShipToPartyCity"/>
                        <xs:element ref="ShipToPartyRegion"/>
                        <xs:element ref="ShipToPartyCountry"
minOccurs="0"/>
```

```
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyNameAddress">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerPartyName1"/>
                        <xs:element ref="BuyerPartyPOBox"
minOccurs="0"/>
                        <xs:element ref="BuyerPartyDepartment"
minOccurs="0"/>
                        <xs:element ref="BuyerPartyStreetSupplement1"
minOccurs="0"/>
                        <xs:element ref="BuyerPartyStreetSupplement2"
minOccurs="0"/>
                        <xs:element ref="BuyerPartyBuilding"
minOccurs="0"/>
                        <xs:element ref="BuyerPartyPostalCode"/>
                        <xs:element ref="BuyerPartyCity"/>
                        <xs:element ref="BuyerPartyRegion"/>
                        <xs:element ref="BuyerPartyCountry"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyNameAddress">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerPartyName1"/>
                        <xs:element ref="SellerPartyPOBox"
minOccurs="0"/>
                        <xs:element ref="SellerPartyDepartment"
minOccurs="0"/>
                        <xs:element ref="SellerPartyStreetSupplement1"
minOccurs="0"/>
                        <xs:element ref="SellerPartyStreetSupplement2"
minOccurs="0"/>
                        <xs:element ref="SellerPartyBuilding"
minOccurs="0"/>
                        <xs:element ref="SellerPartyPostalCode"/>
                        <xs:element ref="SellerPartyCity"/>
```

```
                        <xs:element ref="SellerPartyRegion"/>
                        <xs:element ref="SellerPartyCountry"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderHeaderNoteID">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ItemDetailNoteID">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderDates">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="RequestedShipByDate"
minOccurs="0"/>
                        <xs:element ref="RequestedDeliverByDate"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderDetail">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ListOfItemDetail"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderHeader">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="OrderNumber"/>
                        <xs:element ref="OrderIssueDate"/>
                        <xs:element ref="OrderReferences"
minOccurs="0"/>
```

```
                        <xs:element ref="Purpose" minOccurs="0"/>
                        <xs:element ref="RequestedResponse"
minOccurs="0"/>
                        <xs:element ref="OrderCurrency" minOccurs="0"/>
                        <xs:element ref="OrderLanguage" minOccurs="0"/>
                        <xs:element ref="OrderDates"/>
                        <xs:element ref="OrderParty"/>
                        <xs:element ref="OrderTermsOfDelivery"
minOccurs="0"/>
                        <xs:element ref="OrderHeaderNote"
minOccurs="0"/>
                        <xs:element
ref="OrderHeaderListOfStructuredNote" minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderHeaderNote">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderIssueDate">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderLanguage">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="LanguageCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderNumber">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerOrderNumber"/>
                  </xs:sequence>
            </xs:complexType>
```

```
      </xs:element>
      <xs:element name="OrderParty">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerParty"/>
                        <xs:element ref="SellerParty" minOccurs="0"/>
                        <xs:element ref="ShipToParty"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderReferences">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="AccountCode"/>
                        <xs:element ref="ContractReferences"/>
                        <xs:element ref="OtherOrderReferences"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderSummary">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="OrderTotal"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderTermsOfDelivery">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element
ref="TermsOfDeliveryFunctionCoded"/>
                        <xs:element
ref="ShipmentMethodOfPaymentCoded"/>
                        <xs:element ref="TermsOfDeliveryDescription"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderTotal">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="OrderTotalMonetaryAmount"/>
```

```
                        <xs:element ref="OrderTotalCurrency"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OtherContacts">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="Contact"
maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="OtherOrderReferences">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ReferenceCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyPOBox">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:short"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyPOBox">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:short"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyPOBox">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:short"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartID">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
```

```
      </xs:element>
      <xs:element name="SellerPartID">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="PartNumbers">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerPartNumber"/>
                        <xs:element ref="BuyerPartNumber"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyID">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerAgency"/>
                        <xs:element ref="BuyerIdent"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyID">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ShipToAgency"/>
                        <xs:element ref="ShipToIdent"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyID">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerAgency"/>
                        <xs:element ref="SellerIdent"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyPostalCode">
            <xs:complexType>
```

```
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyPostalCode">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyPostalCode">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="Price">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="UnitPrice"/>
                        <xs:element ref="CalculatedPriceBasisQuantity"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="PricingDetail">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ListOfPrice"/>
                        <xs:element ref="LineItemTotal"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyPrimaryContact">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerPartyContactName"/>
                        <xs:element ref="BuyerPartyContactFunction"
minOccurs="0"/>
                        <xs:element
ref="BuyerPartyListOfContactNumber"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
```

```
      <xs:element name="ShipToPartyPrimaryContact">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ShipToPartyContactName"/>
                        <xs:element ref="ShipToPartyContactFunction"
minOccurs="0"/>
                        <xs:element
ref="ShipToPartyListOfContactNumber"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="PrimaryReference">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="PrimaryReferenceRefNum"/>
                        <xs:element ref="RefDate"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="Purpose">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="PurposeCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="PurposeCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="Quantity">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="QuantityValue"/>
                        <xs:element ref="UnitOfMeasurement"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="QuantityValue">
```

```
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="CalculatedPriceQuantityValue">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="TotalQuantityValue">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="RefDate">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="AccountCodeRefNum">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="PrimaryReferenceRefNum">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ReferenceCoded">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ReferenceTypeCoded"/>
                        <xs:element ref="PrimaryReference"/>
                        <xs:element ref="ReferenceDescription"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ReferenceDescription">
            <xs:complexType>
```

```
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ReferenceTypeCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyRegion">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ShipToPartyRegionCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyRegion">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="BuyerPartyRegionCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyRegion">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerPartyRegionCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyRegionCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyRegionCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
```

```
      <xs:element name="SellerPartyRegionCoded">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="RequestedDeliverByDate">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="RequestedDeliveryDate">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="RequestedResponse">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="RequestedResponseCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="RequestedResponseCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="RequestedShipByDate">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ScheduleLine">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ScheduleLineID"
minOccurs="0"/>
                        <xs:element ref="Quantity"/>
```

```
                        <xs:element ref="RequestedDeliveryDate"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ScheduleLineID">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:short"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartNumber">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerPartID"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerParty">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="SellerPartyID"/>
                        <xs:element ref="SellerPartyNameAddress"/>
                        <xs:element ref="OtherContacts"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToParty">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ShipToPartyID" minOccurs="0"/>
                        <xs:element ref="ShipToPartyNameAddress"/>
                        <xs:element ref="ShipToPartyPrimaryContact"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipmentMethodOfPaymentCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
```

```
      </xs:element>
      <xs:element name="StartDate">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyStreetSupplement1">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyStreetSupplement1">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyStreetSupplement1">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="SellerPartyStreetSupplement2">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="BuyerPartyStreetSupplement2">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ShipToPartyStreetSupplement2">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
```

```
            </xs:complexType>
      </xs:element>
      <xs:element name="OrderHeaderStructuredNote">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="OrderHeaderGeneralNote"/>
                        <xs:element ref="OrderHeaderNoteID"/>
                        <xs:element
ref="OrderHeaderStructuredNoteAgency"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="ItemDetailStructuredNote">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ItemDetailGeneralNote"/>
                        <xs:element ref="ItemDetailNoteID"/>
                        <xs:element
ref="ItemDetailStructuredNoteAgency"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="SystemID">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="TermsOfDeliveryDescription">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="TermsOfDeliveryFunctionCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="TotalQuantity">
```

```
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="TotalQuantityValue"/>
                        <xs:element ref="TotalUnitOfMeasurement"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="TypeOfContract">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="ContractTypeCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="CalculatedPriceUOMCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="UOMCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="TotalUOMCoded">
            <xs:complexType>
                  <xs:attribute name="valor" type="xs:string"
use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="CalculatedPriceUnitOfMeasurement">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="CalculatedPriceUOMCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="UnitOfMeasurement">
            <xs:complexType>
```

```
                  <xs:sequence>
                        <xs:element ref="UOMCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="TotalUnitOfMeasurement">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="TotalUOMCoded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="UnitPrice">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="UnitPriceValue"
maxOccurs="unbounded"/>
                        <xs:element ref="UnitPriceCurrency"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="UnitPriceValue">
            <xs:complexType>
                  <xs:attribute name="valor" use="required"/>
            </xs:complexType>
      </xs:element>
      <xs:element name="ValidityDates">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="StartDate"/>
                        <xs:element ref="EndDate"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Quadro A.3 - XML Schema (fonte) do caso 5

```
CREATE TABLE [dbo].[ORIGEM] (
    [cXML] [varchar] (500),
    [Header] [varchar] (500),
```

```
    [From] [varchar] (500),
    [Credential] [varchar] (500),
    [Identity] [varchar] (500),
    [Credential_domain] [varchar] (500),
    [Credential_type] [varchar] (500),
    [To] [varchar] (500),
    [Sender] [varchar] (500),
    [SharedSecret] [varchar] (500),
    [UserAgent] [varchar] (500),
    [Request] [varchar] (500),
    [OrderRequest] [varchar] (500),
    [OrderRequestHeaderID] [varchar] (500),
    [OrderRequestDate] [varchar] (500),
    [OrderRequesttype] [varchar] (500),
    [Total] [varchar] (500),
    [MoneyCurrency] [varchar] (500),
    [Money] [varchar] (500),
    [ShipTo] [varchar] (500),
    [Address] [varchar] (500),
    [Name] [varchar] (500),
    [PostalAddress] [varchar] (500),
    [DeliverTo] [varchar] (500),
    [Street] [varchar] (500),
    [City] [varchar] (500),
    [State] [varchar] (500),
    [PostalCode] [varchar] (500),
    [CountryCode] [varchar] (500),
    [Country] [varchar] (500),
    [BillTo] [varchar] (500),
    [NameBill] [varchar] (500),
    [PostalAddressBill] [varchar] (500),
    [StreetBill] [varchar] (500),
    [CityBill] [varchar] (500),
    [StateBill] [varchar] (500),
    [PostalCodeBill] [varchar] (500),
    [CountryCodeBill] [varchar] (500),
    [CountryBill] [varchar] (500),
    [Tax] [varchar] (500),
    [MoneyCurrencyTax] [varchar] (500),
    [MoneyTax] [varchar] (500),
    [Description] [varchar] (500),
```

```
    [Payment] [varchar] (500),
    [PCard] [varchar] (500),
    [expiration] [varchar] (500),
    [Comments] [varchar] (500),
    [ItemOut] [varchar] (500),
    [quantity] [varchar] (500),
    [requestedDeliveryDate] [varchar] (500),
    [ItemID] [varchar] (500),
    [SupplierPartID] [varchar] (500),
    [ItemDetail] [varchar] (500),
    [UnitPrice] [varchar] (500),
    [MoneyCurrencyPrice] [varchar] (500),
    [MoneyPrice] [varchar] (500),
    [ItemOutDescription] [varchar] (500),
    [UnitOfMeasure] [varchar] (500),
    [Classification] [varchar] (500),
    [ManufacturerPartID] [varchar] (500),
    [ManufacturerName] [varchar] (500),
    [URL] [varchar] (500),
    [ShippingTrackingId] [varchar] (500),
    [TrackingDomain] [varchar] (500),
    [MoneyCurrencyShipping] [varchar] (500),
    [MoneyShipping] [varchar] (500),
    [Distribution] [varchar] (500),
    [Accounting] [varchar] (500),
    [Segment] [varchar] (500),
    [Type] [varchar] (500),
    [Id] [varchar] (500),
    [Charge] [varchar] (500),
    [MoneyCurrencyCharge] [varchar] (500),
    [MoneyCharge] [varchar] (500)
)
```

Quadro A.4 - Script SQL da tabela criada a partir da instância de origem

```
CREATE TABLE [dbo].[DESTINO] (
    [ShipToPartyRegionCoded] [varchar] (500),
    [ShipToPartyRegion] [varchar] (500),
    [SellerPartyStreetSupplement1] [varchar] (500),
    [PartNumbers] [varchar] (500),
    [SellerPartyStreetSupplement2] [varchar] (500),
```

```
    [OrderReferences] [varchar] (500),
    [ItemDetailGeneralNote] [varchar] (500),
    [ReferenceCoded] [varchar] (500),
    [OrderLanguage] [varchar] (500),
    [OrderDates] [varchar] (500),
    [SellerPartyCity] [varchar] (500),
    [SellerAgency] [varchar] (500),
    [PricingDetail] [varchar] (500),
    [ShipToPartyContactNumberTypeCoded] [varchar] (500),
    [BuyerOrderNumber] [varchar] (500),
    [OtherOrderReferences] [varchar] (500),
    [OtherContacts] [varchar] (500),
    [OrderSummary] [varchar] (500),
    [Contact] [varchar] (500),
    [OrderHeaderStructuredNote] [varchar] (500),
    [CalculatedPriceQuantityValue] [varchar] (500),
    [OrderHeaderListOfStructuredNote] [varchar] (500),
    [LanguageCoded] [varchar] (500),
    [BuyerAgency] [varchar] (500),
    [Order] [varchar] (500),
    [TermsOfDeliveryDescription] [varchar] (500),
    [SellerPartyNameAddress] [varchar] (500),
    [BuyerPartyID] [varchar] (500),
    [OrderTotalCurrencyCoded] [varchar] (500),
    [ContractAgencyCoded] [varchar] (500),
    [UnitPriceCurrency] [varchar] (500),
    [SellerPartyContactNumberValue] [varchar] (500),
    [OrderHeaderStructuredNoteAgency] [varchar] (500),
    [SellerPartyRegionCoded] [varchar] (500),
    [ItemDetailNoteID] [varchar] (500),
    [ContractAgencyCodedOther] [varchar] (500),
    [SellerPartyListOfContactNumber] [varchar] (500),
    [CurrencyCoded] [varchar] (500),
    [ShipToPartyDepartment] [varchar] (500),
    [SellerPartyID] [varchar] (500),
    [ContractAgency] [varchar] (500),
    [OrderNumber] [varchar] (500),
    [OrderHeaderNote] [varchar] (500),
    [OrderHeaderNoteID] [varchar] (500),
    [ListOfPrice] [varchar] (500),
    [CalculatedPriceBasisQuantity] [varchar] (500),
```

```
    [BuyerAgencyCoded] [varchar] (500),
    [BaseItemDetail] [varchar] (500),
    [SellerPartyPOBox] [varchar] (500),
    [TotalQuantity] [varchar] (500),
    [TotalQuantityValue] [varchar] (500),
    [ShipToPartyNameAddress] [varchar] (500),
    [UnitPrice] [varchar] (500),
    [OrderTotalCurrency] [varchar] (500),
    [RequestedResponseCoded] [varchar] (500),
    [SellerPartyContactNumber] [varchar] (500),
    [Purpose] [varchar] (500),
    [BuyerIdent] [varchar] (500),
    [BuyerPartyPOBox] [varchar] (500),
    [ItemDetailStructuredNote] [varchar] (500),
    [SellerAgencyCodedOther] [varchar] (500),
    [PrimaryReference] [varchar] (500),
    [SellerPartyContactNumberTypeCoded] [varchar] (500),
    [ScheduleLineID] [varchar] (500),
    [BuyerPartyRegionCoded] [varchar] (500),
    [OrderTermsOfDelivery] [varchar] (500),
    [EndDate] [varchar] (500),
    [BuyerPartyCountryCoded] [varchar] (500),
    [BuyerPartyCountry] [varchar] (500),
    [SellerPartyBuilding] [varchar] (500),
    [TypeOfContract] [varchar] (500),
    [RequestedResponse] [varchar] (500),
    [SellerPartyName1] [varchar] (500),
    [ShipToPartyContactFunctionCoded] [varchar] (500),
    [BuyerPartyContactNumberTypeCoded] [varchar] (500),
    [ShipToPartyPrimaryContact] [varchar] (500),
    [SellerPartyContactFunction] [varchar] (500),
    [SellerPartyDepartment] [varchar] (500),
    [OrderTotalMonetaryAmount] [varchar] (500),
    [ContractTypeCoded] [varchar] (500),
    [QuantityValue] [varchar] (500),
    [LineItemTotalMonetaryAmount] [varchar] (500),
    [ShipToPartyCity] [varchar] (500),
    [BuyerParty] [varchar] (500),
    [BuyerPartyName1] [varchar] (500),
    [BuyerPartyListOfContactNumber] [varchar] (500),
    [LineItemNote] [varchar] (500),
```

```
    [BuyerPartyDepartment] [varchar] (500),
    [ShipToPartyContactNumber] [varchar] (500),
    [ItemDetail] [varchar] (500),
    [UOMCoded] [varchar] (500),
    [ShipToPartyID] [varchar] (500),
    [ItemDetailStructuredNoteAgencyCodedOther] [varchar] (500),
    [OrderHeaderStructuredNoteAgencyCoded] [varchar] (500),
    [ShipToAgency] [varchar] (500),
    [SellerPartyPostalCode] [varchar] (500),
    [BuyerPartyStreetSupplement1] [varchar] (500),
    [ReferenceDescription] [varchar] (500),
    [BuyerPartyStreetSupplement2] [varchar] (500),
    [BuyerPartyBuilding] [varchar] (500),
    [ItemDetailStructuredNoteAgencyCoded] [varchar] (500),
    [ShipToAgencyCodedOther] [varchar] (500),
    [BuyerPartyPostalCode] [varchar] (500),
    [ValidityDates] [varchar] (500),
    [OrderTotal] [varchar] (500),
    [ShipToAgencyCoded] [varchar] (500),
    [CalculatedPriceUOMCoded] [varchar] (500),
    [SellerPartNumber] [varchar] (500),
    [ShipToPartyListOfContactNumber] [varchar] (500),
    [SellerPartyContactName] [varchar] (500),
    [BuyerAgencyCodedOther] [varchar] (500),
    [TotalUnitOfMeasurement] [varchar] (500),
    [LineItemTotal] [varchar] (500),
    [ShipToPartyCountry] [varchar] (500),
    [TotalUOMCoded] [varchar] (500),
    [StartDate] [varchar] (500),
    [SellerPartID] [varchar] (500),
    [ItemDetailListOfStructuredNote] [varchar] (500),
    [ListOfScheduleLine] [varchar] (500),
    [OrderCurrency] [varchar] (500),
    [RequestedDeliverByDate] [varchar] (500),
    [OrderIssueDate] [varchar] (500),
    [ListOfItemDetail] [varchar] (500),
    [CalculatedPriceUnitOfMeasurement] [varchar] (500),
    [ScheduleLine] [varchar] (500),
    [ReferenceTypeCoded] [varchar] (500),
    [SellerIdent] [varchar] (500),
    [ItemDetailStructuredNoteAgency] [varchar] (500),
```

```
    [UnitOfMeasurement] [varchar] (500),
    [ShipToPartyContactNumberValue] [varchar] (500),
    [ShipToPartyPostalCode] [varchar] (500),
    [BuyerPartID] [varchar] (500),
    [BuyerPartyNameAddress] [varchar] (500),
    [RequestedDeliveryDate] [varchar] (500),
    [ContractIdent] [varchar] (500),
    [ShipToIdent] [varchar] (500),
    [SellerAgencyCoded] [varchar] (500),
    [SystemID] [varchar] (500),
    [BuyerPartyRegion] [varchar] (500),
    [Price] [varchar] (500),
    [BuyerPartyContactNumber] [varchar] (500),
    [TermsOfDeliveryFunctionCoded] [varchar] (500),
    [ShipToPartyCountryCoded] [varchar] (500),
    [ShipToPartyContactName] [varchar] (500),
    [SellerParty] [varchar] (500),
    [DeliveryDetail] [varchar] (500),
    [ShipmentMethodOfPaymentCoded] [varchar] (500),
    [BuyerPartyContactFunctionCoded] [varchar] (500),
    [BuyerPartyCity] [varchar] (500),
    [BuyerLineItemNum] [varchar] (500),
    [OrderHeader] [varchar] (500),
    [ContractID] [varchar] (500),
    [BuyerPartyContactName] [varchar] (500),
    [RequestedShipByDate] [varchar] (500),
    [BuyerPartyPrimaryContact] [varchar] (500),
    [OrderHeaderGeneralNote] [varchar] (500),
    [SellerPartyCountry] [varchar] (500),
    [BuyerPartyContactNumberValue] [varchar] (500),
    [PrimaryReferenceRefNum] [varchar] (500),
    [ShipToParty] [varchar] (500),
    [SellerPartyRegion] [varchar] (500),
    [RefDate] [varchar] (500),
    [ShipToPartyPOBox] [varchar] (500),
    [AccountCodeRefNum] [varchar] (500),
    [Quantity] [varchar] (500),
    [OrderDetail] [varchar] (500),
    [BuyerPartyContactFunction] [varchar] (500),
    [ShipToPartyContactFunction] [varchar] (500),
    [UnitPriceValue] [varchar] (500),
```

```
    [ItemIdentifiers] [varchar] (500),
    [SellerPartyCountryCoded] [varchar] (500),
    [OrderHeaderStructuredNoteAgencyCodedOther] [varchar] (500),
    [LineItemNum] [varchar] (500),
    [ContractReferences] [varchar] (500),
    [ShipToPartyStreetSupplement1] [varchar] (500),
    [OrderParty] [varchar] (500),
    [ShipToPartyName1] [varchar] (500),
    [ShipToPartyBuilding] [varchar] (500),
    [ShipToPartyStreetSupplement2] [varchar] (500),
    [AccountCode] [varchar] (500),
    [PurposeCoded] [varchar] (500),
    [UnitPriceCurrencyCoded] [varchar] (500),
    [Contract] [varchar] (500),
    [BuyerPartNumber] [varchar] (500),
    [ContractItemNumber] [varchar] (500),
    [SellerPartyContactFunctionCoded] [varchar] (500)
)
```

Quadro A.5 - Script SQL da tabela criada a partir do XML Schema de destino

```
<Order>
  <OrderHeader>
    <OrderNumber>
      <BuyerOrderNumber/>
    </OrderNumber>
    <OrderIssueDate/>
    <OrderReferences>
      <AccountCode>
        <AccountCodeRefNum/>
      </AccountCode>
      <ContractReferences>
        <Contract>
          <ContractID>
            <ContractAgency>
              <ContractAgencyCoded/>
              <ContractAgencyCodedOther/>
            </ContractAgency>
            <ContractIdent/>
          </ContractID>
          <TypeOfContract>
```

```
            <ContractTypeCoded/>
          </TypeOfContract>
          <ValidityDates>
            <StartDate/>
            <EndDate/>
          </ValidityDates>
          <SystemID/>
          <ContractItemNumber/>
        </Contract>
      </ContractReferences>
      <OtherOrderReferences>
        <ReferenceCoded>
          <ReferenceTypeCoded/>
          <PrimaryReference>
            <PrimaryReferenceRefNum/>
            <RefDate/>
          </PrimaryReference>
          <ReferenceDescription/>
        </ReferenceCoded>
      </OtherOrderReferences>
    </OrderReferences>
    <Purpose>
      <PurposeCoded/>
    </Purpose>
    <RequestedResponse>
      <RequestedResponseCoded/>
    </RequestedResponse>
    <OrderCurrency>
      <CurrencyCoded/>
    </OrderCurrency>
    <OrderLanguage>
      <LanguageCoded/>
    </OrderLanguage>
    <OrderDates>
      <RequestedShipByDate/>
      <RequestedDeliverByDate/>
    </OrderDates>
    <OrderParty>
      <BuyerParty>
        <BuyerPartyID>
          <BuyerAgency>
```

```
            <BuyerAgencyCoded/>
            <BuyerAgencyCodedOther/>
          </BuyerAgency>
          <BuyerIdent/>
        </BuyerPartyID>
        <BuyerPartyNameAddress>
          <BuyerPartyName1/>
          <BuyerPartyPOBox/>
          <BuyerPartyDepartment/>
          <BuyerPartyStreetSupplement1/>
          <BuyerPartyStreetSupplement2/>
          <BuyerPartyBuilding/>
          <BuyerPartyPostalCode/>
          <BuyerPartyCity/>
          <BuyerPartyRegion>
            <BuyerPartyRegionCoded/>
          </BuyerPartyRegion>
          <BuyerPartyCountry>
            <BuyerPartyCountryCoded/>
          </BuyerPartyCountry>
        </BuyerPartyNameAddress>
        <BuyerPartyPrimaryContact>
          <BuyerPartyContactName/>
          <BuyerPartyContactFunction>
            <BuyerPartyContactFunctionCoded/>
          </BuyerPartyContactFunction>
          <BuyerPartyListOfContactNumber>
            <BuyerPartyContactNumber>
              <BuyerPartyContactNumberValue/>
              <BuyerPartyContactNumberTypeCoded/>
            </BuyerPartyContactNumber>
          </BuyerPartyListOfContactNumber>
        </BuyerPartyPrimaryContact>
      </BuyerParty>
      <SellerParty>
        <SellerPartyID>
          <SellerAgency>
            <SellerAgencyCoded/>
            <SellerAgencyCodedOther/>
          </SellerAgency>
          <SellerIdent/>
```

```
        </SellerPartyID>
        <SellerPartyNameAddress>
          <SellerPartyName1/>
          <SellerPartyPOBox/>
          <SellerPartyDepartment/>
          <SellerPartyStreetSupplement1/>
          <SellerPartyStreetSupplement2/>
          <SellerPartyBuilding/>
          <SellerPartyPostalCode/>
          <SellerPartyCity/>
          <SellerPartyRegion>
            <SellerPartyRegionCoded/>
          </SellerPartyRegion>
          <SellerPartyCountry>
            <SellerPartyCountryCoded/>
          </SellerPartyCountry>
        </SellerPartyNameAddress>
        <OtherContacts>
          <Contact>
            <SellerPartyContactName/>
            <SellerPartyContactFunction>
              <SellerPartyContactFunctionCoded/>
            </SellerPartyContactFunction>
            <SellerPartyListOfContactNumber>
              <SellerPartyContactNumber>
                <SellerPartyContactNumberValue/>
                <SellerPartyContactNumberTypeCoded/>
              </SellerPartyContactNumber>
            </SellerPartyListOfContactNumber>
          </Contact>
        </OtherContacts>
      </SellerParty>
      <ShipToParty>
        <ShipToPartyID>
          <ShipToAgency>
            <ShipToAgencyCoded/>
            <ShipToAgencyCodedOther/>
          </ShipToAgency>
          <ShipToIdent/>
        </ShipToPartyID>
        <ShipToPartyNameAddress>
```

```
          <ShipToPartyName1/>
          <ShipToPartyPOBox/>
          <ShipToPartyDepartment/>
          <ShipToPartyStreetSupplement1/>
          <ShipToPartyStreetSupplement2/>
          <ShipToPartyBuilding/>
          <ShipToPartyPostalCode/>
          <ShipToPartyCity/>
          <ShipToPartyRegion>
            <ShipToPartyRegionCoded/>
          </ShipToPartyRegion>
          <ShipToPartyCountry>
            <ShipToPartyCountryCoded/>
          </ShipToPartyCountry>
        </ShipToPartyNameAddress>
        <ShipToPartyPrimaryContact>
          <ShipToPartyContactName/>
          <ShipToPartyContactFunction>
            <ShipToPartyContactFunctionCoded/>
          </ShipToPartyContactFunction>
          <ShipToPartyListOfContactNumber>
            <ShipToPartyContactNumber>
              <ShipToPartyContactNumberValue/>
              <ShipToPartyContactNumberTypeCoded/>
            </ShipToPartyContactNumber>
          </ShipToPartyListOfContactNumber>
        </ShipToPartyPrimaryContact>
      </ShipToParty>
    </OrderParty>
    <OrderTermsOfDelivery>
      <TermsOfDeliveryFunctionCoded/>
      <ShipmentMethodOfPaymentCoded/>
      <TermsOfDeliveryDescription/>
    </OrderTermsOfDelivery>
    <OrderHeaderNote/>
    <OrderHeaderListOfStructuredNote>
      <OrderHeaderStructuredNote>
        <OrderHeaderGeneralNote/>
        <OrderHeaderNoteID/>
        <OrderHeaderStructuredNoteAgency>
          <OrderHeaderStructuredNoteAgencyCoded/>
```

```
          <OrderHeaderStructuredNoteAgencyCodedOther/>
        </OrderHeaderStructuredNoteAgency>
      </OrderHeaderStructuredNote>
    </OrderHeaderListOfStructuredNote>
  </OrderHeader>
  <OrderDetail>
    <ListOfItemDetail>
      <ItemDetail>
        <BaseItemDetail>
          <LineItemNum>
            <BuyerLineItemNum/>
          </LineItemNum>
          <ItemIdentifiers>
            <PartNumbers>
              <SellerPartNumber>
                <SellerPartID/>
              </SellerPartNumber>
              <BuyerPartNumber>
                <BuyerPartID/>
              </BuyerPartNumber>
            </PartNumbers>
          </ItemIdentifiers>
          <TotalQuantity>
            <TotalQuantityValue/>
            <TotalUnitOfMeasurement>
              <TotalUOMCoded/>
            </TotalUnitOfMeasurement>
          </TotalQuantity>
        </BaseItemDetail>
        <PricingDetail>
          <ListOfPrice>
            <Price>
              <UnitPrice>
                <UnitPriceValue/>
                <UnitPriceCurrency>
                  <UnitPriceCurrencyCoded/>
                </UnitPriceCurrency>
              </UnitPrice>
              <CalculatedPriceBasisQuantity>
                <CalculatedPriceQuantityValue/>
                <CalculatedPriceUnitOfMeasurement>
```

```
                <CalculatedPriceUOMCoded/>
              </CalculatedPriceUnitOfMeasurement>
            </CalculatedPriceBasisQuantity>
          </Price>
        </ListOfPrice>
        <LineItemTotal>
          <LineItemTotalMonetaryAmount/>
        </LineItemTotal>
      </PricingDetail>
      <DeliveryDetail>
        <ListOfScheduleLine>
          <ScheduleLine>
            <ScheduleLineID/>
            <Quantity>
              <QuantityValue/>
              <UnitOfMeasurement>
                <UOMCoded/>
              </UnitOfMeasurement>
            </Quantity>
            <RequestedDeliveryDate/>
          </ScheduleLine>
        </ListOfScheduleLine>
      </DeliveryDetail>
      <LineItemNote/>
      <ItemDetailListOfStructuredNote>
        <ItemDetailStructuredNote>
          <ItemDetailGeneralNote/>
          <ItemDetailNoteID/>
          <ItemDetailStructuredNoteAgency>
            <ItemDetailStructuredNoteAgencyCoded/>
            <ItemDetailStructuredNoteAgencyCodedOther/>
          </ItemDetailStructuredNoteAgency>
        </ItemDetailStructuredNote>
      </ItemDetailListOfStructuredNote>
    </ItemDetail>
  </ListOfItemDetail>
</OrderDetail>
<OrderSummary>
  <OrderTotal>
    <OrderTotalMonetaryAmount/>
    <OrderTotalCurrency>
```

```
        <OrderTotalCurrencyCoded/>
      </OrderTotalCurrency>
    </OrderTotal>
  </OrderSummary>
</Order>
```

Quadro A.6 - Árvore de aceitação do Caso 5

```
<?xml version="1.0" encoding="UTF-8"?>
<Order>
  <OrderHeader>
    <OrderNumber>
      <BuyerOrderNumber valor="DO1234"/>
    </OrderNumber>
    <OrderIssueDate valor="1999-03-12"/>
    <OrderCurrency>
      <CurrencyCoded valor="USD"/>
    </OrderCurrency>
    <OrderDates>
      <RequestedDeliverByDate valor="1999-03-12"/>
    </OrderDates>
    <OrderParty>
      <BuyerParty>
        <BuyerPartyNameAddress>
          <BuyerPartyName1 valor="Acme"/>
          <BuyerPartyPostalCode valor="90489"/>
          <BuyerPartyCity valor="Sunnyvale"/>
          <BuyerPartyRegion>
            <BuyerPartyRegionCoded valor="CA"/>
          </BuyerPartyRegion>
        </BuyerPartyNameAddress>
      </BuyerParty>
      <ShipToParty>
        <ShipToPartyNameAddress>
          <ShipToPartyName1 valor="Acme"/>
          <ShipToPartyStreetSupplement1 valor="123 Anystreet"/>
          <ShipToPartyPostalCode valor="90489"/>
          <ShipToPartyCity valor="Sunnyvale"/>
          <ShipToPartyRegion>
            <ShipToPartyRegionCoded valor="CA"/>
```

```
          </ShipToPartyRegion>
          <ShipToPartyCountry>
            <ShipToPartyCountryCoded valor="US"/>
          </ShipToPartyCountry>
        </ShipToPartyNameAddress>
      </ShipToParty>
    </OrderParty>
  </OrderHeader>
  <OrderDetail>
    <ListOfItemDetail>
      <ItemDetail>
        <BaseItemDetail>
          <LineItemNum>
            <BuyerLineItemNum valor="234"/>
          </LineItemNum>
          <ItemIdentifiers>
            <PartNumbers>
              <SellerPartNumber>
                <SellerPartID valor="1233244"/>
              </SellerPartNumber>
            </PartNumbers>
          </ItemIdentifiers>
          <TotalQuantity>
            <TotalQuantityValue valor="2"/>
            <TotalUnitOfMeasurement>
              <TotalUOMCoded valor="EA"/>
            </TotalUnitOfMeasurement>
          </TotalQuantity>
        </BaseItemDetail>
        <PricingDetail>
          <ListOfPrice>
            <Price>
              <UnitPrice>
                <UnitPriceValue valor="0.34"/>
                <UnitPriceValue valor="1"/>
                <UnitPriceCurrency>
                  <UnitPriceCurrencyCoded valor="USD"/>
                </UnitPriceCurrency>
              </UnitPrice>
            </Price>
          </ListOfPrice>
```

```
          <LineItemTotal>
            <LineItemTotalMonetaryAmount valor="1.34"/>
          </LineItemTotal>
        </PricingDetail>
        <LineItemNote valor="hello"/>
      </ItemDetail>
    </ListOfItemDetail>
   </OrderDetail>
   <OrderSummary>
     <OrderTotal>
       <OrderTotalMonetaryAmount valor="2.68"/>
       <OrderTotalCurrency>
         <OrderTotalCurrencyCoded valor="USD"/>
       </OrderTotalCurrency>
      </OrderTotal>
   </OrderSummary>
</Order>
```

Quadro A.7 - Instância XML de destino gerada