RUI JORGE TRAMONTIN JUNIOR

CONFIGURAÇÃO E INTEGRAÇÃO DE DADOS EM PLATAFORMAS PARA EMPRESAS VIRTUAIS

FLORIANÓPOLIS
2004

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

CONFIGURAÇÃO E INTEGRAÇÃO DE DADOS EM PLATAFORMAS PARA EMPRESAS VIRTUAIS

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica

RUI JORGE TRAMONTIN JUNIOR

Florianópolis, Setembro 2004

CONFIGURAÇÃO E INTEGRAÇÃO DE DADOS EM PLATAFORMAS PARA EMPRESAS VIRTUAIS

Rui Jorge Tramontin Junior

‘Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em Automação e Sistemas, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina’

_____
Prof. Ricardo José Rabelo, Dr.
Orientador

_____
Prof. Denizar Cruz Martins, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

_____
Prof. Ricardo José Rabelo, Dr.
Presidente

_____
Prof. Frank Augusto Siqueira, Dr.

_____
Prof. Jean-Marie Farines, Dr. Eng.

_____
Prof. Rômulo Silva de Oliveira, Dr. Eng.

Agradecimentos

Ao longo da escrita desta dissertação pude contar com a ajuda de diversas pessoas que sempre estiveram presentes e que de alguma forma deram a sua contribuição. Posso afirmar, sem sombra de dúvida, que estas pessoas foram fundamentais para o término bem sucedido deste trabalho.

Primeiramente gostaria de agradecer aos meus pais, que nunca pouparam esforços para me apoiar e sempre me guiaram pelo caminho certo.

Agradeço também à minha noiva Marília, por estar sempre ao meu lado, principalmente nos momentos mais difíceis, e também pela paciência nos fins de semana que mal ficamos juntos.

Agradeço imensamente ao professor e amigo Ricardo Rabelo, não apenas pela essencial tarefa de orientar este trabalho, mas também por me dar todo apoio necessário nos momentos difíceis.

É preciso agradecer também aos meus companheiros do GSIGMA, Carlos Eduardo Gesser, Fabiano Baldo, Leandro Loss, Edmilson Rampazo Klen e Alexandra Pereira Klen, por todo o auxílio tanto na escrita quanto na implementação, e pelo ótimo ambiente de trabalho proporcionado por todos.

Aos professores e funcionários do programa de pós-graduação, por proporcionarem as condições necessárias ao meu crescimento ao longo do curso; e aos membros da banca cujas sugestões e críticas contribuíram para o engrandecimento deste trabalho.

Agradeço também à família Rathje, que sempre me apoiou ao longo desta caminhada: meus sogros, Clóvis e Regina, e meus cunhados, Mayla e Clóvis Filho.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

CONFIGURAÇÃO E INTEGRAÇÃO DE DADOS EM PLATAFORMAS PARA EMPRESAS VIRTUAIS

Rui Jorge Tramontin Junior

Setembro, 2004

Orientador: Prof. Ricardo José Rabelo, Dr.
Área de Concentração: Automação e Sistemas.
Palavras Chave: Empresas Virtuais, Configuração de dados, XML, XML *Data Binding*, *Middleware* XML, *Enterprise Application Integration*.
Número de Páginas: 194

RESUMO: A evolução das tecnologias de informação e comunicação permitiu o desenvolvimento de novos modelos de negócio, onde as relações comerciais entre empresas podem ser realizadas de forma mais ágil, independente de onde estão localizadas. Neste contexto, o paradigma de Empresas Virtuais (EV) vem se destacando, dada a base que cria para as empresas atingirem com eficiência e flexibilidade novos mercados. Uma EV consiste no agrupamento temporário e dinâmico de empresas que cooperam entre si para atender a uma oportunidade de negócio, sendo esta cooperação suportada por redes de computadores. Apesar dos inúmeros trabalhos já desenvolvidos na área da EVs, vários aspectos importantes podem ser considerados como naturalmente pouco explorados, devido ao fato de que a área em si é ainda relativamente nova. Esta dissertação focaliza a etapa de criação da EV, onde são propostos *mecanismos assistidos por computador* como suporte a algumas das atividades realizadas nesta etapa, com o objetivo de aumentar a agilidade da EV em responder a uma oportunidade de negócio. Mais especificamente, esta dissertação aborda o aspecto de configuração de dados em plataformas para EV, em dois níveis: i) a definição do formato da estrutura de dados a ser intercambiada entre os membros da EV, e ii) a integração da plataforma da EV com os sistemas legados de cada membro. A razão para o foco nestas duas atividades está no fato de que dentre todos os trabalhos avaliados, elas têm sido executadas de forma manual ou com pouca assistência, fazendo com que o processo de configuração esteja sujeito a muitos erros, seja moroso (e mais custoso), com pouca ou nenhuma flexibilidade para reconfigurações, e sem um método. Esta dissertação visa contribuir na direção da solução desses problemas. Para validar a proposta apresentada, um protótipo computacional foi implementado, usando como tecnologias de informação a linguagem XML e banco de dados. A validação foi feita no âmbito do projeto *MyFashion.eu*, um projeto de cooperação internacional entre Brasil e Europa que contou com quatro indústrias-piloto.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements of the degree of Master in Electrical Engineering.

DATA CONFIGURATION AND INTEGRATION IN PLATFORMS FOR VIRTUAL ENTERPRISES

Rui Jorge Tramontin Junior

September, 2004

Advisor: Ricardo José Rabelo, Dr.
Area of Concentration: Automation and Systems.
Keywords: Virtual Enterprises, Data Configuration, XML, XML Data Binding, XML Middleware, Enterprise Application Integration.
Number of Pages: 194

ABSTRACT: The evolution of the information and communication technologies has enabled the development of new business models, where the commercial relations among enterprises can be performed in a more agile way, regardless of their locations. In this scenario, the Virtual Enterprise (VE) paradigm has emerged to allow enterprises to achieve new markets in an efficient and flexible way. A VE is a temporary and dynamic alliance of enterprises that cooperate each other to address a given business opportunity, and whose cooperation is supported by computer networks. Although there are many works developed in the VE area, some important aspects still can be considered as not yet fully explored, owing to this area itself is still relatively new. This dissertation focuses the VE creation phase, where *computer supported mechanisms* are proposed to assist some of the activities done in this phase. It aims to increase the agility to the VE to better respond to a given business opportunity. More specifically, this work focuses the aspect of data configuration in VE platforms in two levels: i) the definition of the data structure format to be exchanged by the VE members, and ii) the integration between the VE platform and the legacy systems of each VE member. The main reason to focus on these two activities is that in all studied researches, they have been performed manually or with little support. Therefore, it makes all this configuration process error-prone, slow (and expensive), with none or little flexibility to reconfigurations, and without a method. This dissertation aims to contribute towards the solution of these problems. In order to validate the presented proposal, a computational prototype was developed using information technologies such as XML and databases. The validation was made in the scope of the *MyFashion.eu* project, an international cooperation project between Brazil and Europe. On this project were participating four pilot-industries.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

O mundo dos negócios vive atualmente em um estado de crescente globalização, onde as relações comerciais ocorrem a nível mundial, suportadas pelas tecnologias de informação e de comunicação, predominantemente as baseadas na Internet. Neste cenário, novos paradigmas e tecnologias têm surgido para dar suporte a essas relações, onde as empresas precisam atuar de forma cada vez mais competitiva.

Um dos mais proeminentes paradigmas resultantes dessa tendência é o de Empresas Virtuais (EV), que se apresenta como uma área de crescente interesse em pesquisa e desenvolvimento tecnológico. Em linhas gerais, uma EV consiste no agrupamento temporário de empresas que cooperam entre si para melhor responder a uma oportunidade de negócios. As empresas participantes compartilham informações, habilidades e recursos para este fim, sendo esta cooperação suportada por tecnologias de informação e comunicação (CAMARINHA-MATOS *et al.*, 1999a). Pode-se dizer que o objetivo de uma EV é produzir o que foi pedido dentro dos critérios acordados e entregá-lo a quem o pediu.

O sucesso de uma EV depende essencialmente da *agilidade* com que é criada para atender a uma necessidade ou oportunidade de mercado, da sua *flexibilidade* para absorver os problemas que surjam durante a sua operação (ou seja, durante a produção do que foi pedido), e da sua *capacidade* para executar o que fora pedido dentro dos prazos estipulados e da qualidade requerida (RABELO *et al.*, 2003). Há que se considerar ainda que a composição de uma EV pode ser variável de negócio para negócio, ou seja, há EVs mais estáticas, onde seus parceiros costumam trabalhar em conjunto como que numa aliança, e há as mais dinâmicas, nas quais a composição é bastante variável/volátil, onde novos parceiros podem entrar e outros podem sair sempre que necessário.

O pré-requisito básico para que uma empresa participe de uma EV é a utilização de uma infraestrutura que dê o suporte necessário para a interoperação com os demais membros da EV. Esta infraestrutura é usualmente chamada de *plataforma da EV*, e é através dela que toda troca de informações entre os parceiros de uma EV acontece.

No entanto, para que os membros de uma EV possam intercambiar informações entre si, é necessário definir uma estrutura de dados comum (sintaxe e semântica) para este intercâmbio. Este processo normalmente consome um considerável período de tempo, onde refinamentos progressivos são realizados até se chegar a um modelo final em que todas as

partes estão de acordo. Isto significa que sempre que a plataforma da EV é testada (para validar a estrutura), cada eventual modificação na estrutura da informação a ser intercambiada implicará na redefinição de parte da plataforma da EV, tais como o projeto da base de dados, os processadores das mensagens, etc. Em conseqüência disso, este processo costuma ser bastante trabalhoso, consumindo tempo e dinheiro.

Outra questão importante diz respeito à integração da plataforma da EV com os sistemas legados de cada parceiro. Como os membros da EV precisam intercambiar informações, é necessário integrar a plataforma da EV com seus sistemas legados (isto é, com aqueles que provêm as informações de cada empresa). Entretanto, a natural heterogeneidade entre diferentes ambientes e sistemas representa um desafio para este processo, que também demanda uma parcela considerável de tempo e dinheiro. Estimativas apontam que projetos de integração consomem até 30% do orçamento destinado ao desenvolvimento de tecnologias de informação em grandes organizações (LEAR, 1999), e podem levar vários meses de trabalho.

1.1 Delimitação do Trabalho

Este trabalho é focalizado no aspecto da *agilidade*, contribuindo para a solução de alguns dos problemas no âmbito da *criação* de uma EV. Esta agilidade envolve principalmente fatores tecnológicos, organizacionais, legais e culturais (CAMARINHA-MATOS *et al.*, 2000). Embora tenham igual relevância, neste trabalho apenas os fatores tecnológicos serão considerados. Neste sentido, a proposta introduzida nesta dissertação visa oferecer condições a nível de serviços de *software* que dêem maior rapidez e confiança ao processo de tornar uma EV apta a operar a partir do momento em que seus parceiros tornam-se conhecidos.

Na sua *criação*, ou seja, antes das empresas parceiras da EV passarem a produzir (na fase de *operação*), uma EV deve passar por um extenso e complexo processo de configuração, onde são definidos e negociados vários aspectos essenciais para a sua operação, a saber:

1. Busca e seleção dos parceiros que farão parte da EV;
2. Definição dos contratos, que definem os direitos e deveres de cada membro;
3. Configuração do formato para intercâmbio de dados;

4. Configuração dos dados intercambiáveis entre os parceiros e seus direitos de acesso;
5. Integração com os sistemas legados, como suporte básico ao intercâmbio de dados entre os parceiros.

Uma série de trabalhos têm sido desenvolvidos, com variados graus de profundidade, sobre esses vários aspectos. Por exemplo, no aspecto da busca e seleção, podem ser relacionados os trabalhos de RABELO et al. (2000), ROCHA et al. (1999) e ÁVILA et al. (2002). O aspecto de contratos é de cunho predominantemente legal e foge ao escopo desta dissertação. Tem a ver com a questão da elaboração, assinatura e cumprimento de contratos digitais/eletrônicos, uma área ainda repleta de pontos em aberto apesar de importantes iniciativas sendo realizadas, tais como SCHERER (1997), eLEGAL (ELEGAL, 2003) e a Câmara Brasileira de Comércio Eletrônico (CAMARA-E; 2003). O terceiro aspecto tem a ver com a definição do formato propriamente dito para troca de dados entre empresas heterogêneas e autônomas, membros de uma EV, sobre a infraestrutura de comunicações de suporte que as permite interagir entre si. Uma série de iniciativas internacionais foram lançadas, tais como STEP (STEP, 2003) e EDIFACT (UN/EDIFACT, 2003), mas só atenderam determinados tipos de processos de negócios, portanto não contemplando a contento todos os processos executados numa EV (transacionados entre seus membros). Além disso, a interoperação entre sistemas não era garantida plenamente uma vez que estipulavam formatos de dados não processados por outras aplicações. Com o desenvolvimento do padrão XML (*Extensible Markup Language*), definido pelo W3C[1], uma verdadeira revolução ocorreu, fazendo com que XML seja hoje em dia considerado o padrão *de facto* para intercâmbio de dados entre aplicações distribuídas e heterogêneas, que é a característica encontrada no domínio das EVs. Em suma, pode-se dizer que o primeiro aspecto compreende uma área onde já existe uma certa maturidade e bons resultados, o segundo aspecto está fora dos objetivos desta dissertação, e o terceiro aspecto pode-se considerar como "resolvido", visto a realidade empresarial apontar claramente para a adoção da XML.

Os dois últimos aspectos, 4 e 5, podem ser considerados os menos desenvolvidos, especialmente se se considerar propostas que visem a introduzir algum grau de

[1] *World Wide Web Consortium*, entidade criada com o objetivo de definir especificações, diretrizes e ferramentas para protocolos Web (www.w3.org).

automatismo para elas, tendo-se em vista a agilidade necessária de ser introduzida na criação de EVs, principalmente as mais dinâmicas.

Assim sendo, o foco desta dissertação será nos dois últimos itens, ou seja, na *configuração dos dados intercambiáveis* e na *integração com os sistemas legados*. Os dados intercambiáveis dizem respeito a informações do processo de negócio, tais como ordens de compra, *status* da produção, etc. A integração com os sistemas legados corresponde à ligação do modelo genérico de dados a ser utilizado pela EV com os modelos de dados utilizados nos sistemas legados de cada membro.

1.2 Objetivo

Em face do cenário apresentado, esta dissertação tem como objetivo a proposição de um modelo para configuração e integração de dados em empresas virtuais. O modelo usará XML como formato para representação dos dados visando principalmente agilizar o processo de integração das empresas-membro e, conseqüentemente, reduzir o tempo e o custo envolvidos.

O objetivo proposto será concretizado através de desenvolvimentos no âmbito de duas etapas gerais do processo de configuração e integração:

- Concepção da infraestrutura computacional da EV, via um protótipo computacional para tratar a *configuração dos dados intercambiados* entre os membros da EV;
- Implantação da plataforma da EV, via um segundo protótipo computacional, para tratar a problemática da integração do *modelo de dados* da EV com os dos *sistemas legados* de cada membro. Este modelo segue a filosofia de *Enterprise Application Integration* (EAI).

Este modelo de configuração e integração foi desenvolvido no âmbito do projeto europeu *MyFashion.eu* (IST-2001-32560) (MYFASHION, 2003).

Considerando a relativa incipiência da área de EVs e a quase inexistência de análises e estudos comparativos quantitativos em termos de redução de tempo provocada pela introdução de automatismos em algumas fases da EV, a validação desta dissertação frente ao objetivo-macro da agilidade será medida pelo desenvolvimento desses dois protótipos. Por outras palavras, assume-se que automatizando essas duas etapas, atualmente feitas quase que totalmente de forma manual e sujeitas a muitos erros, um aumento significativo

da agilidade será alcançado. Ainda, que tais protótipos, incorporados dentro de um módulo mais amplo de gestão de EVs (dentro da plataforma MyFashion) e instalado/validado pelas empresas-membro do projeto MyFashion, comprovam a sua operacionalidade e utilidade.

1.3 Organização da Dissertação

O Capítulo 2 descreve o paradigma de Empresas Virtuais, apresentando os seus conceitos, aspectos de configuração, tecnologias utilizadas e alguns trabalhos relacionados.

O Capítulo 3 apresenta algumas das principais tecnologias e padrões para intercâmbio de informações, com especial destaque para a XML.

O Capítulo 4 apresenta a proposta para configuração e integração de dados intercambiáveis em EVs, bem como o seu enquadramento dentro do ciclo de vida e da plataforma de EV.

O Capítulo 5 apresenta o modelo conceitual para a configuração de dados intercambiáveis em EVs.

O Capítulo 6 apresenta o modelo conceitual para a integração de dados em EVs.

O Capítulo 7 discute os aspectos de implementação dos modelos conceituais a partir do desenvolvimento de protótipos.

O Capítulo 8 avalia os protótipos implementados, apresentando as vantagens, limitações e considerações no que diz respeito aos testes realizados.

Finalmente, o Capítulo 9 apresenta as conclusões sobre os resultados obtidos com o presente trabalho, assim como as contribuições da abordagem utilizada e algumas propostas para trabalhos futuros.

2 Empresas Virtuais

2.1 Introdução

Este capítulo tem o objetivo de apresentar o conceito de Empresa Virtual (EV), de forma a contextualizar a proposta deste trabalho. Embora seja uma área de crescente interesse em pesquisa e desenvolvimento, o paradigma de EV ainda é carente de uma padronização na definição dos conceitos e de um acordo na terminologia utilizada (CAMARINHA-MATOS *et al.*, 1999a).

Entretanto, algumas características em comum podem ser encontradas em várias definições. Para PARK et al. (1999) "Empresas Virtuais são criadas para aproveitar uma oportunidade específica de mercado, sendo constituídas de duas ou mais empresas diferentes, e são projetadas para facilitar o agrupamento de recursos de forma rápida e concorrente, sem considerar o tamanho da organização, a localização geográfica, ambiente computacional ou as tecnologias envolvidas". Para WALTON et al. (1996) "uma empresa virtual consiste de uma série de "nós" de competências essenciais (*core competencies*) cooperantes que formam uma cadeia produtiva para responder a uma oportunidade específica de mercado". Já para CAMARINHA-MATOS et al. (1999a) "uma empresa virtual é uma aliança temporária de empresas que se agrupam para compartilhar habilidades ou competências essenciais e recursos para melhor responder às oportunidades de negócios, e cuja cooperação é suportada por redes de computadores".

Neste trabalho utiliza-se a definição de RABELO et al. (2004), que define EV como sendo "uma agregação lógica e temporária de empresas autônomas, cooperantes e heterogêneas, que é estratégica e dinamicamente formada para atender a uma certa oportunidade de negócios, e cujo atendimento é operacionalizado através de um compartilhamento coordenado de habilidades, recursos e informações, integralmente via rede de computadores".

Há duas palavras-chave em comum encontradas em praticamente todas as definições de EV: *interconexão* (em rede) e *cooperação*. Assim, há a tendência de descrever uma EV como uma rede de empresas cooperantes. Isso sugere que o processo de produção não é executado por uma única empresa, mas cada empresa executa uma etapa na "cadeia produtiva" formada. Portanto, o conceito de EV é que um conjunto de empresas ou

organizações com objetivos em comum se agrupam, formando uma rede interoperável que atua como uma única organização. O estabelecimento de acordos de cooperação entre empresas não é um fato novo, mas o uso de tecnologias de informação e comunicação para suportar uma cooperação ágil entre os parceiros é a principal característica diferencial associada ao conceito de EV.

2.1.1 Conceitos Relacionados

A crescimento da variedade de organizações interconectadas e o surgimento de novos paradigmas de gerenciamento da produção têm levado à criação de novos termos relacionados a empresas virtuais, tais como *organização virtual* e *cluster de empresas*. Alguns autores usam alguns destes termos indistintamente a empresas virtuais embora existam diferenças no seu significado. A Figura 1 ilustra a relação entre estes termos, que serão detalhados a seguir (CAMARINHA-MATOS *et al.*, 1999a).

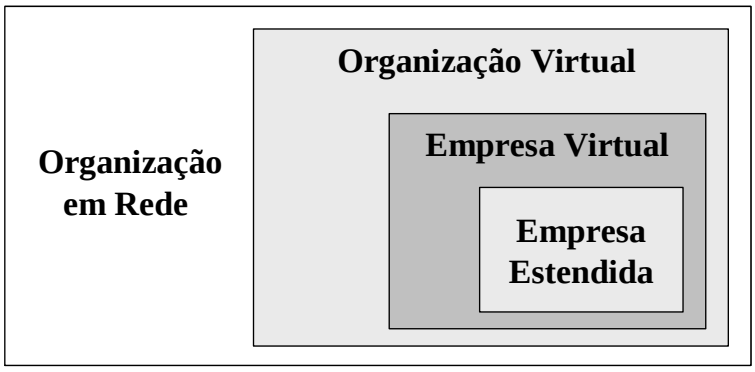

Figura 1: Tipos de Organizações

Empresa Estendida: este termo é o mais próximo do conceito de EV. Este conceito é aplicado a uma organização em que uma empresa dominante “estende” suas fronteiras para todos ou alguns de seus fornecedores, enquanto que uma EV pode ser vista como um conceito mais geral e com uma estrutura mais “democrática”. Neste sentido, uma empresa estendida pode ser vista como um caso particular de empresa virtual.

Organização Virtual: este é um conceito similar ao de EV, consistindo de uma rede de organizações que compartilham recursos e habilidades para alcançar um objetivo, mas que não está limitada a uma aliança de empresas. Um exemplo de organização virtual pode ser a interconexão de várias organizações de um município (prefeitura, serviços de

distribuição de água, serviços cadastrais, etc). Uma EV é, portanto, um caso particular de organização virtual.

Organização em Rede: este é o termo mais geral no que se refere a qualquer grupo de organizações interconectadas por uma rede de computadores, mas sem necessariamente compartilhar habilidades ou recursos, ou possuir um objetivo em comum.

Cadeia Produtiva ou de Suprimentos ou de Fornecedores: agrupamento usualmente "seqüencial" de empresas que ao longo do processo agregam valor ao que se está produzindo, e por esta razão também pode ser chamada de *cadeia de valor*. Normalmente cobrem aspectos desde os provedores de matéria-prima até os consumidores, considerando detalhes de transporte, vendas, distribuição, etc. Este conceito é normalmente aplicado a organizações que são relativamente estáveis, ou seja, os parceiros principais permanecem os mesmos por um longo período de tempo.

***Cluster* de Empresas:** grupo de empresas que possuem alguma ligação estratégica cooperativa – seja por questões geográficas ou por questões tecnológicas/de competências essenciais – para trabalharem juntas em uma EV.

2.2 Classificação de Empresas Virtuais

Embora um grande número de organizações interconectadas seja classificado como sendo EV, é necessário que se faça uma classificação de acordo com suas características e requisitos, para que possam então ser modeladas e implantadas.

Em CAMARINHA-MATOS et al. (1999a) é feita uma classificação de EVs baseada em um conjunto de características, conforme destacado a seguir.

Duração: quanto à duração, uma EV pode ser uma *única oportunidade de negócio*, onde a aliança de empresas é dissolvida no final no processo. Este é talvez o tipo mais comum de EV, tendo como exemplo sistemas de engenharia em grande escala, como um consórcio envolvido na construção de uma ferrovia. Há também *alianças em longo prazo*, onde há um número indefinido de processos de negócio ou um intervalo de tempo muito longo especificado. Neste caso, pode-se citar a indústria de alimentos e automotiva.

Topologia: em relação à topologia da rede, ela pode ser de *natureza variável/dinâmica*, em que algumas empresas (não fundamentais) podem se unir ou deixar a aliança dinamicamente de acordo com as fases do processo de negócio ou outros fatores de mercado. Neste caso faz-se necessário o auxílio de funcionalidades específicas para a

busca e seleção de parceiros (produtores e provedores de serviços) e de procedimentos para o acoplamento/retirada do parceiro. Por outro lado, empresas virtuais podem possuir uma topologia com *estrutura fixa* (com pequenas variações em termos de produtores e clientes).

Em relação a esta característica, OUZOUNIS (2001) classifica empresas virtuais em **Estáticas** ou **Dinâmicas**. Em uma **EV Estática** os parceiros estão ligados entre si de forma estática e fixa, ou seja, os processo de negócios estão firmemente integrados. As relações entre os parceiros (interfaces) são predefinidas, fortemente acopladas, fixas e customizadas. Além disso, a estrutura da EV é estática e pré-determinada. Já em uma **EV Dinâmica** os parceiros são vinculados dinamicamente, sob demanda, de acordo com os requisitos dos consumidores. Os parceiros não possuem relacionamentos de negócios fixos, e assim a EV pode mudar continuamente baseada em critérios dirigidos pelo mercado.

Participação: quanto à participação, uma empresa pode se dedicar a uma *única EV*, ou pode participar simultaneamente de *múltiplas EVs*. No caso de EVs múltiplas, a infraestrutura de suporte deve ser capaz de lidar com vários "espaços" virtuais de participação, e de controlar a visibilidade das informações de acordo com os requisitos individuais de cada empresa.

Coordenação: no que diz respeito à coordenação, existem várias abordagens diferentes. A primeira, chamada *estrutura de coordenação em estrela*, uma empresa dominante centraliza a coordenação, impondo seus próprios padrões, como os modelos de processos de negócio, os mecanismos de troca de informação e direitos de acesso, entre outros. O conceito de empresa estendida se enquadra bem nesta abordagem. Quando os parceiros cooperam de forma igualitária, preservando sua autonomia, mas compartilhando habilidades, temos uma *EV democrática*. Mesmo não havendo uma empresa dominante, faz-se necessário um nó coordenador para administrar e monitorar informações gerais da organização. No outro extremo temos uma *federação*, que consiste de uma EV onde as empresas percebem que há benefícios mútuos no gerenciamento comum dos recursos e habilidades, criando um tipo de estrutura de coordenação comum (federação).

Escopo de Visibilidade: o aspecto da visibilidade está intimamente relacionado com a topologia e a coordenação da rede, significando quais nós (empresas) da rede, um determinado nó pode "ver". Na maioria dos casos, um nó vê apenas os seus vizinhos diretos, tendo então uma *visibilidade de único nível*. Por outro lado, em situações onde há uma coordenação mais avançada, um nó pode ter um grau de visibilidade maior (*visibilidade de múltiplos níveis*). Isto é necessário em tarefas como: planejamento,

escalonamento, previsão de demanda, distribuição de carga de trabalho, gerenciamento otimizado de recursos, entre outros.

2.3 Ciclo de Vida

Um aspecto importante a ser considerado quando se analisa a estrutura de uma EV é o seu ciclo de vida. Para facilitar o entendimento e caracterizar os requisitos e desafios para uma EV, será apresentado um cenário de exemplo (CAMARINHA-MATOS *et al.*, 1999a). Suponha uma grande empresa, como uma construtora, que resolve participar de uma licitação para a construção de uma ponte. Devido a diversos fatores (complexidade, custos, etc.) esta construtora resolve então envolver outras grandes e pequenas empresas no negócio. Neste caso, ela resolve criar uma EV, atuando como o seu *coordenador* durante o seu ciclo de vida. Os seguintes passos podem ocorrer:

1. A construtora entra em contato com algumas grandes empresas, que por sua vez contatam algumas pequenas empresas;
2. A construtora transfere a descrição do trabalho a todas as empresas interessadas;
3. Depois dos acordos iniciais, todas as empresas negociam para a criação da EV;
4. O coordenador (construtora) define a oferta que, caso seja selecionada, a EV começa a sua operação. Caso contrário, a EV se dissolverá;
5. A EV operará corretamente se e somente se suas empresas-membro cooperarem entre si e apoiarem a coordenação da EV:
    - A cooperação entre os membros da EV envolve, na maioria das vezes, o intercâmbio de informações (como modelos de produtos) e a resposta pontual a pedidos de outros parceiros (por exemplo, pedido de *status* de ordem);
    - Cada empresa deve também cooperar com o coordenador da EV no que diz respeito à monitoração do *status* do trabalho, e outras tarefas de cooperação avançadas;
6. No caso de um problema com uma empresa, seja porque não está cumprindo o seu trabalho no tempo certo ou simplesmente se esta decide deixar a EV, o coordenador da EV procurará por outra empresa e a substituirá;
7. Uma vez que o trabalho está completamente feito, a EV se dissolve.

Observando o exemplo é possível perceber algumas etapas bem definidas no ciclo de vida de uma EV. A Figura 2 abaixo ilustra as etapas do ciclo de vida de uma EV: criação, operação, evolução e dissolução (SPINOSA *et al.*, 1998) (CAMARINHA-MATOS *et al.* 1999a).

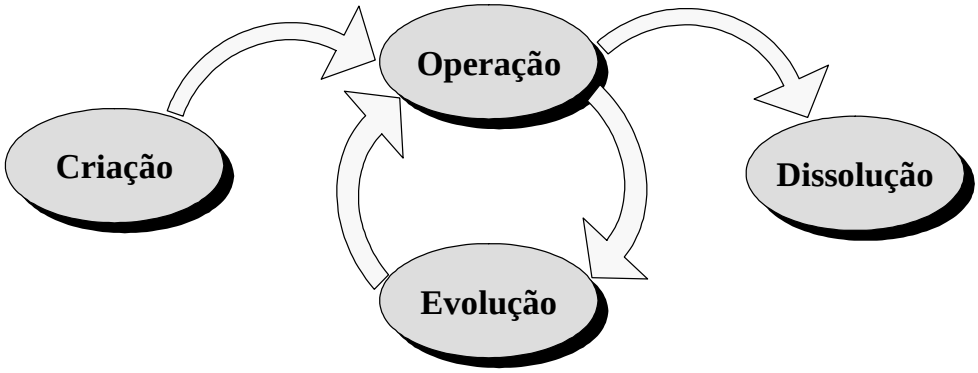

Figura 2: Ciclo de vida de uma EV.

2.3.1 Criação

Esta é a fase inicial, onde a EV é criada e configurada. A seguir são apresentadas as etapas principais no processo de configuração de uma empresa para que esta possa participar de uma EV (CAMARINHA-MATOS *et al.*, 1999e):

Configuração dos Recursos de Infraestrutura: nesta etapa são identificados e instalados os recursos computacionais para a empresa, ou seja, a instalação da plataforma computacional da EV e sua configuração/integração com os sistemas legados, tais como o sistema operacional, sistema de gerenciamento de bases de dados, sistemas de gerenciamento empresarial (ERP), conexão com a Internet, etc.

Manifestação: uma vez tendo instalado a plataforma da EV, a empresa está pronta para se envolver em futuras EVs. Nesta etapa a empresa pode se cadastrar em um catálogo público que armazena informações gerais, tais como áreas/serviços de interesse e o seu perfil geral.

Seleção de Parceiros e Formação da EV: quando uma oportunidade de negócios é identificada, os parceiros adequados são procurados, identificados e selecionados. Alguns acordos preliminares são feitos, e então a topologia da EV e os papéis a serem desempenhados pelos parceiros são especificados.

Negociação de Contrato da EV: uma vez que a topologia da EV e os papéis dos seus parceiros foram definidos, contratos devem ser assinados entre o coordenador da EV e cada um dos parceiros, onde são definidos os direitos e responsabilidades de cada um em relação à EV. Mais especificamente, são definidos os "planos de produção" para as empresas e as "Cláusulas de Supervisão", que definem os direitos de acesso à informação por parte do coordenador da EV (seção 2.4) em termos de observar e monitorar o progresso das atividades que ocorrem nos membros da EV.

Configuração da Empresa: nesta etapa, a plataforma da EV recebe do coordenador as configurações definidas nas etapas anteriores, ou seja, a definição da EV e as cláusulas de supervisão. Baseada nestas definições, a empresa ajustará o acesso preciso à informação e os direitos de visibilidade de cada membro da EV.

Para KANET et al. (1999), a etapa de criação está implícita em três fases (identificação, formação e *design*), sendo que as atividades realizadas são similares às apresentadas acima. Outros incluem também a fase de "mobilização", que é a pré-disposição para buscar alianças e ter uma plataforma de suporte computacional, além do planejamento da EV.

2.3.2 Operação

Nesta fase os parceiros da EV realizam os seus processos de negócio visando alcançar seus objetivos comuns. Para esta etapa várias funcionalidades costumam ser necessárias, tais como (CAMARINHA-MATOS *et al.* 1999b):

- **Intercâmbio de informações:** inclui diversas funcionalidades, como:
  - *Mecanismos para intercâmbio de informações:* informações de negócios, informações técnicas de produtos, informações gerais (como estatísticas e catálogos de produtos/serviços), etc;
  - *Interoperabilidade entre padrões:* pois um membro deve estar apto para interpretar e gerar mensagens de diferentes padrões;
  - *Monitoramento do status de ordens:* utilizado não apenas para a coordenação da EV, mas também para dar suporte à cooperação entre os membros.
- **Coordenação avançada:** necessária para que o coordenador da EV possa supervisionar o *status* geral de execução de toda EV (cobrindo aspectos de

produção, vendas e distribuição/entrega) e tomar as devidas providências no caso de algum problema em relação ao planejamento descrito nos contratos.

- **Funcionalidades relacionadas a materiais/serviços:** necessárias para representar e monitorar o fluxo de produtos e serviços através da rede da EV. São elas:
  - *Gerenciamento do fluxo de materiais/serviços:* identificação, representação e monitoramento dos fluxos de materiais/serviços da EV;
  - *Logística:* planejamento do transporte, inventário e armazenamento.
  - *Previsão:* feita a partir de informações históricas transmitidas pelos pontos de venda.

2.3.3 Evolução

Evoluções acontecem quando há a necessidade de se adicionar ou substituir um parceiro, ou mesmo modificar seus direitos de acesso. Isto pode ocorrer devido a algum evento não previsto, como uma incapacidade (temporária) de um parceiro, ou a necessidade de se aumentar a carga de trabalho. Para esta etapa, são necessárias funcionalidades similares às especificadas para a fase de criação (seção 2.3.1) para que a EV possa ser reconfigurada.

2.3.4 Dissolução

Esta fase ocorre quando a EV termina seus processos de negócio e se dissolve (“separação dos parceiros”). A dissolução pode ocorrer devido a duas situações: quando os objetivos são alcançados com sucesso, ou por decisão dos parceiros para interromper a operação da EV (cancelamento da EV). Dentre as principais tarefas a serem realizadas nesta fase, temos:

- Definição das responsabilidades de cada parceiro depois da dissolução da EV (qualidade/manutenção do produto).
- Redefinição/interrupção dos direitos de acesso à informação depois do fim do processo de cooperação.
- Avaliação do desempenho dos parceiros, gerando informações históricas que poderão ser usadas por ferramentas de seleção de parceiros em futuras EVs.

É preciso deixar claro que a EV não se dissolve quando o produto é entregue ao cliente. Conforme a legislação vigente e o tipo de produto, a EV deverá se manter formalmente conectada para prover serviços de pós-venda (garantia, devolução, reciclagem, etc.).

2.4 Papéis dos Participantes de uma EV

As empresas desempenham diferentes papéis em uma EV durante o seu ciclo de vida. Dentre os diversos papéis que uma empresa pode assumir, destacam-se abaixo os principais (CAMARINHA-MATOS *et al.*, 1999a).

Coordenador da EV: é o componente regulador das atividades relacionadas à EV. O coordenador pode ser um nó especializado em coordenação (organização externa à EV), ou este papel pode ser desempenhado por um membro da EV. Entre outras tarefas, o coordenador da EV é o responsável por:

- Registrar novas empresas na rede;
- Se necessário, prover assistência às novas empresas para instalar e configurar a infraestrutura de suporte;
- Manter as informações sobre os membros no *diretório* da EV;
- Reconfigurar a EV, se necessário, e distribuir as notícias sobre a evolução da rede;
- Pode servir como uma “testemunha” para as empresas que necessitam do apoio de terceiros em suas negociações com outras empresas;
- Supervisionar e coordenar as diferentes atividades das empresas visando os objetivos da EV;
- Supervisionar e auxiliar as empresas durante a dissolução da EV;
- Coletar/averiguar as métricas estabelecidas.

Empresa Membro: são as empresas que participam da EV. As principais funcionalidades realizadas por uma empresa membro incluem:

- Estabelecer contato e interação segura com os outros nós;
- Compartilhar suas atividades dentro do processo de negócio global da EV de acordo com as suas responsabilidades (definidas em seus contratos);

- Gerenciar os seus direitos de visibilidade de informação, de maneira a proteger tanto seus próprios interesses, quanto os interesses da EV;
- Compartilhar e intercambiar informações (e materiais) necessárias para a cooperação com outros nós membros.

***Broker*:** este papel é desempenhado pela empresa (não necessariamente o coordenador da EV) que cria/inicia a EV e procura por parceiros.

Outros Papéis: além destes papéis, uma empresa pode desempenhar outros papéis, normalmente definidos dependendo do ramo de negócios da EV. Por exemplo: gerente de projeto, auditor, provedor de dados, técnico da rede, etc.

2.5 Projetos Relacionados

O paradigma de EVs tem sido abordado por dezenas de projetos de pesquisa e desenvolvimento nos últimos anos (CORDIS, 2004). Como o modelo proposto nesta dissertação está enquadrado neste paradigma, esta seção tem o objetivo de apresentar alguns trabalhos correlatos nesta área.

Os projetos iniciais na área de EV tinham como foco principal a etapa de *operação* (na qual a EV está realizando o seu processo de negócio), pois a idéia básica era desenvolver modelos e ferramentas para demonstrar a relevância deste paradigma às comunidades científica e industrial. Com o passar do tempo surgiu a necessidade do desenvolvimento de modelos/tecnologias para as demais etapas do ciclo de vida.

Como este trabalho está concentrado na etapa de *criação*, e mais especificamente na configuração e integração dos dados, os projetos selecionados estão relacionados a estes aspectos.

Muitos projetos foram localizados nos chamados "*clusters* de projetos", que indexam uma grande variedade de projetos. Alguns dos *clusters* pesquisados foram: COVE (COVE, 2003), EXPIDE (EXPIDE, 2003) e VOSTER (VOSTER, 2003). Apesar da grande variedade de projetos, não foi encontrado nenhum que trate dos aspectos de configuração dos dados intercambiáveis e integração dos dados dos sistemas legados.

Muitos deles, tais como prodAEC (PRODAEC, 2003), PRODNET II (PRODNET, 2003) e DAMASCOS (DAMASCOS, 2003) utilizam padrões para intercâmbio de dados,

como EDIFACT (UN/EDIFACT, 2003), STEP (STEP, 2003) e até mesmo linguagens baseadas em XML, mas normalmente a configuração destes padrões é feita de forma manual, o que implica em um tempo considerável para o desenvolvimento da infraestrutura de comunicação. Quanto à integração com os sistemas legados, destaca-se o projeto PRODNET II.

Na verdade, apesar de o PRODNET II ser um projeto considerado "referência" na área de EVs, não se encontrou nenhum outro projeto que tratasse das duas problemáticas abraçadas nesta dissertação, que são associadas à automatização das etapas de integração dos sistemas legados e de configuração das informações a serem trocadas entre os membros da EV. De fato, cada projeto desenvolvera suas próprias plataformas para EVs a fim de explorar certos tipos de serviços entre empresas, não depositando esforços na concepção de metodologias ou automatismos para certas etapas (CAMARINHA-MATOS, 2004), que é o que propõe esta dissertação.

2.5.1 PRODNET II

PRODNET II (*Production Planning and Management in an Extended Enterprise*) (PRODNET, 2003) foi um projeto financiado pela União Européia que teve como objetivo a modelagem e desenvolvimento de uma plataforma aberta para suporte a empresas virtuais industriais focando principalmente nas necessidades de pequenas e médias empresas. A arquitetura foi implementada com padrões emergentes e tecnologias em comunicação, gerenciamento cooperativo de informações, e tomada de decisões distribuída.

A plataforma básica inclui, entre outras funcionalidades, uma camada de cooperação responsável pela interoperação com os sistemas legados existentes em cada empresa. Esta camada, chamada *Camada de Cooperação PRODNET* (*PRODNET Cooperation Layer* – PCL) fica "acima" dos sistemas legados e aplicações existentes em cada empresa. Assim, as empresas podem interoperar através de suas PCLs. Dentro da infraestrutura PRODNET, cada empresa é representada por dois componentes principais: O módulo Interno (sistema legado da empresa) e a camada de Cooperação, como ilustrado na Figura 3 (CAMARINHA-MATOS *et al.*, 1999d).

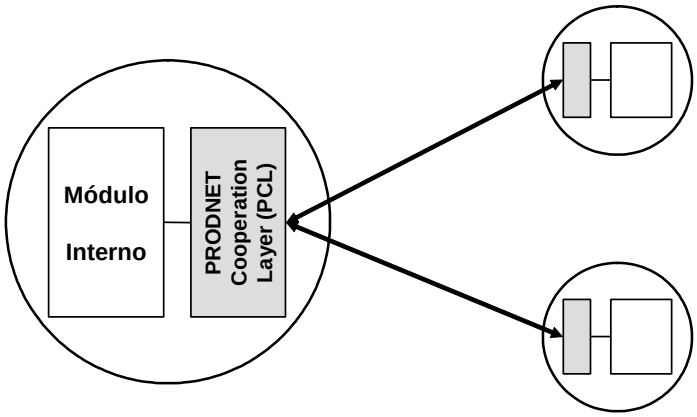

Figura 3: Interoperação usando a infraestrutura PRODNET.

A PCL contém todas as funcionalidades para a interconexão entre a empresa e a EV. Ela atua na cooperação, comunicação (utiliza dois padrões largamente utilizados: STEP e EDIFACT) e coordenação da empresa, cumprindo o papel de interlocutor desta dentro da EV. Entretanto, apesar de adicionar o caráter de cooperação a uma empresa, a PCL visa manter também a completa privacidade, independência e autonomia da empresa.

Embora a PCL facilite o processo de integração com os sistemas legados, este processo é feito manualmente, onde extensões e reengenharias aos sistemas legados são necessárias para suportar as interações com a PCL (CAMARINHA-MATOS *et al.*, 1999e).

Apesar destes aspectos terem sido pouco abordados nos projetos de pesquisa na área de EVs, outras iniciativas internacionais têm abordado a questão da interoperação/integração entre aplicações visando o desenvolvimento de tecnologias no contexto de negócios eletrônicos, conforme será apresentado no próximo capítulo.

2.6 Tecnologias de suporte para EVs

O conceito de EV está vinculado ao uso da tecnologia da informação, fazendo referência de maneira geral às tecnologias de comunicação para suportar a interoperabilidade entre os seus membros. Nesta seção, serão apresentadas as principais tecnologias envolvidas, tal como apresentado por CAMARINHA-MATOS et al. (1999b) e FILOS *et.al.* (2000):

- Protocolos usados na Internet (TCP/IP, HTTP, FTP, etc.), visto que esta se tornou um padrão absoluto como meio de comunicação para negócios em nível mundial.

Recentemente, o WAP (*Wireless Application Protocol*), utilizado por dispositivos móveis, estendeu o alcance da Internet para novos meios;

- No contexto da Internet, há também os *serviços Web*, que permite a interoperação entre aplicações no ambiente Web. Neste contexto, podemos citar o SOAP e o WSDL;
- Plataformas para objetos distribuídos e serviços de *middleware* têm se tornado mais estáveis, oferecendo melhor desempenho e escalabilidade. Nesta categoria, podemos citar o CORBA, Java RMI e DCOM.
- Na área de sistemas de *workflow*, destacam-se os conceitos, arquitetura, e interfaces propostos pelo *Workflow Management Coalition*;
- A XML (*Extensible Markup Language*) tem desempenhado um papel fundamental no desenvolvimento de protocolos para aplicações de e-business, conforme será visto no próximo capítulo;
- Os *sistemas multiagente*, especialmente a utilização de agentes móveis, têm modificado significativamente a forma como os sistemas distribuídos funcionam. Neste contexto, XML também tem desempenhado um importante papel na definição de *ontologias*[2] para os agentes;
- Padrões para troca de informações de negócios, como o EDIFACT, e para intercâmbio de dados técnicos de produtos (STEP) também são de grande importância no contexto de EVs. Nesta área, a XML tem auxiliado nos esforços para a definição de uma nova geração destes padrões.

No próximo capítulo, as tecnologias para intercâmbio de informações serão abordadas em maior detalhe.

[2] *Ontologia* é uma maneira de se formular um esquema conceitual rigoroso dentro de um dado domínio de conhecimento, sendo tipicamente uma estrutura de dados hierárquica contendo todas as entidades relevantes, seus relacionamentos e regras dentro do domínio em questão (WIKIPEDIA, 2004b). Usada para que entidades possam se comunicar utilizando os mesmos termos de uma certa área de conhecimento visando minimizar erros de interpretação dos mesmos.

3 Tecnologias para Intercâmbio de Informações

3.1 Introdução

Como o foco deste trabalho é a configuração das informações intercambiáveis em uma EV e sua integração com os sistemas legados, um dos objetivos deste capítulo é apresentar algumas das principais tecnologias utilizadas para este propósito, lembrando que muitas destas são também utilizadas em outros domínios. Entretanto, a meta principal do capítulo é mostrar que várias das tecnologias a serem apresentadas estão evoluindo para o uso da XML como formato para a representação das informações.

É preciso salientar que serão apresentadas apenas as tecnologias envolvidas na interoperação em um nível mais alto (na modelagem e representação de informações). Por não estarem diretamente relacionadas a este trabalho, as tecnologias envolvidas em um nível mais baixo, ou seja, mecanismos de transporte de dados (como HTTP, CORBA, Java RMI, etc.) ou para coordenação (tais como sistemas de gerenciamento de *workflow*, sistemas multiagente, etc.) não serão apresentadas.

3.2 EDI

EDI (*Electronic Data Interchange*), ou Intercâmbio Eletrônico de Dados pode ser definido como “a transferência eletrônica de dados estruturados segundo uma norma pública, entre aplicações de diferentes organizações” (APEDI, 1998).

Uma outra definição para EDI pode ser: “intercâmbio eletrônico de informações entre entidades usando formatos de dados estruturados, padronizados e processáveis por máquinas”. *Intercâmbio eletrônico* envolve comunicação através de redes de telecomunicações; *formato padronizado* corresponde a um conjunto de regras, definidas em comum acordo, para a organização e transmissão dos dados; e *processável por máquinas* significa que um computador é capaz de ler e processar os dados transmitidos sem a intervenção humana (EDI, 1999).

Mensagens EDI são rapidamente trocadas entre empresas, reduzindo os tempos e os custos envolvidos nas atividades de cada uma, aumentando assim a qualidade na execução de suas tarefas, e conseqüentemente aumentando a sua competitividade no mercado.

As mensagens EDI dizem respeito a documentos específicos, cobrindo diversos ramos de atividade, tais como: Comércio, Transportes, Bancos, Construção, Estatística, Saúde, Segurança Social, Administração Pública, Mercados Públicos, entre outros.

O conceito de EDI surgiu no final dos anos 60, quando padrões de comunicação entre computadores tornaram-se disponíveis. Com o tempo, alguns setores da economia sentiram uma grande necessidade por ferramentas eficientes para comércio eletrônico, resultando no desenvolvimento de um grande número de padrões. Contudo, estes eram especializados em uma região ou país, ou em setores industriais específicos. Dentre estes padrões, considera-se os mais bem sucedidos o ANSI X12 (norte-americano), GTDI e ODETTE (europeus).

Devido à crescente globalização e ao aumento dos negócios entre empresas de diferentes países, viu-se a necessidade do desenvolvimento de um padrão realmente mundial, visando alcançar uma possível convergência entre os padrões existentes. Assim, em 1986 a ONU então cria o UN/EDIFACT[3].

3.2.1 Benefícios

A prática do EDI traz vários benefícios, sendo estes um dos primeiros temas de análise das organizações que o pretendem implementar. Entretanto, alguns destes benefícios não são quantificáveis, podendo ser tanto ou mais importante do que a princípio se pode estimar. Assim, conforme dito em APEDI (1998), os benefícios podem ser classificados em duas áreas distintas: os Operacionais e os Estratégicos.

3.2.1.1 Operacionais

Diminuição de Erros: os mecanismos de comunicação atuais asseguram um certo nível de segurança e a integridade dos dados transmitidos. Além disso, os erros humanos são minimizados, pois não há a necessidade de re-introdução dos dados no sistema.

Redução dos Custos Administrativos: ao automatizar o processo de troca de informações, a empresa libera recursos humanos e técnicos para a sua atividade essencial.

Diminuição de Papel: a quantidade de papel utilizada é fortemente reduzida. Conseqüentemente, os custos de impressão serão reduzidos e a gestão de informação é melhorada. Os custo com correio ou fax são também significativamente reduzidos.

[3] *United Nations/Eletronic Data Interchange For Administration Commerce and Transport*

3.2.1.2 Estratégicos

Diminuição de Tempos: os dados transmitidos pelos meios de telecomunicações e tratados automaticamente pelas aplicações são processados rapidamente, obtendo-se uma redução do tempo de resposta nas transações comerciais.

Melhor Serviço aos Clientes e Relacionamento com Fornecedores: através de entregas a tempo, notificação prévia de envio, redução ao mínimo nos tempos de transporte tal como nos pagamentos de bens e serviços.

Maior Competitividade: este pode ser uma simples conseqüência dos restantes através da redução dos custos, aumento da qualidade do serviço, etc.

Além disso, há casos onde as considerações sobre qualidade não são avaliadas do ponto de vista econômico, mas apenas em relação à imagem que a companhia deseja transmitir (GIBON *et al.*, 1999).

3.2.2 Modelo Básico de Funcionamento do EDI

EDI é normalmente implantada nos pontos de comunicação entre duas empresas. Cada empresa utiliza um servidor que atua como o ponto de comunicações para o EDI. Toda comunicação é enviada por um provedor de serviços especial, que permite o acesso à VAN[4]. Um provedor de VAN garante que toda transação EDI seja direcionada para o destinatário correto, além de prover segurança e outros serviços de suporte (normalmente com a cobrança de uma taxa).

O EDI envolve uma seqüência de atividades, conforme ilustrado na Figura 4:

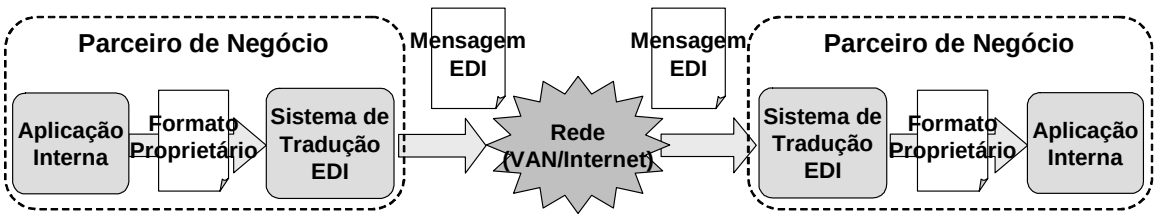

Figura 4: Modelo de funcionamento do EDI

Em um exemplo típico, o sistema interno de um comprador gera uma mensagem e a envia para o sistema de tradução (mapeamento) EDI. Uma transformação é feita, sendo

[4] *Value Added Network* ou Rede de Valor Agregado. É uma rede privada que provê serviços especializados, tais como serviços para EDI ou acesso a uma base de dados particular.

formada uma mensagem no formato EDI. A mensagem é então passada para o operador da VAN, que a direciona de forma segura até o sistema de tradução EDI do vendedor. O sistema EDI do vendedor então traduz a mensagem e salva os trechos relevantes no seu sistema de gerenciamento de ordens. Dependendo do tipo de transação realizada, o parceiro receptor poderá enviar uma resposta para o parceiro emissor seguindo os mesmos passos já apresentados.

3.2.3 Considerações e Perspectivas

EDI é cara de implementar devido a uma série de razões. Por exemplo, a dificuldade de se escrever um programa que interprete as suas mensagens. A Figura 5 abaixo ilustra um exemplo de mensagem EDI, contendo uma Ordem de Compra (*Purchase Order* – ANSI X12 4040 850) (XMLGLOBAL, 2003).

```
ST*850*0001|
BEG*00*SA*A123456**19980507|
DTM*002*19980514|
N1*ST**9*1122334450000|
N2*ABC Co.*Branch 1|
N3*1234 Maple Grove St.|
N4*Apple Grove*CA*98765|
PO1*1*6*EA*5.74**
BP*00054321876547|
DTM*002*19980522|
PO1*2*12*EA*8.7**
BP*00054321647437|
PO1*3*12*EA*10.75**
BP*00054321674310|
SE*12*0001|
```

Figura 5: Mensagem EDI para uma Ordem de Compra.

Uma mensagem EDI é altamente estruturada, sendo normalmente de difícil leitura para um leitor que desconhece a sua estrutura. Para interpretar cada trecho da informação, normalmente é necessário comprar caros dicionários que definem as regras gramaticais e semânticas de cada parte da informação (XMLGLOBAL, 2003).

É preciso considerar também que se eventualmente o padrão EDI é atualizado, geralmente é necessário que o usuário atualize o seu sistema de tradução para o nível do novo padrão, implicando em um contínuo custo de manutenção para continuar usando EDI.

Há, portanto, algumas barreiras para a implantação/manutenção do EDI, tais como alto custo de implantação, incluindo a contratação de serviços especializados para definir

qual o subconjunto de mensagens EDI serão intercambiadas entre duas empresas quaisquer, bem como a complexidade das mensagens EDI. Estes aspectos dificultam a utilização do EDI em pequenas e médias empresas.

Nos últimos anos, o surgimento da XML trouxe uma série de vantagens em relação às limitações apresentadas. Assim, muitas empresas estão migrando para ferramentas e aplicações que utilizam esta tecnologia, pois são menos complexas que EDI e mais baratas para operar e manter. Isto não quer dizer que a XML venha a substituir completamente a EDI, principalmente porque, ao longo do tempo, muitas grandes empresas e organizações governamentais investiram bastante em sistemas "tradicionais" de EDI. Por outro lado, muitos esforços têm sido feitos na direção de uma padronização de sistemas EDI que utilizam XML. A próxima seção apresenta algumas destas iniciativas.

3.3 Linguagens baseadas em XML para intercâmbio de Informações

A XML (*Extensible Markup Language*) é uma linguagem para representação estruturada de dados. É uma linguagem de marcadores com estrutura bastante similar à HTML (*Hyper Text Markup Language*). A especificação da XML (W3C, 2000) é suportada por uma série de padrões, e possui diversas implementações de APIs e ferramentas em diversas linguagens e plataformas, que permitem que documentos XML possam ser facilmente interpretados computacionalmente.

Por hora não vamos explorar em profundidade a XML, mas apenas destacar a sua crescente importância como formato para modelagem e representação de informações. Os aspectos sobre XML que são relevantes a este trabalho serão explicados na seção a seguir.

XML está emergindo como um formato para representar as informações intercambiadas entre sistemas, não apenas no domínio da Internet, mas também como suporte à interoperabilidade entre aplicações heterogêneas entre empresas (OSÓRIO *et al.* 2000). Isto ocorre principalmente devido a sua flexibilidade (define novos marcadores), legibilidade (por pessoas), e de ser facilmente interpretada (com auxílio de ferramentas chamadas *parsers*).

A seguir serão apresentados algumas iniciativas e projetos criados por consórcios e grupos industriais espalhados pelo mundo tais como RosettaNet (ROSETTANET, 2003) e

OASIS[5] (OASIS, 2003a), que visam definir tecnologias aplicadas em domínios específicos. O objetivo aqui é apenas ilustrar a ampla utilização da XML nas mais diversas áreas. Uma lista completa indexando uma grande quantidade destes projetos é apresentada em OASIS (2003b):

1. **XML/EDI:** criado pelo *XML/EDI Group* (XMLEDI, 2003), o *framework* XML/EDI é uma proposta que integra XML com EDI, com o objetivo de permitir que organizações possam conduzir seus negócios eletronicamente de forma rápida, barata e mais fácil de manter, independente do tamanho da organização. XML/EDI aborda o problema a partir do desenvolvimento de um sistema que permita que cada parceiro troque não apenas os dados EDI, mas também *templates* que definem a lógica de seu processamento (WEBER *et al.*, 2000).

2. **ebXML (Electronic Business using EXtensible Markup Language):** patrocinado pelo UN/CEFACT[6] e OASIS, é um conjunto de especificações que permitem que empresas de qualquer tamanho e em qualquer lugar administrem seus negócios pela Internet. Usando ebXML, companhias possuem um método padrão para trocar mensagens de negócios, administrar seus relacionamentos comerciais, comunicar dados em termos comuns, e definir e registrar processos de negócios (EBXML, 2003).

3. **cXML (Commerce XML):** utilizado para aplicações de negócio e comércio eletrônicos (CXML, 2003).

4. **DSML (Directory Services Markup Language):** usado na representação de catálogos de serviços (DSML, 2003).

5. **PDML (Product Data Markup Language):** para intercâmbio de informações de produtos entre sistemas comerciais ou governamentais (PDML, 2003). PDML associa a conectividade oferecida pela Internet, a sintaxe amigável e compreensível da XML

[5] OASIS é um consórcio internacional que guia o desenvolvimento, convergência, e adoção de padrões para e-business.
[6] *United Nations Centre for Trade Facilitation and Electronic Business*. Entidade da ONU responsável por definir padrões para comércio e negócio eletrônicos, como o UN/EDIFACT.

para o intercâmbio de dados, e a metodologia para a integração de informações de produtos definida pelo STEP[7] (BURKETT, 2001).

6. **FinXML:** aplicado no intercâmbio de dados entre instituições financeiras (FINXML, 2003).

7. **WML (Wireless Markup Language):** para especificação de conteúdo e interfaces de usuário para dispositivos móveis, tais como telefones celulares (OASIS, 2003c).

8. **Serviços Web:** arquitetura que permite que aplicações disponibilizem serviços e se comuniquem no ambiente Web. Embora a noção de aplicações se comunicando via Web não represente um conceito novo (CGI, *servlets* Java, etc.), muitos destes sistemas consistem de soluções proprietárias. Uma das vantagens dos serviços Web é a utilização de padrões com o objetivo de diminuir as barreiras na integração entre sistemas (CERAMI, 2002). Mais especificamente, serviços Web utilizam linguagens baseadas em XML como representação nas diferentes camadas que compõem sua arquitetura. São elas:
    - **XML-RPC:** é um protocolo simples que usa mensagens XML para fazer *chamadas de procedimento remotas* (*remote procedure call*). Requisições e respostas são codificadas em XML e enviadas via HTTP.
    - **SOAP (Simple Object Access Protocol):** tal como o XML-RPC, o SOAP é um protocolo baseado em XML para intercâmbio de informações. Pode ser usado em uma variedade de sistemas de troca de mensagens e enviado via diversos tipos de protocolos de transporte (normalmente HTTP).
    - **WSDL (Web Service Description Language):** gramática para especificar uma interface pública de um serviço Web. Pode incluir informações sobre todas as funções públicas disponíveis, tipos de dados, informações sobre o protocolo de transporte a ser utilizado, e o endereço para localizar o serviço especificado.
    - **UDDI (Universal Description, Discovery, and Integration):** faz o papel do *registro de serviços*, onde provedores de serviços publicam os seus serviços Web, e onde clientes podem procurar e localizar estes serviços.

[7] *STandard for the Exchange of Product Model Data* - padrão internacional (ISO 10303) para intercâmbio de dados de produtos dentro de uma organização, ou entre organizações.

9. **Web Semântica:** iniciativa da W3C que visa estender o potencial da *World Wide Web* (WWW) atual em termos de disponibilizar dados que podem ser compartilhados tanto por pessoas quanto por ferramentas automáticas. Seguindo esta iniciativa, idealmente os programas devem ser capazes de compartilhar dados mesmo se estes tenham sido desenvolvidos de forma totalmente independente. Em suma, a idéia básica da *Web Semântica* é que os dados disponíveis na *Web* sejam definidos e vinculados de modo que possam ser usados por máquinas não apenas para propósitos de visualização, mas também para automação, integração, e reuso destes dados entre várias aplicações (W3C, 2004). É baseada em uma série de padrões, a saber:
    - **RDF (Resource Description Framework):** linguagem de propósito geral para representar a semântica de informações na *Web*, através de um conjunto de regras que permitem a descrição destas informações.
    - **OWL (Web Ontology Language):** linguagem para definir ontologias estruturadas e baseadas em Web, permitindo uma interoperação e integração de dados mais ampla. Ou seja, ao contrário de linguagens anteriores usadas para definir ferramentas e ontologias para áreas específicas, esta foi definida para ser compatível com a arquitetura da WWW em geral, e a *Web Semântica* em particular.
    - **DAML+OIL (DARPA Agent Markup Language + Ontology Inference Layer):** linguagem semântica de marcadores (*markup*), utilizada em recursos *Web*, baseada no padrão RDF. Foi definida a partir da DAML original (DAML, 2004), utilizada para ontologias, visando combinar muitos dos componentes da OIL (OIL, 2004).

10. **XMI (XML Metadata Interchange):** criado pelo OMG, XMI tem como principal propósito o *intercâmbio de metadados* entre ferramentas de modelagem UML (Unified Modeling Language) e repositórios de metadados baseados no MOF (Meta Object Facility) utilizando XML. XMI é, portanto, uma maneira de codificar metadados baseados no MOF em documentos XML. Abaixo há uma descrição das tecnologias relacionadas:
    - **XML (Extensible Markup Language):** será detalhado na próxima seção.

- **UML (Unified Modeling Language):** linguagem para visualização, especificação, construção e documentação dos elementos que compõem um sistema de *software*. Embora esta seja a sua principal utilidade, a UML não está limitada à modelagem de *software*, sendo expressiva o bastante para modelar sistemas de diversos domínios diferentes (BOOCH *et al.*, 2000).
- **MOF (Meta Object Facility):** tecnologia adotada pelo OMG (*Object Management Group*) para definir metadados e representá-los como objetos CORBA (OMG, 2002). A OMG definiu uma arquitetura de metamodelos, dentre os quais se encontra a UML. Além disso, MOF suporta qualquer tipo de metadados que possam ser descritos usando técnicas de modelagem de objetos. Por esta razão, a notação gráfica utilizada na UML pode também ser usada para expressar os metamodelos MOF (OMG, 2002).

Um ponto importante é que como a UML está definida como um metamodelo MOF, a especificação XMI leva diretamente a um formato de intercâmbio de modelos UML. Uma conseqüência disto é a interoperabilidade entre ferramentas de modelagem UML, sendo possível que modelos UML definidos em uma ferramenta possam ser lidos por outra. Várias ferramentas já adotaram o XMI como formato para salvar seus modelos UML, tais como Rational Rose (RATIONAL, 2003) e Poseidon (GENTLEWARE, 2003).

3.4 EXTENSIBLE MARKUP LANGUAGE (XML)

Conforme o que foi apresentado até agora, percebe-se o crescente papel da XML no contexto do intercâmbio de informações entre aplicações heterogêneas. Nesta seção a XML será vista em maiores detalhes uma vez que foi adotada na proposta a ser apresentada nesta dissertação. Este detalhamento vai desde a sua estrutura e sintaxe, passando pelos esquemas que definem os vocabulários, as APIs para o seu processamento, e finalmente, os tipos de aplicações.

A XML é uma linguagem de marcadores criada pelo W3C com a intenção de se adicionar contexto e estrutura à informação contida na Web, que até então era feita usando HTML.

3.4.1 Histórico

A origem da XML (e também da HTML) está vinculada à SGML[8]. Nos anos 60, a IBM criou a GML[9], destinada à criação de seus documentos internos. Posteriormente, a própria IBM desenvolve a SGML que em 1986 torna-se um padrão (ISO 8879) e passa a ser adotada em larga escala pela indústria como um padrão de informação para múltiplos propósitos. Embora a SGML seja extremamente poderosa na representação da informação, seu processamento exige uma parcela considerável de recursos computacionais.

No final dos anos 80, foi criada uma aplicação simplificada da SGML, chamada HTML, que posteriormente se tornou o formato padrão para a informação na Web. Com o passar do tempo, novas necessidades foram surgindo, e a HTML evoluiu (suportada por outras tecnologias, como linguagens de *script*, Java, *plug-ins*, etc.). Por mais que a HTML suprisse bem as necessidades das aplicações na Internet, ela possui várias limitações, e talvez a maior delas seja o seu limitado e fixo conjunto de marcadores.

Em 1996, os membros do W3C procuravam meios de introduzir a SGML no contexto da *Web*. Isso porque a SGML oferece três benefícios significativos que foram perdidos na HTML: *extensibilidade*, *estrutura* e *validação*. Tecnicamente HTML é estruturada, mas como os navegadores Web não avaliam com muito critério sua estrutura, em termos práticos HTML é dita uma linguagem fracamente estruturada. O W3C,

[8] *Standart Generalized Markup Language*

[9] *Generalized Markup Language*

portanto, começou a desenvolver uma linguagem de marcadores que agregava os benefícios da SGML e a relativa simplicidade da HTML. Foi então que, em fevereiro de 1998, foi liberada a especificação 1.0 da XML (MORRISON, 2000).

3.4.2 Definição

XML é um subconjunto simplificado da SGML que incorpora muitas de suas características, incluindo as três principais:

- **Extensibilidade:** XML possui a flexibilidade de permitir a definição de novos marcadores que tenham algum sentido para uma dada aplicação. Por esta característica, XML também pode ser vista como uma metalinguagem, pois permite definir outras linguagens de marcadores. Entretanto, o fato de não possuir um conjunto predefinido de marcadores implica também em não haver semântica predefinida. Toda semântica deve ser tratada pela aplicação que processa documentos XML.
- **Estrutura:** ao contrário da HTML, XML possui uma estrutura sintática mais rígida.
- **Validação:** a estrutura de um documento XML é definida formalmente por um *esquema* XML, que pode ser um DTD (*Document Type Definition*), ou um XML *Schema*. Assim, um documento XML pode ter sua estrutura validada.

Outra característica importante é o fato de que em XML é possível descrever os dados e os relacionamentos entre eles. Esta é uma das principais metas da XML: separar o conteúdo (dados), a estrutura (metadados) e a apresentação (a forma como os dados são visualizados).

A XML é definida por várias especificações relacionadas:

- **Extensible Markup Language 1.0**: define a sintaxe para a XML e o DTD (W3C, 2000).
- **XML Pointer Language** e **XML Linking Language**: define uma forma padrão para representar vínculos (*links*) entre recursos. XPointer descreve como endereçar um recurso, enquanto XLink descreve como associar dois ou mais recursos (W3C, 2002a) (W3C, 2001b).
- **Extensible StyleSheet Language**: descreve uma linguagem de folhas de estilo padrão para documentos XML (W3C, 2001c). Diferente da HTML, a XML não possui um conjunto de *tags* fixo e com semântica definida (que no caso da HTML é a definição da estrutura *visual* de um documento). Assim, a forma em que documentos XML são

visualizados é definida separadamente, com o uso de *folhas de estilo* (*style sheet*), convertendo documentos XML para outros formatos (HTML, PFD, PostScript, etc.).

Com o passar do tempo, novas necessidades surgiram e estão sendo resolvidas por outras especificações, tais como:

- ***Namespaces***: mecanismo que permite utilizar marcadores definidos por diferentes conjuntos de marcadores (W3C, 1999a).
- **XML *Query***: linguagem para consultas em um documento ou coleção de documentos XML (W3C, 2003b).
- **XML *Schema***: linguagem para definir a estrutura de documentos XML (W3C, 2001a).

Quase todas estas especificações definem linguagens. Assim, a forma utilizada para a representação destas especificações é a EBNF[10]. EBNF é uma linguagem utilizada para a especificação formal da sintaxe de linguagens (gramática). Ela consiste de um conjunto de regras (produções) que descrevem fragmentos da sintaxe. A aplicação destas regras permite validar a sintaxe de documentos. Este assunto é abordado em detalhes por AHO et al. (1986), ou qualquer livro sobre compiladores modernos.

3.4.3 Sintaxe e Estrutura da Linguagem

XML é uma linguagem permite representar os dados e seus respectivos metadados, sendo estes definidos na forma *marcadores* (*markup*). Portanto, documentos XML são constituídos basicamente por *dados* e *marcadores*. Os principais marcadores definidos pela XML 1.0 (W3C, 2000) são:

- Elementos (*tags*) e atributos;
- Referências a Entidades;
- Comentários;
- Instruções de Processamento;
- *Document Type Declarations*;

[10] *Extended Backus-Naur Form*

A Figura 6 ilustra um exemplo de documento XML que descreve uma ordem de compra. Este exemplo será discutido nas próximas seções para explicar o papel de cada uma das estruturas da linguagem.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ordemDeCompra SYSTEM "ordem.dtd" []>
<!-- Ordem de Compra -->
<ordemDeCompra número="1234" data="18/05/2003">
  <cliente>
    <nome>João da Silva</nome>
    <endereço>Av. Mauro Ramos, 321</endereço>
    <cidade>Florianópolis</cidade>
    <estado>SC</estado>
  </cliente>
  <item>
    <produto código="123">
      <descrição>Camisa Polo</descrição>
      <preço>59.90</preço>
    </produto>
    <quantidade>1</quantidade>
  </item>
  <item>
    <produto código="456">
      <descrição>Meia Azul</descrição>
      <preço>12.50</preço>
    </produto>
    <quantidade>3</quantidade>
  </item>
</ordemDeCompra>
```

Figura 6: Documento XML representando uma Ordem de Compra.

3.4.3.1 Elementos (Tags)

Elementos são os marcadores mais comuns da XML. O escopo de um elemento é definido por um *tag inicial* e um *tag final*. Na Figura 6, o elemento `preço` está definido entre os *tags* `<preço>` e `</preço>`. Note que o *tag inicial* é envolvido pelos caracteres "maior que" (<) e "menor que" (>), enquanto que o *tag final* possui, além destes caracteres, uma "barra invertida" (/) antes o nome do elemento.

O nome de um elemento normalmente identifica o seu conteúdo (daí o fato de ser chamado metadado). Em relação ao conteúdo, um elemento pode conter apenas texto, como o elemento `preço` (contento o texto "`12.50`"); pode conter outros elementos, como `item` (contento o elemento `produto`); ou pode ser vazio (não contém caracteres de texto ou elementos). Neste último caso, há duas formas distintas: a primeira é como já foi apresentado, com os *tag* inicial e final, ou seja, `<elem></elem>`; a outra maneira é com

apenas o *tag* inicial, mas com uma "barra invertida" logo após o nome o elemento, sendo então `<elem/>`.

Elementos também podem possuir *atributos*. Atributos são pares *nome/valor* e servem para oferecer dados adicionais aos elementos. No exemplo da Figura 6, o elemento `ordemDeCompra` possui um atributo `data` com o valor `"18/05/2003"`. Em XML, todos os valores de atributos devem estar entre aspas.

3.4.3.2 Referências a Entidades

Entidades são nomes únicos associados a blocos de dados em XML. Entidades são usadas normalmente para referenciar dados repetidos ou que constantemente variam; podem ser usadas também para referenciar o conteúdo de arquivos externos. O uso de entidades em XML é semelhante ao de constantes em linguagens de programação: visam facilitar a escrita de documentos, sendo expandidas (pré-processamento) antes da avaliação do compilador.

Outra função do uso de entidades é no caso de se usar caracteres especiais (reservados) como texto normal. Um tipo especial de referência à entidade, chamado de *referência a caractere*, é usada para referenciar caracteres *Unicode* no documento XML.

3.4.3.3 Comentários

Comentários são usados para apresentar informações que não fazem parte do conteúdo do documento XML. Tal como em linguagens de programação, comentários são apenas para leitura por parte do usuário, sendo ignorados pelo *parser*[11]. São escritos entre "`<!—`" e "`-->`", conforme pode ser visto na Figura 6 (página 31).

3.4.3.4 Instruções de Processamento

Instruções de Processamento são marcadores especiais usados pela aplicação que processa documentos XML. Tal como os comentários, não são consideradas parte do conteúdo do documento, mas o *parser* deve processá-las e repassá-las à aplicação.

Instruções de processamento são definidas entre "`<?`" e "`?>`". O exemplo mais claro deste tipo de marcador é `<?xml version="1.0"?>`. Esta instrução é reservada da XML e serve para auxiliar a sua identificação. Ela indica que o documento é baseado na

[11] Um *parser* ou *processador de XML* é um componente de *software* usado por aplicações para o processamento de documentos XML (ver seção 3.4.5).

versão 1.0 da XML. Futuramente isto servirá para evitar problemas de incompatibilidade com versões antigas ou não suportadas por um dado *parser* (MORRISON, 2000).

Um parâmetro opcional da instrução de processamento `<?xml ?>` é a declaração da codificação (*encoding*) dos caracteres do texto no documento, ou seja, a forma como os caracteres são representados nos padrões de *bits*. A base de codificação de caracteres em XML é definida pelos padrões ISO/IEC e Unicode (W3C, 2000), aumentando a sua portabilidade, pois permite a flexibilidade de usar caracteres em múltiplas línguas. Como no exemplo da Figura 6 são usados caracteres acentuados, é necessário então declarar um padrão para codificação adequado, que neste caso é `ISO-8859-1`.

3.4.3.5 Document Type Declarations

Para um documento XML ter significado para uma dada aplicação, é necessário haver algumas restrições na seqüência e aninhamento dos marcadores. Estas restrições são expressas por *declarações*.

Declarações permitem que um documento XML informe ao *parser* sobre *meta-informações* a respeito de seu conteúdo. Tais meta-informações correspondem à seqüência e aninhamento dos elementos, seus atributos e tipos de dados, nomes de arquivos externos que podem ser referenciados (e se eles contêm ou não dados em XML), os formatos de dados externos (não XML) que podem ser referenciados, e as entidades que podem ser encontradas (WALSH, 1998).

Estas informações são especificadas em um documento XML na *Document Type Declaration*. Se estiver presente em um documento, esta declaração deve ser a primeira estrutura a aparecer no documento, depois de *instruções de processamento* e *comentários* (ambos opcionais). Uma *document type declaration* realiza as seguintes tarefas:

- Especifica o *elemento raiz* do documento XML.
- Define os elementos, atributos e entidades específicos ao documento (DTD interno).
- Identifica um DTD externo ao documento (em um arquivo separado).

No DTD (*Document Type Definition*, não confundir com ***Document Type Declaration***) são feitas as declarações de elementos, atributos e entidades que podem ocorrer em um documento XML (ver seção 3.4.4).

O exemplo da Figura 6 apresenta a *document type declaration* destacada em negrito:

```
<!DOCTYPE ordemDeCompra SYSTEM "ordem.dtd" []>
```

O elemento raiz do exemplo é o elemento "`ordemDeCompra`". O DTD externo ao documento é "`ordem.dtd`". E caso houvesse um DTD interno, este deveria estar declarado entre os colchetes no final da declaração.

3.4.3.6 Documentos Bem Formados e Válidos

Agora que a estrutura e sintaxe básica da linguagem foram apresentadas, é possível definir dois conceitos fundamentais a respeito de documentos XML: documentos podem ser *bem formados* e *válidos*. Um documento é considerado *bem formado* se obedecer as seguintes regras:

- Deve haver um único elemento (*elemento raiz*) ao qual todos dever estar contidos;
- Todo elemento deve ter um *tag* inicial e um *tag* final correspondente, ou um único *tag* vazio;
- Se a *tag* inicial está no conteúdo de um elemento, a *tag* final também deve estar, ou seja, todas as *tags* não vazias devem estar adequadamente aninhadas.
- Todo valor de atributo deve estar entre aspas;

Um documento é considerado *válido* se:

- For *bem formado*;
- Obedecer à estrutura definida no *esquema* XML correspondente. Basicamente, um *esquema* XML pode ser um DTD ou um XML *Schema*, conforme será visto na seção a seguir.

3.4.4 Definindo a Estrutura de Documentos XML

Como dito na seção anterior, um documento XML é considerado válido se respeitar a estrutura definida no seu *esquema* XML correspondente, ou seja, o conjunto de elementos e atributos que um documento XML deve ter. As próximas seções apresentam de maneira superficial como isto é feito pelos dois tipos básicos de esquemas XML: DTD e XML *Schema*.

3.4.4.1 DTD

Em um DTD (*Document Type Definition*), é possível definir quase todos os aspectos de um conjunto de marcadores. Assim, um documento XML que declara este DTD deve estar de acordo com a estrutura e as restrições definidas neste para que possa ser considerado válido. Uma restrição em relação ao DTD é que não é possível definir um *tipo* para os dados dos elementos, sendo que qualquer valor é tratado como do tipo *string*. Por exemplo, um campo que representa a *idade*, não pode ser definido como do tipo inteiro. Dessa forma, o tratamento de tipos deve ser feito ao nível de aplicação. A Figura 7 abaixo apresenta um exemplo de DTD, que será explicado a seguir.

```
<!ELEMENT ordemDeCompra (cliente, item+)>
 <!ATTLIST ordemDeCompra número CDATA #REQUIRED>
 <!ATTLIST ordemDeCompra data CDATA #REQUIRED>
 <!ELEMENT cliente (nome, endereço, cidade, estado)>
  <!ELEMENT nome (#PCDATA)>
  <!ELEMENT endereço (#PCDATA)>
  <!ELEMENT cidade (#PCDATA)>
  <!ELEMENT estado (#PCDATA)>
 <!ELEMENT item (produto, quantidade)>
  <!ELEMENT produto (descrição?, preço)>
  <!ATTLIST produto código CDATA #REQUIRED>
   <!ELEMENT descrição (#PCDATA)>
   <!ELEMENT preço (#PCDATA)>
  <!ELEMENT quantidade (#PCDATA)>
```

Figura 7: *Document Type Definition* da Ordem de Compra.

Declarações

Dentre as declarações que podem ser feitas em um DTD, as principais são: *declarações de elementos* e *declarações de atributos*.

Declaração de Elementos

Um documento XML é considerado válido quanto todos os elementos que estão contidos nele estão declarados no DTD. A forma geral de uma declaração de elemento é:

```
<!ELEMENT nomeElemento conteúdoElemento >
```

Nesta declaração, `nomeElemento` corresponde ao nome do elemento, e `conteúdoElemento` indica o conteúdo que este elemento pode conter, que pode ser:

1. Outros Elementos: para indicar que um elemento contém outros elementos, basta declarar estes elementos na ordem em que devem aparecer. Há dois símbolos que podem ser usados para separar os elementos (chamados *elementos filhos*):

- Vírgula (,) – Cada elemento deve ocorrer na ordem em que foi declarado;
- Barra Vertical (|) – Apenas um dos elementos declarados deve ocorrer.

Além disso, é também possível indicar o número de vezes que um elemento pode ocorrer. Para isso, utilizam-se caracteres especiais após o nome do elemento para indicar a sua freqüência, conforme a Tabela 1:

Indicador	Significado
(sem indicador)	Elemento deve aparecer uma *única* vez.
?	Zero ou uma ocorrência do elemento.
+	Elemento pode ocorrer uma ou mais vezes.
*	Elemento pode ocorrer zero ou mais vezes.

Tabela 1: Indicadores de freqüência de elementos.

2. Apenas texto: a palavra reservada `#PCDATA` indica que este elemento deve conter apenas texto.

3. Vazio: a palavra reservada `EMPTY` indica que o elemento não pode conter nem texto, nem elementos. Entretanto pode possuir atributos, conforme será explicado a seguir.

4. Conteúdo Misto: é declarada a lista dos elementos válidos, seguida por um `#PCDATA`. A declaração deve estar entre parênteses e seguida pelo operador *. O elemento poderá ter qualquer combinação de texto e elementos (declarados) como conteúdo.

5. Qualquer conteúdo: usa-se a palavra reservada `ANY`, indicando que o elemento pode conter qualquer combinação de elementos ou texto.

Para estes dois últimos tipos de conteúdo, o fato de permitir um conteúdo muito pouco estruturado faz com que seu uso não seja recomendado por alguns tipos de aplicações, conforme será visto mais adiante.

Declaração de Atributos

A declaração de atributos possui a seguinte forma geral:

```
<!ATTLIST nomeElemento nomeAtributo tipoAtributo default >
```

O valor de *nomeElemento* indica o nome do elemento que possui este atributo e *nomeAtributo* contém o nome do atributo. Os valores possíveis de *tipoAtributo* definem o tipo do atributo que, embora possa assumir vários tipos diferentes, os mais comuns são o tipo CDATA, que corresponde a dados de texto, e o *tipo enumerado*, onde os possíveis valores do atributo devem estar declarados em uma lista separados por barras verticais.

E por fim, o campo *default* indica um valor padrão, ou uma palavra reservada indicando um qualificador para o uso deste atributo, conforme a Tabela 2:

Palavra chave	Significado
#REQUIRED	O atributo *deve* ocorrer no elemento.
#IMPLIED	O atributo é opcional.
#FIXED "valor"	O atributo possui um valor fixo (constante).
Valor padrão (default)	Valor padrão do atributo (caso nenhum seja definido).

Tabela 2: Qualificadores de um atributo.

3.4.4.2 XML Schema

Embora o uso de DTDs seja adequado para diversas aplicações, o seu fraco sistema de tipos de dados (aplicando-se apenas a atributos) torna-se um fator limitante em casos onde este aspecto é relevante. Conseqüentemente, esta e outras limitações foram as principais motivações para o desenvolvimento de novas linguagens de esquema, que são genericamente denominadas como XML *Schema*.

Naturalmente, o uso de XML *Schemas* não representa a solução definitiva para todos os casos, pois além das situações em que é mais adequado usar DTDs, há algumas coisas que XML *Schemas* não podem fazer, como por exemplo, declarações de entidades.

XML Schema da W3C

Existem várias linguagens para XML *Schema*, dentre as quais destacam-se: *Relax*, *Schematron*, TREX (*Tree Regular Expressions for XML*), DDML (*Document Definition Markup Language*) e a *XML Schema* da W3C. Embora existam semelhanças entre várias dessas linguagens, esta seção falará apenas do *XML Schema* da W3C. Esta linguagem foi criada pelo *W3C XML Schema Working Group*, e sua especificação é considerada maior e mais complexa do que a própria especificação 1.0 da XML.

Estrutura

A especificação do XML *Schema* da W3C divide elementos em dois tipos: *complexos* e *simples*. Um elemento do tipo *simples* pode apenas conter texto e não deve ter atributos, mas é possível definir o tipo de dados deste elemento como, por exemplo, inteiro, data/hora, decimal, etc. Já os elementos complexos podem ter atributos e/ou elementos filhos. A Figura 8 apresenta o XML *Schema* que descreve uma ordem de compra, que será explicado a seguir.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="ordemDeCompra" type="ordemType"/>

<xs:complexType name="ordemType">
  <xs:sequence>
   <xs:element name="cliente" type="clienteType"/>
   <xs:element name="item" minOccurs="1" maxOccurs="unbounded" type="itemType"/>
  </xs:sequence>
  <xs:attribute name="número" type="xs:int" use="required"/>
  <xs:attribute name="data" type="xs:date" use="required"/>
 </xs:complexType>

<xs:complexType name="clienteType">
  <xs:sequence>
   <xs:element name="nome" type="xs:string"/>
   <xs:element name="endereço" type="xs:string"/>
   <xs:element name="cidade" type="xs:string"/>
   <xs:element name="estado" type="xs:string"/>
 </xs:complexType>

 <xs:complexType name="itemType">
  <xs:sequence>
   <xs:element name="produto">
    <xs:complexType>
     <xs:element name="descrição" minOccurs="0" maxOccurs="1" type="xs:string"/>
     <xs:element name="preço" type="xs:nonNegativeInteger"/>
     <xs:attribute name="código" type="xs:int" use="required"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="quantidade" type="xs:int"/>
</xs:complexType>

</xs:schema>
```

Figura 8: *XML Schema* para a Ordem de Compra.

Percebe-se neste exemplo que XML *Schemas* nada mais são do que documentos XML. A idéia é não sejam necessárias duas sintaxes diferentes para o esquema (no caso dos DTDs) e as informações (XML) (HAROLD, 2001). Isto pode parecer confuso à primeira vista, pois há elementos que definem outros elementos, mas também mostra uma característica muito poderosa da XML que é a de ser *autodescritiva*.

Tal como qualquer documento XML, XML *Schemas* começam com a instrução de processamento **<?xml version="1.0"?>**. O elemento raiz de um XML *Schema* é o elemento **schema**. Nele está declarado o *namespace*[12] utilizado, que neste exemplo é identificado pelo URI **http://www.w3.org/2001/XMLSchema**, (padrão do XML *Schema* da W3C), e possui o prefixo **xs**. Como a especificação é bastante extensa, será mencionado apenas o suficiente para o entendimento básico da sua estrutura. Assim, podemos dividir o conteúdo de um XML *Schema* em dois tipos de declarações: *declarações de elementos*, e *declarações de tipos* (estes últimos podem ser simples ou complexos). A definição dos tipos separadamente permite, por exemplo, que diferentes elementos compartilhem o mesmo tipo.

Declarando Elementos

Para declarar um elemento, deve-se usar o elemento **element**, que possui os seguintes atributos:

1. **name**: define o nome do elemento.
2. **type**: define o tipo do elemento, que pode ser:
    - *Tipo simples*. Existem mais de 40 tipos simples predefinidos na especificação, que vão desde tipos numéricos (inteiros, ponto flutuante, etc.), tipos de data e hora, tipos de texto (*strings*), etc.
    - *Tipo Complexo*. Tipos complexos (próxima seção) correspondem aos tipos que possuem outros elementos no seu conteúdo e/ou atributos. Neste caso, o valor do atributo será o nome de um tipo, que deve estar declarado.
    - Se este atributo for omitido, obrigatoriamente deverá haver uma declaração de tipo complexo dentro do escopo desta declaração (*tipo anônimo*).
3. **minOccurs**: especifica o número mínimo de ocorrências do elemento.
4. **maxOccurs**: define o número máximo de ocorrências do elemento.

Uma observação importante é que com o uso de **minOccurs** e **MaxOccurs** é possível definir exatamente quantas vezes um elemento pode repetir. Por exemplo, pode-se

[12] *Namespaces* são utilizados para evitar que ocorram nomes repetidos em documentos XML, permitindo que se possa utilizar definições de vários XML *Schemas* sem que ocorram conflitos de nomes. Um *namespace* é identificado por um URI (*Uniform Resource Identifier*) declarado no atributo **xmlns**, e pode ser associado a um prefixo, que serve para qualificar os nomes utilizados. Maiores detalhes sobre *namespaces* podem ser encontrados na sua especificação (W3C, 1999a).

definir que um elemento deva ocorrer no mínimo 4 e no máximo 7 vezes, ou que não tem limites (**unbounded**). Se forem omitidos, será assumido o valor 1.

Declarando Tipos Complexos

Na seção anterior foi visto que um elemento pode ser de um tipo predefinido simples, ou que pode ser de um tipo complexo. Para declarar um tipo complexo, usa-se o elemento **complexType**, que possui um atributo **name** que o identifica (para referenciá-lo). Se este atributo for omitido, este será um *tipo anônimo*, e deve estar declarado no escopo da declaração de um elemento.

A definição da estrutura de um tipo complexo é feita por diversos elementos (ditos elementos de agrupamento, ou elementos de grupo), todos declarados dentro do escopo da declaração do tipo complexo. São eles:

1. **sequence:** cada elemento (**element**) declarado como seu filho deve aparecer na ordem especificada.
2. **choice:** especifica que apenas um elemento declarado neste grupo deve ocorrer (análogo ao operador "|" do DTD). Pode possuir os atributos **minOccurs** e **maxOccurs** para definir o número de vezes que o grupo **choice** deve repetir.
3. **all:** cada elemento declarado em um grupo **all** deve ocorrer no máximo uma vez, mas a ordem não é importante. Além disso, ele só permite declarações de elementos, não sendo permitido declarações de **sequence** e **choice** dentro dele.
4. **attribute:** declara um atributo. Se o **complexType** possui elementos **all**, **sequence**, ou **choice**, as declarações de atributos devem estar sempre no final. O elemento **attribute** tem como principais atributos:
    - **name:** nome do atributo.
    - **type:** tipo do atributo.
    - **use:** indica se a ocorrência do atributo é obrigatória (**required**), ou se é opcional (**optional**). Se for omitido, o atributo será considerado opcional.
    - **default:** se o valor de **use** for **optional**, pode-se definir um valor *default*.

Se o tipo complexo não possuir nenhum destes elementos, ou apenas **attribute**, será do tipo EMPTY (análogo ao DTD).

Considerações a Respeito do XML *Schema*

XML *Schemas* representam um grande avanço em relação aos DTDs para modelagem de dados em XML. Embora o que foi apresentado aqui seja suficiente para o entendimento desta linguagem, XML *Schema* possui muitos outros recursos disponíveis, sendo que o mais poderoso sem dúvida é a sua capacidade de definir novos tipos. Existem muitos outros elementos e estruturas que permitem derivar novos tipos baseados nos tipos simples predefinidos na linguagem. Esta e outras características podem ser encontradas na sua especificação (W3C, 2001a), ou na bibliografia relacionada, como em HAROLD (2001) ou VLIST (2002) que apresentam o assunto com maior profundidade.

3.4.5 APIs para Processamento de Documentos XML

Para que a informação codificada em XML possa ser acessada por aplicações, é necessário um mecanismo responsável pelo processamento de documentos XML. Como documentos XML são simples arquivos de texto, qualquer programador pode desenvolver o seu próprio leitor de arquivos XML de forma que possa extrair as informações necessárias para sua aplicação. Entretanto, esta tarefa tende a ser trabalhosa, principalmente porque para cada tipo de documento XML é necessário desenvolver um novo processador.

Visando resolver este problema, o W3C desenvolveu uma forma padronizada para criar estes “leitores de documentos XML” ou *parsers* de XML (IDRIS, 1999b). Assim, várias empresas, como a IBM, Microsoft e Sun, desenvolveram *parsers* de XML que estão disponíveis gratuitamente em diversas linguagens. Portanto, um desenvolvedor não precisa escrever o seu próprio *parser*, basta obter algum que atenda às suas necessidades.

Além de processar documentos XML, um *parser* de XML deve realizar pelo menos a primeira das tarefas abaixo:

- ***Parsing*:** verifica se um documento é *bem formado*, ou seja, se está de acordo com as regras sintáticas da linguagem XML.
- **Validação:** verifica se um documento é válido, ou seja, se é bem formado e se está de acordo com as regras definidas no esquema XML.

Para acessar as informações armazenadas em um documento XML, existem duas abordagens distintas que um *parser* de XML pode utilizar: DOM e SAX.

3.4.5.1 Document Object Model – DOM

O *parser* de XML que implementa esta abordagem gera um modelo hierárquico de objetos a partir de um documento XML. Para isso, ele precisa realizar o processamento completo do documento XML, consistindo basicamente em expandir todas as entidades usadas no documento e realizar o *parsing* e a validação (se houver a declaração de um esquema XML). Se o processamento é bem sucedido, o *parser* cria a representação deste documento, que corresponde a um modelo hierárquico de objetos (em forma de árvore).

Este modelo, chamado DOM (*Document Object Model*), define a estrutura lógica e a forma em que um documento XML é acessado e manipulado. É padronizado pelo W3C (2003a), que especifica a sua API[13]. A estrutura de um documento é composta de nós, que por sua vez podem conter outros nós e/ou outras informações. Cada nó corresponde a um elemento processado pelo *parser*, na mesma ordem em que este se encontra no documento XML. Este modelo é bastante adequado, pois XML é naturalmente hierárquica (Figura 9).

Assim como o *parser*, o programador também não precisa implementar a API do DOM, pois esta já está inclusa no "pacote" de distribuição do *parser*. Portanto, como o *parser* resolve o problema do processamento de documentos XML e gera a representação deste documento na forma de uma árvore de objetos, cabe à aplicação a tarefa de extrair a informação contida nesta estrutura de dados.

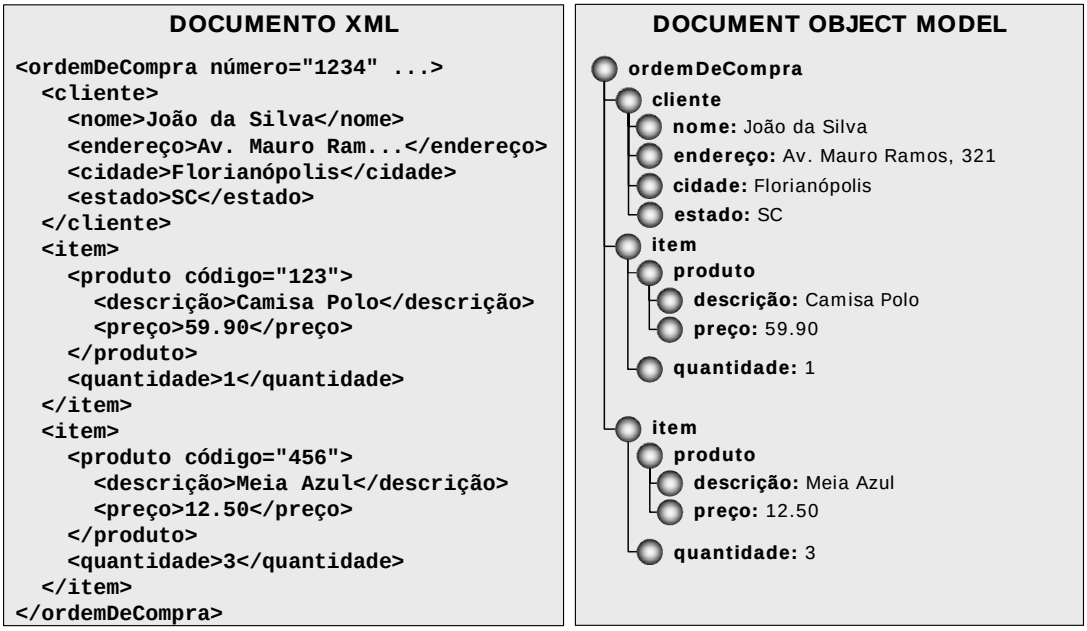

Figura 9: *Document Object Model*.

[13] *Application Programming Interface*

Dentre as principais operações que estão disponíveis na API do DOM, destacam-se:

- Obter a referência do nó raiz;
- Obter/modificar o nome de um nó;
- Obter a lista de nós filhos de um determinado nó;
- Obter/modificar o valor de um nó;
- Obter a lista de atributos de um nó;
- Adicionar e remover um nó;

3.4.5.2 Simple API for XML – SAX

A contrário do DOM, que cria uma representação baseada na estrutura de um documento XML, SAX não possui um modelo de objetos padrão. Ao invés disto, o *parser* SAX dispara eventos à medida que lê o documento XML (IDRIS, 1999a). Os eventos são disparados quando são encontrados:

- *Tag* que abre um elemento;
- *Tag* que fecha o elemento;
- Seções #PCDATA e CDATA;
- Instruções de processamento, comentários, DTD, declarações de entidades.

Por apenas disparar eventos, o *parser* SAX é mais rápido, mas isso também implica que o programador terá duas tarefas adicionais: criar um modelo de objetos próprio e criar uma classe que capture os eventos SAX (*document handler*) para instanciar o modelo de objetos. O *document handler* é então associado ao *parser* SAX.

O *document handler* é a peça mais importante da abordagem SAX, pois é ele quem deve interpretar adequadamente os eventos e instanciar os objetos do modelo de objetos. A Figura 10 mostra um exemplo de seqüência de eventos.

O modelo de processamento de documentos XML usando um *parser* SAX é semelhante ao DOM, com a diferença de que não há a criação de um modelo de objetos padrão, mas sim uma série de eventos que são interceptados por um *document handler*, que por sua vez instancia o modelo de objetos da aplicação.

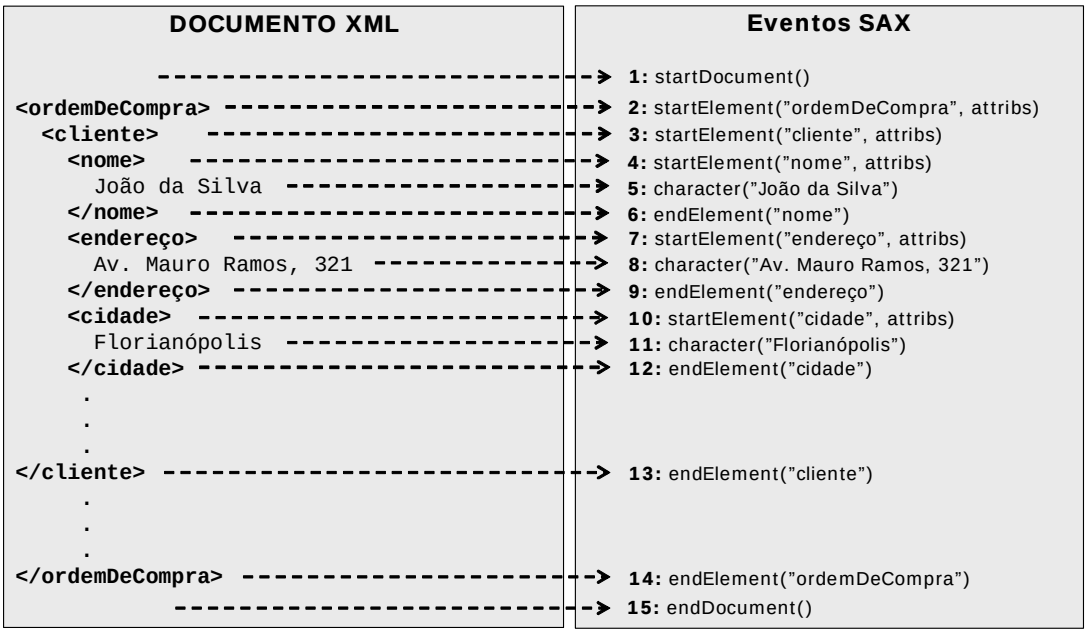

Figura 10: Seqüência de eventos disparados pelo *parser* SAX.

3.4.5.3 Comparando as Abordagens

Nas seções anteriores foram apresentadas duas abordagens diferentes para o processamento de documentos XML. Cada uma apresenta vantagens e desvantagens que devem ser exploradas de acordo com a necessidade de cada aplicação.

Sendo baseada em eventos, a abordagem SAX é rápida, sendo indicada para aplicações que exijam maior desempenho. Outra vantagem é que, lendo o documento por partes, faz com que os dados sejam disponibilizados a medida em que são encontrados, sem a necessidade da aplicação ter que esperar o processamento de todo o documento. Entretanto, estas vantagens demandam um preço: complexidade no seu desenvolvimento. SAX é mais difícil de se desenvolver, pois sendo mais simples, força o desenvolvedor a escrever mais código (*document handler* para capturar os eventos e o modelo de objetos, para armazenar os dados). Devido a esta complexidade, muitos desenvolvedores preferem trabalhar com APIs que já provenham uma estrutura na memória (objetos) para representar os documentos XML processados (MCLAUGHLIN, 2002).

APIs que criam um modelo XML na memória (DOM) são mais populares, pois possuem uma curva de aprendizado menor. Como foi visto, é possível facilmente percorrer a árvore gerada e extrair os atributos de seus nós utilizando a API do DOM. Mas o fato da aplicação ter que esperar o processamento completo do documento antes de disponibilizar o DOM, faz com que esta abordagem seja normalmente mais lenta do que a SAX. É

necessário também mais memória, pois são criados objetos para cada estrutura XML do documento. Entretanto, estas desvantagens devem ser comparadas com o fato de ser um modelo de programação bem mais fácil (MCLAUGHLIN, 2002).

Em suma, cada abordagem deve ser explorada de acordo com o contexto, sendo que no âmbito deste trabalho, ambas serão utilizadas, cada uma envolvida em aspectos diferentes da implementação. Como última consideração, pode-se dizer que estas abordagens são consideradas *APIs de baixo nível*, pois permitem acesso direto aos dados (e estrutura) de documentos XML, mas requerem que se escreva código para tal (MCLAUGHLIN, 2002). Mais adiante serão apresentadas as *APIs de alto nível* como uma alternativa para o desenvolvimento de aplicações integradas com XML.

3.4.6 Vantagens e Desvantagens da XML

Tal como qualquer tecnologia, a viabilidade da aplicação da XML precisa ser avaliada em cada tipo de aplicação. Esta seção citará algumas vantagens e desvantagens no uso da XML, fazendo um breve comentário sobre cada uma delas. Uma descrição bastante abrangente sobre os prós e contras da XML é feita por ZAPTHINK (2001). Em IDRIS (1999a) são apresentados os principais benefícios da XML.

3.4.6.1 Vantagens

XML é um formato de texto estruturado: esta é, provavelmente, a característica mais visível da XML. Isto significa que documentos XML representam não apenas os dados, mas também os metadados (marcadores) que provêm significado e estrutura à informação e facilitam a sua leitura. Por outro lado, em casos que não requerem muita complexidade, formatos mais simples podem ser utilizados de forma mais eficiente.

XML é legível por humanos: conseqüência natural do fato da XML ser estruturada. Esta característica torna mais fácil o processo de depuração e testes, reduzindo o tempo de implementação das suas aplicações. Por outro lado, isto torna os documentos mais volumosos, além de ser possível definir marcadores sem um significado evidente.

XML é independente de plataforma: por ser um formato texto, XML é facilmente processado em diferentes plataformas, permitindo interoperabilidade em sistemas heterogêneos. O seu suporte a Unicode faz com que a XML seja altamente portável e capaz de representar texto em diversas línguas.

É possível definir outras linguagens com XML: conforme já destacado, XML tem a *extensibilidade* como uma das principais características. Ou seja, usando esquemas XML é possível criar vocabulários específicos.

Documentos XML podem ser validados: através de um *parser*, um documento XML pode ser validado em relação à estrutura definida no esquema XML correspondente. Tudo isso é feito de forma automática, sem a intervenção humana. Além disso, estas ferramentas estão amplamente disponíveis e a maioria de forma gratuita.

XML é um padrão aberto: isto significa que é uma especificação "neutra" de fabricante, ou seja, documentos XML podem ser criados, modificados e processados por ferramentas de diferentes fabricantes que sigam a sua especificação. É claro que isto não se aplica aos formatos e vocabulários definidos em XML, que podem ser proprietários.

XML permite separar estrutura, dados e apresentação: com o auxílio de outras tecnologias relacionadas, a XML separa a estrutura (esquemas XML), os dados (documentos XML) e a apresentação (folhas de estilos, definidas em CSS[14] ou XSL[15]). Isto não acontece em HTML, onde os dados estão misturados com a apresentação.

Esquema XML definido em comum acordo resulta em documentos intercambiáveis: em outras palavras, é definido um formato para troca de informações. Em XML é possível definir a estrutura dos dados, mas a semântica deve ser resolvida pela aplicação, que não pode ser representada em XML. Isto é muito importante, pois os marcadores podem ter significados diferentes por diferentes partes.

3.4.6.2 Desvantagens

XML é "volumosa": uma das mais significativas desvantagens da XML é que ela utiliza muito espaço para representar dados que poderiam ser modelados de forma similar em um formato binário ou um formato de texto mais simples. Este é o preço pago pelas vantagens acima enumeradas, e que não pode ser subestimado (pode chegar a ser até 20 vezes maior do que uma representação binária). É claro que, devido à sua redundância, é possível utilizar técnicas de compressão, reduzindo drasticamente o seu tamanho.

XML é adequado para texto, mas não para dados binários: XML é um formato excelente para dados texto, mas por outro lado, não é adequado para representar dados binários, como imagens, músicas e programas executáveis. Dados binários podem ser

[14] *Cascading Style Sheets*
[15] *Extensible Stylesheet Language*

representados em XML através de mecanismos de codificação baseados em Unicode ou ASCII. Uma outra forma é fazer uma referência a estes dados utilizando um URL[16].

3.4.7 Aplicações da XML

As aplicações da XML vão além da definição de textos narrativos, tais como livros, manuais, relatórios, páginas *Web*, etc. XML tem sido utilizada em diferentes domínios de aplicação, atuando principalmente no:

- Armazenamento de informações (arquivos XML/ bases de dados XML nativas);
- Publicação e visualização de informações;
- Integração entre aplicações heterogêneas;
- **Intercâmbio de informações** (protocolo de comunicação de alto nível).

No que diz respeito ao intercâmbio de informações, é preciso destacar que XML torna-se uma forma padronizada para representar informações, sendo independente de como estas são armazenadas pelas aplicações. Não importa se as aplicações utilizam bases de dados (de qualquer plataforma), ou um formato de arquivo proprietário, ou mesmo XML para armazenar as suas informações: se as aplicações trocam suas informações utilizando um vocabulário XML em comum, pode-se dizer que isto corresponde a uma "camada" que é independente dos detalhes de implementação de cada aplicação.

Como neste trabalho o foco será dado neste tipo de aplicação, a próxima seção apresentará alguns conceitos relacionados à integração de aplicações com a XML.

3.4.7.1 Considerações sobre XML e Bases de Dados

Considerando que documentos XML são gerados e lidos por aplicações, é preciso levar em conta que os dados contidos nestes documentos são originados e/ou armazenados em algum repositório associado à aplicação. Assim, é imprescindível destacar o papel das bases de dados relacionais como repositório de dados mais comumente utilizado. Nesta seção serão feitas algumas considerações sobre como a XML pode ser usada com bases de dados. BOURRET (2003a) faz uma discussão mais abrangente sobre este assunto.

A principal questão a ser respondida é: um documento XML pode ser considerado uma base de dados?

[16] *Uniform Resource Locator*

Um documento XML pode ser considerado uma base de dados somente no sentido mais estrito do termo, pois também representa uma coleção de dados. Desta forma, quando considerado como uma base de dados, XML apresenta algumas vantagens. Por exemplo, XML é autodescritiva (marcadores), é portável (Unicode), e pode descrever dados em árvores ou grafos, além de prover muitos aspectos encontrados em bases de dados: armazenamento (documentos XML), esquemas (esquemas XML), linguagens de consulta (XQuery, XPath, XQL, etc.), interfaces para programação (SAX, DOM). Por outro lado, há a falta de muitos outros aspectos encontrados em bases de dados "reais": armazenamento eficiente, índices, segurança, transações e integridade de dados, acesso de múltiplos usuários, entre outros.

Vê-se então que embora seja possível usar um ou mais documentos XML como base de dados em ambientes com pouca quantidade de dados, poucos usuários, e requisitos de desempenho modestos (como arquivos de configuração, listas de contatos, lista de "favoritos" de um navegador *Web*, etc.), esta abordagem não é adequada na maioria dos ambientes, pois possuem muitos usuários, um volume de dados significativo, e a necessidade de bom desempenho.

Um outro fator (talvez o mais importante) na escolha de uma base de dados é saber se esta será usada para armazenar *dados* ou *documentos*. Isto é um aspecto importante, pois documentos ditos **centrados em dados** (*data-centric*) possuem um conjunto de características diferentes dos documentos **centrados em documentos** (*document-centric*), e isto influenciará em como documentos XML são salvos na base de dados (BOURRET, 2003a) (BOURRET, 2003b).

Documentos Centrados em Dados

São documentos que usam XML como transporte de dados e são basicamente interpretados por programas. Documentos centrados em dados são caracterizados por uma estrutura bastante regular, dados bem definidos (a menor unidade independente de dados é um elemento do tipo texto ou um atributo), e pouco ou nenhum conteúdo misto. A ordem em que os campos (elementos PCDATA) ocorrem é geralmente insignificante, exceto quando se precisa *validar* o documento. Exemplos de documentos centrados em dados são ordens de compra, dados bibliográficos, dados científicos, etc. Dados que podem ser encontrados em documentos centrados em dados podem originar de uma base de dados, ou de fora de uma base de dados (neste caso, possivelmente serão salvos em uma base de

dados). Exemplos do primeiro caso correspondem à vasta quantidade de dados legados armazenados em bases de dados relacionais, como por exemplo, uma ordem de compra (Figura 6, na página 31). Um exemplo do segundo caso é um dado científico qualquer obtido por um sistema de medição e convertido para XML.

Documentos Centrados em Documentos

São documentos que são utilizados geralmente em aplicações onde serão interpretados por pessoas, como por exemplo, livros, *e-mail*, anúncios, contratos, páginas *Web*, etc. São caracterizados por uma estrutura irregular ou pouco regular, dados não tão bem definidos (a menor unidade independente de dados pode ser um único elemento ou todo o documento), e bastante conteúdo misto. A ordem em que os elementos PCDATA ocorrem é sempre significante. Ao contrário de documentos centrados em dados, eles geralmente não se originam de uma base de dados, sendo normalmente escritos manualmente ou convertidos de outros formatos de definição de documentos, como RTF, PDF e SGML. Um exemplo de documento centrado em documentos é um contrato de aluguel (Figura 11).

```
<contratoDeAluguel>
 Eu, <proprietário>João da Silva</proprietário>, concordo
 em alugar a propriedade localizada na <logradouro>Av. Mauro
 Ramos, 321</logradouro> por um período de tempo não inferior
 a <tempoDeAluguel unidade="Mês">12</tempoDeAluguel> a um
 preço de <preço unidade="Real">400</preço> por mês.
</contratoDeAluguel>
```

Figura 11: Documento XML centrado em documentos.

Na prática, a distinção entre documentos centrados em dados e centrados em documentos nem sempre é clara. Há casos em que documentos centrados em dados, como uma fatura, podem conter dados pouco estruturados, como uma descrição do produto; e documentos centrados em documentos podem conter dados bem estruturados, como um manual que possui o nome do autor e a data de revisão. Outros exemplos são documentos legais e médicos, que são escritos como um texto não estruturado, mas que contêm trechos bem definidos de dados, como datas, nomes, procedimentos, e geralmente devem ser armazenados como documentos completos por razões legais (BOURRET, 2003a).

Apesar disto, a caracterização de documentos como centrados em dados ou centrados em documentos ajudará na decisão de que tipo de base de dados utilizar. Como regra geral, *dados* podem ser armazenados em uma base de dados tradicional (relacional, orientada a objetos ou hierárquica), que pode ter o suporte de outra aplicação (*middleware XML*) ou ter capacidades embutidas na própria base de dados (*bases de dados integradas com XML*). Já *documentos* podem ser armazenados em uma *base de dados XML nativa*, ou em um *sistema de gerenciamento de conteúdo*. Estas e outras aplicações são apresentadas na próxima seção.

3.4.7.2 Classificação das Aplicações

BOURRET (2003b) classifica as aplicações que integram XML e bases de dados conforme pode ser visto a seguir. Devido à semelhança entre algumas delas, é importante salientar que os limites entre certas categorias podem se tornar arbitrários. São elas:

***Middleware* XML:** programa chamado por aplicações sempre que é necessário transferir dados entre documentos XML e bases de dados. Utilizam documentos centrados em dados.

XML *Data Binding*: programas que podem "ligar" documentos XML aos objetos de uma aplicação. Alguns podem armazenar/recuperar objetos de uma base de dados. Utiliza documentos centrados em dados.

Bases de Dados integradas com XML: bases de dados com extensões para transferir os seus dados para documentos XML e vice-versa. Normalmente utilizam documentos centrados em dados.

Bases de Dados XML Nativas: bases de dados que armazenam XML em uma forma "nativa", geralmente em alguma variante do DOM mapeado em uma estrutura de armazenamento. Este tipo de aplicação inclui a categoria anteriormente conhecida como *Persistent DOM* (PDOM). Pode usar tanto documentos centrados em dados como centrados em documentos.

Servidores XML: servidores *Web*, ou servidores em geral. Podem ser usados em aplicações distribuídas ou simplesmente para publicar documentos XML na *Web*. Inclui a categoria conhecida como *servidores de aplicações XML*. Utilizam tanto documentos centrados em dados como centrados em documentos.

Sistemas de Gerenciamento de Conteúdo: aplicações situadas "acima" das bases de dados nativas e/ou do sistema de arquivos, e são responsáveis pelo gerenciamento do

conteúdo de documentos XML. Inclui funcionalidades para bloqueio de acesso, controle de versão e edição. Utilizam documentos centrados em documentos.

Máquinas de Busca em XML: ferramentas que permitem fazer consultas (*query*) em documentos XML. Usam documentos centrados em dados e centrados em documentos.

Dentre as categorias citadas acima, o *middleware XML* e a *XML Data Binding* são as que melhor se adequam à proposta deste trabalho. Elas são explicadas em maiores detalhes nos próximos capítulos.

3.5 Considerações Finais

Este capítulo apresentou brevemente algumas das principais tecnologias e padrões para intercâmbio de modelagem de informações, com especial destaque para a ampla utilização da XML como base para a maioria destas tecnologias.

Além disso, uma consideração importante sobre a interoperabilidade entre aplicações é que as tecnologias baseadas na Internet, em especial os serviços Web, estão sendo cada vez mais adotadas por diversas aplicações e padrões. SOAP, por exemplo, é largamente utilizado, e padrões como ebXML o utilizam como protocolo de troca de mensagens (FREMANTLE *et al.*, 2002). Seguindo esta tendência, a XML será utilizada na modelagem e implementação do presente trabalho, conforme será visto nos próximos capítulos.

4 Configuração e Integração de Dados em EV: Enquadramento e Proposta

4.1 Introdução

Uma EV necessita passar por uma série de configurações, não apenas para operar adequadamente, mas também para que possa responder a uma oportunidade de mercado em tempo hábil. Estas configurações refletem a forma como as empresas participantes interagirão entre si, cobrindo desde aspectos específicos, como o modelo de negócios da EV, até questões mais gerais, como as tecnologias de informação que darão suporte a estas interações.

Dentre estes aspectos, a configuração das informações intercambiáveis, que tende a consumir uma parcela considerável de tempo, será abordada neste trabalho visando agilizar este processo e conseqüentemente minimizar o tempo e os custos associados. A proposta a ser apresentada trata da configuração do intercâmbio de informações em uma EV, mas focando apenas em um nível mais alto, ou seja, na estrutura (formato e semântica) dessas informações. Os aspectos da infraestrutura de comunicação, tais como protocolos para transporte, segurança, qualidade de serviço, etc., estão fora do escopo desta dissertação. Estes aspectos são abordados, por exemplo, em OSÓRIO et al. (1999).

Outro aspecto que será tratado neste trabalho diz respeito à integração das informações provenientes dos sistemas legados de cada parceiro. Embora a questão da integração seja mais ampla, cobrindo também a integração de processos (coordenação das atividades), esta última não será coberta neste trabalho. Assim, de modo similar à configuração das informações intercambiáveis, esta dissertação apresentará um modelo para integração das informações legadas com o objetivo de tornar este processo mais ágil.

Os modelos a serem apresentados para estas duas atividades são executados por módulos específicos que fazem parte da *plataforma da EV*, que conceitualmente pode ser vista como a infraestrutura tecnológica que provê suporte à interação entre os membros participantes da EV.

A próxima seção enquadrará estas duas atividades dentro do ciclo de vida de uma EV, bem como os respectivos componentes que atuarão na plataforma.

4.2 Enquadramento do Trabalho dentro do Ciclo de Vida e Plataforma da EV

Embora os resultados da configuração das informações intercambiáveis e integração das informações legadas tenham conseqüências em todo o ciclo de vida da EV, ela ocorre antes do ciclo de vida propriamente dito. A Figura 12 apresenta o ciclo de vida da EV (SPINOSA *et al.*, 1998) (CAMARINHA-MATOS *et al.* 1999a) precedido pela etapa de *preparação do ambiente*, onde se situam os modelos e respectivas ferramentas de configuração e integração propostos nesta dissertação (em cinza escuro). Adicionalmente, são apresentados os módulos da plataforma que conceitualmente são os responsáveis por cada atividade durante o ciclo de vida.

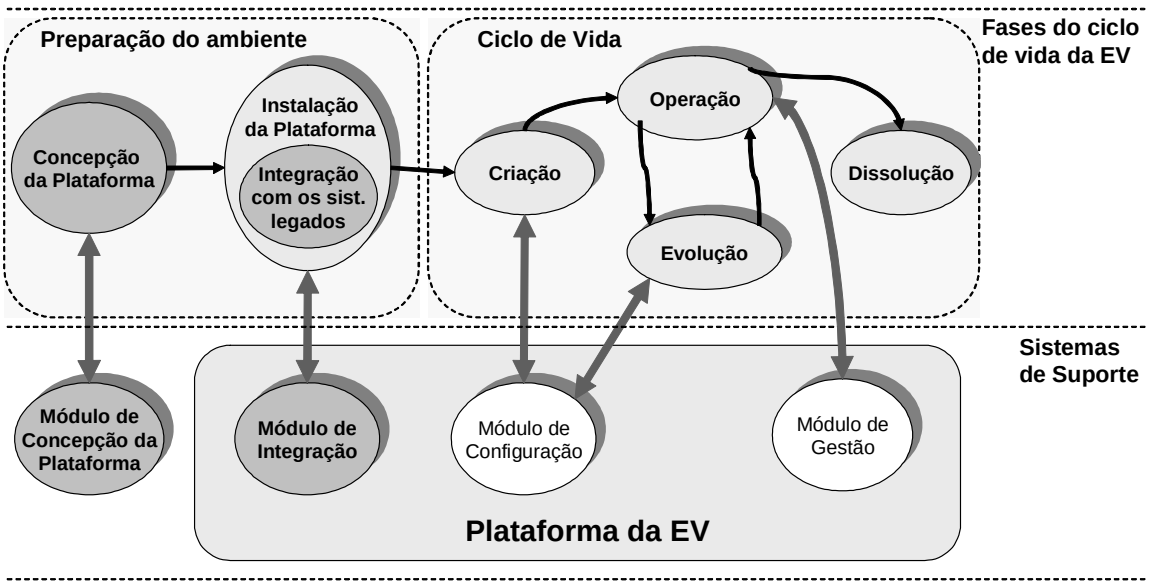

Figura 12: Configuração do Modelo de Informações da EV.

A etapa de *preparação do ambiente* consiste dos seguintes passos:

Concepção da Plataforma: está inserida no processo de modelagem e desenvolvimento da plataforma propriamente dita da EV, ou seja, nesta fase é onde todos os *softwares* de suporte às funcionalidades de todo ciclo de vida da EV são desenvolvidos. Como explicado no Capítulo 1, o que aqui se pretende é automatizar parte do processo de desenvolvimento, focando no aspecto da definição e implementação do formato para o intercâmbio de informações que deverão ser realizadas entre os vários membros da EV, uma vez todos possuindo a plataforma.

Instalação da Plataforma: corresponde à implantação da plataforma da EV em cada parceiro. Nesta etapa, o módulo de integração é utilizado para a integração do modelo genérico de dados utilizado pela plataforma com os repositórios de dados locais dos sistemas legados de cada empresa. Depois de instalar a plataforma, a empresa estará apta para participar de uma EV. Deve-se notar que, neste caso, o módulo de integração representa apenas uma das atividades normalmente executadas quando da instalação de uma plataforma de EV.

Em termos gerais, a lógica desse processo inicial de uma EV dá-se da seguinte forma: primeiramente há que se chegar a um acordo sobre quais informações (incluindo estrutura e semântica, onde a parte da sintaxe estará "resolvida" pela adoção da XML) deverão ser trocadas entre os parceiros. Chegado a este acordo, todos os módulos que compõem uma plataforma para EVs devem ser configurados para receber, entender e armazenar as informações trocadas (concepção da plataforma). Como uma parte significativa dessas informações provém dos sistemas de gestão "internos" já existentes nas empresas-membro da EV (os sistemas legados), há que se construir uma "ponte" entre estes e os módulos da plataforma que têm esta necessidade. Portanto, uma vez concebida, a plataforma deve ser instalada e, para tal, todas essas "pontes"/integrações têm que ser feitas.

Salienta-se novamente que tanto a concepção da plataforma como essas integrações com os sistemas legados têm sido feitas de forma não assistida e fixa no código (*hard-coded*), o que leva usualmente meses de trabalho. Os módulos propostos nesta dissertação visam melhor sistematizar e oferecer uma assistência computacional, um grau de automatismo para essas duas atividades que, além de trazerem maior confiabilidade ao processo, reduzem significativamente o tempo de configuração e preparação da plataforma. Como destacado na seção 2.5, a sistematização e automação (mesmo que parcial) dessas duas etapas são uma inovação introduzida frente aos vários projetos pesquisados na área de EV. Daí que "classicamente" não são contempladas nos arcabouços existentes que delineiam o ciclo de vida de uma EV.

4.2.1 Arquitetura da Plataforma Computacional da EV

Nesta seção será apresentada uma arquitetura geral da plataforma computacional que provê suporte às funcionalidades da EV, com o objetivo de destacar as entidades envolvidas nos modelos de configuração e integração a serem apresentados. Esta

arquitetura representa a plataforma que deve estar presente em cada um dos parceiros da EV, e que se comunicam entre si através da rede (Figura 13).

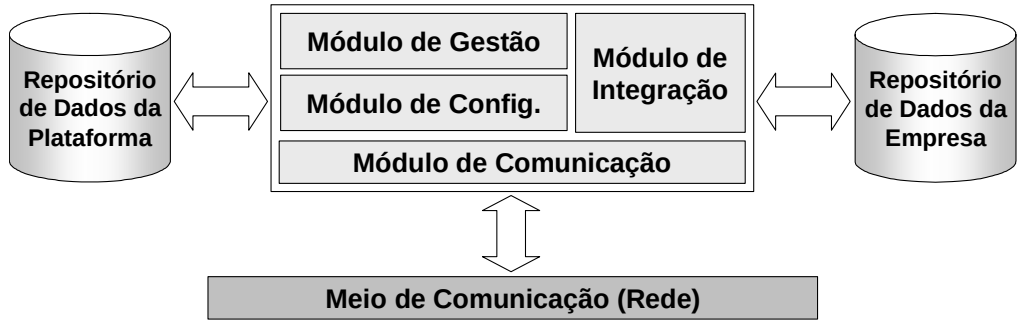

Figura 13: Arquitetura geral de uma plataforma para EVs.

O **Módulo de Gestão** engloba os serviços oferecidos ao nível de aplicação (usuário), tais como gestão e coordenação da EV, como abordado por RABELO et al. (2002).

O **Módulo de Configuração** disponibiliza serviços relacionados à configuração da EV, como busca e seleção de parceiros, configuração da topologia da rede, definição dos direitos de acesso/visibilidade, etc, como abordado em RABELO et al. (2002), SCHMIDT (2003) e BALDO (2003).

No **Repositório de Dados da Plataforma** são armazenadas as informações relevantes aos demais módulos da plataforma. Estas informações são definidas conforme um *Modelo de Referência de Informações*, sendo este definido previamente durante a concepção da plataforma. Este tópico é detalhado na seção a seguir.

O **Módulo de Comunicação** corresponde à interface responsável pelo intercâmbio de informações entre as plataformas, isto é, entre os parceiros da EV. Estas informações são normalmente oriundas do repositório de dados, estando assim de acordo com o *modelo de referência* adotado.

O papel do **Módulo de Integração** é mapear as informações contidas no **Repositório de Dados da Empresa** (relativo aos sistemas legados) para o *Modelo de Referência de Informações* adotado pela plataforma da EV. Este módulo é de importância fundamental para o funcionamento da EV visto que cada empresa participante possui sistemas de gestão/coordenação internos já implantados cujas informações serão utilizadas pelas funcionalidades da plataforma da EV. Este assunto é detalhado no Capítulo 6.

Deve-se salientar que o trabalho proposto nesta dissertação independe do modelo de troca de informações entre os membros da EV. Conforme o modelo adotado por uma certa

plataforma, a troca (isto é, a comunicação propriamente dita) pode ser feita variando desde simples mensagens ponto-a-ponto com TCP/IP e *sockets*, passando por sistemas de *workflow* interorganizacionais (com agentes, CORBA, etc.), e indo até a outros mecanismos de troca como os existentes em bases de dados distribuídas/federadas.

4.3 Modelo de Referência de Informações

Tendo em vista que a proposta deste trabalho está vinculada ao intercâmbio de informações entre os membros de uma EV, e que uma mesma informação pode ser interpretada de diferentes maneiras por diferentes parceiros, faz-se necessária uma definição em comum, que neste caso corresponde a um *Modelo de Referência de Informações* (MR). CAMARINHA-MATOS et al. (1999d) identifica a definição de modelos de referência em comum como pré-requisito para a cooperação entre os parceiros de uma EV, bem como a adoção de padrões (como EDI e STEP) para a definição de uma infraestrutura aberta. O MR define formalmente a estrutura da informação (meta-informações) a ser adotada por empresas (no caso, pela EV), eliminando eventuais problemas de interpretação.

O MR está fortemente ligado ao tipo de negócio (ramo de atuação) que a EV está envolvida. Assim, por exemplo, o MR de uma EV no ramo automobilístico tem algumas naturais diferenças em relação ao MR de uma EV do ramo têxtil. Além disso, EVs diferentes que atuam no mesmo tipo de negócio podem possuir MR diferentes (ou parcialmente diferentes), pois podem atuar em áreas específicas. Por exemplo, duas EVs atuando no ramo têxtil, mas a primeira trabalha apenas com trajes sociais enquanto a outra trabalha com moda infantil esportiva.

O que atualmente tem-se buscado é uma padronização de processos, ou seja, qualquer atividade empresarial, vista como um processo independentemente da implementação, é modelada com um certo âmbito funcional, tendo entradas, saídas e requerendo meios e regras de ativação para a sua execução. Formalizado isso, para todos os processos ter-se-ia naturalmente os modelos de dados genéricos associados, ou seja, os MR. Esta é a idéia sendo perseguida atualmente por várias iniciativas internacionais[17], fortemente estimuladas pelas principais empresas de *software* que querem diminuir

[17] ROSETTANET (2003), EBXML (2003), XMLEDI (2003), COVE (2003), VOMAP (2003), CIMOSA (2003) e VOSTER (2003).

sensivelmente o tempo e o custo de implantação/integração de sistemas. Apesar destes esforços, até o momento não existem MR disponíveis no mercado ou sendo largamente usados entre grupos de empresas, e tampouco específicos para EV, o que leva em certos casos à utilização de modelos proprietários e específicos. Desta forma, vem-se trabalhando com um MR concebido a partir de 1997 e que vem sendo gradualmente complementado até então, o chamado "DBP[18] Model" (RABELO *et al.*, 1999). Este MR foi utilizado nos testes das ferramentas desenvolvidas neste trabalho.

É preciso deixar claro que toda a problemática associada à definição de MRs de informações está, no entanto, além do escopo deste trabalho, sendo então considerado que um dado MR já foi definido. Além disso, para este trabalho, o MR adotado será o ponto de partida para a configuração da plataforma computacional da EV. Esta segunda afirmação é muito importante, visto que ela define a forma como o problema será abordado. Isto quer dizer que a abordagem deste trabalho é essencialmente *top-down*, ou seja, a partir de um MR já definido chega-se à configuração das estruturas de dados a serem intercambiados através da plataforma da EV. A outra abordagem, que não será usada neste trabalho, é *bottom-up*, que consiste em extrair os modelos de informações dos repositórios legados de cada membro da EV e, a partir destes modelos, chegar ao MR da EV. Todavia, mesmo sendo *top-down*, a proposta apresentada permite que as particularidades legadas das empresas-membro da EV sejam contempladas e harmonizadas, como será posteriormente detalhado.

4.3.1 XML Como Formato Para Definição de MR

Conforme visto no Capítulo 3, XML é uma linguagem padronizada para modelagem e representação de dados que possui a vantagem de ser extensível, facilmente legível, estruturada e com baixo custo de implementação. XML está emergindo como o padrão para intercâmbio de dados e interoperação entre aplicações nos mais diversos domínios, caracterizados pela heterogeneidade de modelos e plataformas. Portanto, neste trabalho, XML será a linguagem adotada para representar as informações intercambiáveis entre os parceiros de uma EV. Isto não visa apenas seguir a tendência atual onde cada vez mais aplicações utilizam esta tecnologia, mas também devido a todas as vantagens e benefícios que ela oferece para este fim, os quais a tornaram o "padrão" que é hoje. Desta maneira, os

[18] *Distributed Business Process*

resultados aqui obtidos poderão ser úteis às aplicações que utilizam XML como formato para troca de informações, ou seja, teve-se a preocupação de buscar uma solução aberta.

4.4 Proposta

Como o MR define formalmente a estrutura das informações, é possível implementar/configurar de forma semi-automática (em alguns casos automaticamente) algumas entidades da plataforma da EV. Com base nisto, este trabalho propõe dois modelos que tomam como base um MR (definido em XML):

- No **Capítulo 5** será proposto um modelo para *configuração de dados*, que consiste na geração semi-automática de código. Este processo é executado na etapa de *concepção da plataforma*, e toma como base o MR adotado pela EV. Conforme a *arquitetura da EV*, os seguintes módulos podem ser gerados:
  - **Repositório de dados da plataforma:** onde ficam armazenadas as informações relevantes aos outros módulos da plataforma da EV.
  - **Módulo de comunicação:** responsável pelo processamento das mensagens recebidas, geração das mensagens a serem enviadas, e o envio propriamente dito.
- O **Capítulo 6** propõe um modelo de *integração de dados* a ser utilizado na etapa de *instalação da plataforma*. Este processo consiste basicamente no mapeamento entre o MR adotado pela EV e o modelo de dados legado, sendo desempenhado pelo **módulo de integração**.

Embora sejam apresentados em seqüência, é fundamental salientar que estes modelos podem ser adotados independentemente, de acordo com os requisitos específicos de cada plataforma de EV. Mais ainda: embora o cenário apresentado se enquadre na problemática da configuração e integração de dados em uma plataforma de suporte a EV, ambos os modelos podem naturalmente ser aplicados em outros domínios de aplicação que apresentem requisitos semelhantes. Isto se torna bastante visível devido à disponibilidade de um grande número de ferramentas baseadas em XML, atuando em contextos variados.

5 Configuração de Dados em EV: Modelo Conceitual

5.1 Introdução

A etapa de concepção da plataforma envolve, em termos práticos, todo o processo de desenvolvimento da plataforma, cobrindo a sua modelagem, implementação, e finalmente, a etapa de testes. Assim, é preciso salientar a importância das metodologias destinadas a este fim, com ênfase nas técnicas de desenvolvimento orientado a objetos. Neste contexto, a UML merece destaque por ser uma linguagem padrão cujo principal objetivo é a representação dos diferentes aspectos que compõem um sistema (modelos), além de cobrir as diferentes etapas do seu desenvolvimento.

Embora apenas uma pequena parte deste processo seja abordada neste trabalho, o objetivo neste momento é chamar a atenção para a importância das boas práticas para o desenvolvimento de sistemas, e a sua importância para que a contribuição deste trabalho seja realmente efetiva.

Assim, tendo em vista o que foi dito, será considerada como premissa a utilização da abordagem orientada a objetos para a modelagem da plataforma (seguindo a UML). Embora isto não seja um requisito fundamental, a sua adoção facilitará a utilização dos resultados aqui apresentados.

5.1.1 Delimitação do Escopo

Considerando a arquitetura genérica apresentada anteriormente na Figura 13 (página 55), percebe-se que os diferentes módulos que a compõem são desenvolvidos na etapa de concepção. No modelo proposto neste trabalho, em termos de concepção da plataforma, será abordada apenas a concepção dos módulos vinculados ao MR adotado pela EV: o *repositório de dados da plataforma* e o *módulo de comunicação*.

De modo a tornar o mais claro possível os limites deste trabalho, algumas considerações devem ser feitas:

- O *módulo de comunicação* não será definido completamente, mas apenas o que diz respeito à representação dos dados que serão trocados na EV. Os aspectos de baixo

nível relacionados ao transporte das informações não serão tratados. Entretanto, a solução será apresentada de maneira independente destes aspectos, permitindo uma ampla liberdade para a escolha de qualquer mecanismo de transporte.

- Embora o *módulo de integração* esteja vinculado ao MR, neste trabalho ele é visto como uma ferramenta genérica, ou seja, não é concebido a partir do MR, mas sim *configurado* por ele. Por isto este módulo não será abordado neste capítulo, sendo visto em maiores detalhes no capítulo que segue.

5.1.2 Considerações Iniciais

Tendo o MR como base para o intercâmbio de dados entre as empresas participantes de uma EV, o *módulo de comunicação* de cada uma deve estar preparado para processar mensagens cujas estruturas de dados estejam de acordo com o MR adotado. Este processamento implica em ler a mensagem (validar, se for necessário) e armazenar os seus dados nas estruturas de dados utilizadas pela *plataforma* (objetos ou registros da base de dados). Adicionalmente, o *módulo de comunicação* deve ser capaz de gerar uma mensagem a partir dos dados dos objetos/registros, para o seu posterior envio.

Com o objetivo de justificar a adoção da XML como formato para intercâmbio de dados (representado pelo MR), a seguir será discutida a utilização de outras abordagens: linguagem definida em EBNF[19] ou o padrão EDIFACT.

Com EBNF é possível definir formalmente uma linguagem para intercâmbio de dados, mas, por esta ser mais simples se comparada com uma linguagem de programação, não há a necessidade de todo o poder e flexibilidade oferecidos por este formato. Outro caso seria a adoção do padrão EDIFACT, usado para intercâmbio de dados na área de comércio eletrônico e também em aplicações de EVs. Embora o EDIFACT seja um padrão largamente utilizado, tem como desvantagem o alto custo de implantação, além de definir um formato de representação praticamente ilegível (por pessoas). Este padrão vem sendo gradativamente substituído por diversas especificações baseadas em XML: ebXML, XML/EDI, entre outras. Por esta razão, EDIFACT não será analisado aqui, sendo feita uma comparação apenas entre XML e EBNF.

A primeira atividade (realizada pelo *módulo de comunicação*) a ser analisada é a leitura e validação de mensagens. Com XML, esta tarefa é feita facilmente, pois este

[19] *Extended Backus-Naur Form*. Formato usado para especificar formalmente gramáticas de linguagens livres de contexto, aplicado geralmente em implementações de compiladores para linguagens de programação.

padrão define um mecanismo para o processamento de documentos XML: o *parser* de XML. Além disso, dado um esquema XML (ver seção 3.4.4), o *parser* é capaz de verificar se um documento XML é ou não válido. No caso do uso de EBNF, existem ferramentas que fazem a geração automática de parte de compiladores de linguagens, ou seja, do código relativo apenas à leitura e validação de mensagens.

A próxima atividade é o armazenamento dos dados da mensagem nos objetos internos e/ou no repositório de dados. A implementação desta atividade já não é tão fácil quando a leitura e validação, pois envolve o modelo de objetos particular adotado pela plataforma. Assim, apesar de um *parser* XML oferecer a API do DOM para armazenamento da mensagem na forma de objetos, é necessário ainda escrever código para armazenar os dados no modelo de objetos adotado pela plataforma. O mesmo acontece para a geração de mensagens a partir dos dados armazenados no modelo de objetos e/ou no repositório de dados.

À primeira vista, não há uma diferença significativa que permita decidir sobre qual é a melhor abordagem (XML ou EBNF). Porém, há duas considerações que apontam a XML como abordagem mais adequada. Em primeiro lugar, há a questão da simplicidade: EBNF requer um conhecimento mais profundo na área de linguagens formais, bem como a intrínseca complexidade associada ao desenvolvimento de compiladores e processadores de linguagens. Com XML, o *parser* é visto como uma “caixa preta” pelo desenvolvedor, sem a necessidade deste se preocupar com a sua implementação.

Finalmente, e mais importante, é o papel essencial de um tipo de aplicação da XML, chamado *XML Data Binding* (seção 3.4.7.2). Este tipo de aplicação consiste na geração de um modelo de classes a partir de um esquema XML. Dessa forma, além do processamento automático de mensagens (feito pelo *parser* XML), a implementação das classes que armazenam estes dados pode ser feita semi-automaticamente. Este tipo de aplicação será a base do modelo conceitual apresentado neste capítulo, conforme será visto na seção a seguir.

5.2 Modelo Conceitual

Conforme dito no capítulo anterior, a abordagem para configuração dos dados é *top-down*, ou seja, o MR é o ponto de partida para o desenvolvimento da plataforma da EV. Mais especificamente, como o MR define um modelo de informações, a parte relevante da

plataforma da EV a ser gerada a partir deste MR consiste em estruturas de dados, que tipicamente são representadas como classes e tabelas de uma base de dados relacional, sendo então usadas pelos módulos da plataforma da EV. A seguir é apresentado um modelo que, a partir de um MR (definido em XML), faz a geração semi-automática das classes e tabelas do repositório de dados de acordo com a estrutura definida no mesmo.

O modelo geral de geração das classes e tabelas pode ser visto na Figura 14. Uma *especificação XML* serve de base para o *gerador* gerar uma série de classes e/ou tabelas. Associada à especificação, há um *esquema de ligação*, onde são definidos alguns parâmetros auxiliares ao processo de geração.

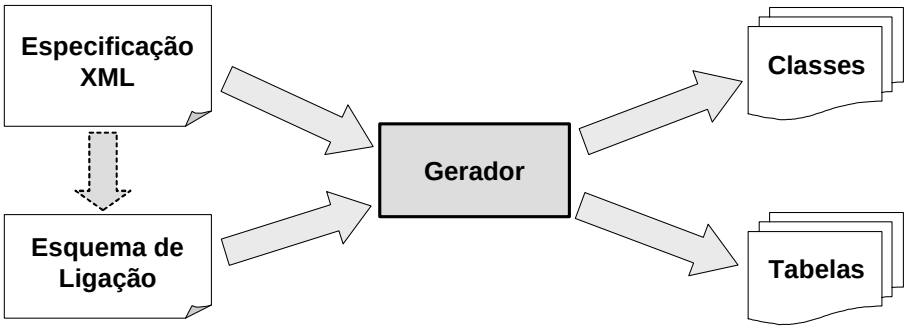

Figura 14: Modelo geral para a geração de classes/tabelas.

O modelo se enquadra na categoria de aplicação chamada *XML Data Binding* (apresentada na seção 3.4.7.2) que consiste na ligação (*binding*) de documentos XML com objetos cujas classes foram definidas especialmente para um certo esquema XML. Isto permite que aplicações manipulem dados de documentos XML de maneira mais natural e simples do que usando DOM ou SAX (BOURRET, 2003c). Assim, por ser mais abstrata que DOM e SAX (*APIs de baixo nível*), a *XML Data Binding* é vista como uma *API de alto nível*. É importante ressaltar que nesta abordagem o processamento de documentos XML (usando DOM ou SAX) ainda é feito. Entretanto, a *data binding* oculta estes detalhes, fornecendo os dados de uma forma mais adequada (objetos) à aplicação em questão (MCLAUGHLIN, 2002).

Baseado neste conceito, os módulos que compõem a plataforma da EV, e que utilizam estruturas de dados geradas a partir do MR adotado, podem tratar de maneira adequada as instâncias deste, que neste caso estarão representadas por documentos XML.

Como as mensagens intercambiadas entre os módulos da plataforma da EV contêm conjuntos de dados estruturados (pedidos, notas fiscais, especificação de produtos, etc.),

estes dados, quando codificados em XML, correspondem a documentos XML *centrados em dados* (seção 3.4.7.1). Portanto, a *XML Data Binding* é adequada neste caso, pois é preciso que a informação seja estruturada de forma a se obter um modelo de classes coerente.

5.2.1 Especificação XML

Uma *especificação XML*[20] (Figura 14) pode ser definida de três maneiras diferentes, a saber:

- **Document Type Definition (DTD):** como possui a limitação de não definir tipos de dados, estes podem ser definidos através do *esquema de ligação*.
- **XML *Schema* da W3C:** permite a definição mais detalhada de modelos de dados, como os tipos de dados e a cardinalidade dos elementos.
- **XML for Metadata Interchange (XMI):** permite a definição de modelos e metamodelos, conforme definido pelo OMG (2002). Neste trabalho, uma especificação codificada em XMI deve definir apenas modelos UML, e mais especificamente, apenas o diagrama de classes.

Embora conceitualmente o uso de qualquer tipo de especificação tenha como resultado um modelo de classes, percebe-se que com XMI há um modelo *explícito* de classes, enquanto que com esquemas XML (DTD ou XML *Schema*) deve haver uma conversão dos dados modelados em XML para um modelo de classes correspondente. XMI é também um documento XML, mas tem uma estrutura predefinida para representar modelos UML. Assim, é possível gerar as classes conforme o modelo UML apresentado (diagrama de classes), e estas serão instanciadas a partir de outros documentos XMI que contêm instâncias deste modelo (diagrama de objetos).

5.2.2 Esquema de Ligação

No *esquema de ligação* (*binding schema*) são especificados os detalhes a respeito de como as classes serão geradas a partir da especificação XML. O esquema de ligação permite modificar os nomes das classes/atributos a serem gerados (que podem, por

[20] Neste ponto é importante deixar claro que o termo *esquema XML* denota apenas DTDs ou XML *Schemas*, ou seja, apenas esquemas para documentos XML "tradicionais". O termo *especificação XML* engloba, além dos esquemas XML, documentos XMI contendo *diagramas de classes UML*.

exemplo, inicialmente receber o nome do elemento XML correspondente). O fato de permitir que o usuário modifique os nomes das estruturas geradas é importante não apenas por uma questão de flexibilidade (resultando em um modelo mais próximo dos seus requisitos), mas principalmente para se evitar erros no código que será gerado posteriormente, como por exemplo, uma classe com mesmo o nome de uma palavra reservada da linguagem. Além disso, um esquema de ligação pode ter um papel mais ou menos importante, dependendo do tipo de especificação XML: se for usado um DTD, é possível definir tipos de dados para os campos que serão gerados; no caso do XML *Schema* isto pode não ser necessário visto que o mesmo permite a definição de tipos de dados; já um diagrama de classes codificado em XMI representa a entrada *exata* para o gerador de classes, pois define claramente as classes, atributos e relacionamentos entre classes.

5.2.3 Serialização e Desserialização

As classes geradas são capazes de se salvarem/carregarem para/de documentos XML através de dois mecanismos implementados em métodos destas classes. São eles:

- **Serialização (*Marshalling*):** a classe é capaz de converter o seu conteúdo em um documento XML, normalmente passando por uma validação prévia dos seus dados. Naturalmente, o documento XML gerado estará de acordo (*válido*) com a especificação com a qual foram geradas as classes.
- **Desserialização (*Unmarshalling*):** processo inverso à serialização, que consiste em converter um documento XML em objetos. A maioria das ferramentas de *data binding* permite que documentos possam ser validados em relação ao esquema XML com o qual as classes foram geradas, evitando assim problemas de incompatibilidade quanto ao armazenamento dos dados nos objetos.

Além do conceito “tradicional” de XML *Data Binding*, o modelo aqui apresentado permite também a definição das tabelas do repositório de dados, sendo que os métodos relacionados à conversão entre classes e tabelas (salvar/carregar) são também gerados automaticamente.

Estas classes podem então ser acopladas à aplicação que processa XML (Figura 15).

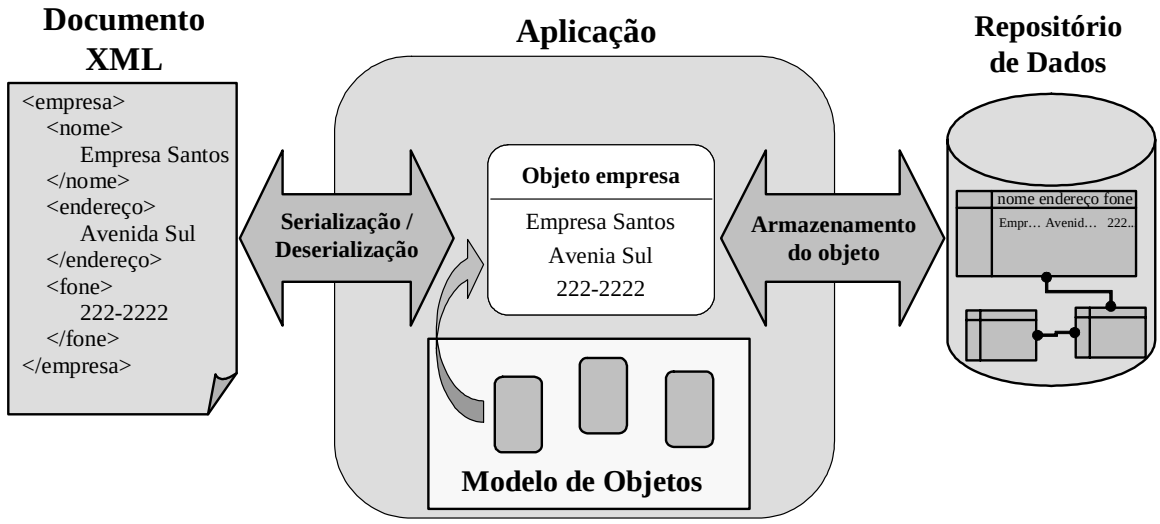

Figura 15: Relação entre XML, objetos de aplicação e repositório de dados.

Embora no contexto da plataforma da EV esta aplicação represente o *módulo de comunicação*, é interessante que do ponto de vista conceitual esta possa ser vista como uma aplicação genérica que processa documentos XML. Isto para mostrar que este modelo não está necessariamente vinculado ao contexto de EVs, sendo aplicado também em outros domínios.

Além disso, a implementação das classes geradas não trata dos aspectos de comunicação de nível mais baixo (a forma na qual um documento XML será transportado pela rede: via CORBA, *sockets*, Java RMI, HTTP, etc.), ou seja, os requisitos de cada aplicação em particular definirão qual o mecanismo mais adequado para o transporte dos documentos XML. A conseqüência disto é que cabe ao desenvolvedor implementar este aspecto. Por outro lado, há uma maior flexibilidade durante a implementação, pois o modelo de classes não está "amarrado" a nenhum mecanismo de transporte.

5.2.4 Geração de Classes a partir de um Esquema XML

Nesta seção será explicado em linhas gerais o processo de geração de classes a partir de uma especificação definida por um esquema XML (DTD ou XML *Schema*). Para cada elemento e atributo[21] declarado no esquema XML, o processo consiste em decidir se este

[21] O termo *atributo* pode denotar tanto um atributo de um elemento XML quanto um atributo de uma classe. Para evitar confusão entre os termos utilizados, quando for relativo à XML, será chamado apenas de *atributo*, e quando relativo a uma classe será chamado *atributo de classe*.

será uma classe ou atributo de classe. Para resolver isto, algumas regras básicas foram definidas:

1. Se um elemento é declarado contendo outros elementos (no XML *Schema*, o tipo do elemento é *complexType*) ele será mapeado como uma classe;
2. Se um elemento é declarado contendo um tipo simples ou com conteúdo vazio:
    a. Se possuir atributos ele será mapeado como uma classe;
    b. Se não possuir atributos ele será mapeado como um atributo da classe definida pelo elemento que o contém;
3. Se um elemento qualquer "x" mapeado como classe (regras 1 e 2.a) não for raiz, será criado um atributo na classe que foi definida pelo elemento "pai" do elemento "x". Este atributo de classe referencia a classe gerada pelo elemento "x", e corresponde a um relacionamento entre a classe "mãe" e a classe "x";
4. Um atributo é mapeado como um atributo da classe definida pelo elemento que o contém;
5. A cardinalidade de um atributo de classe é definida de acordo com a cardinalidade do elemento que o define. Se um elemento puder ocorrer mais de uma vez (operadores "*" ou "+" no DTD, ou o atributo *maxOccurs* > 1, no XML *Schema*), o atributo de classe será uma lista (*array*).

Vale lembrar que os tipos dos atributos de classe gerados a partir de um DTD são sempre texto (*string*), mas que podem ser modificados pelo *esquema de ligação*. Os atributos de classe gerados a partir de XML *Schemas* terão os mesmo tipos definidos para os elementos ou atributos, mas que podem também ser modificados pelo *esquema de ligação*.

Para explicar melhor estas regras, um exemplo simples será apresentado, conforme pode ser visto na Figura 16.

Em (A) há um DTD que define informações a respeito de uma empresa e seus produtos. Em (B) temos o código gerado através das regras anteriormente mencionadas. Os elementos "empresa" e "produto" correspondem a classes, conforme a *regra 1*. Conforme a *regra 2.b*, os elementos "nome", "tipo", "descrição" e "valor" são mapeados como atributos de classe, sendo os dois primeiros pertencentes à classe "Empresa" e os últimos à classe "Produto". O atributo "código" do elemento "produto" foi mapeado como um atributo da classe "Produto" (*regra 4*). Conforme a *regra 3*, foi criado um atributo

(listaProduto) na classe “Empresa” que faz referência à classe “Produto”, e cuja cardinalidade é definida pela *regra 5*.

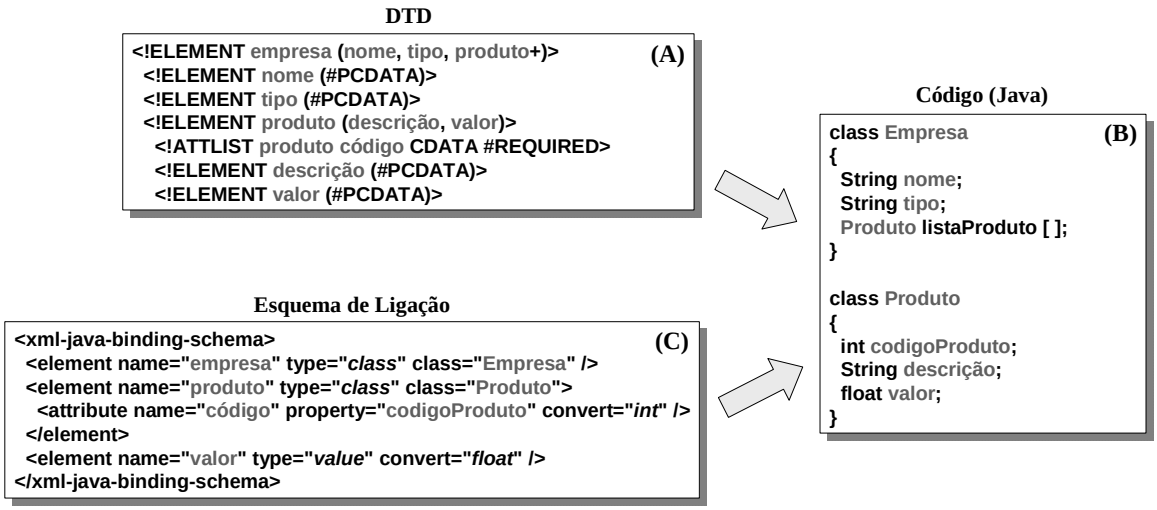

Figura 16: Processo de geração de classes a partir de um DTD.

Em (C) é apresentado um *esquema de ligação* simples, que se baseia no esquema de ligação descrito por MCLAUGHLIN (2002). Nele, são mencionados somente os elementos/atributos que precisam ser convertidos para um tipo de dados em particular (diferente de *string*), ou cuja classe/atributo de classe correspondente deva ter um nome diferente. As duas primeiras declarações indicam os nomes das respectivas classes dos elementos “empresa” e “produto”, e as duas últimas correspondem a conversões de tipos para o atributo “código” e para o elemento “valor”.

Por uma limitação de espaço e para tornar o exemplo simples de entender, foram omitidos vários aspectos que também são gerados automaticamente, tais como: métodos de acesso aos atributos (“get” e “set”), métodos para serialização/desserialização para XML e métodos para acesso às tabelas do repositório de dados. O processo de geração destas tabelas será detalhado na seção a seguir.

5.2.5 A Geração de Tabelas

A partir das classes geradas, é possível gerar as tabelas correspondentes no repositório de dados. As regras para a geração das tabelas são:

1. Para cada classe, é gerada uma tabela.
2. Para cada atributo de classe que seja de tipo simples:

   a. Se a cardinalidade for 1 é gerado um campo na tabela correspondente à sua classe.
   b. Se a cardinalidade for maior que 1 (ou seja, o atributo armazena uma lista de valores) é gerada uma tabela que contém o campo relativo a este atributo.
3. Através do *esquema de ligação* o usuário deve definir, para cada tabela, quais campos são chaves primárias. Caso a classe não possua nenhum atributo que possa ser chave primária, um campo será criado *automaticamente* para este papel, cujo valor será incremental.
4. Para cada relacionamento entre duas classes, será criada uma chave estrangeira na tabela referente à classe "filha" que estará vinculada à chave primária da tabela referente à classe "mãe". A chave estrangeira gerada também será definida como chave primária.

O processo de geração de tabelas é simples, visto que cada classe se torna uma tabela e que geralmente os atributos desta classe se tornam os campos da tabela (*regras 1* e *2.a*). Em casos onde a classe possui um atributo que é uma lista (*regra 2b*) é necessário criar uma tabela separada para armazenar os valores desta lista.

Em um modelo relacional, as tabelas precisam de chaves para manter a consistência de seus dados. Assim, é preciso definir quais campos correspondem a chaves primárias (*regra 3*).

Quando houver um relacionamento entre tabelas (*regra 4*), os campos que são as chaves estrangeiras podem ser definidos automaticamente a partir das chaves primárias definidas. Este processo é extremamente importante, visto que o resultado dele implicará na definição consistente das tabelas do repositório de dados.

É preciso destacar que embora o processo de geração de tabelas não seja completamente automatizado, pois *a priori* não é possível inferir de forma automática quais atributos de classe podem ser chaves primárias de uma tabela, a definição das chaves estrangeiras a partir das chaves primárias pode ser feita automaticamente. Além disso, no caso extremo de nenhuma das classes possuir algum atributo que possa ser usado como chave primária, e em conseqüência novas chaves serem geradas, este processo tende a ser completamente automático.

Por outro lado, outras abordagens poderiam ser adotadas. Por exemplo, ao invés de serem definidas automaticamente, as chaves estrangeiras poderiam ser definidas pelo

usuário. Entretanto, esta abordagem implica em um trabalho maior por parte do usuário, além de exigir uma verificação de eventuais erros que possam ser cometidos. Assim, optou-se por um mecanismo que procura minimizar a parte manual do processo, e conseqüentemente tornando-o mais ágil.

Um exemplo de geração de um modelo relacional a partir de um modelo de classes pode ser vista na Figura 17.

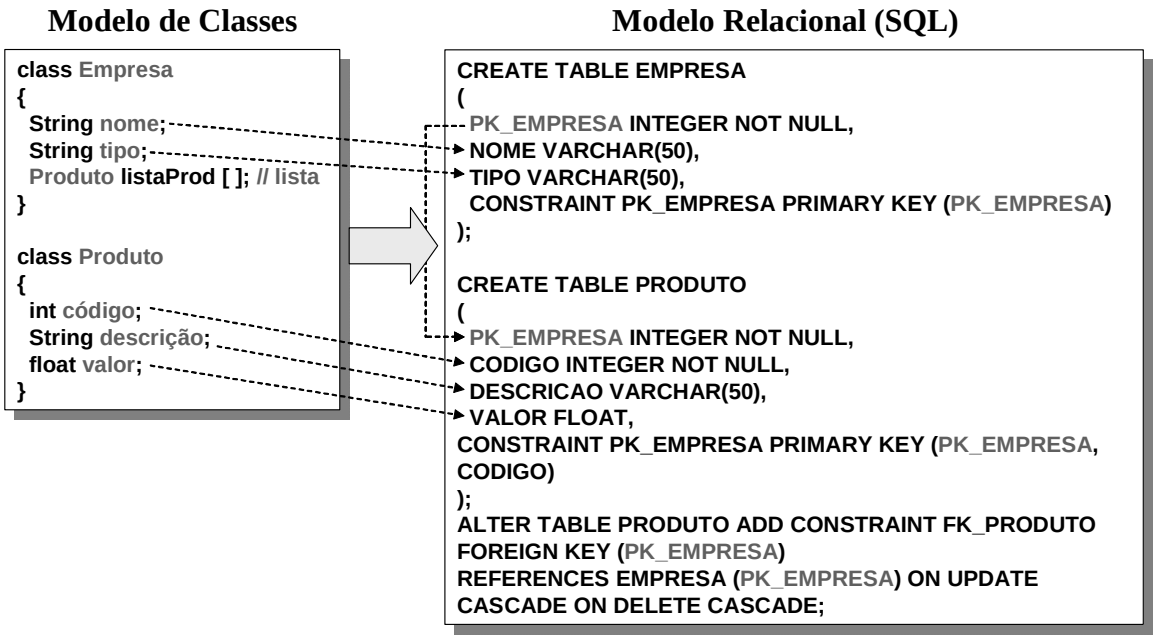

Figura 17: Relação entre o modelo de classes e o modelo relacional.

Seguindo a *regra 1*, as classes “Empresa” e “Produto” darão origem às tabelas “EMPRESA” e “PRODUTO” respectivamente. Os atributos destas classes que são de tipo simples darão origem a campos nas respectivas tabelas, conforme a *regra 2.a* (relacionados pelas setas pontilhadas). Como a classe “Empresa” não possui um atributo que possa ser usado como chave primária, o campo “PK_EMPRESA” é gerado automaticamente na tabela “EMPRESA”, sendo definido como chave primária desta tabela (*regra 3*). A classe “Produto” possui um atributo (“código”) que pode ser chave primária, sendo isto definido pelo usuário através do *esquema de ligação* (*regra 3*).

Como as classes estão relacionadas entre si, esta relação deve ser feita conforme a *regra 4*, ou seja, deve ser expressa através de uma chave estrangeira definida na tabela “PRODUTO” que está vinculada à chave primária da tabela “EMPRESA”. Assim, o campo “PK_EMPRESA” foi criado automaticamente na tabela “PRODUTO”, sendo

definido como uma chave estrangeira (*foreign key*) para o campo “PK_EMPRESA” da tabela “EMPRESA”. Além disso, este campo também é, junto com o campo “CODIGO”, chave primária da tabela “PRODUTO”, pois para referenciar unicamente um produto, são necessários o seu código e o código da empresa que o produz.

Adicionalmente ao processo de geração de tabelas, são gerados automaticamente os métodos para as classes carregarem e salvarem seu conteúdo nas respectivas tabelas.

5.3 Considerações sobre a Abordagem Apresentada

Uma consideração a respeito do modelo para configuração das informações a serem trocadas entre parceiros de EVs é que ele é um modelo semi-automático, ou seja, há uma etapa automática (a geração das classes e tabelas a partir da especificação XML), mas alguns detalhes devem inerentemente ser feitos manualmente, tais como a definição de nomes e tipos (quando necessária), e das chaves para as tabelas do repositório de dados.

Apesar disso, a implementação deste modelo, conforme será vista no próximo capítulo, visa reduzir o tempo de desenvolvimento de aplicações que processam XML, além de reduzir a quantidade de erros que possam ocorrer neste processo.

Embora o modelo tenha sido apresentado de maneira genérica com o intuito de mostrar a sua aplicabilidade em outros domínios, o módulo de comunicação e o repositório de dados da plataforma da EV são contemplados pelo modelo, ou seja:

- O *módulo de comunicação* pode ser desenvolvido usando as classes geradas a partir do MR. Todo o processamento de mensagens XML está embutido na implementação destas classes, exceto a definição do mecanismo de transporte das mensagens XML. Conforme será visto no Capítulo 7, as classes são geradas de modo a tornar simples a sua utilização seja qual for o mecanismo de transporte adotado.
- As tabelas do *repositório de dados* são geradas a partir do MR, com a vantagem de o seu acesso estar também embutido na implementação das classes geradas. Assim, uma conseqüência disto é que outros módulos da plataforma da EV também podem aproveitar as classes geradas para facilitar o acesso ao repositório de dados.

De acordo com o modelo geral proposto nesta dissertação, viu-se como uma plataforma para EVs pode ser definida/configurada de forma assistida pelo computador,

especificando-se as estruturas de informação a serem trocadas na EV. O próximo passo do modelo proposto, descrito no Capítulo 6, consiste em estabelecer a integração dos sistemas legados com as estruturas de informação especificadas.

5.4 Trabalhos Relacionados

Diversos trabalhos descrevem o modelo conceitual da *XML Data Binding*. Embora existam diferenças entre si, os conceitos apresentados são similares. Por exemplo, o conceito de *XML Data Binding* apresentado neste trabalho segue a estrutura apresentada por MCLAUGHLIN (2002).

Outros trabalhos serviram de fonte de pesquisa para a definição do modelo de geração de classes e tabelas. BOURRET (2001b) apresenta o mapeamento de um DTD para um modelo de objetos seguido do mapeamento para um modelo relacional. Em BOURRET (2001a) é apresentado o processo de maneira mais genérica, descrevendo o mapeamento feito também a partir de um XML *Schema*. Em SHANMUGASUNDARAN et al. (1999) são apresentadas técnicas para a geração de um esquema relacional a partir de um DTD.

Adicionalmente aos trabalhos que apresentam o modelo de forma conceitual, muitas ferramentas têm sido desenvolvidas, não apenas para *data binding*, mas para diversos tipos de aplicações. Existem inúmeras destas ferramentas disponíveis, implementadas para diversas plataformas e linguagens, e possuindo características próprias dependendo da forma como foram implementadas. Como neste trabalho propõe-se a implementação de uma ferramenta que utiliza o conceito de *XML data binding* (Capítulo 7), aqui serão citadas algumas ferramentas correlatas. Uma lista completa e atualizada pode ser encontrada em BOURRET (2003c).

Breeze XML Binder: ferramenta comercial que cria classes Java que encapsulam o processamento e validação da XML e que possuem métodos que mapeiam diretamente para elementos e atributos XML. Breeze permite que o usuário acesse elementos XML como propriedades e atributos de classes Java *Beans*. As classes podem ser geradas a partir de esquemas XML e relacionais. Possui uma interface gráfica para especificar tipos e modificar os nomes (BREEZE, 2003).

Castor: projeto de ferramenta de código aberto que gera classes a partir de XML *Schemas*. Inclui uma linguagem para mapeamento de classes Java para XML, bases de

dados e diretórios LDAP. A ferramenta de geração de código tem uma interface de linha de comando e uma interface de programação (CASTOR, 2003).

JAXB: é um mapeamento padrão desenvolvido pela Sun Microsystems (JAXB, 2003). O gerador de classes do JAXB gera classes e interfaces a partir de XML *Schemas*. Ele permite que o usuário controle como as classes podem ser geradas (nomes e propriedades das classes e se/como a validação deve ser feita) através de anotações no arquivo do XML *Schema*. Há planos de que no futuro isto seja feito em um documento separado (*esquema de ligação*). Objetos das classes geradas podem ser instanciados a partir de documentos XML na forma de um *input source*, URL, árvore DOM, ou eventos SAX. Durante este processo, os dados podem ou não ser validados. Objetos das classes geradas podem ser validados a qualquer hora, o que permite às aplicações validar o seu conteúdo antes de *serializá-los* como XML. Objetos podem ser *serializados* para um *output stream*, árvore DOM, ou como um conjunto de eventos SAX.

XGen: ferramenta de código aberto desenvolvida tendo como base o código do Castor, mas com um número de mudanças substanciais no mapeamento e funcionalidade. Ela consiste de um compilador de esquema e um módulo de tempo de execução. O compilador de esquema gera classes Java *Beans* a partir de um XML *Schema*. O modulo de tempo de execução converte documentos XML em objetos e vice-versa (XGEN, 2003).

Delphi/C++ Builder 6: ambientes de desenvolvimento desenvolvidos pela Borland (BORLAND, 2003). O *Data Binding Wizard* do Delphi/C++ Builder permite a geração de classes a partir de esquemas XML e documentos XML. Ele também gera uma classe "empacotadora" (*wrapper*) responsável por instanciar os objetos destas classes a partir de documentos XML e vice-versa.

.NET Framework: ambiente de desenvolvimento desenvolvido pela Microsoft (MICROSOFT, 2003). Em tempo de projeto, a *XML Schema Definition Tool* permite tanto a geração de um conjunto de classes C# ou Visual Basic a partir de um XML *Schema*, quanto a geração de um XML *Schema* a partir de um conjunto de classes. Em tempo de execução, a classe *XmlSerializer* permite a serialização de objetos para XML e a desserialização de objetos a partir de XML. É possível controlar quais propriedades são serializadas/desserializadas, além da forma que estão representadas em XML (como elementos, atributos, etc.). É possível também serializar/desserializar objetos em documentos SOAP.

5.4.1 Considerações sobre as Ferramentas Relacionadas

Apesar da variedade de ferramentas disponíveis, cada uma apresenta características específicas, que podem ser vantajosas ou não, dependendo do tipo de aplicação. Por exemplo, algumas trabalham com formatos específicos como objetos Java *Beans*, ou documentos SOAP. Em outras, o esquema de ligação é definido por *anotações* no próprio esquema XML. Alguns permitem a geração das classes a partir do modelo da base de dados (uma abordagem inversa – *bottom-up*). Algumas são gratuitas, outras não.

Assim, embora a utilização de uma ou mais destas ferramentas possa se enquadrar no cenário proposto, neste trabalho optou-se por desenvolver uma ferramenta que se enquadra completamente no cenário apresentado, ou seja, uma ferramenta que faça a geração semi-automática de código para classes e tabelas de um repositório de dados, e que também seja genérica e flexível para que possa ser aplicada em outros casos.

Por existir um número considerável de ferramentas disponíveis e a tecnologia sendo relativamente nova, a tarefa de se avaliar as características suportadas por cada uma nem sempre é fácil. HEDIN (2003) compara algumas ferramentas de *XML Data Binding* para Java, enumerando alguns aspectos que servem como critério de comparação, tais como:

- **Geração de Código:** verifica a conformidade com o XML *Schema* definido pela W3C e verifica se o código é gerado sem erros.
- **Teste em tempo de execução:** converte um documento XML em objetos, seguido da conversão dos mesmos objetos de volta para XML.
- **Comparação das características:** aspectos mais gerais, que podem ser: se a ferramenta possui o código aberto (*open source*); custo para aquisição; interface gráfica ou linha de comando; tratamento de *namespaces*; validação do documento XML; validação dos valores para a serialização; otimização do código;

Conforme será visto no Capítulo 7, a ferramenta implementada neste trabalho possui alguns pontos positivos, como interface gráfica, validação dos dados para a serialização (assim como do próprio documento), e possui código aberto. Como limitação, não trabalha com *namespaces* e não está em completa conformidade com o XML *Schema* da W3C. Embora este último seja um critério muito importante, esta limitação pode ser observada (em diferentes níveis) por muitas ferramentas, tal como apontado por HEDIN (2003).

6 Integração de Dados em EV: Modelo Conceitual

6.1 Introdução

A instalação da plataforma da EV consiste na implantação e configuração de todo o arcabouço computacional necessário à participação de uma empresa em EVs. Este processo pode envolver investimentos de tempo e dinheiro consideráveis, sendo uma tarefa essencial para a participação da empresa na EV. Dentre as atividades realizadas nesta etapa, este trabalho focará na *integração com os sistemas legados*, tarefa que representa um requisito obrigatório para qualquer infraestrutura de suporte a EVs (CAMARINHA-MATOS *et al.*, 1999b).

Conforme visto na arquitetura da plataforma da EV, é necessário que haja a integração dos sistemas legados dos participantes com a plataforma da EV para que seja possível cooperarem entre si. Esta integração ocorre no nível de dados, ou seja, os parceiros integram o modelo de dados utilizado nos seus respectivos sistemas legados com o modelo de referência adotado pela EV, permitindo que os seus dados legados sejam intercambiados entre os membros da EV.

Há que se salientar que a abordagem de sistemas legados é uma entre outras possíveis em teoria. No entanto, especialmente por questões econômicas, essa abordagem tem vindo a ser largamente aplicada uma vez que visa manter os sistemas ora existentes em uma empresa, apenas criando-se interfaces destes para com sistemas de informação (centralizados ou distribuídos) corporativos (RABELO, 2003). Ou seja, esta abordagem hoje faz uso de bases de dados alimentadas pelos sistemas CIM[22] e/ou ERP[23] de cada empresa, e a partir deste nível de integração interna são estabelecidas as interfaces de integração (*wrappers*) com os módulos da plataforma para EVs.

Além disso, a integração deve ocorrer no nível de processos, ou seja, é preciso configurar as atividades que cada parceiro irá executar (chamadas de *business process*), cujas execuções são devidamente coordenadas para que o objetivo global da EV seja alcançado (CAMARINHA-MATOS *et al.*, 1999c). As abordagens mais populares para este fim são os *sistemas de workflow* e os *sistemas multiagente*.

[22] *Computer Integrated Manufacturing*
[23] *Enterprise Resource Planning*

6.1.1 Delimitação do Escopo

Embora a integração tanto de dados quanto de processos compreendam tarefas essenciais quando da instalação da plataforma da EV em uma empresa, os detalhes referentes à integração de processos estão fora do escopo desta dissertação. Assim, este trabalho focará apenas a questão da integração dos modelos de dados legados de cada parceiro com o MR adotado pela EV. Tal como foi visto no capítulo anterior, este MR serve de base para a definição de parte da plataforma da EV.

Em relação à integração de dados, primeiramente é preciso definir a abordagem que será adotada neste trabalho. Conforme KONTIO (2003), há duas abordagens básicas para a integração orientada a dados, que são:

- **Base de Dados Distribuída/Federada:** consiste na criação de uma base de dados "virtual" logicamente centralizada que abstrai o acesso às bases de dados dos parceiros, tal como descrito por AFSARMANESH et al. (1999). O acesso às informações é feito por consultas (*queries*) federadas, usualmente descritas em formato proprietário. Nesta abordagem, não há duplicação de informações e a integridade global é intrinsecamente garantida. É uma abordagem que vai mais na direção de modelos de referência, onde toda a EV partilha um único modelo de BD.
- **Entre bases de dados (*database-to-database*):** consiste no compartilhamento de dados através de *middlewares* ou da replicação da base de dados. Cada parceiro mantém sua BD legada e a integração é feita de forma lógica, materializada por esquemas de mais alto nível, no caso, via XML.

Este trabalho utilizará a segunda abordagem (entre bases de dados), que embora seja menos complexa que a utilização de bases de dados federadas, é também eficiente. Além disso, é francamente a mais usada nas empresas, pois é mais rápida e fácil. Falando mais especificamente, a abordagem neste trabalho utilizará o conceito de *middleware*.

Middleware é um termo usado com freqüência na área de sistemas distribuídos, onde corresponde à camada de *software* que provê serviços às aplicações, tornando o acesso à infraestrutura de comunicação (rede) o mais abstrato possível. No contexto das aplicações baseadas em XML, o termo *middleware* XML possui um significado diferente, correspondendo ao *software* usado por aplicações para transferir dados entre documentos XML e bases de dados (BOURRET, 2003d). Em outras palavras, *middleware* XML é o

software responsável pelo mapeamento entre os dados contidos em documentos XML para dados correspondentes em uma base de dados (seção 3.4.7.2).

Considerando a arquitetura conceitual da plataforma da EV, o papel do *middleware* XML caberá ao *módulo de integração,* que será o responsável por mapear os dados dos repositórios legados para o MR adotado pela EV. Os detalhes desta abordagem são explicados a seguir.

Convém aqui mencionar que o modelo de integração a ser apresentado está relacionado ao conceito de *Enterprise Application Integration* (EAI). EAI consiste no uso de *software* para permitir a comunicação entre aplicações, tanto dentro de uma empresa quanto entre múltiplas empresas. EAI está relacionada a tecnologias de *middleware* tal como *middlewares* orientados a mensagens (MOM) e tecnologias de representação de dados como a XML, além de estar relacionada também às antigas tecnologias de EDI (WIKIPEDIA, 2004a).

6.2 Modelo Conceitual

Para que o módulo de integração possa realizar a tarefa da mapear diferentes modelos de dados é necessária uma configuração inicial, o que é feito na etapa de instalação da plataforma. Além disso, deve-se considerar a abordagem (*top-down*) que foi definida inicialmente, que consiste na definição da plataforma a partir de um MR comum para os parceiros da EV. Isto significa que o MR adotado por uma EV (e que auxiliou na definição das estruturas de dados da sua plataforma) será o parâmetro para o mapeamento com os modelos de dados de cada parceiro. Este detalhe é importante, pois outras políticas podem ser adotadas. Por exemplo, uma outra abordagem poderia definir que os mapeamentos sejam feitos entre cada par de parceiros. Portanto, a idéia aqui é que cada empresa faça o mapeamento do modelo de dados dos seus sistemas legados para o MR adotado pela EV, conforme a Figura 18.

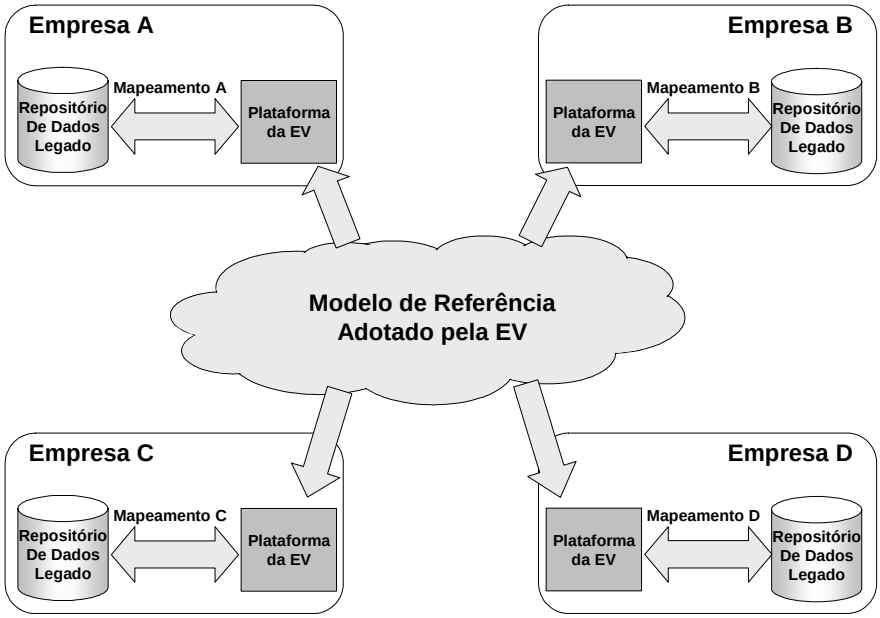

Figura 18: Mapeamento dos modelos de dados legados em função do MR.

Assim, para cada empresa que instale a plataforma, um mapeamento é feito. Seguindo esta abordagem, o intercâmbio de informações entre os membros da EV será sempre feito em função do MR adotado (envio de mensagens codificadas em XML), independente dos modelos adotados em cada sistema legado. A Figura 19 abaixo ilustra o modelo geral do processo de integração.

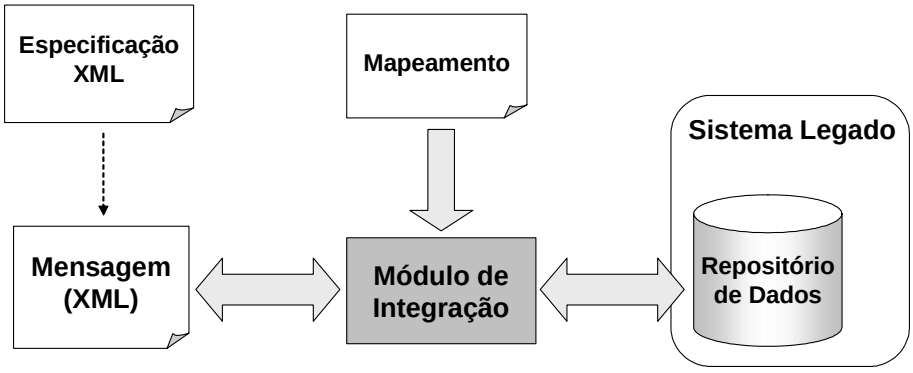

Figura 19: Processo de integração.

O modelo apresentado na Figura 18 e detalhado na Figura 19 mostra o módulo de integração atuando durante o ciclo de vida de EV, ou seja, uma vez definidos os mapeamentos (na etapa de instalação), os mesmos podem ser utilizados pelo módulo de integração nas demais etapas do ciclo de vida para o acesso aos repositórios legados

(Figura 19). A definição dos mapeamentos, que corresponde à configuração do módulo de integração, será apresentada a seguir.

6.2.1 A Definição dos Mapeamentos

Como neste trabalho o MR é definido em XML, temos novamente uma ou mais especificações XML como parâmetro para a definição dos mapeamentos. Além disso, é necessário obter o modelo de dados do repositório de dados legado. O processo de definição dos mapeamentos é ilustrado na Figura 20 e consiste nos seguintes passos:

1) O módulo de integração lê a especificação XML. Embora conceitualmente seja mostrada apenas uma especificação XML, na prática um MR pode ser definido por mais de uma. Se este for o caso, um mapeamento deve ser feito para cada especificação XML.
2) O módulo de integração lê o modelo de dados do repositório de dados legado.
3) O usuário responsável pela configuração, com o auxílio do módulo de integração, faz o mapeamento entre os campos definidos na especificação XML e os campos do repositório de dados legado.

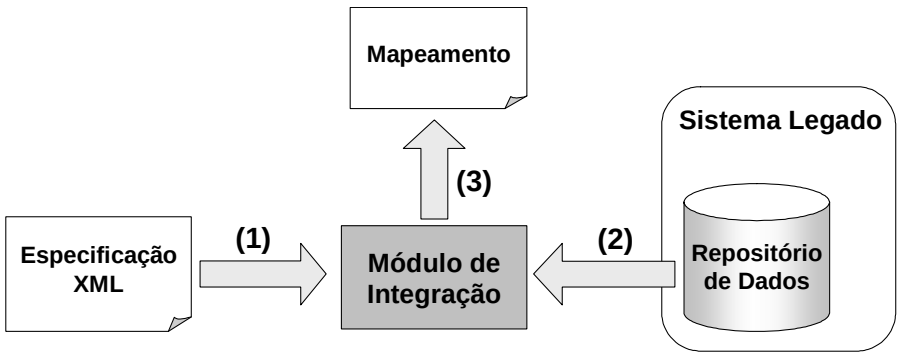

Figura 20: Processo de definição dos mapeamentos.

Enquanto que os dois primeiros passos são feitos automaticamente, o terceiro (e mais importante) é feito manualmente. Isto porque cada repositório de dados tem um modelo de dados próprio (com uma estrutura e semântica particular), sendo que o indivíduo responsável pelo mapeamento deve ser alguém que conheça em detalhes o modelo de dados em questão. Embora o processo continue sendo trabalhoso e demorado, o uso do módulo de integração visa minimizar este custo, pois além de garantir a consistência do mapeamento entre os dois modelos (conforme será visto a seguir), este é definido

dinamicamente, ou seja, a cada novo mapeamento definido não há a necessidade de recompilar o código da plataforma.

Depois de previamente configurado (definição dos mapeamentos), o módulo de integração está apto para fazer a integração dos dados legados. Dessa forma, o módulo de integração processa (e valida) a mensagem XML e, a partir do mapeamento, gera dinamicamente as funções (SQL) para incluir as informações no repositório de dados legado. De modo inverso, o módulo de integração pode recuperar informações do repositório de dados e gerar uma mensagem em formato XML.

6.2.2 O Mapeamento

Nesta seção será explicado em maiores detalhes como é feito um mapeamento. Embora esta seja uma tarefa essencialmente humana (pois requer um conhecimento prévio da semântica do modelo de dados), o módulo de integração resolve de forma automática alguns aspectos relacionados à consistência do mapeamento.

O primeiro aspecto está relacionado à restrição do tipo de mapeamento que será realizado em cada elemento/atributo XML. Conforme foi visto na seção 5.2.4, é possível aplicar um conjunto de regras para mapear automaticamente uma especificação XML em um modelo de classes correspondente. Embora isto não seja feito neste caso, percebe-se que é possível determinar previamente se um elemento XML deve ser mapeado como uma classe (neste caso, uma tabela) ou como um atributo de classe (neste caso, um campo de uma tabela). Em outras palavras, o módulo de integração evita que um elemento que deva ser mapeado como uma tabela seja mapeado como um campo, e vice-versa.

Outra restrição, que é conseqüência da anterior, ocorre quando o usuário define um mapeamento para tabela. Neste caso, os elementos “filhos” do elemento mapeado (somente os que deverão ser mapeados como campos) só poderão ser mapeados para campos da tabela mapeada.

O outro aspecto é a garantia da compatibilidade entre os tipos de dados. Ou seja, o módulo de integração só permite mapeamentos entre campos de tipos compatíveis. No caso da especificação XML ser um DTD, os elementos que devem ser mapeados como campos serão do tipo texto (*string*), que pode ser mapeado para os outros tipos. Já com um XML *Schema* o módulo de integração verifica se o tipo do elemento é compatível com o campo da tabela, garantindo a consistência em termos do tipo de dados.

A estrutura do mapeamento é análoga à estrutura da especificação XML, ou seja, para cada elemento definido na especificação há um mapeamento (para campo ou tabela) correspondente. Além disso, a relação entre os elementos é mantida no mapeamento, ou seja, se o elemento A é filho do elemento B, o mapeamento A também será filho do mapeamento B. A Figura 21 a seguir ilustra um exemplo de DTD e o mapeamento correspondente.

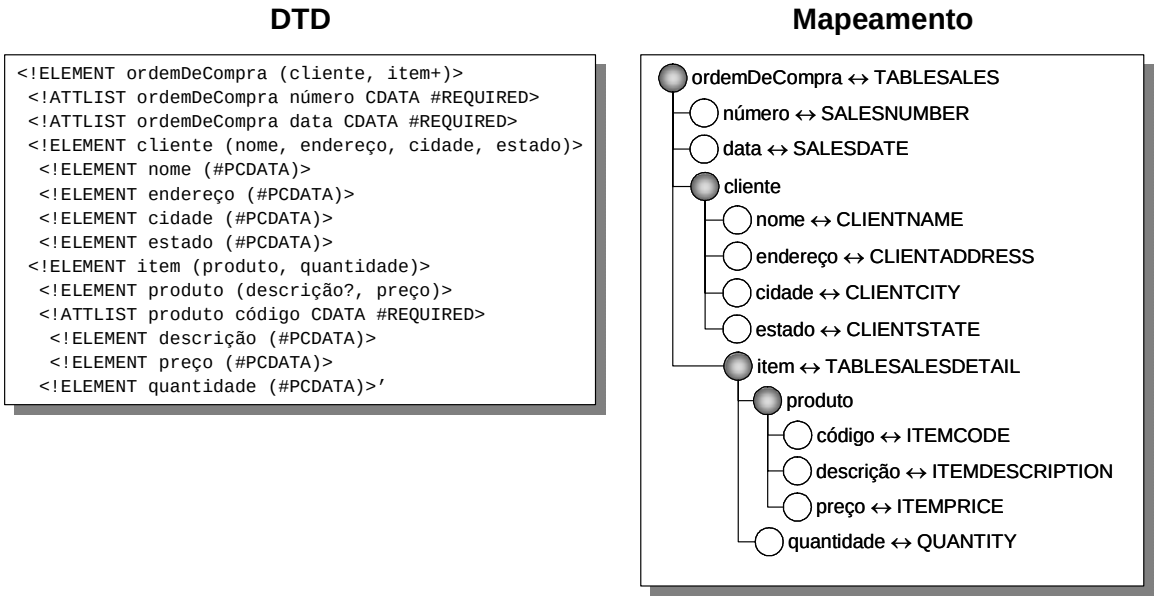

Figura 21: DTD da ordem de compra com o mapeamento correspondente.

Na Figura 21 os *mapeamentos para tabela* estão destacados em cinza, e os *mapeamentos para campo* estão em branco. A partir destas estruturas (que são detalhas a seguir), o módulo de integração pode então armazenar adequadamente os valores codificados em XML para o repositório de dados e vice-versa.

6.2.2.1 Mapeamento para Tabela

Indica um mapeamento para uma tabela do repositório de dados. Contém os seguintes metadados:

- Nome do elemento XML.
- Nome da tabela relacionada.

- Chave primária incremental (opcional). Caso não exista um elemento/atributo XML que possa ser mapeado como chave primária, é possível indicar uma das chaves primárias (da tabela relacionada) para que esta receba um valor incremental.
- Chaves estrangeiras. O mapeamento para tabela possui uma lista de estruturas que armazenam o nome da chave estrangeira, bem como o nome da tabela e do campo relacionados. Estes metadados são utilizados para que o módulo de integração possa resolver os detalhes relativos à consistência dos dados do repositório legado.

Adicionalmente, um mapeamento para tabela contém uma lista de *mapeamentos para campo* e *mapeamentos para tabela* (relativos aos elementos filhos).

Há casos onde um elemento deve ser ignorado por não haver uma tabela correspondente que possa ser mapeada. Entretanto, seus elementos filhos podem ser mapeados, e neste caso serão mapeados para campos da tabela que foi mapeada pelo elemento superior ao elemento ignorado. A Figura 22 ilustra um fragmento do mapeamento da Figura 21 destacando um elemento ignorado.

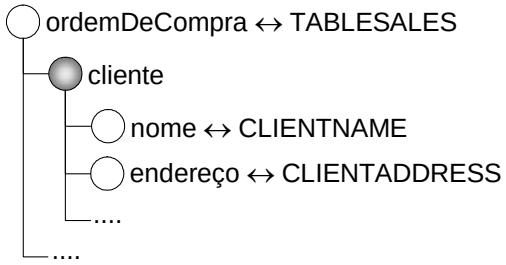

Figura 22: Fragmento do mapeamento destacando um elemento ignorado.

Na situação representada acima, o repositório de dados legado possui uma tabela relacionada ao elemento “ordemDeCompra” (TABLESALES). Entretanto, não há um equivalente para o elemento “cliente”. Assim, este pode ser ignorado, mas o seu nome, endereço, etc., podem ser mapeados para os campos da tabela “TABLESALES”. Este recurso provê uma maior flexibilidade para mapeamentos entre diferentes modelos de dados.

6.2.2.2 Mapeamento para Campo

Indica um mapeamento para campo de uma tabela. Contém os metadados:

- Nome do elemento/atributo XML.

- Tipo do elemento/atributo XML.
- Nome do campo (da tabela mapeada pelo elemento superior).
- *Flag* que indica se é um atributo ou elemento.
- *Flag* que indica se o campo (da tabela) é chave primária.
- Identificador único para este mapeamento.

É importante destacar que existe um *mapeamento para campo* para todos os elementos que devam ser mapeados como tal. Caso um elemento não possua um campo com que possa se relacionar, o mapeamento conterá apenas o nome do elemento. Isto é feito desta maneira para que o mapeamento mantenha a informação completa sobre todos os elementos, e assim permitindo que o módulo de integração possa gerar um documento XML válido.

6.2.3 O Processo de Integração

Depois de configurado, ou seja, com os devidos mapeamentos definidos, o módulo de integração está apto a acessar os dados do repositório legado quando for necessário. É importante mencionar novamente que os aspectos de coordenação e comunicação não serão abordados. Entretanto, o modelo a ser detalhado ilustra o acesso ao repositório de dados legado de maneira independente destes aspectos.

O acesso ao repositório legado pode ocorrer de duas formas:

1. Dados definidos em função do MR adotado pela EV (documento XML) são armazenados no repositório de dados legado.
2. Dados são carregados do repositório legado e são convertidos para o “formato” definido pelo MR adotado pela EV.

6.2.3.1 Armazenamento dos Dados

Para armazenar os dados no repositório legado, o módulo de integração recebe um documento XML (definido conforme o MR adotado pela EV) e, com o auxílio do mapeamento adequado, gera os comandos SQL apropriados (“Insert” ou “Update”) para executar esta tarefa (Figura 23).

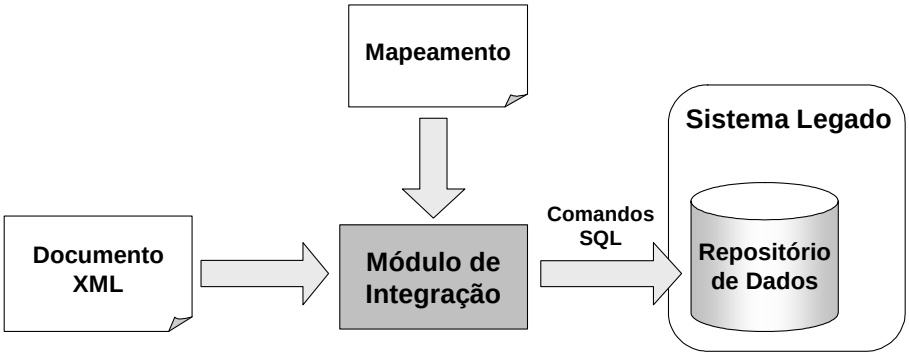

Figura 23: Armazenamento dos dados no repositório legado.

O processo de armazenamento consiste nos seguintes passos:

1. O documento XML é validado conforme a sua especificação;
2. O módulo de integração procura o mapeamento relativo à especificação deste documento;
3. O módulo de integração monta uma lista de "comandos abstratos". Um comando abstrato consiste em uma estrutura de dados que armazena informações necessárias para a posterior geração do comando SQL apropriado, tais como o nome da tabela, a lista de campos e seus respectivos valores. Isto é feito a partir do processamento da estrutura do documento que, guiado pelo mapeamento, avalia os seguintes casos:
    a. Se o elemento foi mapeado como tabela, é criado um comando abstrato (associado a esta tabela) que é adicionado à lista. Caso o mapeamento para esta tabela indique uma chave primária incremental, é gerado um valor baseado no valor máximo que este campo possui no repositório de dados, e ambos (nome da chave primária e seu valor) são adicionados ao comando. Finalmente, cada chave estrangeira que não tenha sido mapeada como campo será adicionada ao comando, cujo valor é obtido do campo respectivo contido no comando abstrato referente à tabela "estrangeira";
    b. Se o elemento foi mapeado como campo, o nome do campo (da tabela) e o valor contido no elemento são adicionados ao comando abstrato atual (correspondente à tabela que contém este campo);
    c. Elementos definidos como "ignorados" serão, naturalmente, ignorados. Entretanto, seus elementos filhos serão mapeados (se assim indicados) como campos da tabela mapeada pelo elemento superior a este.

4. Para cada comando abstrato da lista gerada no passo 3 será gerado um comando SQL para consulta (SELECT) que é então enviado ao repositório de dados. Caso a consulta retorne algum valor, é gerado um comando de atualização (UPDATE), caso contrário é gerado um comando de inserção (INSERT) para inserir os novos registros no repositório de dados.

Um exemplo para ilustrar o processo de geração dos comandos SQL para armazenamento dos dados pode ser visto na Figura 24.

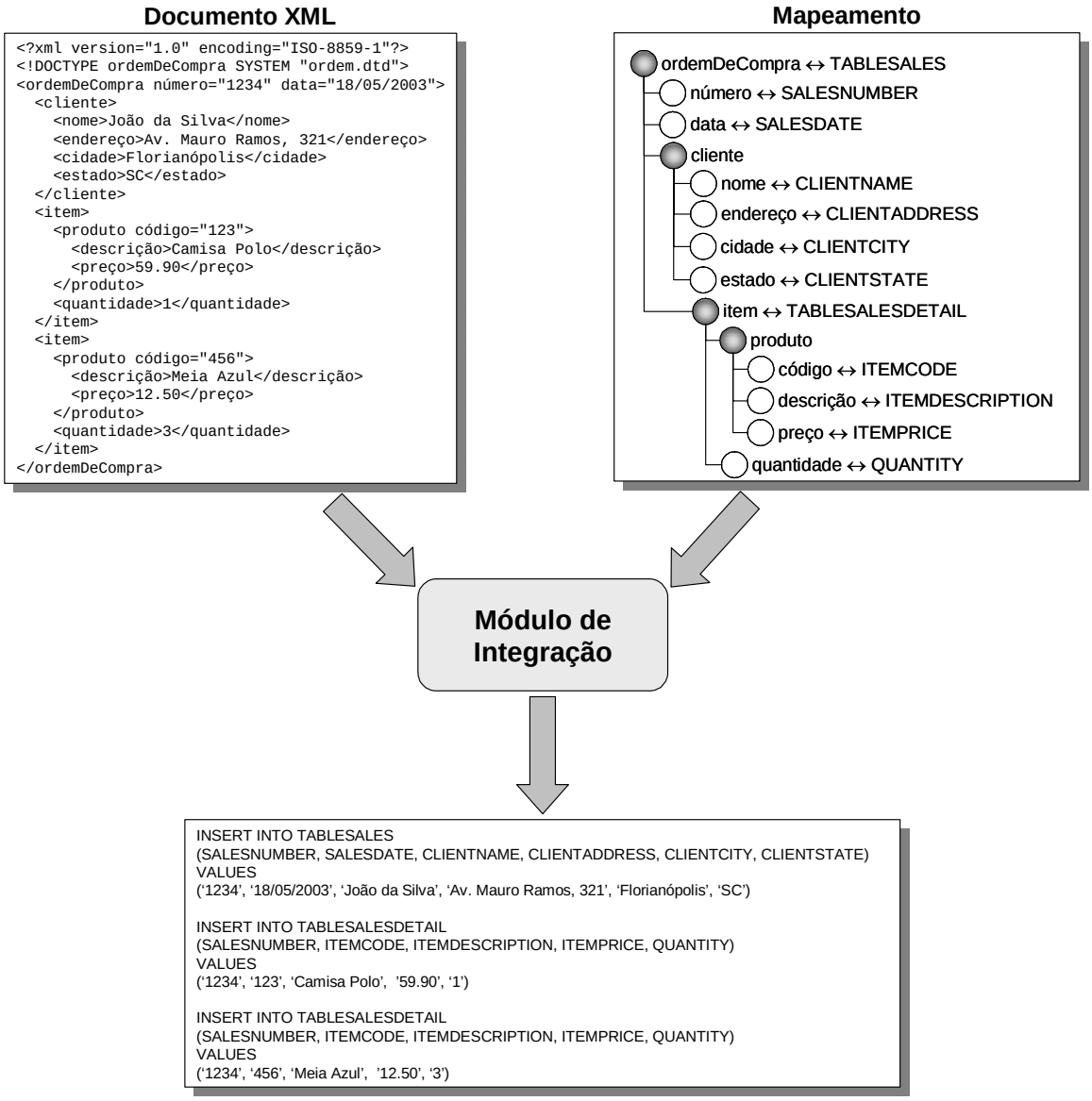

Figura 24: Geração dos comandos SQL para armazenamento dos dados.

Neste exemplo, considera-se que os dados do documento XML não estão armazenados no repositório e, portanto, são gerados comandos para inserção de dados. Repare que a tabela *TABLESALESDETAIL* (mapeada para o elemento *item*), possui um campo (chave primária) chamado *SALESNUMBER*, que referencia o número da ordem à qual este item pertence. Este campo não faz parte do mapeamento, mas como ele é também chave estrangeira, o módulo de integração se encarregou de obter o seu valor do comando abstrato referente à tabela "estrangeira", que neste caso é a tabela *TABLESALES*.

6.2.3.2 Carregamento dos Dados

O carregamento de dados do repositório legado consiste em uma consulta feita ao módulo de integração, que gerará uma nova consulta ao repositório de dados legado, e tendo como resultado um documento XML (válido perante o MR adotado pela EV). Os passos gerais deste processo são enumerados a seguir, e ilustrados na Figura 25:

A. O módulo de integração recebe uma consulta definida em termos do MR da EV (1), e com o mapeamento apropriado (2) gera a consulta SQL relacionada ao modelo de dados do repositório legado;
B. A consulta SQL é efetuada (3) no repositório de dados legado, cujo resultado (4) é recebido pelo módulo de integração;
C. O módulo de integração gera o documento XML convertendo o resultado da consulta com o auxílio do mapeamento (5 e 6).

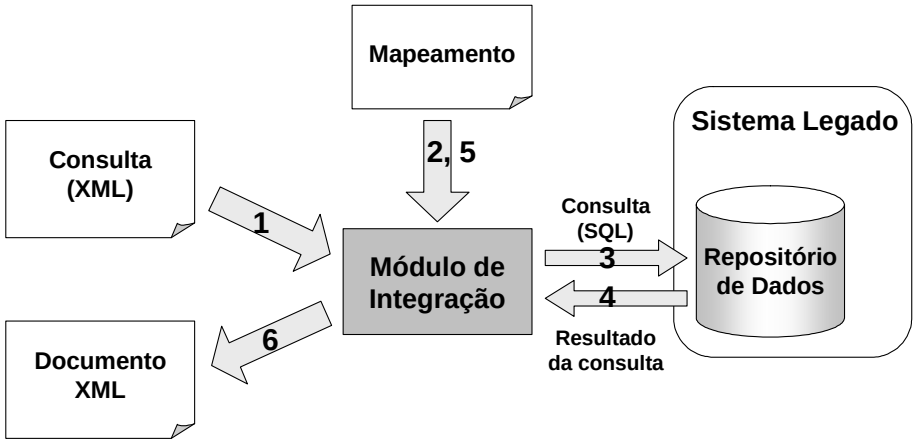

Figura 25: Carregamento dos dados do repositório legado.

Destes três passos gerais, somente os passos A e C serão detalhados, pois abrangem justamente o processo de conversão (mapeamento) entre o modelo de referência da EV com o modelo de dados legado.

Conversão de uma Consulta XML para uma Consulta SQL

A consulta recebida pelo módulo de integração é feita em função do MR adotado pela EV. Como este é definido em XML, a consulta está vinculada a elementos e atributos XML. Isto garante a transparência no acesso aos dados, pois a consulta é feita em termos do MR, ou seja, quem solicita a consulta não precisa conhecer os detalhes relacionados ao modelo de dados de um repositório legado em particular. Cabe então ao módulo de integração a responsabilidade de gerar uma consulta compatível com o modelo de dados legado, e retornar o seu resultado no formato definido pelo MR adotado pela EV.

Para este trabalho, foi especificada uma linguagem (baseada em XML) utilizada para definir consultas conforme o MR adotado. A justificativa para a definição de uma linguagem específica ao invés da adoção de uma das já existentes para consulta em XML – tal como a XQuery (W3C, 2003b) –, é que estas linguagens definem consultas visando o acesso aos dados contidos em documentos XML, o que não corresponde ao cenário apresentado. Como o objetivo aqui é o acesso aos dados contidos em um repositório de dados qualquer, as construções destas linguagens (utilizadas para o acesso a documentos XML) tornam-se pouco adequadas. Portanto, a abordagem adotada neste trabalho consiste no acesso a um repositório de dados (também com consultas), mas cujo modelo de dados é abstraído por um MR definido em XML.

A estrutura da linguagem de consulta XML pode ser vista na Figura 26 e é explicada a seguir.

- **select:** este é o elemento raiz da consulta XML;
- **element:** indica o nome do elemento (definido pelo MR) cujos conteúdo deve ser consultado;
- **from:** indica o nome da especificação XML que define o elemento indicado por *element*;
- **where:** contém uma lista de condições, que podem ser agrupadas por elementos *and* (operação lógica “e”) e/ou elementos *or* (operação lógica “ou”);
- **condition:** contém uma expressão condicional que compara dois parâmetros (igualdade, desigualdade, maior-quê e menor-quê).

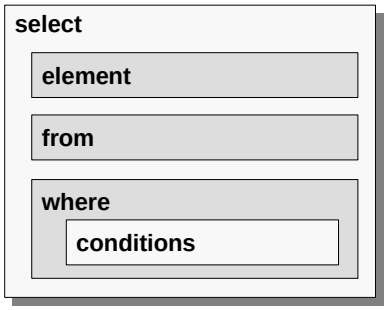

Figura 26: Estrutura da consulta XML.

O DTD que especifica esta linguagem encontra-se no **Anexo A**. A Figura 27 apresenta um exemplo de como é na prática uma consulta XML. Para facilitar o entendimento do processo de integração como um todo, este exemplo está relacionado ao DTD e mapeamento já apresentados nos exemplos anteriores.

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

<!DOCTYPE select SYSTEM "XMLQuery.dtd">

<select>
  <element>ordemDeCompra</element>
  <from>ordem.dtd</from>
  <where>
    <condition op1="ordemDeCompra/@número" operator="=" op2="1234"/>
  </where>
</select>
```

Figura 27: Consulta XML.

Conforme a estrutura já apresentada, esta consulta solicita obter a ordem de compra (elemento *ordemDeCompra*) definida pelo DTD *ordem.dtd*, cuja condição seja que o seu *número* seja igual a *1234*. Uma observação importante diz respeito à forma de referenciar elementos XML. Embora esta linguagem seja específica (não padronizada), os elementos contidos nas condições são referenciados usando a linguagem *XPath*.

XPath é também um padrão definido pelo W3C (1999b), e consiste em uma linguagem utilizada para referenciar partes de um documento XML. A estrutura se assemelha a caminho de um arquivo dentro de uma árvore de diretórios. Assim, neste exemplo temos o *xpath "ordemDeCompra/@número"* que referencia o atributo *número* (indicado pelo "@") pertencente ao elemento *ordemDeCompra*.

Antes de apresentar o algoritmo para conversão da consulta XML para a consulta SQL, será apresentada a estrutura geral de uma consulta SQL (Figura 28).

```
SELECT
     (lista de campos)
FROM
     (lista de tabelas)
WHERE
     (lista de condições)
```

Figura 28: Estrutura de uma consulta SQL.

Percebe-se que há algumas diferenças entre os dois tipos de consultas. A principal é que uma consulta SQL é mais flexível do que a consulta XML definida neste trabalho. Isto porque, como a consulta é feita para obter uma lista de campos que podem ser de diferentes tabelas, é possível fazer consultas mais elaboradas e obter um conjunto de dados bastante complexo. Entretanto, esta flexibilidade não é necessária neste trabalho, pois o resultado de uma consulta deve ter os dados estruturados conforme definido pelo MR adotado pela EV.

O processo de conversão de uma consulta XML para uma consulta SQL é simples, consistindo nos seguintes passos:

1. O mapeamento referente à especificação XML (indicada pelo elemento *from*) é obtido dentre os mapeamentos já definidos.
2. O *mapeamento para tabela* referente ao elemento indicado por *element* é localizado dentro do mapeamento obtido no passo 1.
3. Para cada *campo mapeado* neste *mapeamento para tabela*, bem como nos *mapeamentos para tabela* "filhos", é adicionado um campo à *lista de campos* da consulta SQL.
4. Todas as tabelas cujos filhos estão na *lista de campos* são adicionadas à *lista de tabelas* da consulta SQL.
5. Cada condição definida na consulta XML é mapeada para as tabelas e campos correspondentes e adicionada à *lista de condições* da consulta SQL.
6. Para cada chave estrangeira das tabelas pertencentes à *lista de tabelas* é adicionada uma nova condição à *lista de condições*. Isto é fundamental para restringir adequadamente o conjunto retornado pela consulta.

Para ilustrar a conversão entre os dois tipos de consultas, a consulta XML apresentada na Figura 27 foi convertida para a consulta SQL apresentada na Figura 29.

```
SELECT
  TABLESALES.SALENUMBER AS f1,
  TABLESALES.SALESDATE AS f2,
  TABLESALES.CLIENTNAME AS f3,
  TABLESALES.CLIENTADDRESS AS f4,
  TABLESALES.CLIENTCITY AS f5,
  TABLESALES.CLIENTSTATE AS f6,
  TABLESALESDETAIL.ITEMCODE AS f7,
  TABLESALESDETAIL.ITEMDESCRIPTION AS f8,
  TABLESALESDETAIL.ITEMPRICE AS f9,
  TABLESALESDETAIL.QUANTITY AS f10
FROM
  TABLESALES,
  TABLESALESDETAIL
WHERE
  TABLESALES.SALESNUMBER = '1234' AND
  TABLESALESDETAIL.SALESNUMBER = TABLESALES.SALESNUMBER
```

Figura 29: Consulta SQL gerada a partir da consulta XML.

Como foi dito anteriormente, um dos atributos que um *mapeamento para campo* possui é um identificador único. Assim, além da consulta conter todos os campos que foram mapeados, estes são referenciados através destes identificadores (*f1*, *f2*, etc.). Isto é feito para evitar ambigüidades no momento de extrair os dados do resultado da consulta. Isto é bastante comum visto que campos de tabelas diferentes podem ter nomes iguais (por exemplo, *número*, *código*, *data*, etc.).

Outra observação sobre este exemplo relaciona-se à última condição, que não faz parte da consulta XML. Conforme foi dito, a vantagem de se adotar uma linguagem de consulta que é independente do modelo de dados legado é a total transparência no que diz respeito aos detalhes de sua implementação. Assim, além das condições explícitas na consulta XML (neste caso, número da ordem igual a ‘1234’), há outras condições implícitas que devem ser adicionas à consulta SQL, referentes às restrições de chave estrangeira (neste caso, número da ordem da tabela referente ao item de compra deve ser igual ao número da ordem na tabela referente à ordem de compra).

Conversão do Resultado da Consulta SQL para Documentos XML

O resultado de uma consulta SQL, também chamado de "*result set*", consiste em um conjunto de dados que pode ser representado como uma tabela contendo todos os campos que foram requisitados na consulta. A Tabela 3 abaixo mostra o resultado da consulta apresentada na seção anterior.

f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
1234	18/05/2003	João da Silva	Av. Mauro Ramos, 321	Florianópolis	SC	123	Camisa Polo	59.90	1
1234	18/05/2003	João da Silva	Av. Mauro Ramos, 321	Florianópolis	SC	456	Meia Azul	12.50	3

Tabela 3: Resultado da consulta SQL.

O primeiro detalhe a ser observado no resultado desta consulta são os nomes dos campos, que correspondem aos definidos na consulta. Este recurso mostra sua utilidade agora, pois embora este exemplo não tenha campos com o mesmo nome, estes podem ser identificados unicamente sem ambigüidades.

Outro detalhe diz respeito aos valores contidos no resultado da consulta. Os seis primeiros campos (f1 ao f6) correspondem aos campos da tabela *TABLESALES*, cujos valores estão repetidos em ambos os registros. Isto ocorre porque foram encontrados dois registros da tabela *TABLESALESDETAIL* (campos f7 ao f10) que se relacionam com a ordem '1234'.

A partir do *result set* da consulta, o próximo passo consiste em converter este conjunto de dados em um ou mais documentos XML. Para tornar o processo mais simples, este foi dividido em duas etapas: a primeira consiste em extrair os dados do *result set* e armazená-los em uma estrutura em árvore; e a segunda consiste em converter esta árvore em um ou mais documentos XML, conforme o definido pela sua especificação.

Cada nodo desta árvore "intermediária" está vinculado ao seu mapeamento para tabela correspondente e contém uma lista de valores correspondentes aos campos da tabela mapeada e que estão no *result set*. O processo consiste então em construir esta árvore a partir do *result set*, ou seja, converter o resultado que está na forma de um conjunto para um resultado na forma de árvore (e assim mais próximo da estrutura de um documento

XML). O algoritmo básico para a construção desta árvore consiste em, para cada registro do *result set*, realizar os seguintes passos:

- Para cada *mapeamento para tabela* que possui algum campo no *result set*:
  1. Criar um novo *nodo*;
  2. Para cada *mapeamento para campo* contido neste *mapeamento para tabela* (inclusive nos contidos nos elementos ignorados), adicionar ao *nodo* o nome e o valor deste campo (que estão no registro atual do *result set*);
  3. Se o *nodo* não existe na lista de nodos do mapeamento, este é adicionado à lista de nodos do mapeamento, caso contrário, obtém a referência deste *nodo* da lista de nodos;
  4. O *nodo* atual é adicionado como filho do *nodo* relativo ao *mapeamento para tabela* que é "pai" do *mapeamento para tabela* atual.

Em linhas gerais, este algoritmo visa selecionar os valores que estão agrupados no *result set* e adicioná-los em uma série de nodos organizados em forma de árvore. Ele cria um novo nodo para cada tabela cujos campos estão no *result set*, mas descarta os nodos repetidos. Como foi visto no exemplo anterior, os dois registros do *result set* possuem valores repetidos para os campos da tabela *TABLESALES*. Através do algoritmo (no passo 3), estes valores darão origem a um único nodo, que por sua vez conterá os dois nodos criados pelos valores dos campos da tabela *TABLESALESDETAIL*, conforme pode ser visto na próxima página (Figura 30).

A árvore gerada neste exemplo possui como nodo raiz o nodo relacionado ao mapeamento "raiz", ou seja o mapeamento para tabela relacionado ao elemento requisitado na consulta XML (*ordemDeCompra*). Como o documento a ser gerado deve ser válido, é necessário também obter dados referentes aos mapeamentos (para tabela) "filhos" (relativos aos elementos filhos) que originarão novos nodos, e que por conseqüência, são filhos do nodo raiz, e assim sucessivamente. Neste exemplo, há apenas mais dois nodos descendentes do nodo raiz, que são relacionados ao elemento *item*.

Percebe-se que agora, além do *mapeamento* que serve para armazenar a relação entre os dois *modelos* (XML e relacional), há também uma *árvore*, utilizada para armazenar temporariamente uma *instância* que dará origem a um ou mais documentos XML.

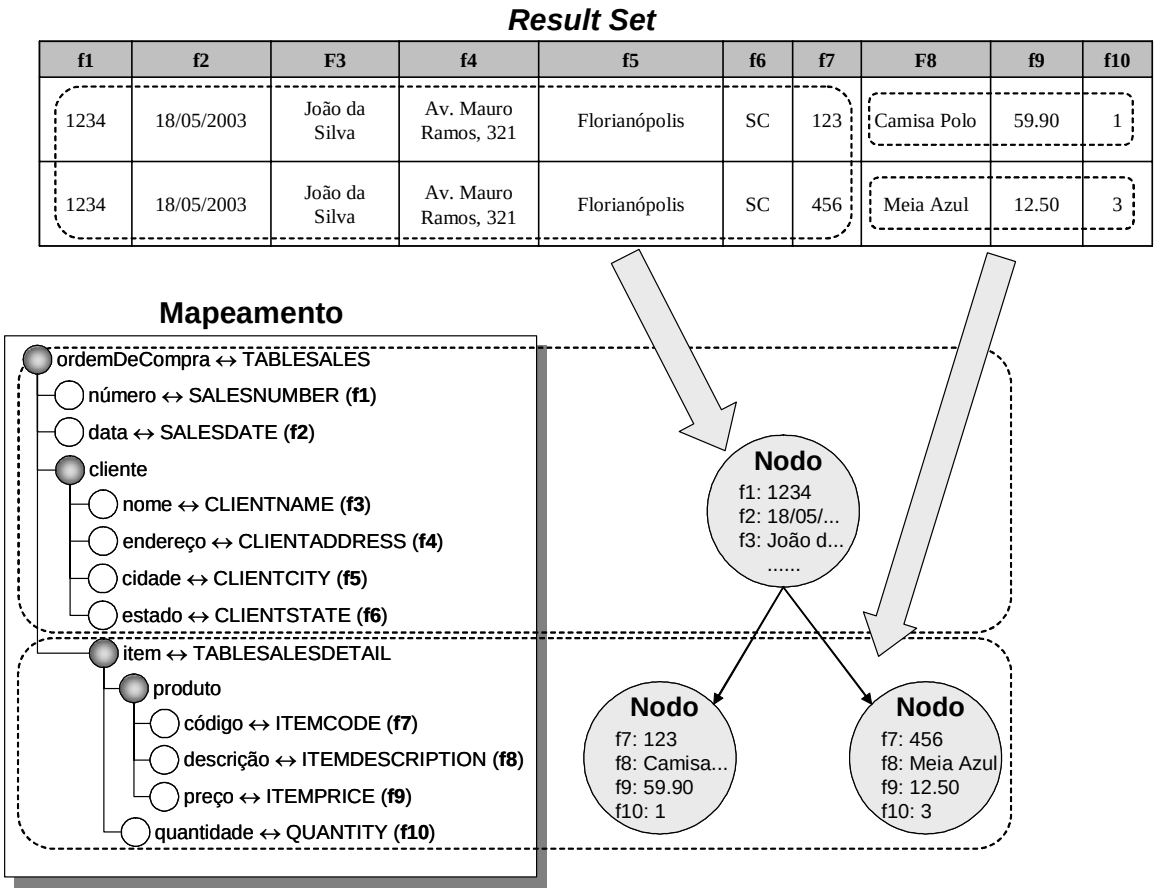

Figura 30: Construção da árvore a partir dos dados do *result set*.

Com a definição da árvore que contém os valores do *result set* (instância), o próximo passo é mais simples, consistindo na geração dos documentos XML a partir destes dados. Um aspecto que não foi explicado até agora foi o fato do mapeamento possuir uma estrutura semelhante à especificação XML correspondente. Isto será útil agora, pois é necessário gerar os elementos do documento XML na ordem em que foram declarados para que o documento seja válido.

O processo consiste em percorrer o mapeamento e escrever, para cada campo, o seu valor correspondente (contido no nodo atual) cercado pelo nome do elemento. Note que os elementos que foram marcados como ignorados (como *cliente* e *produto*), agora devem estar presentes no documento. Sempre que for encontrado um mapeamento para tabela, é criado um elemento para cada um dos seus nodos relacionados que também sejam filhos do nodo relativo ao mapeamento superior. Finalmente, se houver mais de um nodo relativo ao mapeamento raiz, é verificada cardinalidade do elemento: se for possível, os elementos são escritos no mesmo documento XML, caso contrário, é feito um documento XML para cada

nodo relacionado ao elemento raiz. Ao final do processo, o documento XML é gerado, conforme a Figura 31 (próxima pagina).

Como o DTD deste exemplo permite apenas um elemento “ordemDeCompra” por documento, se a consulta retornasse mais de uma ordem de compra, seriam gerados tantos documentos quantas forem as ordens recuperadas.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE ordemDeCompra SYSTEM "ordem.dtd" []>
<!-- Ordem de Compra -->
<ordemDeCompra número="1234" data="18/05/2003">
  <cliente>
    <nome>João da Silva</nome>
    <endereço>Av. Mauro Ramos, 321</endereço>
    <cidade>Florianópolis</cidade>
    <estado>SC</estado>
  </cliente>
  <item>
    <produto código="123">
      <descrição>Camisa Polo</descrição>
      <preço>59.90</preço>
    </produto>
    <quantidade>1</quantidade>
  </item>
  <item>
    <produto código="456">
      <descrição>Meia Azul</descrição>
      <preço>12.50</preço>
    </produto>
    <quantidade>3</quantidade>
  </item>
</ordemDeCompra>
```

Figura 31: Documento XML carregado pelo módulo de integração.

6.3 Considerações sobre a Abordagem Apresentada

Novamente aqui tem-se um modelo semi-automático, ou seja, há uma etapa manual onde o usuário define o(s) mapeamento(s) entre o modelo de referência da EV e o modelo de dados legado (etapa de instalação da plataforma da EV); e há uma etapa que pode ser feita sem intervenção humana que consiste no processo de integração propriamente dito (nas demais etapas do ciclo de vida da EV). Evidentemente, o modelo apresentado pode ser aplicado em diferentes domínios de aplicação, visto que a questão da integração de dados está presente em qualquer sistema heterogêneo. Na próxima seção serão apresentadas algumas ferramentas utilizadas em outros contextos.

É importante salientar que podem ocorrer casos onde não seja possível realizar um mapeamento de forma satisfatória, seja porque não há campos suficientes (por exemplo, chaves primárias), ou porque não é possível relacionar as tabelas do modelo de dados legado com as estruturas definidas no MR. Entretanto, este é um problema intrínseco a este tipo de aplicação, visto que *a priori* não se pode afirmar que dois modelos quaisquer são compatíveis entre si. Assim, a solução desta questão está fortemente vinculada às particularidades de cada modelo de dados e, portanto, está fora do escopo deste trabalho.

6.4 Trabalhos Relacionados

Conceitualmente, há duas abordagens básicas e distintas que um *middleware* XML pode adotar: *baseada em templates* e *baseada em modelos*. Esta distinção é importante de modo a se destacar algumas características que possam ser comparadas entre as abordagens. A seguir é feita uma breve explicação sobre cada uma delas, bem como são apresentadas algumas ferramentas relacionadas. Devido ao grande número de implementações de *middlewares* XML, aqui serão listadas as mais relevantes. Uma lista mais abrangente destas ferramentas pode ser encontrada em BOURRET (2003d).

6.4.1 Middlewares XML baseados em Templates

Nesta abordagem, comandos são embutidos em um *template* que é processado pelo *middleware* XML. O *template* pode conter, por exemplo, um comando SELECT que será processado, e o resultado desta consulta é então formatado em XML. As principais desvantagens são a quantidade de código (*templates*) que o usuário deve escrever, e o fato de que poucos dos produtos disponíveis desta categoria são capazes de transferir dados oriundos de documentos XML para a base de dados, ou seja, geralmente permitem apenas carregar dados em documentos XML. Esta abordagem não é utilizada neste trabalho. Exemplos de *middlewares* XML desta categoria:

- **ODBC2XML:** biblioteca (DLL) do Windows para a transferência de dados de uma base de dados ODBC[24] para um documento XML a partir de declarações SELECT dentro de *templates*. (ODBC2XML, 2003).

[24] ODBC, *Open Data Base Connectivity*, é uma API padrão que permite acesso a bases de dados independente de linguagem de programação, sistema operacional ou sistema de banco de dados.

- **SXQL:** classes Java que transferem dados entre uma base de dados JDBC[25] e um documento XML, utilizando *templates* que contêm declarações SELECT. Como a anterior, possui licença *shareware* (SXQL, 2003).
- **XSYNC-ML:** *middleware* XML multiplataforma (desenvolvido em Java) utilizado para transferir dados entre bases de dados relacionais e documentos XML. Além de prover o acesso através de *templates*, esta ferramenta possibilita a geração dinâmica de DTDs ou de esquemas relacionais (SOUSA *et al.,* 2003).

6.4.2 Middlewares XML baseados em Modelos

Nessa abordagem, um modelo de dados relativo à especificação XML é definido para então ser mapeado implícita ou explicitamente para a base de dados. Mapeamento implícito significa que a própria ferramenta já implementa internamente o mapeamento, o que não provê flexibilidade. Por exemplo, algumas ferramentas só permitem o mapeamento se os nomes dos elementos XML forem iguais aos nomes das tabelas/campos da base de dados. Já um mapeamento explícito é definido de forma separada pelo usuário, dando uma maior flexibilidade ao processo. Este último caso é o que foi apresentado neste trabalho. Exemplos de *middlewares* XML desta categoria:

- **XML-DBMS:** usado para transferir dados entre um documento XML e uma base de dados relacional. Ele usa um mapeamento objeto-relacional que é descrito por uma linguagem de mapeamento baseada em XML. Mapeamentos podem ser escritos manualmente ou gerados automaticamente a partir de um DTD ou esquema da base de dados. Adicionalmente, permite a geração de esquemas de base de dados a partir de DTDs e vice-versa (XML-DBMS, 2003). BOURRET et al. (2000) descreve alguns detalhes de implementação desta ferramenta que, dentre as pesquisadas, apresentou um maior grau de flexibilidade e funcionalidade, além de estar disponível gratuitamente como *open source*.
- **DB2XML:** classes Java usadas para transferir dados de uma base de dados relacional para um documento XML. Estas classes podem ser usadas em uma aplicação individual ou como um *servlet*. O produto (*open source*) modela o documento XML como um conjunto de tabelas, onde o usuário especifica um ou

[25] *Java Data Base Connectivity*. API da plataforma Java que define como um cliente pode acessar uma base de dados. Implementa uma "ponte" que permite a comunicação com *drivers* ODBC, e conseqüentemente o acesso a uma grande variedade de fontes de dados.

mais comandos SELECT. As suas opções incluem a especificação dos nomes dos *tags* a ser usados no resultado, bem como incluir metadados da base de dados no documento. O documento pode ser retornado como um arquivo, fluxo (*stream*) ou objeto DOM, além de prover suporte de enviar o resultado para um processador XSL. O mapeamento é feito implicitamente, fazendo com que o documento resultante tenha um formato não muito flexível (DB2XML, 2003).

- **XML SQL Utility for Java:** conjunto de classes Java desenvolvido pela Oracle (ORACLE, 2003a) para transferir dados entre bases de dados relacionais e um documento XML. Estas classes podem ser usadas em uma aplicação particular. Pode usar um mapeamento objeto-relacional (se a base de dados suporta SQL 3) ou usar um mapeamento baseado em tabela para um única tabela. No carregamento dos dados da base de dados para XML, o usuário pode definir um comando SELECT ou um *result set* JDBC. Os resultados são retornados como um documento XML, um objeto DOM ou eventos SAX. No armazenamento dos dados, o usuário provê um documento XML ou um objeto DOM. Algumas opções incluem a especificação de alguns *tags* usados no documento XML. Esta ferramenta também utiliza mapeamento implícito.

6.4.3 Considerações

Tal como no caso de aplicações de XML *Data Binding*, há uma considerável variedade de aplicações implementadas para *Middlewares* XML. De fato, desde que a XML foi concebida, um grande número de aplicações para os mais diversos propósitos foram desenvolvidas, não apenas no contexto de *Middlewares* XML ou XML *Data Binding*. Desta maneira, este trabalho também visa desenvolver um *middleware* XML em particular, visto que apesar da grande diversidade, não foi possível encontrar uma ferramenta que satisfizesse os requisitos definidos para a etapa de instalação da plataforma de uma EV. Dentre os aspectos que levaram ao desenvolvimento desta ferramenta em particular, temos:

- Algumas das ferramentas estão vinculadas a aspectos muito específicos, tais como sistema operacional, plataforma de base de dados, ou protocolo de comunicação. Por exemplo, ODBC2XML, funciona apenas na plataforma Windows;
- Algumas ferramentas geram documentos XML com um formato pouco ou nada flexível, tais como DB2XML e XML SQL Utility for Java.

- Algumas possuem uma abordagem mais “intrusiva”, ou seja, atuam diretamente na manipulação da estrutura da base de dados, como a geração de novas tabelas. Isto não é o caso neste trabalho, visto que o objetivo não é modificar a estrutura interna dos repositórios legados. Este subconjunto de ferramentas já atuam na linha tênue ente as aplicações de *Middleware* XML e as de XML *Data Binding*. Exemplos: DB2XML, Castor, XSYNC-ML.
- Finalmente, o fator crucial para o desenvolvimento desta ferramenta foi a dificuldade de se encontrar uma ferramenta que faça o mapeamento dinamicamente e que, além disso, mantenha o documento válido perante o esquema XML configurado. Até mesmo XML-DBMS (BOURRET *et al.* 2000), que foi a ferramenta mais flexível encontrada, não mantém a estrutura dos documentos intacta.

Assim, o próximo capítulo apresentará uma implementação de um *middleware* XML que atende aos requisitos definidos para a etapa de instalação da plataforma da EV, ou seja:

1. Através de mapeamentos, converter dinamicamente os dados entre bases de dados e documentos XML, e vice-versa, e;
2. Manter os documentos gerados válidos perante a sua especificação XML.

Adicionalmente, outras categorias de aplicações que utilizam XML e bases de dados (conforme enumerado na seção 3.4.7.2) têm relação com *middlewares* XML, embora com abordagens sensivelmente diferentes. São elas: *bases de dados integradas com XML* e *bases de dados XML nativas*.

As *bases de dados integradas com XML* provêm funcionalidades adicionais que permitem a utilização de XML, mas com a desvantagem de normalmente serem soluções “pesadas” em termos de processamento, e de vincularem esta solução ao uso obrigatório do sistema em questão. Exemplo: Oracle 9i (ORACLE, 2003b). Já as *bases de dados XML nativas* armazenam documentos XML na forma nativa, pelo menos do ponto vista de quem acessa a base de dados. Exemplo: Tamino (TAMINO, 2003). Tamino também oferece funcionalidades de *middleware* XML, permitindo mapeamentos entre XML e banco de dados relacional através de uma linguagem de esquema proprietária.

7 Implementação

7.1 Introdução

Este capítulo apresentará os detalhes mais relevantes acerca da modelagem e implementação de duas ferramentas e de uma biblioteca de classes que foram desenvolvidas a partir dos modelos apresentados nos Capítulos 5 e 6. O próximo capítulo apresentará os testes que foram realizados através da utilização destas ferramentas de modo que os conceitos apresentados possam ser avaliados.

Inicialmente, estas ferramentas começaram a ser implementadas em C++ (Builder 5), mas a escassez de APIs para XML disponíveis em C++ levou à migração para o Java, que além da vantagem de ser multiplataforma (aumentando a portabilidade e facilidade de interoperação), possui muitas APIs para XML disponíveis.

Para a implementação dos protótipos, foram utilizadas as seguintes ferramentas:

- Java Development Kit (JDK) 1.4.2 (java.sun.com);
- Ambiente de desenvolvimento Eclipse versão 2.1.1. Eclipse é uma plataforma desenvolvida em linguagem Java e que opera sob o paradigma de código aberto (*open source*). Possui uma arquitetura de *plug-ins* que permite a integração com outras ferramentas (ECLIPSE, 2003);

As ferramentas implementadas neste trabalho estão disponíveis publicamente em http://sourceforge.net/projects/gsigma-xml/, sob as normas da GPL (*GNU General Public License*).

Antes de se aprofundar nos detalhes de implementação de cada ferramenta, é preciso deixar claro o papel de cada uma de acordo com o apresentado no modelo conceitual. São elas:

1. Ferramenta de *geração de código*: utilizada para auxiliar a etapa de *concepção da plataforma*, onde o usuário (desenvolvedor da plataforma) gera código (classes e tabelas de uma base de dados) de forma semi-automática a partir de um MR definido em XML.
2. Ferramenta de *definição dos mapeamentos*: utilizada na etapa de *instalação de plataforma*, onde o usuário (por exemplo, o administrador da base de dados de uma

empresa em particular) define os mapeamentos entre um MR definido em XML e o modelo de dados da base de dados legada.

3. Biblioteca de classes que implementa um *middleware* XML. Esta biblioteca deve ser ligada à aplicação que atuará como o *módulo de integração*, cujo papel é o de realizar a integração de dados a partir dos mapeamentos definidos na etapa de instalação de plataforma (pela ferramenta citada no item anterior). O *middleware* XML não foi implementado como uma aplicação completa de modo a permitir uma maior flexibilidade a quem implementar um *módulo de integração* em particular, independente dos seus requisitos de comunicação.

É importante destacar duas observações sobre estas implementações. Em primeiro lugar, a etapa de atuação de cada uma: a primeira é utilizada na etapa de *concepção da plataforma*; a segunda, na etapa de *instalação de plataforma*; e a terceira (vinculada a uma aplicação completa), nas demais etapas do ciclo de vida de EV. Além disso, é preciso observar os tipos de usuário de cada uma delas: a primeira e a última são tipicamente utilizadas por desenvolvedores, seja para a geração de código ou para a implementação do módulo de integração. A ferramenta de definição dos mapeamentos é tipicamente utilizada por um administrador de base de dados.

7.2 Ferramenta para Geração de Código

A ferramenta de geração de código implementa o modelo de geração de classes e tabelas (*XML Data Binding*) apresentado no Capítulo 5. Para implementar esta ferramenta, foi preciso levantar algumas considerações que podiam ser ignoradas no modelo conceitual. São elas:

1. O fato de haver vários tipos de entrada diferentes – que neste caso são DTD, XML *Schema*, e XMI – implica em um processador diferente para cada tipo de entrada.
2. Para que seja possível efetuar alterações nas classes antes da geração de código propriamente dita é necessário armazenar as classes em uma estrutura interna.
3. Também é preciso considerar que existem diversas linguagens disponíveis e, assim, a ferramenta deve possuir um gerador de código para cada linguagem que foi escolhida para ser gerada.

A partir destes apontamentos é possível definir um modelo de implementação que representa a estrutura geral da ferramenta, conforme pode ser visto na Figura 32.

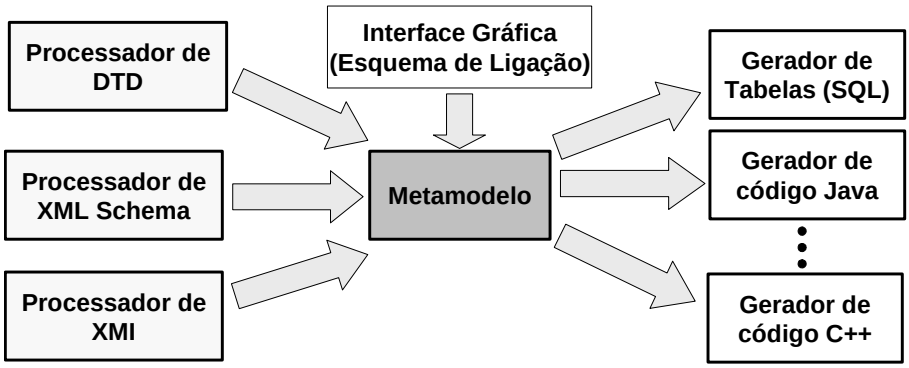

Figura 32: Modelo de implementação da ferramenta de geração de código.

Conforme a figura, a aplicação é constituída por quatro tipos de estruturas:

1. Os processadores para cada tipo de entrada, cuja saída será armazenada na estrutura interna (metamodelo);
2. Um *metamodelo*, que armazena de maneira uniforme as especificações processadas, e que sofre as modificações do *esquema de ligação*;
3. A interface gráfica, que implementa a funcionalidade do *esquema de ligação*.

4. Os geradores de código para diversas linguagens. Por limitações de tempo, apenas o gerador para linguagem Java foi implementado neste protótipo. Entretanto, a forma como a aplicação foi modelada permite que geradores para outras linguagens possam ser facilmente acoplados. Alem disso, há o gerador de tabelas da base de dados.

Estas estruturas serão detalhadas nas próximas seções.

7.2.1 O Metamodelo

O metamodelo será a primeira estrutura a ser explicada, pois representa o ponto central do modelo de implementação, sendo o responsável por armazenar o que é lido da especificação de entrada. Ele pode sofrer modificações do *esquema de ligação* (que nesta implementação é definido via interface gráfica), e finalmente o código gerado é obtido a partir dos seus dados. A idéia básica é que, tendo como objetivo a geração de um modelo de classes/tabelas, este possa ficar armazenado em uma estrutura intermediária de maneira que fosse independente da especificação de entrada, e também independente da linguagem à qual o código será gerado.

Assim, foi definido um modelo de classes que fosse capaz de armazenar todas as informações de um modelo qualquer de classes. Em outras palavras, foi criado um metamodelo. A Figura 33, na próxima página, ilustra o seu diagrama de classes.

Os métodos foram suprimidos por uma questão de espaço, e foram mantidos os principais atributos de cada classe. O objetivo deste metamodelo é armazenar metadados a respeito das classes que serão geradas a partir da especificação de entrada. A classe “MetaModel” armazena informações relativas à especificação de entrada, como o seu nome e tipo, além de possuir uma lista de classes “MetaClass”, que como o próprio nome diz, representa uma metaclasse, isto é, uma classe que armazena informações relativas a uma classe, tais como nome, nome do elemento XML, etc. Conforme o diagrama, uma “MetaClass” é composta por um ou mais “MetaField”. De modo similar, um “MetaField” armazena metadados de atributos de classe, como o nome, nome XML, o tipo, se o atributo é simples ou lista, etc.

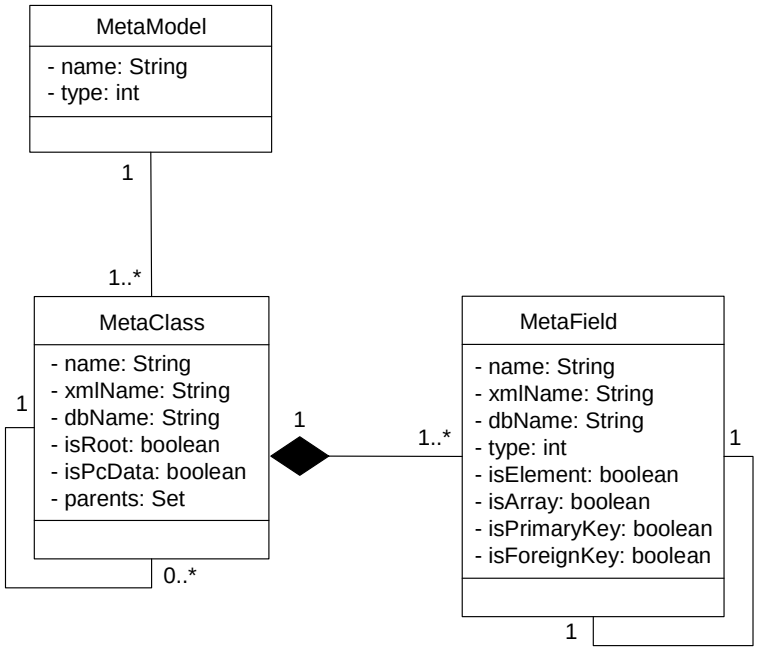

Figura 33: Diagrama de classes do metamodelo.

Embora seja uma peça fundamental no modelo de implementação, é preciso salientar que o metamodelo é passivo, ou seja, serve apenas para armazenar a estrutura das classes em um formato interno. Assim, cabe aos processadores processar a especificação de entrada, converter para uma estrutura de classes e armazená-la no metamodelo. De maneira análoga, é tarefa dos geradores ler a estrutura contida no metamodelo e gerar o código correspondente.

Nas próximas seções será explicado como cada uma das outras estruturas foi implementada.

7.2.2 Processadores das Especificações XML

Esta ferramenta implementa três processadores para os diferentes tipos de entrada: DTD, XML *Schema* e XMI. Embora cada um tenha uma implementação diferente, todos implementam uma interface comum, conforme pode ser visto na Figura 34.

A interface “Processor” declara os dois métodos que cada tipo de processador implementa: um método para ler uma especificação XML, e outro para obter o *metamodelo* gerado a partir da especificação lida. Os detalhes da implementação de cada um dos processadores são apresentados a seguir.

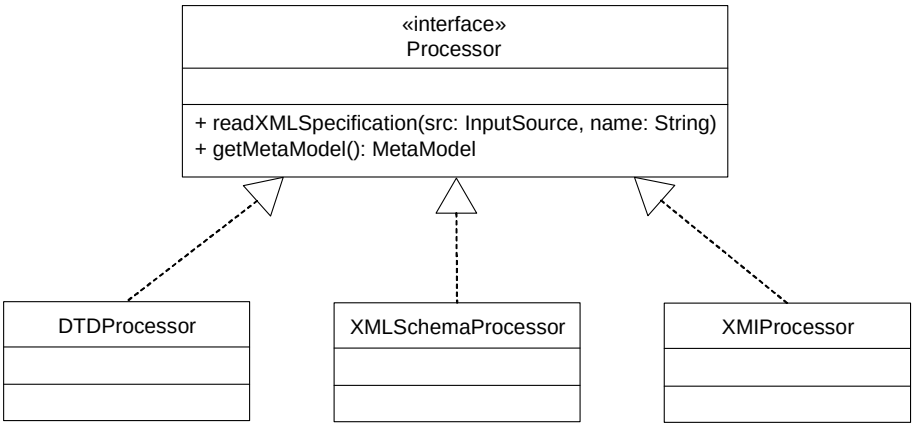

Figura 34: Hierarquia de classes dos processadores de entrada.

Embora conceitualmente e até mesmo no modelo de implementação tenha-se mencionado o processamento de documentos XMI, a implementação de um processador para este tipo de especificação não foi efetuada em tempo hábil de forma satisfatória, o que deve ser providenciado para próximas versões das ferramentas.

7.2.2.1 Processador de DTD

Por possuir uma sintaxe diferente da XML, não é possível usar um *parser* de XML para processar uma *document type definition* (DTD). Embora existam diversas ferramentas para o processamento da XML, normalmente o processamento de DTDs está embutido na implementação destas ferramentas. É claro que esta situação se justifica pelo fato de que o objetivo é a utilização da XML como formato ou linguagem para as aplicações, e não a DTD. Assim, não há a necessidade de processadores específicos para esta linguagem.

Para não ter que escrever um *parser* de DTD – o que iria levar um tempo considerável, e desviaria a atenção para outro problema –, foi utilizado um *parser* de DTD implementado por BOURRET (2002). Este *parser* foi implementado em Java, é gratuito, possui código aberto, documentação, e é acompanhado de um programa exemplo para auxiliar o seu entendimento. Ele é acompanhado de um conjunto de classes que armazenam a estrutura dos DTDs processados. Por todas estas características, este componente se apresentou bastante adequado e foi utilizado na implementação do *processador de DTDs* desta ferramenta.

Entretanto, foi necessário fazer algumas modificações no código deste *parser* para se adequar às necessidades da ferramenta de geração de código. Isto porque, depois de processado pelo *parser*, o DTD (representado na forma de objetos) não armazenava a

ordem dos elementos, o que impossibilitaria a geração do código responsável pela serialização das classes. Como foi visto, é necessário respeitar a ordem em que os elementos são declarados no DTD para que os documentos XML sejam considerados válidos. O fato do *parser* de DTD ser de código aberto facilitou esta tarefa.

A tarefa do *processador de DTDs* consiste em "navegar" pela estrutura de objetos (que representa um DTD) gerada pelo *parser* de DTDs e, aplicar as regras apresentadas no modelo conceitual (seção 5.2) para a geração do *metamodelo*.

7.2.2.2 Processador de XML Schema

Tendo o XML *Schema* uma sintaxe baseada em XML, o seu processamento pode facilmente ser feito utilizando um *parser* de XML. Entretanto a sua especificação (W3C, 2001a) define uma linguagem muito extensa, o que tomaria muito tempo para que o *processador de XML Schema* desse suporte a todas as suas características. Assim, de modo semelhante ao *parser* de DTD, foi procurado um componente que já fizesse este trabalho. Como não foi possível até o momento encontrar um que suprisse as necessidades requeridas, foi implementado um processador de XML *Schema* que suporta parcialmente a sua especificação.

Conforme visto na seção 3.4.4.2, um XML *Schema* é composto basicamente de dois tipos de declarações: de elementos e de tipos de dados. O processador de XML *Schema* armazena os elementos e os tipos em estruturas separadas e, aplicando as regras apresentadas no modelo conceitual (seção 5.2), gera o *metamodelo*.

Outras características mais avançadas suportadas pelo XML *Schema*, como a flexibilidade para a definição de novos tipos a partir de tipos já definidos, não são suportadas por esta implementação. Embora isto possa ser visto como uma desvantagem, esta limitação se justifica pelo fato da especificação do XML *Schema* ser muito longa e o trabalho envolvido desviaria a atenção do problema principal. Além disto, este módulo pode ser melhorado em versões futuras, ou mesmo um componente que venha a ser disponível futuramente pode ser acoplado à ferramenta.

7.2.3 Gerador de Código

O gerador de código é o responsável por ler as informações contidas no metamodelo e gerar o código para uma dada linguagem. Falando especificamente desta implementação, dois geradores foram implementados: para linguagem Java, e para SQL (para a geração das

tabelas). Entretanto, o algoritmo para a geração de código é análogo para qualquer que seja a linguagem cujo código será gerado.

7.2.3.1 O Processo de Geração de Código

A tarefa do gerador de código consiste extrair informações do metamodelo (cuja estrutura foi apresentada na Figura 33) e gerar o código apropriado. Isso quer dizer que para cada “MetaClass” que for encontrada, o gerador deve gerar código para uma classe. Por exemplo, o gerador para Java criará um arquivo para cada classe (pois em Java cada classe deve ser salva em um arquivo separado) e escreverá as declarações iniciais da classe (em sintaxe Java) neste arquivo. O gerador então percorre todos os “MetaFields” desta “MetaClass” e gera o código da para os atributos. Continuando o exemplo, o gerador escreverá a declarações de cada atributo (em sintaxe Java) no arquivo da classe. Percebe-se, portanto, que independente da linguagem a ser gerada o processo será basicamente o mesmo.

Entretanto, o gerador deve gerar mais algumas declarações. Em primeiro lugar, como cada atributo é declarado como privado (só visível no escopo da sua classe), devem ser gerados métodos para acesso (leitura/escrita) a cada atributo gerado. Esta tarefa é simples, pois a geração dos métodos de acesso é feita adicionalmente à geração de cada atributo.

É preciso gerar também os métodos para carregar/salvar o conteúdo das classes de/para documentos XML. Finalmente, se forem geradas tabelas correspondentes a estas classes na base de dados, os métodos para o carregar e salvar nas tabelas devem ser gerados. Embora no modelo conceitual tenha-se mencionado que as classes implementam o métodos relativos à XML e ao repositório de dados, na implementação os métodos são gerados em classes separadas, denominadas “XMLProcessor” e “DataBaseManager”, respectivamente. Isto foi feito com o objetivo de se gerar classes fracamente acopladas, ou seja, a implementação das classes não está “misturada” com a forma em que estas serão serializadas em XML ou armazenadas na base de dados. A geração destas classes é detalhada nas seções a seguir.

Geração dos Métodos para XML

Antes de explicar como é feita a geração de código para a conversão para XML, é preciso ter em mente que potencialmente mais de uma classe será gerada a partir da especificação XML. Além disso, embora a estrutura hierárquica de documentos XML

implique na existência de um único elemento raiz, este elemento raiz pode ser qualquer um dos elementos definidos no esquema XML. Isto quer dizer que dentre as classes geradas, qualquer uma delas poderá ser a classe “raiz”, dependendo do documento XML.

Portanto, embora todas as classes geradas necessitem de métodos para converterem-se em XML, é preciso definir quais as classes cujos elementos serão as potenciais raízes dos documentos XML a serem utilizados. É claro que na maioria das vezes as aplicações trabalharão com documentos XML que contemplam toda a estrutura declarada na especificação XML, onde os elementos potenciais raízes correspondem justamente aos elementos que não estão na declaração de nenhum outro, ou seja, não possuem “pais”.

Para permitir esta flexibilidade, o usuário pode, via interface gráfica (esta assume o papel do *esquema de ligação*), definir quais serão as classes potenciais “raízes”, ou seja, as classes que podem ser instanciadas a partir de documentos XML, ou que podem gerar documentos XML a partir de suas instâncias.

Além disso, as classes do metamodelo armazenam, entre outras coisas, os seus respectivos nomes em XML. Isto é utilizado neste momento para a geração dos métodos para acesso à XML, pois é necessário saber o nome do elemento XML correspondente a cada classe e seus atributos, além da ordem em que foram declarados.

Por ser mais simples, a primeira tarefa a ser explicada é a geração do método para salvar o conteúdo de um objeto em XML (*serialização*). Conforme mencionado, o modelo conceitual não trata dos aspectos que definem a forma que os documentos XML serão transportados via rede, se são armazenados em arquivos, etc. Para que o código gerado seja o mais genérico possível e ofereça uma maior flexibilidade para o usuário, o método para serialização para XML recebe como parâmetro um objeto da classe abstrata “Writer” onde o documento XML será escrito. “Writer” é uma classe abstrata da biblioteca do Java que é utilizada para a escrita em fluxos (*streams*) de caracteres (SUN, 2003a). Assim, qualquer mecanismo que adote o conceito de *fluxo*, tais como uma conexão por *socket* ou HTTP, ou mesmo salvar em um arquivo ou *string* na memória, pode utilizar as diversas classes derivadas também disponíveis na biblioteca do Java.

Assim, para cada classe a ser gerada há um método correspondente responsável por converter seus dados em XML (a respectiva classe é passada como parâmetro). O conteúdo deste método consiste em escrever no *writer* os marcadores XML correspondentes a cada atributo da classe “cercando” o valor contido neste atributo. Se este atributo for uma referência a outra classe, o método responsável por salvar esta classe em XML é invocado

(que fará a mesma tarefa, mas com a nova classe) com o *writer* sendo passado de parâmetro. Para que os objetos sejam salvos em XML, o método deve ser invocado tendo como parâmetro um objeto das possíveis classes raízes, que por sua vez invocará os métodos relativos aos seus "filhos", e assim sucessivamente até mapear toda a hierarquia de classes. Ao final, o *writer* terá então escrito o documento XML completo.

Já o método para carregar o conteúdo de um documento XML (*desserialização*) representa um processo mais trabalhoso. Em primeiro lugar o *parser* XML estará encapsulado na implementação deste método. Além disso, o *parser* utiliza a abordagem SAX que, conforme visto na seção 3.4.5.3, apresenta um melhor desempenho em relação ao DOM. Além disso, como também já foi visto, este tipo de *parser* necessita de um modelo de classes específico e de um *document handler* (ambos são gerados automaticamente). O *document handler* é a entidade responsável por capturar os eventos SAX e alimentar adequadamente o modelo de classes (seção 3.4.5.2). O método para a *desserialização* gerado tem apenas a classe raiz como parâmetro, pois ao contrário da *serialização* (que é feita aos poucos com os dados de cada classe), a *desserialização* é realizada na sua totalidade pelo *document handler*, que só precisa do objeto da classe raiz.

De modo análogo ao processo de serialização, é necessário definir uma forma genérica e flexível de se definir o parâmetro de entrada que contém o documento XML. Para isto, foi utilizada a classe "InputSource", que está disponível na biblioteca do Java. Esta classe permite encapsular informações a respeito de uma fonte de dados codificada em XML. Ela é usada pelo *parser* SAX para determinar como um documento XML deve ser lido: se for um fluxo (*stream*) de caracteres ou fluxo de *bytes* (neste caso com o auxílio de um formato específico de codificação), o *parser* lerá diretamente o fluxo; caso não haja um fluxo disponível, o *parser* tentará abrir uma conexão com o recurso identificado pelo URI (*Uniform Resource Identifier*) declarado em um atributo de classe chamado *"public identifier"* (SUN, 2003b).

Geração dos Métodos para acesso à Base de Dados

No caso de também serem geradas as tabelas correspondentes às classes do *metamodelo*, os métodos para acessá-las também devem ser gerados. Desta forma, para cada classe do metamodelo serão gerados os métodos para carregar e salvar o seu conteúdo da base de dados. Novamente as informações contidas no metamodelo serão úteis, mais especificamente, as informações que indicam se um dado atributo é chave primária da sua

tabela. No conteúdo dos métodos gerados há uma variável do tipo *string* que, de maneira análoga ao *writer* da conversão para XML, irá concatenar as declarações em sintaxe SQL, que em seguida é passada de parâmetro para a variável que guarda a conexão com a base de dados.

O método para carregar terá como parâmetros os atributos que forem definidos como chave primária da tabela correspondente, pois são os campos necessários para identificar unicamente um registro de uma tabela. A *string* conterá uma declaração "SELECT" que fará a consulta na tabela passando de parâmetro os campos (chaves primárias). Os valores do registro retornado são salvos nos respectivos atributos. No caso de esta classe se relacionar com outra classe, haverá uma relação (chave estrangeira) com a tabela correspondente à classe filha. Assim, novamente de maneira análoga à conversão para XML, é invocado o método que contém a classe filha como parâmetro para que esta também seja carregada da base de dados, e assim sucessivamente.

No método para se salvar na base de dados, caso esta classe não esteja salva na tabela, a *string* conterá uma declaração "INSERT" passando de parâmetro todos os valores dos atributos desta classe. Caso este registro já exista, será feita uma declaração "UPDATE" com os mesmo parâmetros. Novamente, no caso de um atributo ser uma referência a classe, o mesmo método é invocado para a classe filha.

Além das informações contidas no metamodelo, é preciso considerar também que são necessários alguns parâmetros que permitirão o acesso a uma base de dados específica, tais como: a localização da base de dados, a classe que atua como interface (*driver* JDBC) para o acesso à base de dados, bem como nome e a senha do usuário da base de dados. Estes parâmetros também fazem parte do esquema de ligação conforme será visto mais adiante.

7.2.4 Considerações sobre o acesso à base de dados

Uma característica que se buscou implementar em ambas as ferramentas foi a flexibilidade quanto ao acesso a diferentes bases de dados. Esta característica pôde ser obtida pela utilização da API JDBC (ver nota na página 95) disponível na plataforma Java através do pacote (*package*) de classes chamado "java.sql". Esta API permite o acesso e processamento de dados armazenados em uma fonte de dados (normalmente uma base de dados relacional) usando a linguagem Java. JDBC inclui também uma estrutura onde

diferentes *drivers* podem ser dinamicamente instalados para acessar diferentes fontes de dados (SUN, 2003c). Dentre as funções disponíveis nesta API, destacam-se (SUN, 2003c):

- **Fazer uma conexão com uma base de dados:** dentre outras coisas, permite a conexão com um *driver* (usando a classe *DriverManager*), ou registrar novos *drivers* (interface *Driver*).
- **Enviar comandos SQL para uma base de dados:** por exemplo, a classe *Statement* pode ser usada para enviar comandos básicos.
- **Recuperar e atualizar os resultados de uma consulta**, usando a interface *ResultSet*.
- **Acesso aos metadados de uma base de dados:** a interface *DatabaseMetaData* provê metadados a respeito de uma base de dados.
- **Exceções:** por exemplo, a exceção *SQLException* é lançada pela maioria dos métodos quando há algum problema no acesso aos dados.

Embora a API JDBC seja bastante abrangente, aqui foram destacadas as funcionalidades principais relacionadas à implementação deste trabalho. Para ilustrar como tudo isto funciona, será mostrado um pequeno exemplo:

1. A classe que implementa o *driver* de acesso à base de dados (e que é indicada pelo usuário), é registrada através de interface *Driver*.
2. Uma conexão com a base de dados pode ser feita (usando a classe *DriverManager*) e que tem como parâmetros a localização da base de dados, o nome e a senha do usuário.
3. A partir da classe *Statement* e da interface *ResultSet* (que estão vinculadas à conexão criada) é possível enviar comandos e obter os seus resultados (dados).
4. Exceções podem ser lançadas, caso surjam problemas no acesso à base de dados.

A única das funcionalidades apresentadas que não fez parte do exemplo foi o acesso aos metadados, pois não é utilizada na ferramenta de geração de código. Esta funcionalidade foi utilizada na implementação da ferramenta de definição dos mapeamentos, e será detalhada mais adiante.

7.2.5 Interface Gráfica

Conforme visto no modelo conceitual (seção 5.2.2), a definição do esquema de ligação corresponde à parte manual do processo de geração, onde o usuário define certos parâmetros para que o processo como um todo seja bem sucedido e adequado às suas necessidades. Para facilitar ao máximo esta tarefa, a ferramenta possui uma interface gráfica amigável, onde o usuário pode visualizar os diferentes aspectos envolvidos (especificação XML, classes, tabelas), e facilmente efetuar a modificações necessárias que conceitualmente são definidas no esquema de ligação, como renomear as classes, definir as chaves primárias, etc., conforme a Figura 35 abaixo.

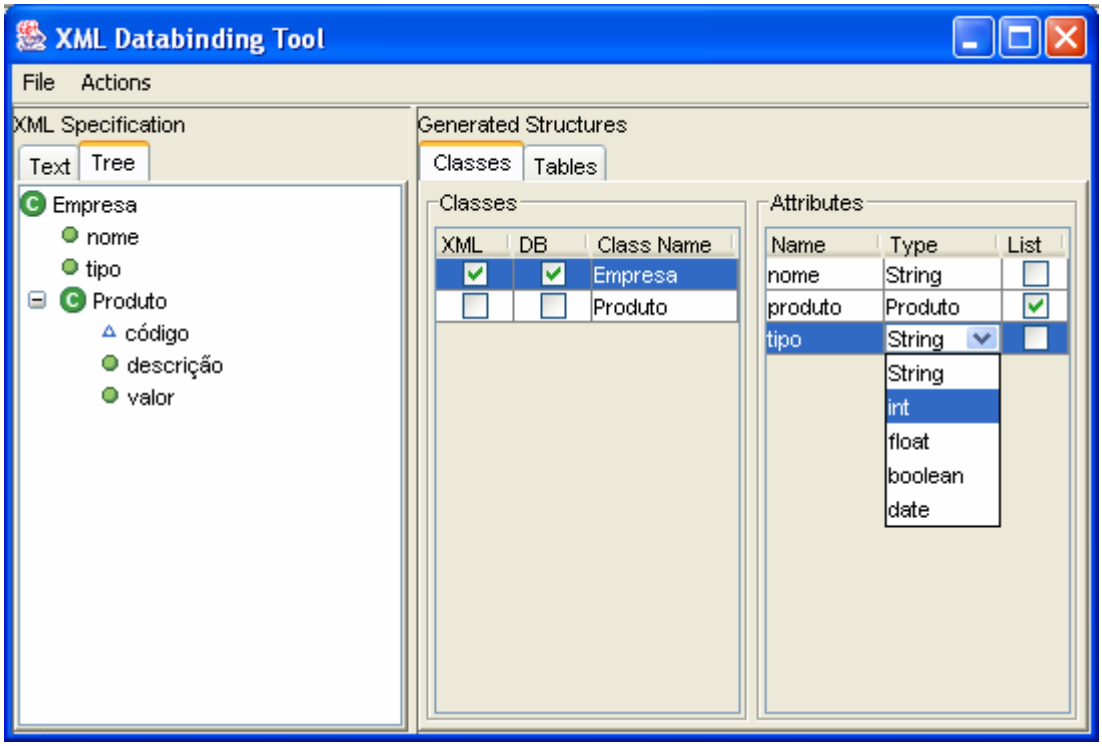

Figura 35: Interface gráfica da ferramenta de geração de código.

A interface é composta basicamente por duas partes:

1. **XML Specification:** pode ser vista no lado esquerdo e permite a visualização da especificação XML de duas formas diferentes: o texto que compreende a especificação (na aba *Text*), ou a especificação apresentada na forma de árvore (na aba *Tree*).

2. **Generated Structures:** compreende o centro e o lado direito da interface e apresenta duas abas, uma para mostrar as classes, e a outra para mostrar as tabelas, ambas relacionadas à especificação XML.
    a. Na aba *Classes*, conforme pode ser visto na Figura 35, é possível visualizar as classes (no lado esquerdo) e os atributos na classe selecionada (no lado direito). Para renomear tanto uma classe quanto um atributo, basta clicar duas vezes na classe/atributo em questão. A definição do tipo de dados também é feita facilmente, conforme pode ser visto na figura. Para cada classe também é possível definir se ela será ou não uma possível classe "raiz", conforme apresentado na última seção. Para isto, basta selecionar as caixas de seleção "XML" (que indica se a classe poderá gerar um documento XML válido) ou "DB" (que indica se a classe pode se salvar/carregar na base de dados).
    b. Na aba *Tables*, destacada na Figura 36, é possível visualizar as tabelas e seus respectivos campos. Tal como na aba *Classes*, é possível renomear as tabelas/campos bem como "marcar" os campos que devam ser chaves primárias (caixa de seleção "PK").

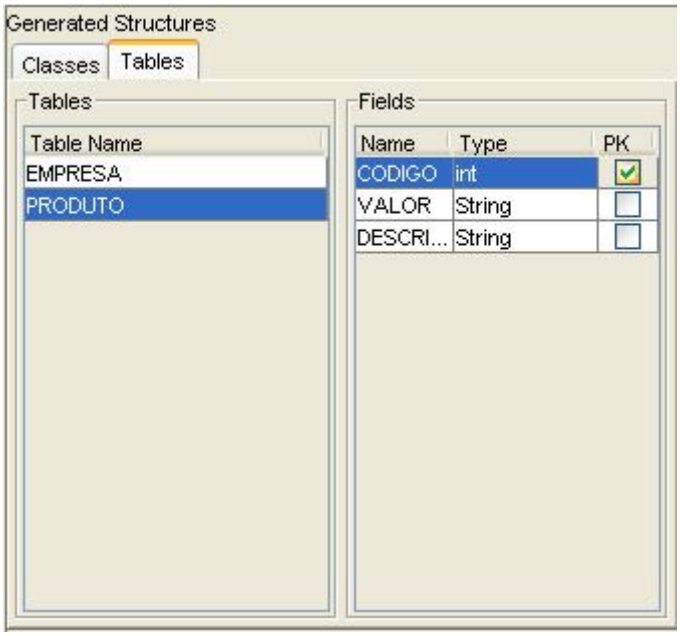

Figura 36: Definição das tabelas e campos a serem gerados.

A interface também possui um menu com algumas opções. Uma especificação XML pode ser aberta através da opção *File→Load XML Specification*. Alguns parâmetros adicionais devem ser definidos através da opção *Actions→Options*, que abrirá uma nova janela, conforme a Figura 37. Finalmente, depois de carregar uma especificação XML e as estruturas geradas (classes/tabelas) sofrerem as modificações por parte do usuário, é possível gerar o código (em linguagem Java) através da opção *Actions→Generate Code*.

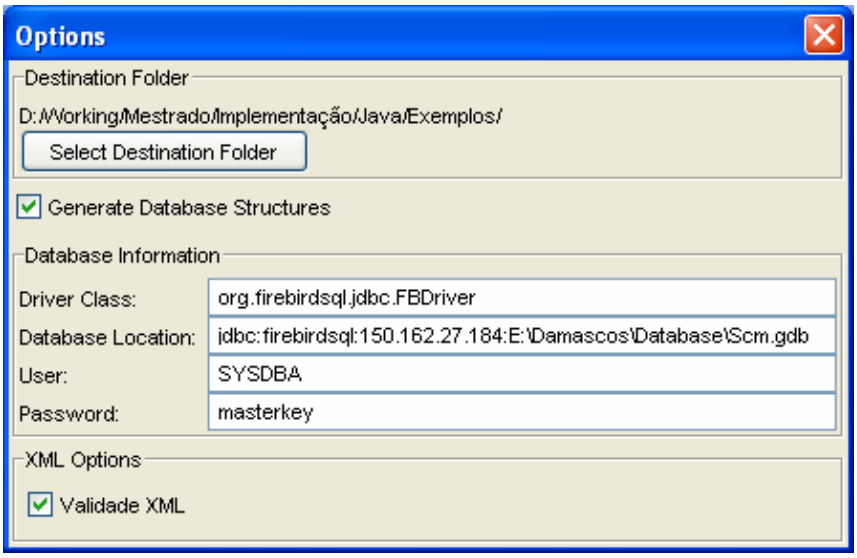

Figura 37: Tela de opções da ferramenta de geração de código.

A janela de opções permite ao usuário a definição de alguns parâmetros relacionados à geração de código, tais como:

- **Destination Folder:** indica o diretório onde o código será gerado.
- **Generate Database Structures:** indica se as tabelas e suas respectivas funções de acesso devem ser geradas. Em caso negativo, as opções definidas em *Database Information* tornam-se desabilitadas.
- **Driver Class:** indica o nome da classe que fará o papel de interface (*driver*) de acesso à base de dados.
- **Database Location:** indica a localização da base de dados.
- **User:** nome do usuário da base de dados.
- **Password:** senha utilizada pelo usuário da base de dados.
- **Validate XML:** indica se o *parser* XML utilizado na função para desserialização (*unmarshalling*) validará ou não documentos XML.

7.3 Ferramenta de Definição dos Mapeamentos

Para agilizar o processo de integração entre os dados legados e o MR adotado por uma EV (etapa de instalação da plataforma), foi desenvolvida uma ferramenta que auxilia o usuário no processo de definição dos mapeamentos necessários (seção 6.2.1). O modelo de implementação desta ferramenta possui alguns aspectos em comum com a ferramenta de geração de código, pois também será responsável por processar uma especificação XML. Por outro lado, ao contrário da ferramenta apresentada anteriormente, esta deve também extrair os metadados de uma base de dados previamente escolhida.

A Figura 38 abaixo ilustra os módulos/estruturas principais que compõem a ferramenta de definição dos mapeamentos.

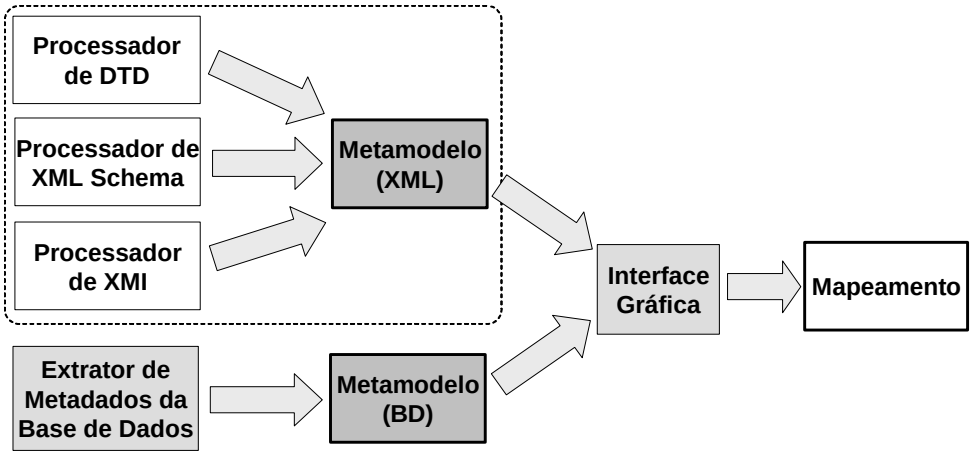

Figura 38: Módulos da ferramenta de definição dos mapeamentos.

Observando a Figura 38, percebe-se que por processar especificações XML, esta ferramenta conterá os mesmos processadores: DTD, XML *Schema* e XMI. Conseqüentemente, a especificação XML lida é armazenada no *metamodelo*, tal como é feito na ferramenta de geração de código.

Além das estruturas já apresentadas, há o *extrator de metadados da base de dados* que, como o próprio nome diz, obtém os metadados de todas as tabelas e campos de uma base de dados relacional. Em seguida estes metadados são armazenados em uma outra instância do *metamodelo*. Isto é feito assim porque o *metamodelo* é genérico o bastante para representar de forma neutra tanto um modelo XML quanto um modelo de relacional.

A conseqüência vantajosa desta abordagem é que é possível aproveitar as estruturas já implementadas para a outra ferramenta.

Os dois metamodelos são então apresentados na *interface gráfica* para que o usuário possa gerar a estrutura do *mapeamento*. As próximas seções descreverão as novas estruturas apresentadas: o *extrator de metadados da base de dados*, a *interface gráfica* e o *mapeamento*.

7.3.1 Extrator de Metadados da Base de Dados

Conforme dito anteriormente, um dos objetivos na implementação das ferramentas é que ambas possuíssem a flexibilidade no acesso a diversas bases de dados. Para isto, foi utilizada a API JDBC que provê acesso flexível a diferentes fontes de dados.

Dentre as diversas funcionalidades disponíveis nesta API, uma em especial foi utilizada na implementação do extrator de metadados: a interface *DatabaseMetadata*. Esta interface é implementada pelos diversos desenvolvedores de *drivers* e tem como função apresentar as capacidades específicas de cada tipo de base de dados (SUN, 2003d). Em outras palavras, a API do Java oferece uma interface uniforme onde informações a respeito de uma base de dados qualquer podem ser recuperadas, e cuja implementação é feita pelos desenvolvedores de cada tipo de base de dados.

Diferentes tipos de metadados podem ser recuperados através desta interface. Por exemplo, é possível saber quais tipos de dados podem ser usados em um comando "CREATE TABLE", ou se a base de dados suporta atualizações em lote (*batch updates*). Além disso, é possível obter o modelo de dados utilizado, ou seja, as tabelas, campos e as restrições de integridade (chaves) implementadas na base de dados, e é justamente isto que o extrator de metadados necessita. Alguns métodos desta interface (como os relacionados ao acesso ao modelo de dados) retornam os metadados na forma de um *ResultSet*, pois alguns destes estão armazenados em tabelas reservadas.

Por possuir uma estrutura genérica, o metamodelo (apresentado na seção 7.2.1) foi também utilizado para o armazenamento do modelo relacional da base de dados, ou seja:

- Informações a respeito das tabelas são recuperadas pelo método da API chamado "getTables(...)" e armazenadas em objetos da classe *MetaClass*;
- Informações sobre os campos de uma tabela são obtidas através do método "getColumns(...)", onde é passado o nome da tabela como parâmetro, e são salvas em objetos da classe *MetaField*;

- Informações sobre os relacionamentos entre os campos (chaves primárias e estrangeiras) são recuperadas pelo método “getCrossReference(...)”, onde os nomes das tabelas relacionadas são passados como parâmetros, e são salvas nos respectivos *MetaFields*.

7.3.2 Mapeamento

O mapeamento é armazenado em uma estrutura de classes conforme o diagrama apresentado na Figura 39.

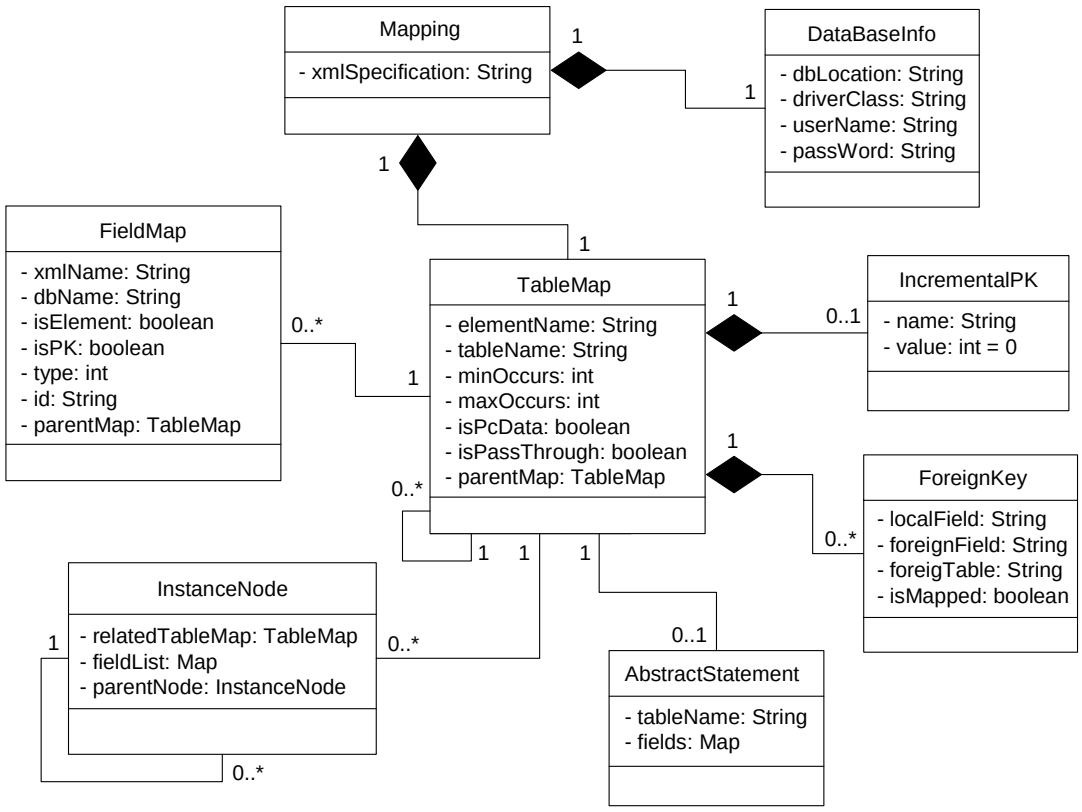

Figura 39: Diagrama de Classes do Mapeamento.

Conforme pode ser visto no diagrama, a estrutura principal é a classe “Mapping”, que agrega as demais classes. Um “Mapping” possui um atributo que armazena o nome da especificação XML e outro que guarda o tipo da especificação (DTD, XML *Schema*, XMI). Esta classe agrega a classe “DataBaseInfo”, responsável por armazenar os já conhecidos parâmetros relacionados à base de dados: a localização da base de dados, a classe que implementa o *driver* de acesso, o nome do usuário e a sua senha. Finalmente, a

classe “Mapping” possui um “TableMap”, que corresponde ao mapeamento para tabela relacionado ao elemento raiz da especificação XML.

Um “TableMap” possui uma lista de mapeamentos para campo (classe “FieldMap”), bem como uma lista de mapeamentos para tabela (representado pelo relacionamento reflexivo). Ela possui também uma lista de chaves estrangeiras (classe “ForeignKey”) e pode conter uma chave primária incremental (“IncrementalPK”). Cada uma das classes possui atributos correspondentes aos metadados apresentados no modelo conceitual.

Foi mencionado anteriormente que o mapeamento é salvo para que possa ser usado posteriormente. Nesta implementação o mapeamento é salvo em XML, que além de ser adequada para este propósito, serve para mostrar que este padrão pode ser aplicado nos diferentes aspectos de uma aplicação. Mais ainda: por esta razão a ferramenta de geração de código foi utilizada para gerar as classes que compõem o mapeamento (mostradas no diagrama da Figura 39). O DTD que define a estrutura do mapeamento em XML encontra-se no **Anexo A**, juntamente com um exemplo de documento XML.

7.3.3 Interface Gráfica

Um ponto positivo desta ferramenta é o fato de possuir uma interface gráfica amigável que auxilia o usuário no processo de definição dos mapeamentos (Figura 40).

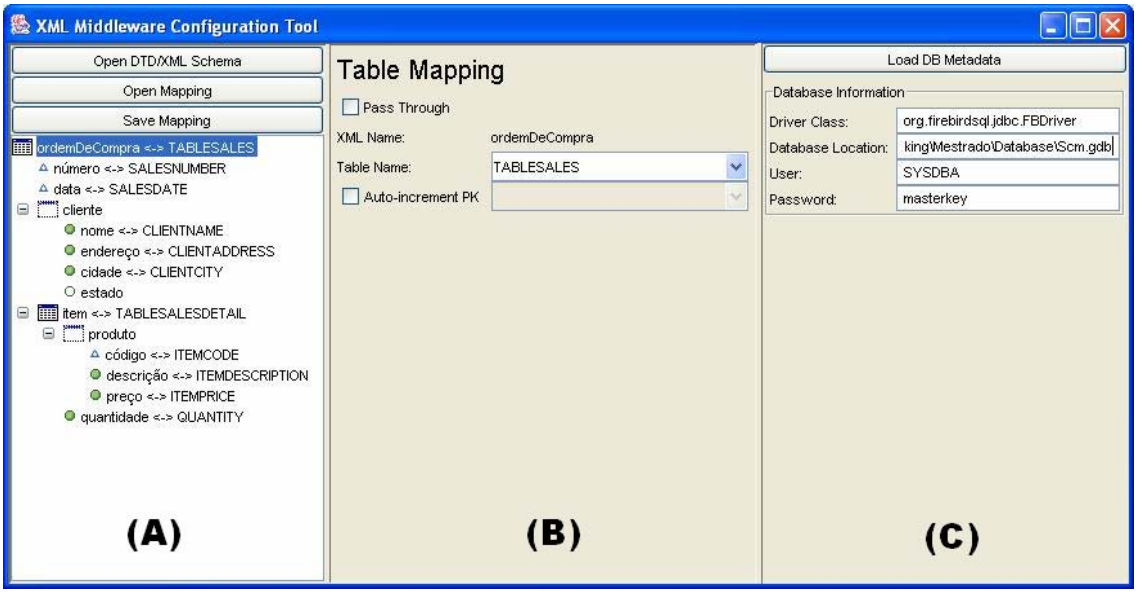

Figura 40: Interface gráfica da ferramenta de definição dos mapeamentos.

A interface pode ser dividida em três partes, conforme destacado na Figura 40:

A. Estrutura geral do mapeamento: possui botões para abrir uma especificação XML a ser mapeada, ou abrir e salvar mapeamentos já definidos. Quando um mapeamento é aberto, sua estrutura completa pode ser visualizada na forma de árvore, mostrando os elementos/atributos da especificação XML e qual tabela/campo estão relacionados (se for o caso). No caso de abrir uma especificação XML (para começar o processo de configuração), um mapeamento é gerado baseado na sua estrutura. A árvore que representa o mapeamento possui ícones diferentes de acordo com a estrutura mapeada:

-  : elemento (que deve ser mapeado para tabela) não mapeado.
-  : elemento (que deve ser mapeado para tabela) mapeado.
-  : elemento (que deve ser mapeado para campo) não mapeado.
-  : elemento (que deve ser mapeado para campo) mapeado.
-  : atributo (sempre é mapeado para campo) não mapeado.
-  : atributo mapeado.

B. Mapeamento selecionado: cada mapeamento selecionado na árvore que contém o mapeamento geral é visualizado nesta interface, sendo esta diferente para os dois tipos de mapeamento:

- **Mapeamento para tabela:** pode ser visto na Figura 40. Contém uma caixa de checagem chamada “PassThrough”, que indica que este elemento é mapeado ou ignorado. Por não estar mapeado ainda, o valor inicial indica que o elemento é ignorado. Ele contém ainda o nome do elemento e uma caixa de combinação (*combo box*) onde o usuário seleciona a tabela a ser mapeada. Por fim, há outra caixa de seleção onde o usuário define qual chave primária (da tabela selecionada) terá um valor incremental.
- **Mapeamento para campo:** a Figura 41 (B) ilustra um mapeamento para campo. Esta interface mostra o nome do elemento/atributo XML, seu tipo (se é atributo ou elemento), e uma caixa de combinação onde o usuário escolhe o campo a ser mapeado. **Nota:** caso o elemento exatamente superior (pai) a este elemento não foi sido mapeado, serão mostrados os campos da

tabela mapeada pelo primeiro elemento ascendente mapeado. Caso isto não aconteça, não será possível mapear este elemento/atributo para um campo.

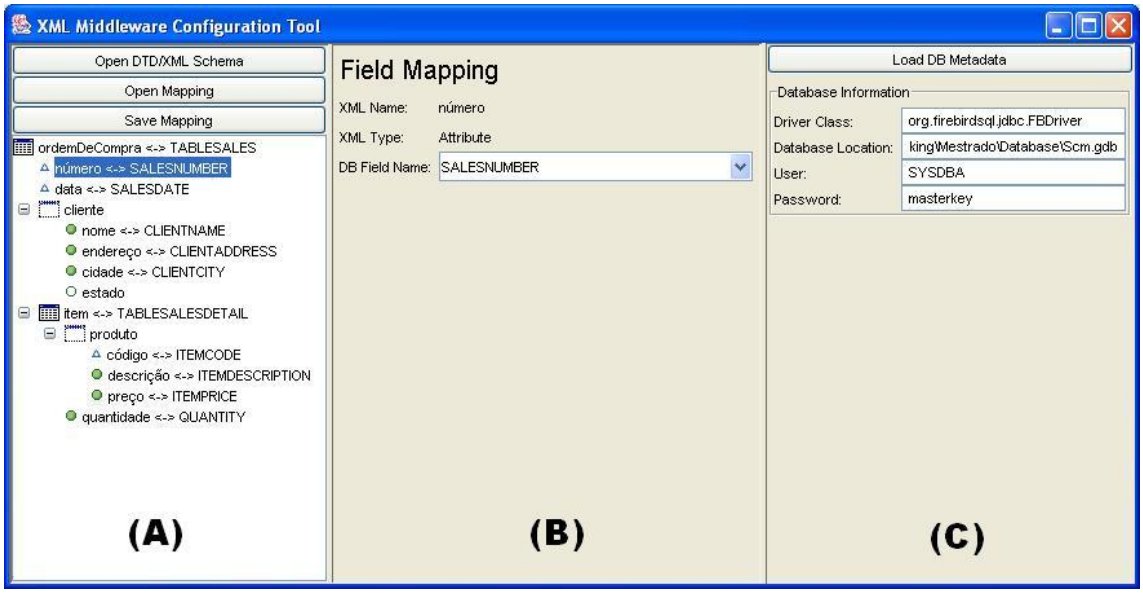

Figura 41: Mapeamento para campo selecionado na interface gráfica.

C. **Informações sobre a base de dados:** interface onde devem ser inseridas as já conhecidas informações sobre a base de dados a ser integrada. **Nota:** para que se possa começar o processo de configuração, ou seja, associar os elementos a tabelas/campos, é necessário antes carregar os metadados da referida base de dados.

O funcionamento geral da ferramenta é então o seguinte: o usuário abre uma especificação XML ou um mapeamento já definido. A estrutura do mapeamento é então mostrada na forma de árvore. A partir daí o usuário deve carregar os metadados da base de dados para que a ferramenta possa mostrar ao usuário as tabelas e campos disponíveis. Depois disso, o usuário pode iniciar o processo de mapeamento, mapeando primeiro uma tabela e depois os seus campos, pois a ferramenta só permite mostrar os campos de uma tabela já mapeada, e estes serão da tabela mapeada pelo elemento ascendente ao que deve ser mapeado como campo (conforme visto na seção 6.2.2, isto visa manter a consistência do mapeamento). Por fim, o usuário pode salvar o mapeamento em um arquivo, na forma de um documento XML. Os mapeamentos salvos têm a extensão “.map”, para que possam ser apropriadamente localizados pelo *middleware* XML implementado neste trabalho.

7.3.4 Conclusões sobre a Ferramenta de Definição dos Mapeamentos

A ferramenta apresentada nesta seção permite a definição dos mapeamentos que serão posteriormente utilizados por um *middleware* XML. No caso particular de uma EV, esta ferramenta é utilizada na etapa de instalação da plataforma, cujos mapeamentos auxiliarão o *módulo de integração* (que utiliza um *middleware* XML) no acesso e recuperação de dados das bases de dados legadas.

Como os mapeamentos são armazenados na forma de documentos XML, estes podem ser definidos e modificados através de um simples editor de textos. Entretanto, é altamente recomendável que se utilize esta ferramenta de modo e se evitar tanto erros de digitação quanto erros de inconsistência nos mapeamentos definidos.

Finalmente, percebe-se que o uso de ferramentas desse tipo associadas a *middlewares* XML minimizam o tempo e os custo do processo de configuração da integração dos dados. Mesmo que este processo ainda demande uma parcela de tempo considerável e de pessoal especializado, a consistência do mapeamento é garantida e, com o uso de um *middleware* XML, não há a necessidade de se escrever novas aplicações ou modificar aplicações já existentes. Na próxima seção será apresentada a implementação de um *middleware* XML que utiliza os mapeamentos definidos por esta ferramenta.

7.4 Biblioteca que implementa o Middleware XML

A idéia de se implementar o *middleware* XML na forma de uma biblioteca, ao invés de uma aplicação completa, surgiu devido a duas motivações principais. Primeiro porque como o modelo não contempla os aspectos de coordenação e comunicação, não se achou interessante desenvolver uma aplicação abrangendo um caso específico. Além disso, e mais importante, a definição de uma biblioteca permite a total liberdade de quem vier a implementar uma aplicação deste tipo (por exemplo, o *módulo de integração* da plataforma da EV), pois pode facilmente vincular esta biblioteca a qualquer aplicação Java que utilize um mecanismo de comunicação/coordenação em particular.

O *middleware* XML implementa o algoritmo de conversão entre dados codificados em XML e uma base de dados, conforme apresentado na seção 6.2.3, disponibilizando dois serviços básicos:

1. **Salvar um documento XML em uma base dados:** o *middleware* XML recebe um documento XML como parâmetro, e com o auxílio do mapeamento apropriado, gera os comandos SQL para armazenar os dados do documento na base de dados;
2. **Carregar um ou mais documentos XML de uma base de dados:** o *middleware* XML recebe uma consulta XML como parâmetro, e com o auxílio do mapeamento apropriado, gera uma consulta SQL equivalente e retorna o resultado da consulta na forma de um ou mais documentos XML.

7.4.1 A Classe XmlMiddleware

Embora conceitualmente o *middleware* XML disponibilize estes dois serviços, a sua implementação levou ao desenvolvimento de alguns outros, conforme pode ser visto no diagrama UML ilustrado na página a seguir (Figura 42). Este diagrama apresenta apenas a classe “XmlMiddleware”, que corresponde à classe pública, ou seja, a classe onde os serviços estão disponibilizados. Mais adiante serão mostradas as demais classes que compõem o *middleware* XML, cuja implementação serve de apoio aos serviços disponibilizados pela classe publica.

Conforme o diagrama, a classe “XmlMiddleware” possui apenas um atributo, chamado “mappingFolder”, que é uma referência ao diretório que contém os mapeamentos

utilizados pelo *middleware* XML. Esta referência é um objeto da classe “File”, disponível na biblioteca padrão do Java, utilizada para representar arquivos e diretórios.

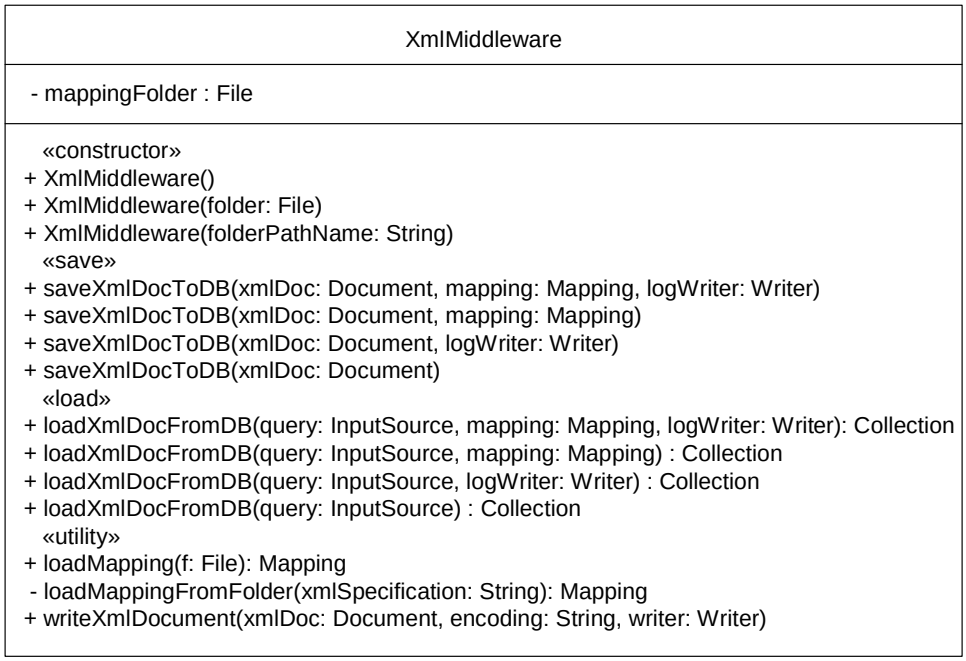

Figura 42: Classe que disponibiliza os serviços do *middleware* XML.

7.4.1.1 Métodos

Os métodos são organizados em diferentes categorias, conforme a sua utilidade, ou seja, métodos construtores (*constructor*), de armazenamento (*save*), de carregamento (*load*) e de utilidade geral (*utility*), conforme é detalhado a seguir.

Construtores

Esta classe possui três métodos construtores:

1. *XmlMiddleware():* neste caso, como não é especificado o diretório que contém os mapeamentos, a cada operação de carregamento/armazenamento o mapeamento relacionado deve ser especificado.
2. *XmlMiddleware(folder: File):* recebe como parâmetro a referência do diretório que contém os mapeamentos utilizados pelo *middleware* XML. Esta referência é armazenada no atributo “mappingFolder”.
3. *XmlMiddleware(folderPathName: String):* recebe como parâmetro uma *string* (texto) que contém o caminho (*path*) do diretório que contém os mapeamentos utilizados pelo *middleware* XML. Ele então instancia uma referência ao diretório

(classe File), e passa de parâmetro para o construtor anterior. Este método pode lançar a exceção “NotDirectoryException”.

Método para armazenamento de dados

A classe sobrecarrega os parâmetros do método “saveXmlDocToDB” de modo a cobrir diferentes combinações de parâmetros, possibilitando uma maior flexibilidade quanto ao uso deste serviço. Há, portanto, quatro implementações deste método, o qual pode ter alguns ou todos os seguintes parâmetros, conforme pode ser visto no diagrama da Figura 42:

- **xmlDoc:** um objeto da classe “Document” (disponível na biblioteca padrão do Java) que corresponde a um DOM (*Document Object Model*, ver seção 3.4.5.1). Como este parâmetro representa um documento XML, naturalmente ele está presente em todas as implementações deste método. Os demais parâmetros são opcionais.
- **mapping:** objeto da classe “Mapping” (apresentado na seção 7.3.2), e representa o mapeamento relacionado ao documento XML representado por “xmlDoc”. Caso este parâmetro seja omitido, o *middleware* XML procura um mapeamento no diretório definido no método construtor (referenciado pelo atributo “mappingFolder”).
- **logWriter:** objeto de uma classe descendente da classe abstrata “Writer” (também disponível na biblioteca do Java). Como já dito no final da seção 7.2.3.1, um *writer* é utilizado para escrever em fluxos de caracteres, como por exemplo em um arquivo ou uma *string*. Este parâmetro é utilizado caso a aplicação que utilize o *middleware* XML requeira algum tipo de registro (como um arquivo de *log*) das operações realizadas. Assim, caso este parâmetro esteja presente, ele registrará todas as operações de conversão XML-SQL realizadas.

Método para carregamento de dados

De modo análogo ao método de armazenamento, a classe sobrecarrega os parâmetros do método “loadXmlDocFromDB”, que contém também quatro implementações diferentes, de acordo com os atributos utilizados:

- **query:** um objeto da classe “InputSource” (disponível na biblioteca padrão do Java) que conforme visto no final da seção 7.2.3.1, permite encapsular informações

a respeito de uma fonte de dados codificada em XML. Como este parâmetro representa uma consulta XML, naturalmente ele está presente em todas as implementações deste método. Os demais parâmetros são opcionais.

- **mapping:** objeto da classe "Mapping" (apresentado na seção 7.3.2), e representa o mapeamento relacionado à consulta XML representada por "query". Caso este parâmetro seja omitido, o *middleware* XML procura um mapeamento no diretório definido no método construtor (referenciado pelo atributo "mappingFolder").
- **logWriter:** idem ao método para armazenamento de dados.

Métodos de utilidade geral

A classe implementa três métodos de cunho geral, que definem serviços adicionais, tanto para suporte aos métodos de carregamento/armazenamento, quanto para auxílio das aplicações que utilizam o *middleware* XML. São eles:

1. *loadMapping(f: File):* utilizado para carregar um mapeamento (classe "Mapping") a partir do arquivo referenciado pelo parâmetro "f". É utilizado internamente pelo *middleware* XML, mas sendo um método público, pode ser acessado externamente.
2. *loadMappingFromFolder(xmlSpecification: String):* método privado utilizado no processo de carregamento/armazenamento quando não há um mapeamento definido explicitamente. Ele procura no diretório referenciado pelo atributo "mappingFolder", e retorna o mapeamento correspondente (se existir).
3. *writeXmlDocument(xmlDoc: Document, encoding: String, writer: Writer):* utiliza o atributo "writer" para escrever um objeto DOM ("xmlDoc"). Pode ser utilizado no processo de carregamento/armazenamento para escrever o *log*, mas também pode ser acessado externamente pela aplicação para propósitos gerais, como a escrita dos documentos retornados no carregamento de dados. Um parâmetro adicional é o "encoding" onde é definido o padrão de codificação de caracteres a ser utilizado na escrita do documento XML.

7.4.1.2 Exceções

Na seção anterior foi descrito o funcionamento normal dos métodos disponibilizados pelo *middleware* XML. Normal porque não leva em conta eventuais erros que possam acontecer. Isto é importante, visto que os diferentes aspectos manipulados pelo *middleware* XML podem gerar situações de erro – as *exceções* – e estas devem ser tratadas

apropriadamente. Estes aspectos cobrem o acesso à base de dados, erros em documentos XML, acesso a arquivos, etc., e para cada caso foi utilizado ou definido um tipo específico de exceção que deve ser tratada caso venha a ocorrer. As exceções foram divididas em dois grupos: um com exceções específicas que podem ocorrer nesta implementação de *middleware* XML, e outro com exceções de cunho mais geral e já implementadas na biblioteca padrão do Java.

Exceções Específicas do Middleware XML

De modo a permitir que a aplicação que utiliza o *middleware* XML possa capturar todas as exceções desta categoria, foi criada uma exceção "XmlMiddlewareException", da qual todas descendem, conforme pode ser visto no diagrama UML da Figura 43.

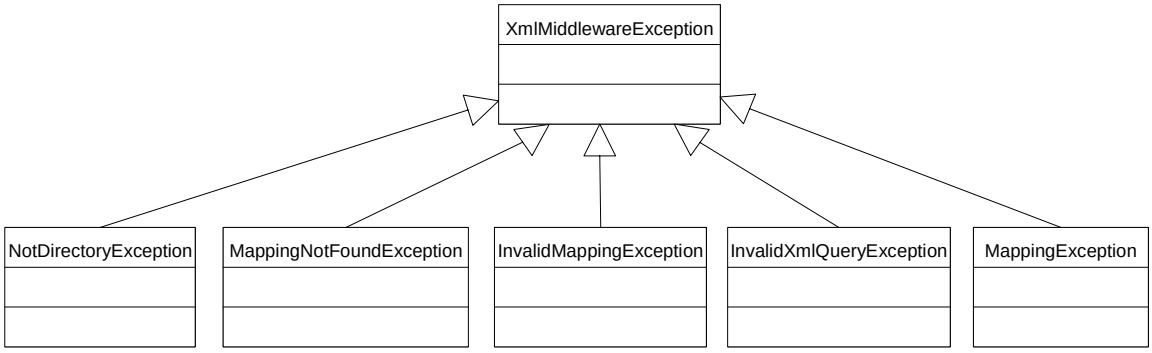

Figura 43: Hierarquia de exceções específicas ao *middleware* XML.

- **NotDirectoryException:** pode ser lançada pelo método construtor caso a referência ao diretório de mapeamentos não seja de fato um diretório.
- **MappingNotFoundException:** lançada se o arquivo XML que contém o mapeamento não está localizado no diretório de mapeamentos ("mappingFolder") ou se o atributo "mappingFolder" não foi definido pelo método construtor. Lançada pelos métodos de armazenamento/carregamento ("saveXmlDocToDB" e "loadXmlDocFromDB") e "loadMappingFromFolder".
- **InvalidMappingException:** se o arquivo XML do mapeamento está fora de contexto (ou seja, não relacionado ao "mapping.dtd") ou contém erros. Lançada pelo método "loadMapping".

- **InvalidXmlQueryException:** se a consulta XML está fora de contexto (ou seja, não relacionado ao “XMLQuery.dtd”) ou contém erros. Lançada pelo método “loadXmlDocFromDB”.
- **MappingException:** lançada se existem inconsistências entre a estrutura de dados XML (documento ou consulta XML) e o mapeamento. Por exemplo, um *xpath* inválido, elementos mapeados inapropriadamente, etc. É lançada pelos métodos de armazenamento/carregamento.

Exceções Livres de Contexto

Por acessar bases de dados, arquivos e documentos XML, uma série de outras exceções podem ocorrer, exceções que estão livre do contexto do *middleware* XML. São elas:

- **FileNotFoundException:** ocorre quando um arquivo referenciado por um objeto do tipo “File” não existe. Lançado pelo método “loadMapping”.
- **ClassNotFoundException:** ocorre quando uma classe que deve ser instanciada não é encontrada. No caso específico do *middleware* XML isto ocorre caso a classe que implementa o *driver* de acesso à base de dados não é encontrada. É lançada pelos métodos de armazenamento/carregamento, pois são os responsáveis pelo acesso à base de dados.
- **SQLException:** lançada quando há algum erro no acesso à base de dados. Esta exceção é bem genérica e pode ser lançada devido a diversos fatores, como por exemplo, uma consulta que referencia uma tabela que não existe. É lançada pelos métodos de armazenamento/carregamento, pois são os responsáveis pelo acesso à base de dados.
- **IOException:** lançada quando algum tipo de exceção de entrada/saída (I/O) ocorre. Nesta implementação ela pode ser lançada pelo objeto *writer* que é utilizado no *log* dos métodos de armazenamento/carregamento ou pelo *writer* do método “writeXmlDocument”.

Obs: há mais uma exceção pré-definida, chamada “SAXException”, lançada quando há algum erro apontado pelo *parser* SAX que é utilizado para processar os mapeamentos e as consultas XML. Entretanto esta exceção é capturada, sendo substituída por

"InvalidMappingException" e "InvalidXmlQueryException" que indicam mais claramente onde ocorreu um erro (no mapeamento ou na consulta XML).

7.4.2 Classes que provêm suporte ao Middleware XML

Adicionalmente à classe "XmlMiddleware", que corresponde à classe pública onde os serviços estão disponibilizados, há uma série de outras classes responsáveis por resolver detalhes internos que darão suporte aos serviços do *middleware* XML. A Figura 44 mostra o diagrama UML contento as principais classes que implementam o *middleware* XML.

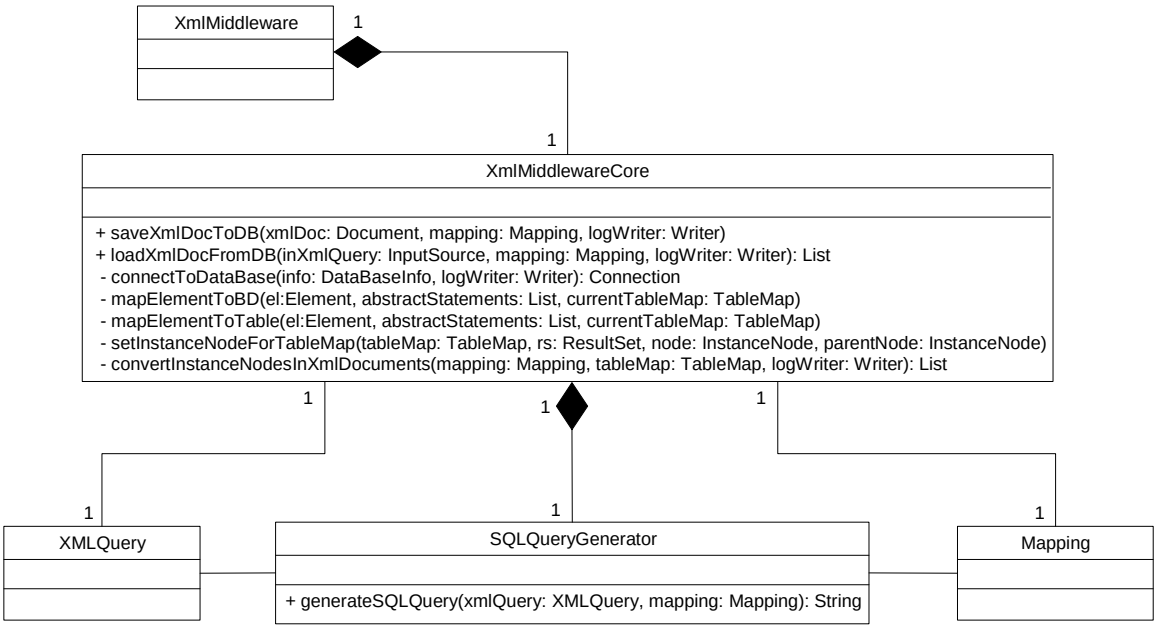

Figura 44: Classes que implementam o *middleware* XML.

Este diagrama apresenta a classe "XmlMiddleware" (a classe pública, que disponibiliza os serviços) agregando a classe "XmlMiddlewareCore". Esta classe é a que realmente implementa os serviços do *middleware* XML, sendo que a classe "XmlMiddleware" simplesmente repassa as chamadas de métodos para a classe "XmlMiddlewareCore". Isto é feito desta maneira para encapsular diversos métodos adicionais que servem de suporte à implementação dos serviços, como a conexão com a base de dados, métodos recursivos para a navegação da árvore (DOM) de um documento XML, etc. Como estes métodos não são relevantes a quem utilizará os serviços disponíveis, decidiu-se que estes ficassem em uma classe privada.

A classe “XmlMiddlewareCore” agrega a classe “SQLQueryGenerator”, que é utilizada para converter uma consulta XML em uma consulta SQL quando do carregamento de dados. Esta classe também é privada, pois a sua implementação também não é relevante à aplicação que utilizará o *middleware* XML. Além disso, esta classe utiliza um modelo de classes que representa uma consulta XML, cuja classe principal é chamada “XMLQuery”. Como foi visto, o método para carregar dados disponibilizado pelo *middleware* XML utiliza um objeto “InputSource” como formato para representar uma consulta XML. Este “InputSource” (que representa uma consulta XML) é processado tendo como resultado uma instância da classe “XMLQuery” como representação interna da consulta XML. Esta classe representa a estrutura da linguagem de consulta XML definida na seção 6.2.3. Tal como foi feito para o mapeamento, esta e outras classes relacionadas foram geradas pela ferramenta de geração de código a partir do DTD que define esta linguagem (**Anexo A**).

7.4.3 Conclusões sobre o Middleware XML

O *middleware* XML desenvolvido neste trabalho pode ser utilizado por qualquer parte interessada, e um fator que facilita a sua utilização é que os parâmetros que são passados para ele são representados de forma padronizada, ou seja:

- Utiliza objetos DOM (classe “Document” do Java) para representação de documentos XML, tanto no armazenamento quanto no carregamento.
- Para a representação das consultas XML, utiliza objetos “InputSource”, que permitem representar documentos XML de diversas formas.
- Utiliza um objeto “Writer” para escrever o *log* das operações, que pode ser implementado conforme a necessidade específica de cada aplicação.

Por outro lado, o *middleware* XML possui algumas limitações que por uma questão de tempo não puderam ser resolvidas, tais como:

- Os documentos XML que são armazenados ou carregados pelo *middleware* XML são representados na forma de objetos DOM. Embora isto não seja propriamente uma limitação, a utilização da abordagem DOM possui um desempenho inferior em relação à abordagem SAX. Uma das alterações para novas versões do *middleware* XML consiste em utilizar também esta abordagem, e neste caso os documentos estarão representados na forma de objetos “InputSource”.

- Não trata das diferentes codificações de caracteres. Tal como apresentado, XML permite utilizar diferentes codificações de caracteres (*encoding*), como o *Unicode*. A maioria das bases de dados não suporta *Unicode*, sendo necessário um tratamento adicional por parte do *middleware* XML para tratar e converter os caracteres necessários.
- Não permite a conversão entre unidades de medida, tais como unidades do sistema métrico e formatos regionais para data e hora (calendário). Estas conversões devem ser implementadas em versões futuras do *middleware* XML.

Além disso, esta implementação não trata de dados binários, que na base de dados são salvos em campos do tipo *BLOB*. Entretanto, o uso de dados binários é pouco utilizado (principalmente em documentos XML), não consistindo uma limitação muito importante. BOURRET et al. (2000) apresenta um *middleware* XML que trabalha também com dados binários.

No próximo capítulo será apresentada uma aplicação simples cujo propósito foi o de testar o *middleware* XML aqui implementado.

8 Avaliação das Ferramentas Implementadas

Este capítulo descreve os testes realizados para avaliar as ferramentas apresentadas no capítulo anterior, apontado os pontos positivos e eventuais limitações práticas. Basicamente, foram feitos três tipos de testes, que abrangem as diferentes formas de atuação de cada ferramenta. São eles:

1. Testes na geração de código, usando a *ferramenta de geração de código* (*XML Data Binding*), que atua na etapa de concepção da plataforma da EV.
2. Geração de mapeamentos entre DTDs e uma base de dados, para testar a *ferramenta de definição dos mapeamentos* (etapa de instalação da plataforma).
3. Testes de integração, usando o *middleware* XML com o auxílio dos mapeamentos definidos. Para estes testes, foi implementada uma aplicação simples que simula o *módulo de integração* (demais etapas do ciclo de vida de uma EV).

Para os testes, foi utilizado o sistema de banco de dados *Interbase 6* (INTERBASE, 2003), que representou uma alternativa com boa relação custo-benefício, pois é independente de plataforma, possui código aberto e sua licença é pública. Para o acesso ao banco de dados por parte das ferramentas (que são implementadas em Java), foi utilizado o *driver* JDBC *Firebird* (FIREBIRD, 2003). Além dos testes realizados, no final deste capítulo há uma seção que discute sobre estimativas de ganho em termos de agilidade através da utilização destas ferramentas.

8.1 Geração de Código

Para avaliar a ferramenta de geração de código, vários testes foram realizados. Dentre estes, dois foram mais relevantes: o primeiro consistiu na implementação de parte da plataforma desenvolvida no projeto MyFashion (MYFASHION, 2003); o outro serviu para auxiliar a implementação de alguns módulos da ferramenta de definição dos mapeamentos e do *middleware* XML.

8.1.1 Projeto MyFashion

Conforme foi mencionado na Capítulo 1, parte do trabalho foi desenvolvida no âmbito do projeto de cooperação internacional *MyFashion.eu*, terminado em junho de

2004. A seguir será feita uma breve explicação do projeto para contextualizar onde foi utilizada a ferramenta de geração de código.

O Projeto MyFashion.eu teve como objetivo prover modelos de negócio inovadores para dar suporte ao gerenciamento de cadeias de fornecimento dinâmicas[26]. Ele atua na indústria de produtos têxteis feitos sob medida, ou seja, a cada pedido feito por um cliente, uma nova EV é criada especialmente para atender a este pedido. Isto resulta em um fluxo de produção de uma peça só, fornecendo aos consumidores produtos personalizados e de alta qualidade, com preços adequados e tempo de entrega compatível com este tipo de produção.

O projeto conta com parceiros de diversos países da Europa, tendo o grupo GSIGMA/DAS/UFSC como o único parceiro não europeu. Estes parceiros foram agrupados conforme o seu papel:

- **Coordenador do projeto:** FIR, da Alemanha.
- **Coordenador técnico:** INESC Porto, de Portugal.
- **Responsáveis pela exploração dos resultados:** ACE HOLDING, de Portugal; EURO IT&C, da Holanda.
- **Empresas piloto:** In.Ca.Ma, da Itália; Silva & Sistelo, de Portugal; Odermark, da Alemanha; MSC Shoe Corporation, da Suíça.
- **Desenvolvedores:** INESC Porto e Wintouch Sistemas de Informação, de Portugal; EURO IT&C e Possen, da Holanda; UFSC/DAS/GSIGMA, do Brasil.

A plataforma desenvolvida é baseada na *Web*, e consiste de módulos que se comunicam através de *Serviços Web* (mencionados na seção 3.3). A plataforma também possui um *portal* que disponibiliza uma série de funcionalidades para usuários autenticados. A Figura 45, na página a seguir, ilustra a arquitetura simplificada da plataforma MyFashion.

As funcionalidades disponíveis no Portal MyFashion são implementadas por dois módulos: o POS Web (*Point of Sale*, Ponto de Venda) que permite, entre outras coisas, que o vendedor possa entrar com os pedidos (que contêm as medidas específicas) do cliente; e o SC^2 (*Supply Chain Smart Coordination*), que é um sistema que provê suporte ao gerenciamento e coordenação por parte o coordenador da EV.

[26] Para efeitos de simplificação, neste nível de projeto de sistemas de suporte, “cadeias de fornecimento” (*supply chain*) podem ser vistas como sinônimos de “empresas virtuais” (EV).

Adicionalmente, há o DWFMS (*Distributed Workflow Management System*), que é um sistema de *workflow* responsável pelo gerenciamento dos processos de negócio (por exemplo, um novo pedido feito pelo cliente). Há também o POS Local, que representa os sistemas de POS já implantados e que foram integrados à arquitetura. A comunicação entre os módulos é feita através de *Serviços Web*.

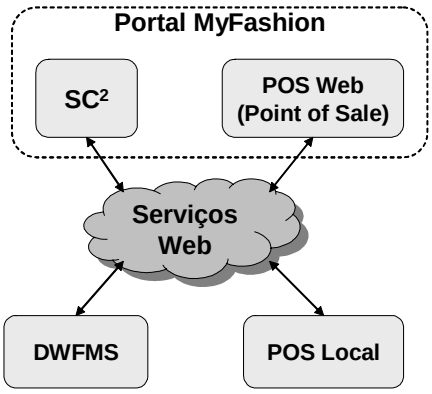

Figura 45: Arquitetura simplificada da Plataforma MyFashion.

Naturalmente, a plataforma desenvolvida é muito mais complexa, mas o seu detalhamento foge ao objetivo deste trabalho. O objetivo aqui é destacar o sistema SC^2, cujo desenvolvimento ficou a cargo do laboratório GSIGMA (Grupo de Sistemas Inteligentes de Manufatura) do Departamento de Automação e Sistemas da UFSC.

O SC^2 (*Supply Chain Smart Coordination*) é um sistema de suporte à decisão que visa oferecer um ambiente integrado para melhor gerenciar cadeias de fornecimento dinâmicas, cobrindo aspectos de produção, vendas e distribuição (RABELO *et al.*, 2002). O SC^2 foi concebido como um sistema multiagente na forma de uma aplicação que executa em ambiente Windows, e como foi desenvolvido no âmbito do Projeto DAMASCOS (DAMASCOS, 2004), está integrado aos demais módulos da arquitetura DAMASCOS através de uma plataforma baseada em CORBA e XML.

Para o Projeto MyFashion, o SC^2 foi remodelado para se adequar aos novos requisitos, ou seja, migrou de um ambiente local (executa apenas no ambiente Windows) para a Internet (Portal MyFashion), podendo ser acessado através de um navegador *Web*. O sistema foi então adaptado para executar em um servidor *Web* que disponibiliza páginas JSP (*Java Server Pages*) e *applets* Java. Foi neste desenvolvimento que a ferramenta de geração de código teve participação, o que serviu para testar o seu funcionamento.

Como o objetivo não é se aprofundar no funcionamento do SC^2, os detalhes acerca das suas funcionalidades não serão mencionados. Estes aspectos podem ser vistos em RABELO *et al.* (2002). Por outro lado, a ferramenta de geração de código auxiliou no desenvolvimento de duas das suas funcionalidades. Portanto, será feita uma breve explicação sobre o seu funcionamento. São elas: *Configurator* e *Smart Map*.

O *Configurator* permite a configuração, de forma visual e interativa, da topologia de uma EV, ou seja, é utilizado na definição dos relacionamentos entre os membros participantes, bem como os papéis que cada um deverá atuar no contexto desta EV. Esta configuração é feita na etapa de criação da EV.

O *Smart Map* permite a visualização de EVs já configuradas, onde é possível visualizar não apenas os membros e seus relacionamentos, mas também informações sobre as empresas e produtos envolvidos nos relacionamentos. Esta funcionalidade é utilizada durante a etapa de operação da EV.

Ambas as funcionalidades são apresentadas em maiores detalhes por BALDO (2003). Por serem funcionalidades que necessitam de interação constante com o usuário (interface gráfica), a forma mais natural de migração para o ambiente Web foi a implementação na forma de *applets* Java. A Figura 46 mostra a versão Web do *Smart Map*. O *Configurator* é bastante semelhante, mas permite, além da visualização, a criação e edição da topologia de uma EV. Por ser uma aplicação Web, as funcionalidades do SC^2 podem ser acessadas de qualquer máquina conectada à Internet e que possua um navegador Web (*browser*).

Uma questão que surgiu durante a implementação destas funcionalidades está relacionada à troca de informações que deve ser efetuada entre o SC^2 (localizado no servidor) e o *applet* (localizado no cliente). Isto é necessário, pois embora os dados da topologia de uma EV fiquem armazenados na base de dados do SC^2, estes são criados, modificados ou acessados pelas funcionalidades que executam no lado do cliente. Assim, a idéia inicial foi utilizar *serviços Web*, tal como é feito na comunicação entre os demais módulos da plataforma MyFashion. Entretanto, algumas limitações práticas dificultaram a sua implementação. A principal limitação encontrada refere-se ao tamanho total do *applet* e a conseqüente carga na rede quando do seu *download* pelo cliente. Originalmente, o *applet* possui menos de 100 Kbytes, mas a utilização de *serviços Web* implicaria no *download* adicional de uma biblioteca que ocupa mais de 2 Mbytes. Além disso, a utilização de *serviços Web* implica que o *applet* seja assinado digitalmente, pois embora

possa fazer conexões com o seu servidor, a plataforma de *serviços Web* instalada necessita que o *applet* faça algumas coisas a mais, como escrever em um arquivo e, portanto, o *applet* precisa estar autenticado para que possa realizar estas tarefas.

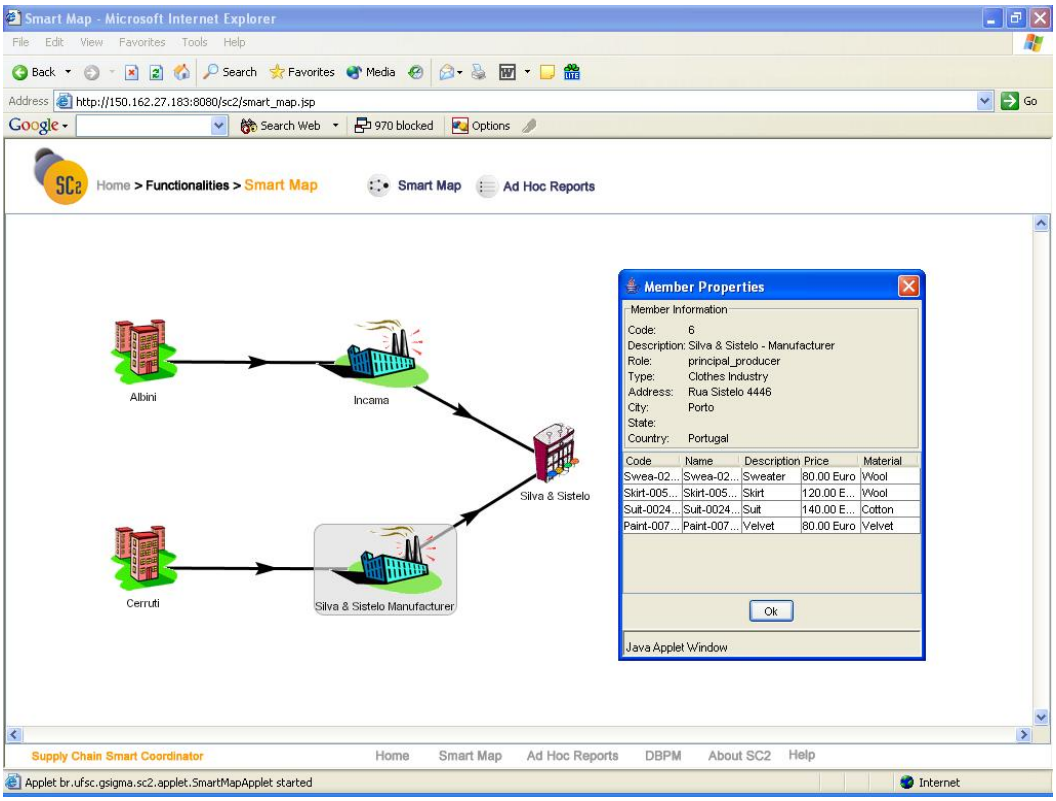

Figura 46: Versão Web da funcionalidade *Smart Map*.

Devido principalmente à primeira limitação, decidiu-se utilizar um vocabulário XML "proprietário" ao invés de *serviços Web*. Foi criada então uma especificação XML que define o formato para intercâmbio de dados entre o SC^2 (servidor) e o *applet* (cliente). Esta especificação foi utilizada na ferramenta de geração de código para a geração das classes (tanto do cliente quanto do servidor) e das tabelas da base de dados do servidor.

Desta forma, a topologia da EV configurada no cliente pode ser armazenada nos objetos cujas classes foram geradas, e estes podem ser serializados na forma de documentos XML e enviados via HTTP para o servidor. No servidor, o documento XML é desserializado e a topologia da EV pode ser salva nas tabelas geradas da base de dados. De maneira inversa, o cliente pode requisitar uma dada topologia ao servidor. Esta é carregada da base de dados, enviada para o cliente e visualizada, por exemplo, pelo *Smart Map* (Figura 47).

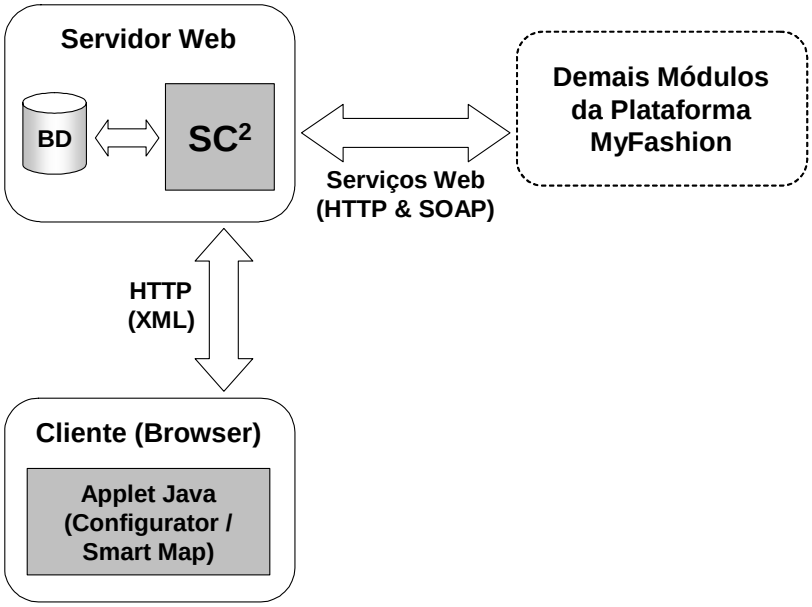

Figura 47: Modelo geral de acesso às funcionalidades do SC^2.

Como a estrutura necessária para representar a topologia da EV não exigia muita complexidade, um DTD foi suficiente como especificação XML, como pode ser visto na Figura 48. Este DTD define dois tipos básicos de documentos, ou seja, define dois elementos que são potenciais raízes:

1. O elemento “enterprises”, que corresponde a uma lista de empresas (representadas pelo elemento “enterprise”). Cada empresa tem um endereço (“address”), uma lista de contatos (“contact”) e uma lista de produtos (“product”);
2. O elemento “supplyChain”, que representa a topologia da EV. Ele contém uma lista de empresas (“enterprise”), uma lista de membros (“member”), uma lista de conexões entre os membros (“connection”), e uma lista de *clusters* ou agrupamentos de empresas (elemento “cluster”).

Portanto, os documentos XML válidos perante este DTD devem conter uma das duas opções: uma lista de empresas ou a topologia de uma EV.

```
<!ELEMENT enterprises (enterprise*)>
 <!ELEMENT enterprise (name, type, webSite, address, contact*, product*)>
 <!ATTLIST enterprise code CDATA #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT type (#PCDATA)>
  <!ELEMENT webSite (#PCDATA)>
  <!ELEMENT address (street, number, aditional, zipCode, poBox, city, region, state, country)>
   <!ELEMENT street (#PCDATA)>
   <!ELEMENT number (#PCDATA)>
   <!ELEMENT aditional (#PCDATA)>
   <!ELEMENT zipCode (#PCDATA)>
   <!ELEMENT poBox (#PCDATA)>
   <!ELEMENT city (#PCDATA)>
   <!ELEMENT region (#PCDATA)>
   <!ELEMENT state (#PCDATA)>
   <!ELEMENT country (#PCDATA)>
  <!ELEMENT contact (name, phone, fax, mail, function, department)>
   <!ELEMENT phone (#PCDATA)>
   <!ELEMENT fax (#PCDATA)>
   <!ELEMENT mail (#PCDATA)>
   <!ELEMENT function (#PCDATA)>
   <!ELEMENT department (#PCDATA)>
  <!ELEMENT product (name, description, price, material)>
  <!ATTLIST product code CDATA #REQUIRED>
   <!ELEMENT description (#PCDATA)>
   <!ELEMENT price (#PCDATA)>
   <!ATTLIST price currency CDATA #REQUIRED>
   <!ELEMENT material (#PCDATA)>

<!ELEMENT supplyChain (name, enterprise*, member*, connection*, cluster*)>
<!ATTLIST supplyChain code CDATA #REQUIRED>
 <!ELEMENT member (role, label, position, cluster?)>
 <!ATTLIST member code CDATA #REQUIRED>
 <!ATTLIST member enterpriseRef CDATA #REQUIRED>
  <!ELEMENT role (#PCDATA)>
  <!ELEMENT label (#PCDATA)>
  <!ELEMENT position (x, y)>
   <!ELEMENT x (#PCDATA)>
   <!ELEMENT y (#PCDATA)>
 <!ELEMENT connection (prodRef*)>
 <!ATTLIST connection code CDATA #REQUIRED>
 <!ATTLIST connection producerRef CDATA #REQUIRED>
 <!ATTLIST connection consumerRef CDATA #REQUIRED>
  <!ELEMENT prodRef (#PCDATA)>
 <!ELEMENT cluster (label, position)>
 <!ATTLIST cluster code CDATA #REQUIRED>
  <!ELEMENT memberRef (#PCDATA)>
```

Figura 48: DTD que define a topologia de uma EV.

8.1.1.1 O Código Gerado

O DTD mostrado na Figura 48 foi o parâmetro de entrada para a geração de um conjunto de classes e tabelas. O diagrama UML das classes geradas pode ser visto na Figura 49 a seguir.

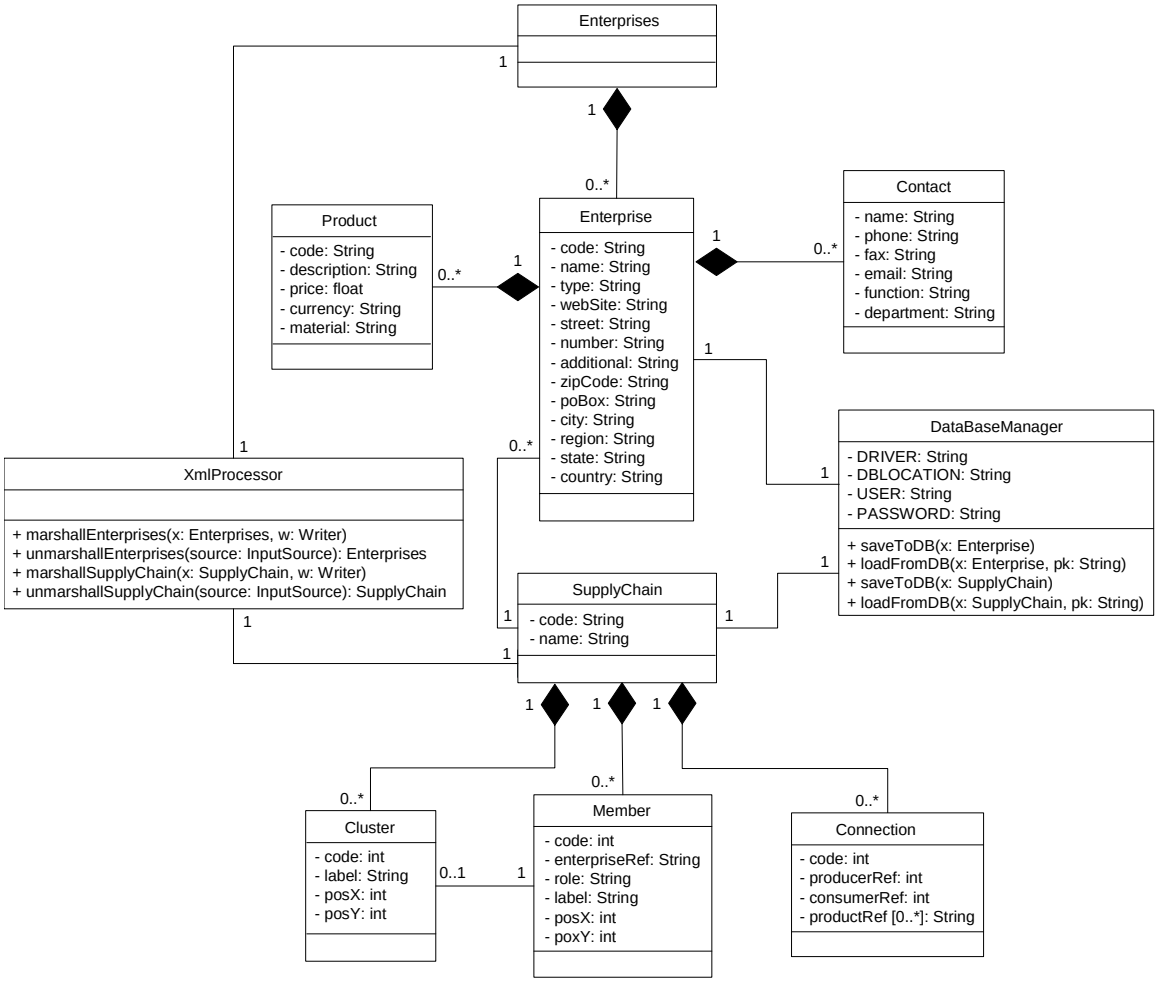

Figura 49: Classes geradas pela ferramenta de geração de código.

Observando o diagrama, percebe-se que as relações entre as classes respeitam a hierarquia dos elementos definidos pelo DTD, além de cada classe gerada possuir atributos relativos aos elementos e atributos XML correspondentes. Outro detalhe diz respeito aos tipos de dados que foram definidos para alguns atributos, como “posX” e “posY”, do tipo “int” (inteiro), ambos pertencentes às classes “Cluster” e “Member”. Como os atributos são declarados como privados, foram gerados também os métodos de acesso correspondentes, mas que foram suprimidos por uma questão de espaço.

Além das classes geradas a partir da estrutura definida pelo DTD, há também mais duas classes: “XmlProcessor”, responsável pelo processamento de documentos XML; e “DataBaseManager”, responsável pelo armazenamento dos objetos na base de dados.

A ferramenta também gerou uma série de tabelas na base de dados que têm relação com as classes geradas, conforme pode ser visto na Figura 50.

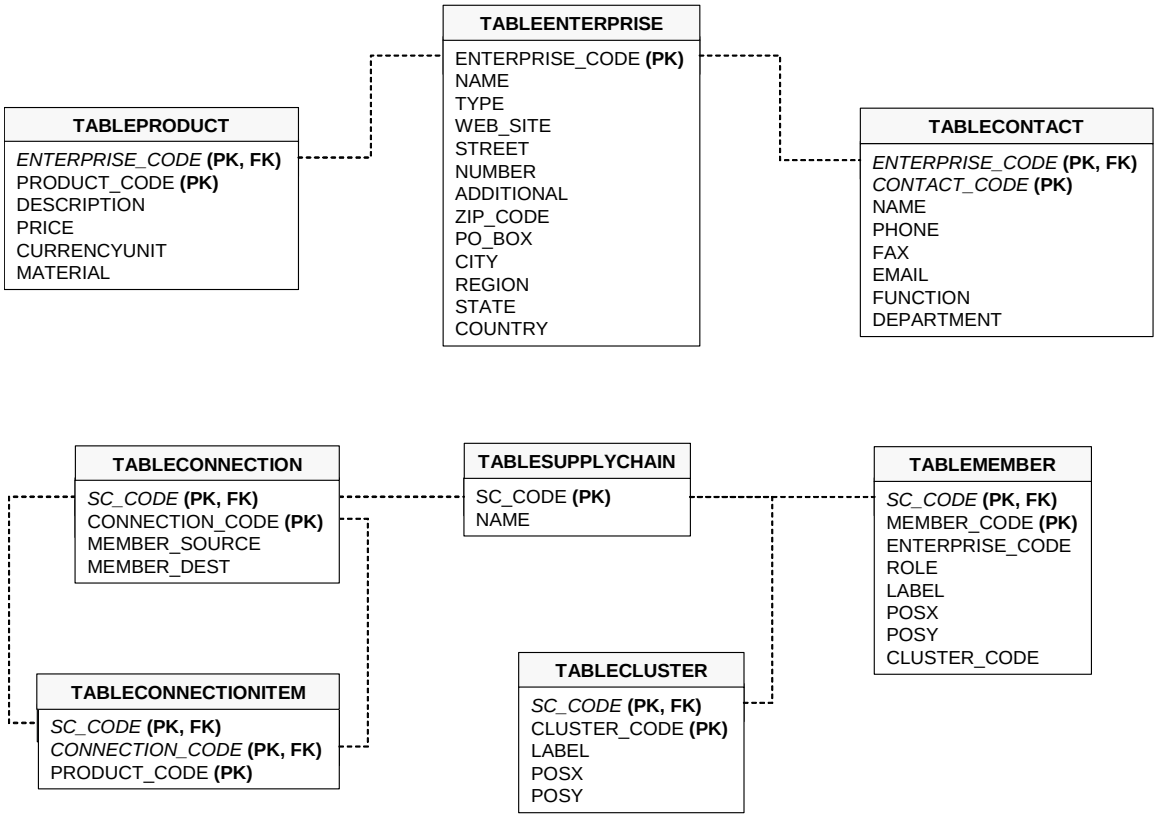

Figura 50: Tabelas geradas pela ferramenta de geração de código.

Nas tabelas apresentadas na Figura 50, as chaves primárias estão marcadas com um “PK”, e as chaves estrangeiras estão marcadas com um “FK” e ligadas por uma linha tracejada.

Os campos em itálico são os que foram gerados pela ferramenta, pois não existem no modelo de classes, mas são necessários no modelo relacional. Por exemplo, a classe “Contact” não possui um atributo que possa ser usado para identificá-la unicamente (chave primária). Assim, a ferramenta gera automaticamente um campo incremental para este papel (CONTACT_CODE). O mesmo ocorre com as chaves estrangeiras de todas as tabelas.

Sobre a geração das tabelas, duas observações são importantes. Em primeiro lugar, a tabela gerada para a classe “Enterprises” foi removida, visto que ela contém apenas uma lista de empresas (classe “Enterprise”). Isto foi feito manualmente, de modo a otimizar o código gerado. O outro detalhe diz respeito à tabela “TABLECONNECTIONITEM”, que foi gerada não a partir de uma classe, mas a partir do atributo “productRef” da classe “Connection”. Isto foi feito de forma automática, e ocorre porque o atributo pode conter uma lista de valores.

8.1.2 Geração de Classes para a Ferramenta de Definição dos Mapeamentos e para o Middleware XML

Outro teste considerado bastante relevante consistiu na geração de classes que foram utilizadas na implementação da ferramenta de definição dos mapeamentos e do *middleware* XML. Isto é importante, pois naturalmente uma ferramenta servirá para a testar da outra, ou seja, o bom funcionamento da última implicará na correta geração de código feita pela primeira.

Conforme mencionado no capítulo anterior, dois conjuntos de classes foram gerados semi-automaticamente: um que representa um *mapeamento* (seção 7.3.2) e outro que representa um consulta XML (seção 7.4.2).

8.1.3 Considerações sobre a Geração de Código

Apesar do código ser gerado de modo semi-automático, algumas alterações posteriores foram necessárias. Entretanto, isto é uma conseqüência natural, visto que alguns aspectos vão além do que pode ser representado em XML, tais como:

- As classes geradas geralmente agregam funcionalidades específicas à aplicação em questão. Por exemplo, no caso da geração das classes que representam um mapeamento, além das listas de mapeamentos para campo e de mapeamentos para tabelas (geradas pela ferramenta), foi necessário adicionar uma lista que contém todos os mapeamentos organizados em ordem (independente do seu tipo).
- Alguns relacionamentos entre as classes não são adequadamente representados em XML. Este aspecto é bastante dependente de cada aplicação em particular. No caso da geração das classes para a plataforma MyFashion, alguns atributos de classe gerados por elementos XML simples foram melhor representados como referências a classes ao invés de atributos de tipo simples.

Além dos aspectos dependentes do tipo de aplicação, algumas limitações foram detectadas, e que serão resolvidas futuramente. São elas:

- Falta de flexibilidade para escolher o que será ou não gerado. Neste teste, viu-se que não era necessário gerar a tabela relativa à classe “Enterprises”. Entretanto, a ferramenta não permitiu fazer esta restrição, e ela foi removida manualmente. É

desejável também que se possa fazer este tipo de restrição para as classes. Assim, a classe “Enterprises” poderia também não ser gerada.

- Limitações do tamanho para identificadores utilizados na base de dados. Foi detectado que o banco de dados *Interbase* possui um limite de no máximo 31 caracteres para os identificadores, o que pode levar a erros na geração das tabelas. Embora esta limitação não seja propriamente da ferramenta, é desejável que esta possa controlar o tamanho máximo dos identificadores a partir, por exemplo, de um parâmetro definido pelo usuário.

8.2 Integração de Dados

Nesta seção serão descritos os testes relacionados ao aspecto da integração dos dados provenientes de uma base de dados legada com os dados definidos por um MR em particular. Esta avaliação consiste de dois passos principais: o primeiro, mostrado na próxima seção, utiliza a ferramenta de definição dos mapeamentos; no passo seguinte (na seção subseqüente), uma aplicação – que utiliza o *middleware* XML – realiza a integração dos dados com o auxílio dos mapeamentos definidos.

8.2.1 Definição dos Mapeamentos

Para o primeiro passo do processo de integração, ou seja, a definição dos mapeamentos, é necessário ter-se dos dois modelos de dados, ou seja, o modelo de referência (MR), e o modelo de dados legado.

O MR utilizado para este teste consiste em três DTDs, que cobrem diferentes aspectos acerca de uma cadeia de fornecimento (*supply chain*), conforme mostrado na Figura 51. É preciso deixar claro que estes DTDs não cobrem completamente os requisitos de dados necessários a uma aplicação real de EVs, sendo escolhidos apenas com o propósito de testar os conceitos aqui apresentados. Além disso, o número de esquemas XML utilizados como MR é particular para cada tipo de aplicação.

Para facilitar o entendimento, foram mantidos apenas os elementos principais definidos por cada DTD. Os DTDs completos encontram-se no **Anexo B.1**. São eles:

- **enterprise_information:** contém informações gerais de uma empresa: seu nome, ramo de atuação, endereços, contatos, produtos, etc.

- **production_information:** define uma ordem de produção, e contém informações referentes a: número da ordem, data de criação, data de entrega, cadeia de fornecimento (*supply chain*) que está relacionada, lista de detalhes que contêm o produtos a serem produzidos, etc.
- **sales_information:** define uma ordem de compra (pedido), e contém as seguintes informações: número da ordem, cadeia de fornecimento (*supply chain*) que está relacionada, data de entrega, lista de detalhes que contêm os produtos pedidos, etc.

enterprise_information.dtd

```
<!ELEMENT enterprise_info (generic_info, product_info*)>
 <!ELEMENT generic_info (code_ent, ..., contact*, address*)>
  <!ELEMENT code_ent (#PCDATA)>
  ...
  <!ELEMENT contact (contact_name, ...)>
  <!ELEMENT contact_name (#PCDATA)>
  ...
  <!ELEMENT address (street, ...)>
   <!ELEMENT street (#PCDATA)>
   ...
 <!ELEMENT product_info (code_prod, ...)>
  <!ELEMENT code_prod (#PCDATA)>
  ...
```

production_information.dtd

```
<!ELEMENT production_order (supply_chain_id, order_number, ..., production_detail+)>
 <!ELEMENT supply_chain_id (#PCDATA)>
 <!ELEMENT order_number (#PCDATA)>
 ...
 <!ELEMENT production_detail (order_number, detail_number, ..., productSpecification)>
  <!ELEMENT detail_number (#PCDATA)>
  ...
  <!ELEMENT productSpecification (cod_prod, ...)>
   <!ELEMENT cod_prod (#PCDATA)>
   ...
```

sales_information.dtd

```
<!ELEMENT sales_order (supply_chain_id, ..., sales_detail+)>
 <!ELEMENT supply_chain_id (#PCDATA)>
 ...
 <!ELEMENT sales_detail (line_number, ...)>
  <!ELEMENT line_number (#PCDATA)>
  ...
```

Figura 51: MR utilizado para a definição dos mapeamentos.

Estes DTDs foram utilizados em um projeto anterior ao MyFashion (Projeto DAMASCOS) e fazem parte do chamado “DBP (*Distributed Business Process*) Model” (RABELO *et al.*, 1999).

O próximo passo consiste em fazer os mapeamentos entre os DTDs e o modelo de dados de uma base de dados legada. Para este teste, como não havia a disponibilidade de uma base de dados "real", foi utilizada a base de dados do SC[2] (versão MyFashion), cuja estrutura possui muitos aspectos em comum com a estrutura definida pelos DTDs. Seu modelo pode ser visto na Figura 52.

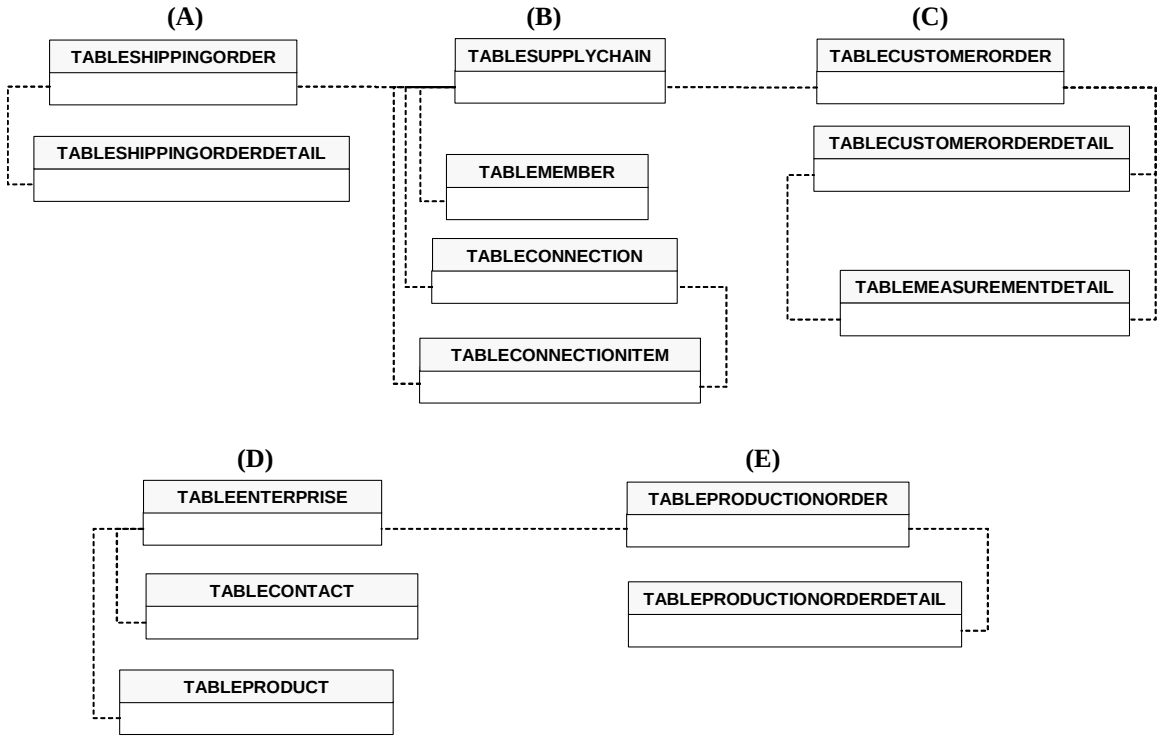

Figura 52: Modelo da base de dados utilizada para o teste de integração.

Novamente, para facilitar o entendimento, a maioria dos detalhes foi omitida, sendo mostrados apenas as tabelas e seus relacionamentos. Um modelo mais abrangente desta base de dados pode ser visto no **Anexo B.2**. Conforme a Figura 52, a tabelas estão organizadas de acordo com o tipo de informações que armazenam, que são:

A) Ordens de distribuição: armazenadas na tabela "TABLESHIPPINGORDER", cujos detalhes (produtos) estão na tabela "TABLESHIPPINGORDERDETAIL".

B) Estrutura da *Supply Chain*: armazenada na tabela "TABLESUPPLYCHAIN". As empresas que compõem esta *supply chain* (os membros) estão na tabela "TABLEMEMBER", os relacionamentos entre os membros estão na tabela "TABLECONNECTION", e os produtos intercambiados nos relacionamentos estão em "TABLECONNECTIONITEM".

C) Ordens de compra (pedidos): estão na tabela “TABLECUSTOMERORDER”, cujos detalhes (produtos) estão na “TABLECUSTOMERORDERDETAIL”. Cada detalhe pode ter uma ou mais informação de medidas (os produtos são personalizados), que ficam na tabela “TABLEMEASUREMENTDETAIL”.

D) Informações sobre empresas: informações cadastrais (nome, endereço, etc.) de uma empresa estão salvas na tabela “TABLEENTERPRISE”, sua lista de contatos, na tabela “TABLECONTACT”, e seus produtos, na tabela “TABLEPRODUCT”.

E) Ordens de produção: estão na tabela “TABLEPRODUCTIONORDER”, cujos detalhes (produtos a serem produzidos) estão armazenados na tabela “TABLEPRODUCTIONORDERDETAIL”.

Observando o modelo de dados, é possível ter uma idéia geral de quais tabelas serão mapeadas para os DTDs. A Figura 53 a seguir mostra os mapeamentos realizados.

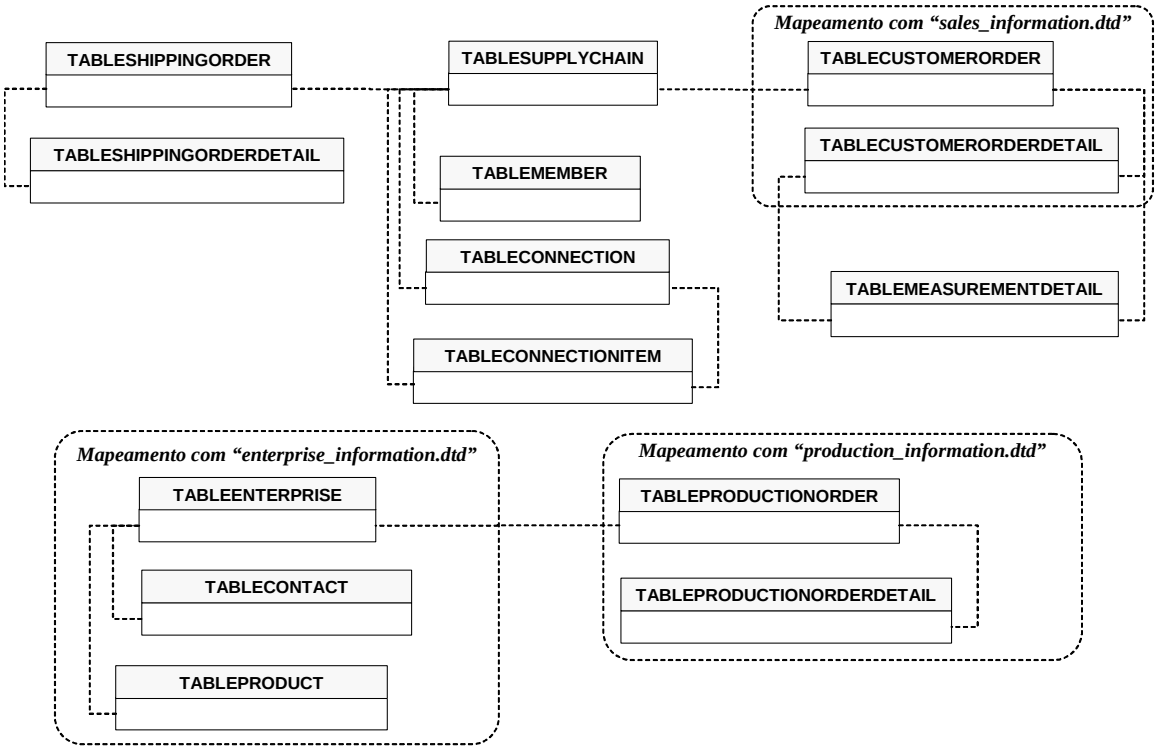

Figura 53: Mapeamentos entre o MR (DTDs) e o modelo da base de dados.

Foram feitos três mapeamentos, relativos a cada um dos DTDs, e que estão relacionados a algumas tabelas do modelo de dados legado. Repare que o mapeamento com o DTD “sales_information” não cobre a tabela “TABLEMEASUREMENTDETAIL”

porque o DTD não possui este tipo de informação. Os mapeamentos podem ser vistos de forma simplificada na Figura 54, pois além de ocuparem um espaço muito grande no texto, o seu detalhamento não é obrigatório para o entendimento do que está sendo descrito. Para maiores detalhes, os mapeamentos encontram-se no **Anexo B.3** na forma de documentos XML.

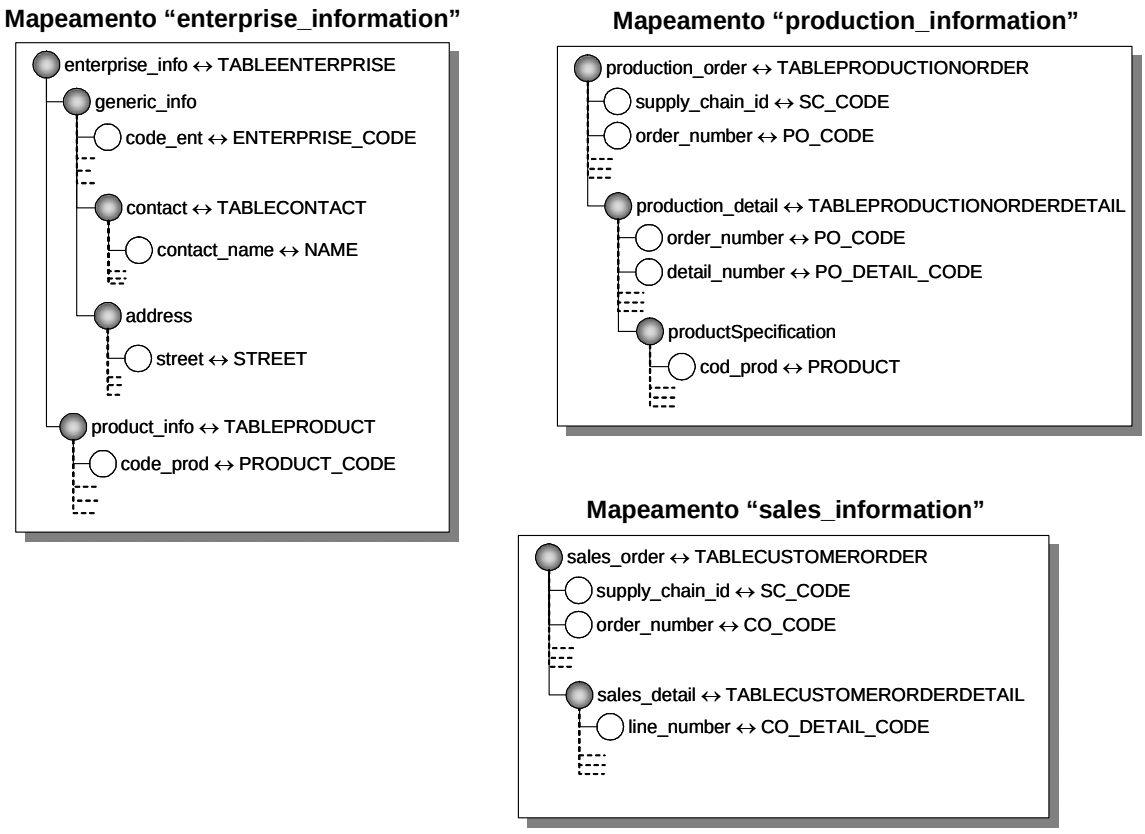

Figura 54: Mapeamentos definidos para o teste de integração.

A próxima seção apresentará uma aplicação simples que utiliza o *middleware* XML desenvolvido neste trabalho, com o propósito de testar o processo de integração com os três mapeamentos definidos.

8.2.2 Testes de Integração

Após a definição dos mapeamentos, o *middleware* XML está apto a realizar o processo de integração. Para testá-lo, foi desenvolvida uma aplicação simples, que permite salvar documentos XML na base de dados configurada nos mapeamentos, ou carregar

documentos XML contendo dados extraídos da mesma. No segundo caso são utilizadas consultas XML. A Figura 55 a seguir mostra a interface desta aplicação.

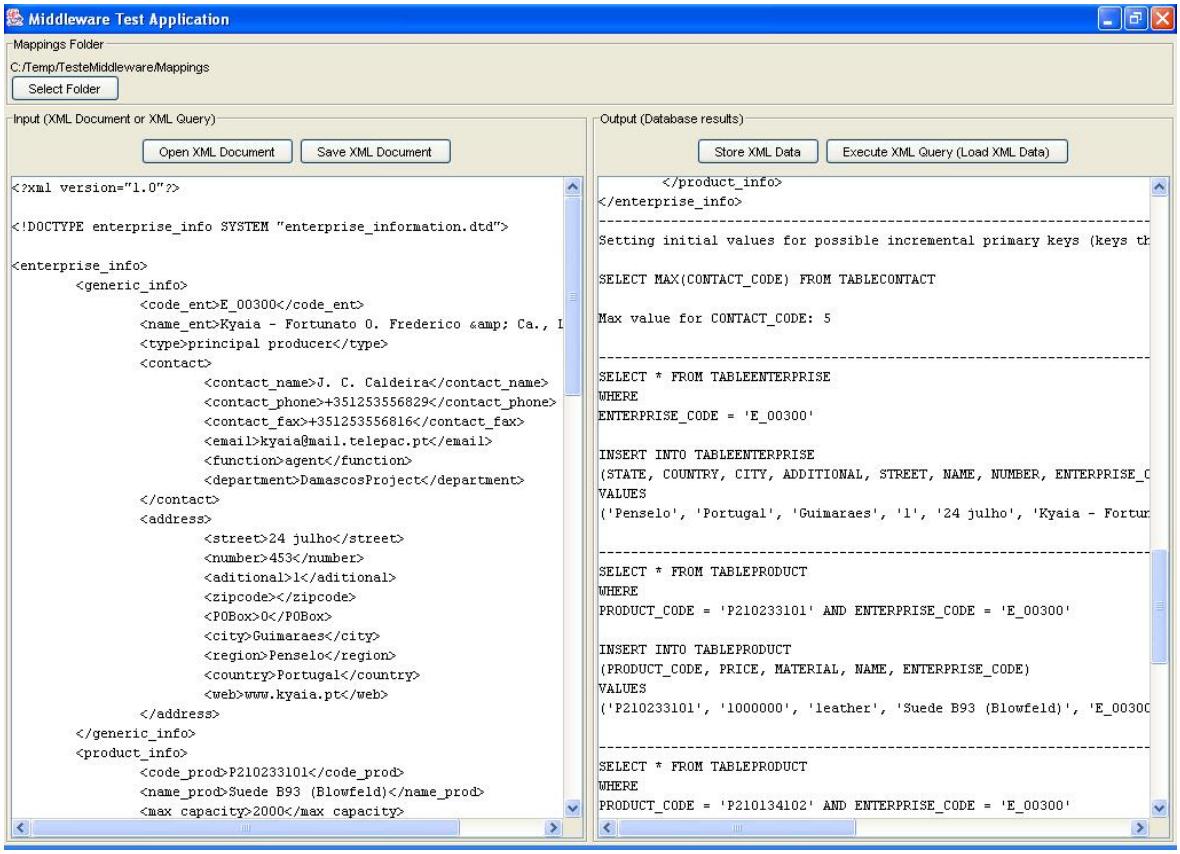

Figura 55: Interface da aplicação que utiliza o *middleware* XML.

Embora uma interface gráfica não seja obrigatória para uma aplicação que utilize um *middleware* XML, a idéia aqui é mostrar as operações que são realizadas. Assim sendo, a interface desta aplicação pode ser dividida em três partes, conforme a Figura 55:

1. No painel superior, o usuário indica o diretório onde estão localizados os mapeamentos definidos. De acordo com o que foi visto no modelo de implementação, o *middleware* XML procurará os mapeamentos neste diretório conforme a necessidade.
2. O painel à esquerda, chamado “Input (XML Document or XML Query)”, permite que o usuário carregue e/ou salve documentos XML (para serem salvos na base de dados) ou consultas XML (para carregar documentos XML da base de dados).
3. O painel à direita, chamado “Output (Database Results)”, permite salvar ou carregar documentos XML da base de dados, de acordo com o conteúdo do painel à

esquerda (Input). Este painel apresenta o registro (*log*) das operações realizadas pelo *midleware* XML.

Embora nesta aplicação todo o processo seja feito de forma manual, o objetivo aqui é a avaliação do modelo em si, sem entrar nos detalhes relacionados à comunicação e coordenação. Dependendo dos requisitos de cada tipo de aplicação (por exemplo, o *módulo de integração* da plataforma da EV), um sistema de *workflow*, *serviços Web*, etc., podem tratar destes aspectos, tornando o processo de integração automático.

Para auxiliar os testes realizados, a aplicação *IBConsole*, disponível no pacote de instalação do *Interbase*, foi utilizada para a visualização dos dados armazenados na base de dados legada.

O teste consistiu de uma série de armazenamentos de documentos XML na base de dados, além de uma série de documentos XML carregados da mesma através de consultas XML. Para o armazenamento dos dados foram utilizados:

- 6 documentos relativos ao DTD “enterprise_information”;
- 4 documentos relacionados ao DTD “production_information”;
- 4 documentos relacionados ao DTD “sales_information”.

Devido ao tamanho destes documentos, o **Anexo B.4** contém, além do seu conteúdo, algumas das operações realizadas pelo *middleware* XML para o armazenamento destes documentos.

Para o carregamento dos dados, foram utilizadas algumas consultas XML para cada tipo de mapeamento:

- 2 consultas relacionadas ao DTD “enterprise_information”;
- 2 consultas relativas ao DTD “production_information”;
- 4 consultas relacionadas ao DTD “sales_information”.

De modo análogo aos documentos XML utilizados no armazenamento, algumas das consultas e operações realizadas pelo *middleware* XML estão no **Anexo B.5**.

A escolha destes documentos e consultas XML foi arbitrária, mas com a intenção de mostrar casos diferentes de utilização do *middleware* XML.

8.3 Considerações Relacionadas à Agilidade

Conforme visto neste capítulo, os testes realizados para a avaliação das ferramentas apresentaram resultados adequados sob o ponto de vista da correção dos valores, ou seja, tanto a geração de código quanto o processo de integração ocorreram sem erros. Entretanto, em face aos objetivos deste trabalho, faz-se necessário adotar métricas para que se possa estimar o ganho em agilidade ao se utilizar tais ferramentas.

Em termos práticos, a *agilidade* é medida pela velocidade com que as atividades são realizadas e, portanto, no conseqüente ganho em tempo. A seguir serão apresentadas considerações e estimativas relacionadas à agilidade associada ao uso das ferramentas aqui implementadas. Conforme será visto, é preciso salientar que normalmente a estimativa precisa desta agilidade não é uma tarefa trivial.

8.3.1 Geração de Código

Sobre a geração semi-automática de código, há dois aspectos opostos com relação à agilidade alcançada. Se por um lado o ganho em agilidade – que neste caso corresponde à redução do tempo de desenvolvimento – é bastante evidente, visto que não é necessário escrever nenhuma linha de código (envolvendo classes, tabelas, o processamento XML e o acesso à base de dados), por outro lado o tempo ganho nesta atividade (que corresponde ao tempo de desenvolvimento manual do mesmo código) não é uma tarefa fácil de se medir com precisão. Isto porque o tempo de desenvolvimento depende de uma série de variáveis, tais como:

- **Tamanho e complexidade do MR:** este fator deve ser levado em conta, pois um MR maior e possivelmente mais complexo implica em um trabalho maior para a definição das estruturas de dados e códigos relacionados.
- **Tamanho da equipe de desenvolvedores:** de um modo geral, uma equipe de desenvolvedores realizará esta tarefa em um tempo menor do que alguém trabalhando sozinho.
- **Experiência do desenvolvedor:** principalmente em relação a alguma tecnologia em particular, como XML ou banco de dados.
- **Tempo relacionado a testes e eventuais modificações:** como tipicamente o processo de desenvolvimento não está livre de erros, eventuais correções devem ser feitas e testes posteriores devem ser realizados. É preciso destacar que ao longo do

tempo a equipe de desenvolvimento vai ganhando em experiência, resultando em um tempo de desenvolvimento gradativamente menor.

A análise destes fatores torna evidente a dificuldade de se estimar o tempo ganho através da geração semi-automática de código. Entretanto, o ganho em tempo torna-se mais perceptível (mas não de forma precisa) quando é levado em conta, além do desenvolvimento propriamente dito, o tempo associado à etapa de testes, onde são feitos refinamentos que visam minimizar o número de erros. Portanto, além do tempo ganho por não realizar o desenvolvimento em si, o usuário da ferramenta de geração de código também economiza em tempo por não necessitar realizar testes específicos para o código gerado.

Apesar das dificuldades apresentadas, é possível fazer estimativas baseadas no *tamanho do software*. Isto é feito a partir da medição do número de *linhas do código-fonte* (*Source Lines of Code* – SLOC). Embora seja uma medida simples e não conclusiva em relação à funcionalidade e eficiência do código, ela serve como base para a estimativa de esforço envolvido no desenvolvimento. Existem duas medidas básicas para SLOC: a *física*, que conta todas as linhas "não vazias" e não comentadas, e a *lógica*, que conta todos os comandos (normalmente comandos terminados com ";") independente da formatação do código (WIKIPEDIA, 2004c).

Além da medida de SLOC, outras medidas podem ser efetuadas, tais como o número de pessoas envolvidas, o número de erros encontrados, o número de páginas documentadas, o custo total do desenvolvimento, etc. A partir destas medidas, um conjunto de *métricas orientadas a tamanho* (SLOC) pode ser desenvolvido. Por exemplo:

- Erros por KLOC[27];
- Custo por KLOC;
- Páginas de documentação por KLOC;
- SLOC por pessoa-mês.

Apesar de oferecerem meios para medir o processo de desenvolvimento de *software*, estas métricas não permitem que se possa fazer uma estimativa do esforço e custo envolvidos. Para tal, existem diversas técnicas usadas para prever esforço do

[27] KLOC – milhares de linhas de código.

desenvolvimento em função do SLOC e de outras medidas. Dentre estes, destacam-se as *técnicas de decomposição* (estimativa baseada no problema ou no processo) e os *modelos empíricos de estimativa*. Estes e outros modelos de estimativa podem ser vistos em maiores detalhes em PRESSMAN (2002).

Pelo fato de as técnicas de decomposição envolverem aspectos específicos de cada projeto, neste trabalho será utilizado um modelo de estimativa empírico, que toma como base o SLOC para prever o esforço de desenvolvimento. Um modelo de estimativa típico é derivado a partir da análise de dados coletados em projetos de *software* anteriores. Como este não é o caso, será utilizado um modelo em particular, chamado COCOMO (*COnstructive COst MOdel*).

8.3.1.1 Modelo COCOMO

O *modelo construtivo de custo* (COCOMO) original tornou-se um dos modelos de estimativa de custo de *software* mais amplamente usados e discutidos na indústria (PRESSMAN, 2002). Ele posteriormente evoluiu para um modelo mais abrangente, chamado COCOMO II que, tal como seu predecessor, é na verdade uma hierarquia de modelos que trata de diversas etapas do desenvolvimento de *software*. Maiores informações sobre COCOMO II podem ser obtidas em COCOMO (2004).

Embora o modelo COCOMO II apresente muitos melhoramentos em relação ao original, são necessários muitos parâmetros para que se possa fazer uma estimativa. Portanto, de modo a ficar mais simples de entender, e também porque neste trabalho não se consideram todos os parâmetros do modelo, será utilizado o modelo COCOMO original. A hierarquia no modelo COCOMO é dividida em três níveis (BOEHM, 1981):

1. **Modelo Básico:** modelo simples e estático, que computa o esforço (e custo) de desenvolvimento de *software* em função do tamanho do programa (SLOC).
2. **Modelo Intermediário:** computa o esforço de desenvolvimento em função do tamanho do programa (SLOC) e de um conjunto de "fatores de custo" (*cost drivers*) que incluem aspectos subjetivos como os já mencionados anteriormente: tamanho e experiência de equipe, uso de ferramentas de suporte, etc.
3. **Modelo Avançado:** incorpora todas as características do modelo intermediário, mas levando em conta o impacto dos "fatores de custo" em cada etapa do processo de desenvolvimento de *software* (análise, projeto, etc.).

Dentre estes modelos, o intermediário é que melhor se adequa a este trabalho, pois leva em conta os aspectos subjetivos já mencionados sem considerar outros que o tornariam mais complexo de utilizar, como no modelo avançado. Antes de apresentar a fórmula utilizada para a estimativa, é necessário caracterizar as três classes de projetos de *software* abordadas pelo modelo COCOMO (BOEHM, 1981):

- **Projetos "Orgânicos":** relativamente pequenos e simples, onde pequenos grupos com uma boa experiência trabalham em um conjunto de requisitos não tão rígidos.
- **Projetos "*Semi-detached*":** projetos intermediários (em tamanho e complexidade) onde equipes com níveis de experiência variados devem atender a um conjunto variado de requisitos rígidos e não tão rígidos.
- **Projetos "*Embedded*":** devem ser desenvolvidos dentro de um conjunto rígido de restrições operacionais, de *hardware* e de *software*.

A classe de projeto influencia nos parâmetros aplicados no cálculo da estimativa. O modelo intermediário define a seguinte fórmula (BOEHM, 1981):

$$E = a \cdot KLOC^{b} \cdot EAF$$

onde E é o esforço aplicado em pessoas-mês[28], $KLOC$ é o número estimado de linhas de código para o projeto (expresso em milhares) e EAF representa o conjunto de "fatores de custo". Os coeficientes a, e b são dados pela Tabela 4.

Tipo do Projeto	***A***	***B***
Orgânico	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

Tabela 4: Valores para o modelo COCOMO intermediário.

Cada "fator de custo" (*cost driver*) pode possuir uma classificação que varia entre "muito baixa" até "extra alta" (em importância ou valor). Baseado nesta classificação, um *multiplicador de esforço* é determinado, conforme a Tabela 5 (BOEHM, 1981). O produto de todos os multiplicadores resulta no fator de ajustamento de esforço (*EAF*). Valores típicos para *EAF* estão entre 0,9 e 1,4 (BOEHM, 1981).

[28] Trabalho realizado por uma pessoa durante um mês, em tempo integral.

Uma calculadora para o modelo COCOMO intermediário está disponível *on-line* em COCOMO81 (2004). Nela, o usuário define cada parâmetro envolvido na fórmula (inclusive cada fator de custo) para fazer a estimativa. A página é bem explicativa, onde cada parâmetro possui uma breve explicação sobre o significado dos seus valores.

Fatores de Custo	**Classificação**					
	Muito baixa	***Baixa***	***Normal***	***Alta***	***Muito alta***	***Extra alta***
Atributos do Produto						
Confiabilidade requerida do *software*	0,75	0,88	1,00	1,15	1,40	
Tamanho da base de dados		0,94	1,00	1,08	1,16	
Complexidade do produto	0,70	0,85	1,00	1,15	1,30	1,65
Atributos de *Hardware*						
Restrições de desempenho em tempo de execução			1,00	1,11	1,30	1,66
Restrições de memória			1,00	1,06	1,21	1,56
Volatilidade da *máquina virtual*[29]		0,87	1,00	1,15	1,30	
Tempo de resposta		0,87	1,00	1,07	1,15	
Atributos Pessoais						
Capacidade do analista	1,46	1,19	1,00	0,86	0,71	
Capacidade do engenheiro de *software*	1,29	1,13	1,00	0,91	0,82	
Experiência na área de aplicação	1,42	1,17	1,00	0,86	0,70	
Experiência na *máquina virtual*	1,21	1,10	1,00	0,90		
Experiência na linguagem de programação	1,14	1,07	1,00	0,95		
Atributos de Projeto						
Uso de ferramentas de *software*	1,24	1,10	1,00	0,91	0,82	
Aplicação de métodos de engenharia de *software*	1,24	1,10	1,00	0,91	0,83	
Restrições no cronograma de desenvolvimento	1,23	1,08	1,00	1,04	1,10	

Tabela 5: Fatores de custo para o modelo COCOMO intermediário.

8.3.1.2 Exemplo de Estimativa

Para apresentar uma estimativa de ganho em termos de agilidade, será apresentado um exemplo que toma como base o DTD apresentado na seção 8.1 (Figura 48, página 135), que representa a topologia de uma EV. Para medir o número de linhas de código geradas,

[29] Neste contexto, *máquina virtual* significa todo o *hardware* e *software* necessário à execução do *software* a ser desenvolvido. Por exemplo: sistema operacional, sistema de gerenciamento de banco de dados, etc.

foi utilizada a ferramenta *SLOC Counter* (SLOC, 2004), cujo resultado (1243 linhas de código) pode ser visto na Figura 56. Repare que são também discriminados os números de linhas para cada arquivo (classe Java) gerado, onde se pode perceber o número elevado de linhas para as classes relativas ao processamento XML e acesso à base de dados, em comparação com as demais classes.

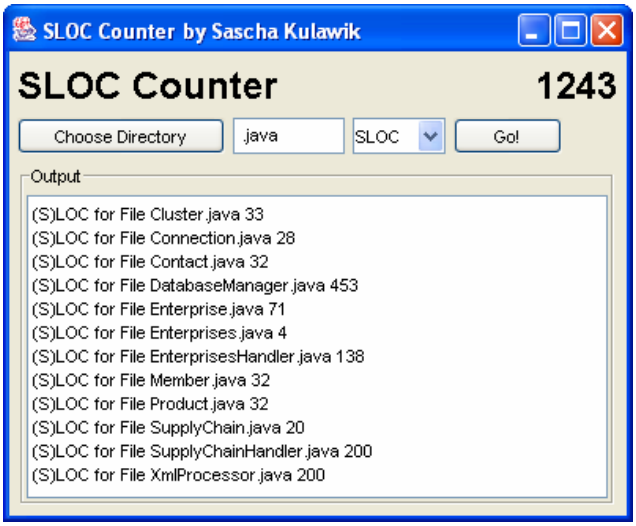

Figura 56: Ferramenta para a medição do SLOC.

A partir do valor de SLOC (1243), pode-se aplicar a fórmula apresentada na seção anterior para se fazer a estimativa. Devido ao seu tamanho, o projeto foi classificado como do tipo “orgânico”. Como neste caso é difícil de estimar os “fatores de custo” (pois o código foi gerado), serão utilizados os valores típicos de *EAF*, ou seja, 0,9 e 1,4. Aplicando a fórmula com estes dois valores de *EAF*, o esforço estimado (*E*) fica entre 3,6 e 5,6 pessoas-mês, aproximadamente. A estimativa de custos é uma conseqüência dos resultados aqui obtidos, sendo calculada pela atribuição de valores de mão-de-obra às atividades relacionadas ao processo de desenvolvimento de *software* (PRESSMAN, 2002).

8.3.2 Definição dos Mapeamentos e Integração de Dados

Nos testes realizados foi possível verificar que o uso de mapeamentos torna o processo de integração mais ágil, visto que esta tarefa é nitidamente mais rápida e fácil do que uma eventual reengenharia dos sistemas legados, além de esta última ter a agravante de poder apresentar dificuldades, como por exemplo, a indisponibilidade de profissionais

especializados em uma linguagem em particular. Entretanto, para este caso não foi possível estimar este ganho visto que é necessário medir o tempo que um usuário levaria para definir os mapeamentos e compará-lo com o tempo relacionado à reengenharia nos sistemas legados, sabendo-se ainda que esta última envolve os aspectos enumerados na seção anterior, tais como tamanho e experiência da equipe, etc.

Em relação ao processo de integração propriamente dito (armazenamento e carregamento dos dados) realizado pelo *middleware* XML, é preciso destacar que, sendo genérico, este processo pode ser considerado mais lento do que em um sistema que é desenvolvido especificamente para mapear um MR com um modelo de dados legado em particular. Apesar de neste trabalho não ter sido feita nenhuma análise em tempo de execução relacionada ao desempenho do *middleware* XML, o que teria a carga adicional de se implementar um sistema específico para ser parâmetro de comparação, é possível observar alguns aspectos relacionados aos algoritmos utilizados no processo de integração (seção 6.2.3) que reforçam esta situação.

Em relação ao armazenamento de documentos XML, dois passos básicos são feitos em qualquer um dos casos (seja com *middleware* XML ou específico): o processamento dos documentos XML (neste trabalho, usando o DOM), considerado o passo inicial; e a execução dos comandos SQL para o armazenamento dos dados (passo final). As diferenças podem ser vistas nos passos intermediários: uma aplicação específica pode ler os nodos da árvore DOM de forma mais eficiente, visto que não necessita consultar um mapeamento; além disso, a mesma aplicação pode não necessitar fazer uma consulta prévia pelos dados, o que implica em um menor número de operações na base de dados.

Sobre o carregamento dos dados em documentos XML, novamente há certas semelhanças e diferenças. Entretanto, de semelhante há apenas a execução de uma consulta SQL inicial, realizada após a conversão da consulta XML. A diferença está na leitura do *result set*, onde uma aplicação específica pode ler os campos diretamente relacionados a cada elemento XML, além de não ler desnecessariamente linhas repetidas no *result set* (como é feito pelo *middleware* XML).

Apesar de haver diferenças que tornem uma aplicação específica mais rápida do que um *middleware* XML, estas podem variar de acordo com cada caso. Além disso, este é o preço a ser pago quando do uso de aplicações genéricas, como *middlewares* XML, visando uma maior agilidade no processo de integração de dados em sistemas heterogêneos.

9 Conclusões

A evolução e posterior popularização das tecnologias de informação e comunicação, em particular as relacionadas à *World Wide Web*, tornaram mais ágeis as relações comerciais, permitindo que organizações – que podem estar localizadas em diferentes partes do mundo – realizem os seus processos de negócio de forma mais rápida e eficiente. Além disso, o uso destas tecnologias levou estas relações a um nível de automação e agilidade de modo a se tornar um requisito praticamente obrigatório para uma empresa se manter competitiva.

A aplicação destas tecnologias levou ao desenvolvimento de novos paradigmas de negócios entre empresas, com especial destaque para o de Empresas Virtuais (EV). O conceito de EV está fortemente vinculado à cooperação entre empresas para atender a uma oportunidade de negócio, e no uso de tecnologias de informação e comunicação como suporte a esta cooperação. Desde o surgimento deste paradigma, diversos grupos têm desenvolvido modelos e ferramentas que atuam nas diversas etapas do seu ciclo de vida.

Neste sentido, este trabalho visou contribuir para a solução de alguns problemas no âmbito da etapa de *criação* de uma EV. Mais especificamente, este trabalho abordou a problemática associada aos aspectos da configuração das estruturas de dados utilizadas para o intercâmbio de informações entre os seus parceiros e sua posterior integração com os modelos de dados utilizados pelos sistemas legados. Dentre os projetos de pesquisa na área de EVs relacionados neste trabalho, estes aspectos, principalmente a questão da integração dos dados, foram pouco abordados, sendo normalmente resolvidos de forma manual.

Assim, o objetivo deste trabalho consistiu em adicionar *agilidade* ao processo de criação de uma EV através da adoção de um modelo semi-automático para configuração e integração de dados em plataformas para EVs, o que representou uma inovação frente aos vários projetos pesquisados nesta área. O modelo utilizou a linguagem XML (*Extensible Markup Language*) como formato para representação de dados.

XML surgiu como um formato padrão para intercâmbio de informações entre aplicações distribuídas e heterogêneas, pois, como foi visto, é um padrão aberto, representa os dados de forma neutra e é independente de plataforma. Como conseqüência natural, diversas iniciativas internacionais têm buscado desenvolver plataformas e ferramentas para

este propósito, tais como Rosettanet, ebXML, XML/EDI, entre outras, levando a uma gradual migração das *antigas* ferramentas (que utilizam formatos proprietários, são mais complexas e caras) para *novas* ferramentas que utilizam um formato padronizado e aberto.

O modelo de configuração e integração de dados descrito neste trabalho levou ao desenvolvimento de três protótipos computacionais, que serviram não apenas para avaliar os conceitos apresentados, mas que também estão disponíveis publicamente a qualquer parte interessada em utilizar este tipo de aplicação. Os resultados práticos são enumerados abaixo:

1. Ferramenta de geração de código, que permite a geração semi-automática de classes Java e tabelas de uma base de dados a partir de um esquema XML. Possui uma interface gráfica amigável que disponibiliza uma série de opções ao usuário. Esta ferramenta se enquadra na categoria de aplicações chamada *XML Data Binding*.
2. Ferramenta para a definição dos mapeamentos, utilizada para permitir a integração dos dados provenientes dos sistemas legados. Esta ferramenta permite ao usuário definir, de maneira amigável, um mapeamento entre um esquema XML e o modelo de dados de uma base de dados legada. Os mapeamentos definidos podem ser usados por um *middleware* XML.
3. *Middleware* XML. Utiliza os mapeamentos definidos pela ferramenta de definição dos mapeamentos para integrar dados entre documentos XML e uma base de dados legada. É um componente bastante flexível, pois realiza a integração de forma dinâmica, a partir de mapeamentos predefinidos. Foi implementado na forma de um pacote de classes Java, podendo ser utilizado por qualquer aplicação que necessite deste tipo de serviço.

As ferramentas e a biblioteca de classes foram implementadas na linguagem Java, o que é um ponto positivo no caso das ferramentas, pois podem ser utilizadas em qualquer plataforma que possua uma máquina virtual Java.

É importante salientar que, embora este trabalho esteja enquadrado na área de EVs, os modelos aqui apresentados e suas respectivas implementações podem ser usados em outros contextos, visto que os aspectos de configuração e integração de dados são requisitos normalmente presentes em sistemas distribuídos.

9.1 Trabalhos Futuros

Com base no que foi visto nos capítulos de implementação e avaliação, as ferramentas apresentaram algumas limitações, principalmente porque, devido ao tempo disponível, alguns detalhes de menor prioridade foram deixados de lado ou pouco abordados. A seguir são enumeradas algumas sugestões de melhoramentos para as futuras versões.

Sugestões gerais, que afetam todas as ferramentas:

- Testes com diferentes bases de dados. Nos testes realizados para este trabalho foi utilizado apenas o sistema de banco de dados *Interbase*. Os comandos SQL utilizados para o acesso à base de dados foram definidos em SQL padrão (ANSI) visando justamente a portabilidade no uso de diferentes bases de dados. Entretanto, testes com outros sistemas de bancos de dados são necessários de modo a efetivamente validar este aspecto.
- Implementar o processador de XMI. O modelo conceitual apresentou documentos XMI como um tipo de especificação XML, mas devido ao tamanho da sua especificação (padrão), não foi possível implementar em tempo hábil um processador para este tipo de documento.
- Melhorar o processador de XML *Schema*. Tal como no caso do XMI, a especificação do XML *Schema* é muito extensa, e não foi possível implementar um processador que cobrisse todos os seus aspectos. Entretanto, a gradual adoção de XML *Schemas* ao invés de DTDs torna este um componente obrigatório para as ferramentas implementadas.

Para a ferramenta de geração de código:

- Implementar geradores de código para outras linguagens. Como a idéia inicial era apenas avaliar os conceitos apresentados, não se viu a necessidade de se gerar código para mais de uma linguagem. Assim, a versão atual da ferramenta gera código apenas na linguagem Java, mas é desejável que no futuro outras linguagens possam ser contempladas, como por exemplo, C++.
- Fazer com que o “projeto” das classes a serem geradas possa ser salvo. A idéia aqui é que a ferramenta permita salvar o projeto de “*binding*” em um arquivo

(possivelmente também XML), para que este possa ser editado/modificado caso sejam necessárias alterações posteriores.

Para o *middleware* XML:

- Utilizar um *parser* SAX. A versão atual do *middleware* XML requer que os documentos XML a serem salvos estejam no formato DOM. Embora seja mais lento que a abordagem SAX, isto foi feito desta maneira porque neste caso foi mais fácil de se implementar. A idéia é que no futuro o *middleware* XML utilize também um *parser* SAX para melhorar o seu desempenho.
- Permitir a conversão entre diferentes unidades de medidas. Um aspecto importante e que foi ignorado na implementação do *middleware* XML está relacionado à conversão entre unidades. Exemplos típicos para este aspecto são conversões entre unidades métricas e entre formatos regionais de data e hora. Assim, é recomendável que uma versão futura forneça meios para que se possa tratar também este aspecto. Como esta informação deve ficar no mapeamento, esta alteração afetaria também a ferramenta de definição dos mapeamentos.

Para a ferramenta de definição dos mapeamentos, uma evolução interessante seria a possibilidade de ser utilizada remotamente, ou seja, a aplicação é instalada no local onde devem ser integrados os dados (como é feito atualmente), mas ela poderia também ser utilizada remotamente, por exemplo, via Web.

Anexo A – Document Type Definitions

A.1 – Mapeamento

Os mapeamentos definidos pela ferramenta de definição dos mapeamentos (seção 7.3) são armazenados em documentos XML cujo DTD é apresentado na Figura 57.

```
<!ELEMENT mapping (header, (tableMapping | passThrough))>
 <!ELEMENT header (xmlSpecification, dbInformation)>
  <!ELEMENT xmlSpecification (#PCDATA)>
  <!ATTLIST xmlSpecification type (DTD|XSD|XMI) #REQUIRED>
  <!ELEMENT dbInformation (location, driver, user, pass)>
   <!ELEMENT location (#PCDATA)>
   <!ELEMENT driver (#PCDATA)>
   <!ELEMENT user (#PCDATA)>
   <!ELEMENT pass (#PCDATA)>
  <!ELEMENT tableMapping (incrementalPK?, (foreignKey | fieldMapping | tableMapping | passThrough)*)>
  <!ATTLIST tableMapping xmlName CDATA #REQUIRED>
  <!ATTLIST tableMapping dbName CDATA #REQUIRED>
  <!ATTLIST tableMapping minOccurs CDATA #IMPLIED>
  <!ATTLIST tableMapping maxOccurs CDATA #IMPLIED>
  <!ATTLIST tableMapping isPCDATA (true|false) "false">
   <!ELEMENT incrementalPK (#PCDATA)>
   <!ELEMENT foreignKey (from, to)>
   <!ATTLIST foreignKey mapped (true|false) "false">
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT to (#PCDATA)>
    <!ATTLIST to table CDATA #REQUIRED>
   <!ELEMENT fieldMapping (xmlName, dbName, type?)>
   <!ATTLIST fieldMapping id ID #REQUIRED>
   <!ATTLIST fieldMapping isElement (true|false) "false">
   <!ATTLIST fieldMapping isPK (true|false) "false">
    <!ELEMENT xmlName (#PCDATA)>
    <!ELEMENT dbName (#PCDATA)>
    <!ELEMENT type (#PCDATA)>
   <!ELEMENT passThrough (fieldMapping | tableMapping | passThrough)*>
   <!ATTLIST passThrough name CDATA #REQUIRED>
```

Figura 57: DTD que define o mapeamento.

Os elementos e atributos definidos neste DTD têm relação com as classes e atributos de classe mostrados no diagrama da Figura 39 (página 115). A única observação a respeito deste DTD está nos *mapeamentos para tabela* que são ignorados, onde foi utilizado o elemento "*passThrough*". Isto foi feito assim simplesmente para facilitar a diferenciação entre um mapeamento para tabela e um elemento ignorado (pois são implementados na mesma classe, "*TableMap*"). Naturalmente, esta facilidade é apenas quando da sua leitura por pessoas. Quando o mapeamento é processado pelo *middleware* XML e carregado na forma de objetos, elementos "*passThrough*" passam a ser objetos da classe "*TableMap*".

Um documento XML contendo um mapeamento pode ser visto na Figura 58.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE mapping SYSTEM "mapping.dtd">
<mapping>
  <header>
    <xmlSpecification type="DTD">ordem.dtd</xmlSpecification>
    <dbInformation>
      <location>jdbc:firebirdsql:150.162.27.185:D:\\Working\\Mestrado\\Database\\Scm.gdb</location>
      <driver>org.firebirdsql.jdbc.FBDriver</driver>
      <user>SYSDBA</user>
      <pass>masterkey</pass>
    </dbInformation>
  </header>
  <tableMapping xmlName="ordemDeCompra" dbName="TABLESALES">
    <fieldMapping id="f1" isPK="true">
      <xmlName>número</xmlName>
      <dbName>SALESNUMBER</dbName>
    </fieldMapping>
    <fieldMapping id="f2">
      <xmlName>data</xmlName>
      <dbName>SALESDATE</dbName>
    </fieldMapping>
    <passThrough name="cliente">
      <fieldMapping id="f3" isElement="true">
        <xmlName>nome</xmlName>
        <dbName>CLIENTNAME</dbName>
      </fieldMapping>
      <fieldMapping id="f4" isElement="true">
        <xmlName>endereço</xmlName>
        <dbName>CLIENTADDRESS</dbName>
      </fieldMapping>
      <fieldMapping id="f5" isElement="true">
        <xmlName>cidade</xmlName>
        <dbName>CLIENTCITY</dbName>
      </fieldMapping>
      <fieldMapping id="f6" isElement="true">
        <xmlName>estado</xmlName>
        <dbName>CLIENTSTATE</dbName>
      </fieldMapping>
    </passThrough>
    <tableMapping xmlName="item" dbName="TABLESALESDETAIL" maxOccurs="unbounded">
      <foreignKey>
        <from>SALESNUMBER</from>
        <to table="TABLESALES">SALESNUMBER</to>
      </foreignKey>
      <passThrough name="produto">
        <fieldMapping id="f7" isPK="true">
          <xmlName>código</xmlName>
          <dbName>ITEMCODE</dbName>
        </fieldMapping>
        <fieldMapping id="f8" isElement="true">
          <xmlName>descrição</xmlName>
          <dbName>ITEMDESCRIPTION</dbName>
        </fieldMapping>
        <fieldMapping id="f9" isElement="true">
          <xmlName>preço</xmlName>
          <dbName>ITEMPRICE</dbName>
        </fieldMapping>
      </passThrough>
      <fieldMapping id="f10" isElement="true">
        <xmlName>quantidade</xmlName>
        <dbName>QUANTITY</dbName>
      </fieldMapping>
    </tableMapping>
  </tableMapping>
</mapping>
```

Figura 58: Documento XML contendo um mapeamento.

A.2 – Consulta XML

O DTD que define a linguagem para consulta XML é ilustrado na Figura 59. Maiores detalhes sobre esta linguagem podem ser vistos na seção 6.2.3.2.

```
<!ELEMENT select (element, from, where?)>
  <!ELEMENT element (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT where (and|or|condition)>
    <!ELEMENT and (or|condition)+>
    <!ELEMENT or (and|condition)+>
    <!ELEMENT condition EMPTY>
    <!ATTLIST condition op1 CDATA #REQUIRED>
    <!ATTLIST condition operator CDATA #REQUIRED>
    <!ATTLIST condition op2 CDATA #REQUIRED>
```

Figura 59: DTD que define a linguagem de consulta XML.

Anexo B – Detalhes do Teste de Integração de Dados

B.1 – DTDs utilizados como Modelo de Referência

Os DTDs completos são mostrados a seguir: “enterprise_information.dtd” (Figura 60), “production_information.dtd” (Figura 61) e “sales_information.dtd” (Figura 62).

```
<!ELEMENT enterprise_info (generic_info, product_info*)>
 <!ELEMENT generic_info (code_ent, name_ent, type, contact*, address*)>
  <!ELEMENT code_ent (#PCDATA)>
  <!ELEMENT name_ent (#PCDATA)>
  <!ELEMENT type (#PCDATA)>
 <!ELEMENT contact (contact_name, contact_phone, contact_fax?, email?, function?, department?)>
  <!ELEMENT contact_name (#PCDATA)>
  <!ELEMENT contact_phone (#PCDATA)>
  <!ELEMENT contact_fax (#PCDATA)>
  <!ELEMENT email (#PCDATA)>
  <!ELEMENT function (#PCDATA)>
  <!ELEMENT department (#PCDATA)>
 <!ELEMENT address (street, number, aditional, zipcode, POBox, city, region?, country, web)>
  <!ELEMENT street (#PCDATA)>
  <!ELEMENT number (#PCDATA)>
  <!ELEMENT aditional (#PCDATA)>
  <!ELEMENT zipcode (#PCDATA)>
  <!ELEMENT POBox (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT region (#PCDATA)>
  <!ELEMENT country (#PCDATA)>
  <!ELEMENT web (#PCDATA)>
 <!ELEMENT product_info (code_prod, name_prod, max_capacity, allocated_capacity, production_price,
                         color?, volume?, height?, width?, weight?, material?, productType?,
                         productionLot?, productAcquisition?, min_stock_limit?, max_stock_limit?,
                         components_list)>
  <!ELEMENT code_prod (#PCDATA)>
  <!ELEMENT name_prod (#PCDATA)>
  <!ELEMENT max_capacity (#PCDATA)>
  <!ELEMENT allocated_capacity (#PCDATA)>
  <!ELEMENT production_price (#PCDATA)>
  <!ELEMENT color (#PCDATA)>
  <!ELEMENT volume (#PCDATA)>
  <!ATTLIST volume unit CDATA #REQUIRED>
  <!ELEMENT height (#PCDATA)>
  <!ATTLIST height unit CDATA #REQUIRED>
  <!ELEMENT width (#PCDATA)>
  <!ATTLIST width unit CDATA #REQUIRED>
  <!ELEMENT weight (#PCDATA)>
  <!ATTLIST weight unit CDATA #REQUIRED>
  <!ELEMENT material (#PCDATA)>
  <!ELEMENT productType (#PCDATA)>
  <!ELEMENT productionLot (#PCDATA)>
  <!ELEMENT productAcquisition (#PCDATA)>
  <!ELEMENT min_stock_limit (#PCDATA)>
  <!ELEMENT max_stock_limit (#PCDATA)>
  <!ELEMENT components_list (component*)>
   <!ELEMENT component (component_cod, supplier_cod, qty, name)>
    <!ELEMENT component_cod (#PCDATA)>
    <!ELEMENT supplier_cod (#PCDATA)>
    <!ELEMENT qty (#PCDATA)>
    <!ELEMENT name (#PCDATA)>
```

Figura 60: DTD “enterprise_information”.

```
<!ELEMENT production_order (supply_chain_id, order_number, order_date, deliveryDate, supplier,
                            status, plannedStartDate, plannedEndDate, startDate, endDate,
                            slackOverrun?, progress?, lastFeedback?, leadTime?, production_detail+)>
 <!ELEMENT supply_chain_id (#PCDATA)>
 <!ELEMENT order_number (#PCDATA)>
 <!ELEMENT order_date (#PCDATA)>
 <!ELEMENT deliveryDate (#PCDATA)>
 <!ELEMENT supplier (#PCDATA)>
 <!ELEMENT status (#PCDATA)>
 <!ELEMENT plannedStartDate (#PCDATA)>
 <!ELEMENT plannedEndDate (#PCDATA)>
 <!ELEMENT startDate (#PCDATA)>
 <!ELEMENT endDate (#PCDATA)>
 <!ELEMENT slackOverrun (#PCDATA)>
 <!ELEMENT progress (#PCDATA)>
 <!ELEMENT lastFeedback (#PCDATA)>
 <!ELEMENT leadTime (#PCDATA)>
 <!ATTLIST leadTime unit CDATA #REQUIRED>
 <!ELEMENT production_detail (order_number, detail_number, plannedStartDate, plannedEndDate,
                              startDate, endDate, deliveryDate, priceTotal, slackOverrun?,
                              status, progress?, lastFeedback?, requestedQuantity?, producedQuantity?,
                              requestedLotSize?, producedLotSize?, lotValue?, cycleTime?,
                              earliestStartDate?, latestStartDate?, earliestEndDate?, latestEndDate?,
                              related_sales_order*, productSpecification)>
  <!ELEMENT detail_number (#PCDATA)>
  <!ELEMENT priceTotal (#PCDATA)>
  <!ELEMENT requestedQuantity (#PCDATA)>
  <!ELEMENT producedQuantity (#PCDATA)>
  <!ELEMENT requestedLotSize (#PCDATA)>
  <!ELEMENT producedLotSize (#PCDATA)>
  <!ELEMENT lotValue (#PCDATA)>
  <!ELEMENT cycleTime (#PCDATA)>
  <!ATTLIST cycleTime unit CDATA #REQUIRED>
  <!ELEMENT earliestStartDate (#PCDATA)>
  <!ELEMENT latestStartDate (#PCDATA)>
  <!ELEMENT earliestEndDate (#PCDATA)>
  <!ELEMENT latestEndDate (#PCDATA)>
  <!ELEMENT related_sales_order (sales_number,sales_line)>
  <!ELEMENT sales_number (#PCDATA)>
  <!ELEMENT sales_line (#PCDATA)>
  <!ELEMENT productSpecification (cod_prod, size, colour, price)>
   <!ELEMENT cod_prod (#PCDATA)>
   <!ELEMENT size (#PCDATA)>
   <!ELEMENT colour (#PCDATA)>
   <!ELEMENT price (#PCDATA)>
```

Figura 61: DTD “production_information”.

```
<!ELEMENT sales_order (supply_chain_id, order_number,order_date, delivery_date, sales_agent,
                       client, season, status, total_amount, sales_detail+)>
 <!ELEMENT supply_chain_id (#PCDATA)>
 <!ELEMENT order_number (#PCDATA)>
 <!ELEMENT order_date (#PCDATA)>
 <!ELEMENT delivery_date (#PCDATA)>
 <!ELEMENT sales_agent (#PCDATA)>
 <!ELEMENT client (#PCDATA)>
 <!ELEMENT season (#PCDATA)>
 <!ELEMENT status (#PCDATA)>
 <!ELEMENT total_amount (#PCDATA)>
 <!ELEMENT sales_detail (line_number, cod_prod, size, colour, price, qty, class, segment)>
  <!ELEMENT line_number (#PCDATA)>
  <!ELEMENT cod_prod (#PCDATA)>
  <!ELEMENT size (#PCDATA)>
  <!ELEMENT colour (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ELEMENT qty (#PCDATA)>
  <!ELEMENT class (#PCDATA)>
  <!ELEMENT segment (#PCDATA)>
```

Figura 62: DTD “sales_information”.

B.2 – Modelo de dados da base de dados utilizada

O Modelo de dados da base de dados utilizada nos testes é apresentado na Figura 63.

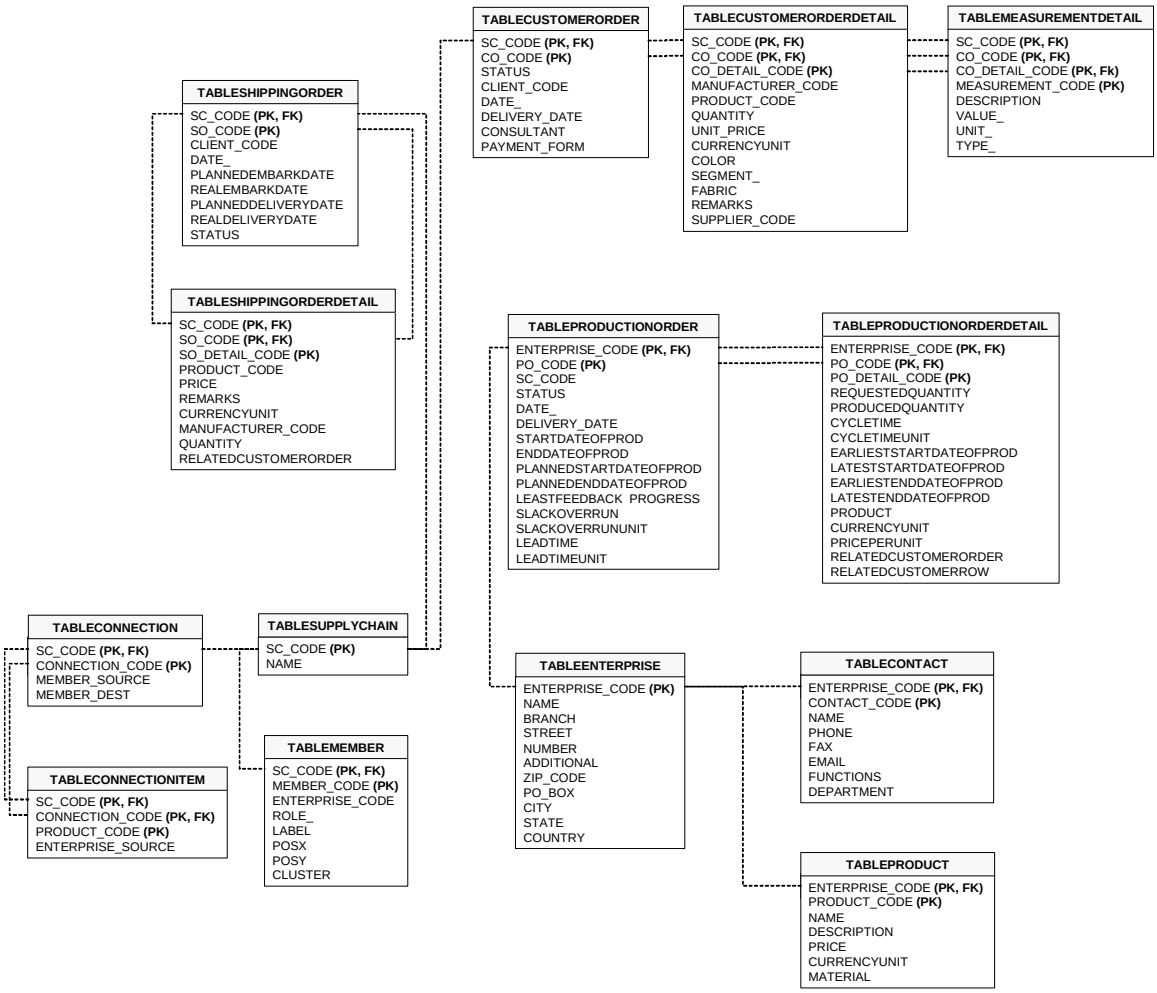

Figura 63: Modelo de dados da base de dados utilizada no teste.

B.3 – Mapeamentos

Os mapeamentos definidos para os testes são mostrados a seguir, na forma de documentos XML.

1) Mapeamento relacionado ao DTD “enterprise_information”:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapping SYSTEM "mapping.dtd">
<mapping>
  <header>
    <xmlSpecification type="DTD">enterprise_information.dtd</xmlSpecification>
    <dbInformation>
```

```
      <location>
        jdbc:firebirdsql:150.162.27.185:D:/Working/Mestrado/Database/MyFashionDM.GDB
      </location>
      <driver>org.firebirdsql.jdbc.FBDriver</driver>
      <user>SYSDBA</user>
      <pass>masterkey</pass>
    </dbInformation>
  </header>
  <tableMapping xmlName="enterprise_info" dbName="TABLEENTERPRISE">
    <passThrough name="generic_info">
      <fieldMapping id="f1" isElement="true" isPK="true">
        <xmlName>code_ent</xmlName>
        <dbName>ENTERPRISE_CODE</dbName>
      </fieldMapping>
      <fieldMapping id="f2" isElement="true">
        <xmlName>name_ent</xmlName>
        <dbName>NAME</dbName>
      </fieldMapping>
      <fieldMapping id="f3" isElement="true">
        <xmlName>type</xmlName>
        <dbName>BRANCH</dbName>
      </fieldMapping>
      <tableMapping xmlName="contact" dbName="TABLECONTACT" maxOccurs="unbounded">
        <incrementalPK>CONTACT_CODE</incrementalPK>
        <foreignKey>
          <from>ENTERPRISE_CODE</from>
          <to table="TABLEENTERPRISE">ENTERPRISE_CODE</to>
        </foreignKey>
        <fieldMapping id="f4" isElement="true">
          <xmlName>contact_name</xmlName>
          <dbName>NAME</dbName>
        </fieldMapping>
        <fieldMapping id="f5" isElement="true">
          <xmlName>contact_phone</xmlName>
          <dbName>PHONE</dbName>
        </fieldMapping>
        <fieldMapping id="f6" isElement="true">
          <xmlName>contact_fax</xmlName>
          <dbName>FAX</dbName>
        </fieldMapping>
        <fieldMapping id="f7" isElement="true">
          <xmlName>email</xmlName>
          <dbName>EMAIL</dbName>
        </fieldMapping>
        <fieldMapping id="f8" isElement="true">
          <xmlName>function</xmlName>
          <dbName>FUNCTIONS</dbName>
        </fieldMapping>
        <fieldMapping id="f9" isElement="true">
          <xmlName>department</xmlName>
          <dbName>DEPARTMENT</dbName>
        </fieldMapping>
      </tableMapping>
      <passThrough name="address">
        <fieldMapping id="f10" isElement="true">
          <xmlName>street</xmlName>
          <dbName>STREET</dbName>
        </fieldMapping>
        <fieldMapping id="f11" isElement="true">
          <xmlName>number</xmlName>
          <dbName>NUMBER</dbName>
        </fieldMapping>
        <fieldMapping id="f12" isElement="true">
          <xmlName>aditional</xmlName>
          <dbName>ADDITIONAL</dbName>
        </fieldMapping>
        <fieldMapping id="f13" isElement="true">
          <xmlName>zipcode</xmlName>
          <dbName>ZIP_CODE</dbName>
        </fieldMapping>
        <fieldMapping id="f14" isElement="true">
          <xmlName>POBox</xmlName>
          <dbName>PO_BOX</dbName>
        </fieldMapping>
        <fieldMapping id="f15" isElement="true">
```

```
          <xmlName>city</xmlName>
          <dbName>CITY</dbName>
        </fieldMapping>
        <fieldMapping id="f16" isElement="true">
          <xmlName>region</xmlName>
          <dbName>STATE</dbName>
        </fieldMapping>
        <fieldMapping id="f17" isElement="true">
          <xmlName>country</xmlName>
          <dbName>COUNTRY</dbName>
        </fieldMapping>
        <fieldMapping id="f18" isElement="true">
          <xmlName>web</xmlName>
          <dbName></dbName>
        </fieldMapping>
      </passThrough>
    </passThrough>
    <tableMapping xmlName="product_info" dbName="TABLEPRODUCT" maxOccurs="unbounded">
      <foreignKey>
        <from>ENTERPRISE_CODE</from>
        <to table="TABLEENTERPRISE">ENTERPRISE_CODE</to>
      </foreignKey>
      <fieldMapping id="f19" isElement="true" isPK="true">
        <xmlName>code_prod</xmlName>
        <dbName>PRODUCT_CODE</dbName>
      </fieldMapping>
      <fieldMapping id="f20" isElement="true">
        <xmlName>name_prod</xmlName>
        <dbName>NAME</dbName>
      </fieldMapping>
      <fieldMapping id="f21" isElement="true">
        <xmlName>max_capacity</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f22" isElement="true">
        <xmlName>allocated_capacity</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f23" isElement="true">
        <xmlName>production_price</xmlName>
        <dbName>PRICE</dbName>
      </fieldMapping>
      <fieldMapping id="f24" isElement="true">
        <xmlName>color</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <passThrough name="volume" isPCDATA="true">
        <fieldMapping id="f25">
          <xmlName>unit</xmlName>
          <dbName></dbName>
        </fieldMapping>
        <fieldMapping id="f26" isElement="true">
          <xmlName>value</xmlName>
          <dbName></dbName>
        </fieldMapping>
      </passThrough>
      <passThrough name="height" isPCDATA="true">
        <fieldMapping id="f27">
          <xmlName>unit</xmlName>
          <dbName></dbName>
        </fieldMapping>
        <fieldMapping id="f28" isElement="true">
          <xmlName>value</xmlName>
          <dbName></dbName>
        </fieldMapping>
      </passThrough>
      <passThrough name="width" isPCDATA="true">
        <fieldMapping id="f29">
          <xmlName>unit</xmlName>
          <dbName></dbName>
        </fieldMapping>
        <fieldMapping id="f30" isElement="true">
          <xmlName>value</xmlName>
          <dbName></dbName>
        </fieldMapping>
```

```
      </passThrough>
      <passThrough name="weight" isPCDATA="true">
        <fieldMapping id="f31">
          <xmlName>unit</xmlName>
          <dbName></dbName>
        </fieldMapping>
        <fieldMapping id="f32" isElement="true">
          <xmlName>value</xmlName>
          <dbName></dbName>
        </fieldMapping>
      </passThrough>
      <fieldMapping id="f33" isElement="true">
        <xmlName>material</xmlName>
        <dbName>MATERIAL</dbName>
      </fieldMapping>
      <fieldMapping id="f34" isElement="true">
        <xmlName>productType</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f35" isElement="true">
        <xmlName>productionLot</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f36" isElement="true">
        <xmlName>productAcquisition</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f37" isElement="true">
        <xmlName>min_stock_limit</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f38" isElement="true">
        <xmlName>max_stock_limit</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <passThrough name="components_list">
        <passThrough name="component">
          <fieldMapping id="f39" isElement="true">
            <xmlName>component_cod</xmlName>
            <dbName></dbName>
          </fieldMapping>
          <fieldMapping id="f40" isElement="true">
            <xmlName>supplier_cod</xmlName>
            <dbName></dbName>
          </fieldMapping>
          <fieldMapping id="f41" isElement="true">
            <xmlName>qty</xmlName>
            <dbName></dbName>
          </fieldMapping>
          <fieldMapping id="f42" isElement="true">
            <xmlName>name</xmlName>
            <dbName></dbName>
          </fieldMapping>
        </passThrough>
      </passThrough>
    </tableMapping>
  </tableMapping>
</mapping>
```

2) Mapeamento relacionado ao DTD “production_information”:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapping SYSTEM "mapping.dtd">
<mapping>
  <header>
    <xmlSpecification type="DTD">production_information.dtd</xmlSpecification>
    <dbInformation>
      <location>
        jdbc:firebirdsql:150.162.27.185:D:/Working/Mestrado/Database/MyFashionDM.GDB
      </location>
      <driver>org.firebirdsql.jdbc.FBDriver</driver>
      <user>SYSDBA</user>
      <pass>masterkey</pass>
```

```
    </dbInformation>
  </header>
  <tableMapping xmlName="production_order" dbName="TABLEPRODUCTIONORDER">
    <foreignKey mapped="true">
      <from>ENTERPRISE_CODE</from>
      <to table="TABLEENTERPRISE">ENTERPRISE_CODE</to>
    </foreignKey>
    <fieldMapping id="f1" isElement="true">
      <xmlName>supply_chain_id</xmlName>
      <dbName>SC_CODE</dbName>
    </fieldMapping>
    <fieldMapping id="f2" isElement="true" isPK="true">
      <xmlName>order_number</xmlName>
      <dbName>PO_CODE</dbName>
    </fieldMapping>
    <fieldMapping id="f3" isElement="true">
      <xmlName>order_date</xmlName>
      <dbName>DATE_</dbName>
    </fieldMapping>
    <fieldMapping id="f4" isElement="true">
      <xmlName>deliveryDate</xmlName>
      <dbName>DELIVERY_DATE</dbName>
    </fieldMapping>
    <fieldMapping id="f5" isElement="true" isPK="true">
      <xmlName>supplier</xmlName>
      <dbName>ENTERPRISE_CODE</dbName>
    </fieldMapping>
    <fieldMapping id="f6" isElement="true">
      <xmlName>status</xmlName>
      <dbName>STATUS</dbName>
    </fieldMapping>
    <fieldMapping id="f7" isElement="true">
      <xmlName>plannedStartDate</xmlName>
      <dbName>PLANNEDSTARTDATEOFPROD</dbName>
    </fieldMapping>
    <fieldMapping id="f8" isElement="true">
      <xmlName>plannedEndDate</xmlName>
      <dbName>PLANNEDENDDATEOFPROD</dbName>
    </fieldMapping>
    <fieldMapping id="f9" isElement="true">
      <xmlName>startDate</xmlName>
      <dbName>STARTDATEOFPROD</dbName>
    </fieldMapping>
    <fieldMapping id="f10" isElement="true">
      <xmlName>endDate</xmlName>
      <dbName>ENDDATEOFPROD</dbName>
    </fieldMapping>
    <fieldMapping id="f11" isElement="true">
      <xmlName>slackOverrun</xmlName>
      <dbName>SLACKOVERRUN</dbName>
    </fieldMapping>
    <fieldMapping id="f12" isElement="true">
      <xmlName>progress</xmlName>
      <dbName>PROGRESS</dbName>
    </fieldMapping>
    <fieldMapping id="f13" isElement="true">
      <xmlName>lastFeedback</xmlName>
      <dbName>LEASTFEEDBACK</dbName>
    </fieldMapping>
    <passThrough name="leadTime" isPCDATA="true">
      <fieldMapping id="f14">
        <xmlName>unit</xmlName>
        <dbName>LEADTIMEUNIT</dbName>
      </fieldMapping>
      <fieldMapping id="f15" isElement="true">
        <xmlName>value</xmlName>
        <dbName>LEADTIME</dbName>
      </fieldMapping>
    </passThrough>
    <tableMapping xmlName="production_detail" dbName="TABLEPRODUCTIONORDERDETAIL"
maxOccurs="unbounded">
      <foreignKey mapped="true">
        <from>PO_CODE</from>
        <to table="TABLEPRODUCTIONORDER">PO_CODE</to>
      </foreignKey>
```

```
      <foreignKey>
        <from>ENTERPRISE_CODE</from>
        <to table="TABLEPRODUCTIONORDER">ENTERPRISE_CODE</to>
      </foreignKey>
      <fieldMapping id="f16" isElement="true" isPK="true">
        <xmlName>order_number</xmlName>
        <dbName>PO_CODE</dbName>
      </fieldMapping>
      <fieldMapping id="f17" isElement="true" isPK="true">
        <xmlName>detail_number</xmlName>
        <dbName>PO_DETAIL_CODE</dbName>
      </fieldMapping>
      <fieldMapping id="f18" isElement="true">
        <xmlName>plannedStartDate</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f19" isElement="true">
        <xmlName>plannedEndDate</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f20" isElement="true">
        <xmlName>startDate</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f21" isElement="true">
        <xmlName>endDate</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f22" isElement="true">
        <xmlName>deliveryDate</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f23" isElement="true">
        <xmlName>priceTotal</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f24" isElement="true">
        <xmlName>slackOverrun</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f25" isElement="true">
        <xmlName>status</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f26" isElement="true">
        <xmlName>progress</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f27" isElement="true">
        <xmlName>lastFeedback</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f28" isElement="true">
        <xmlName>requestedQuantity</xmlName>
        <dbName>REQUESTEDQUANTITY</dbName>
      </fieldMapping>
      <fieldMapping id="f29" isElement="true">
        <xmlName>producedQuantity</xmlName>
        <dbName>PRODUCEDQUANTITY</dbName>
      </fieldMapping>
      <fieldMapping id="f30" isElement="true">
        <xmlName>requestedLotSize</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f31" isElement="true">
        <xmlName>producedLotSize</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f32" isElement="true">
        <xmlName>lotValue</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <passThrough name="cycleTime" isPCDATA="true">
        <fieldMapping id="f33">
          <xmlName>unit</xmlName>
```

```
          <dbName>CYCLETIMEUNIT</dbName>
        </fieldMapping>
        <fieldMapping id="f34" isElement="true">
          <xmlName>value</xmlName>
          <dbName>CYCLETIME</dbName>
        </fieldMapping>
      </passThrough>
      <fieldMapping id="f35" isElement="true">
        <xmlName>earliestStartDate</xmlName>
        <dbName>EARLIESTSTARTDATEOFPROD</dbName>
      </fieldMapping>
      <fieldMapping id="f36" isElement="true">
        <xmlName>latestStartDate</xmlName>
        <dbName>LATESTSTARTDATEOFPROD</dbName>
      </fieldMapping>
      <fieldMapping id="f37" isElement="true">
        <xmlName>earliestEndDate</xmlName>
        <dbName>EARLIESTENDDATEOFPROD</dbName>
      </fieldMapping>
      <fieldMapping id="f38" isElement="true">
        <xmlName>latestEndDate</xmlName>
        <dbName>LATESTENDDATEOFPROD</dbName>
      </fieldMapping>
      <passThrough name="related_sales_order">
        <fieldMapping id="f39" isElement="true">
          <xmlName>sales_number</xmlName>
          <dbName>RELATEDCUSTOMERORDER</dbName>
        </fieldMapping>
        <fieldMapping id="f40" isElement="true">
          <xmlName>sales_line</xmlName>
          <dbName>RELATEDCUSTOMERROW</dbName>
        </fieldMapping>
      </passThrough>
      <passThrough name="productSpecification">
        <fieldMapping id="f41" isElement="true">
          <xmlName>cod_prod</xmlName>
          <dbName>PRODUCT</dbName>
        </fieldMapping>
        <fieldMapping id="f42" isElement="true">
          <xmlName>size</xmlName>
          <dbName></dbName>
        </fieldMapping>
        <fieldMapping id="f43" isElement="true">
          <xmlName>colour</xmlName>
          <dbName></dbName>
        </fieldMapping>
        <fieldMapping id="f44" isElement="true">
          <xmlName>price</xmlName>
          <dbName>PRICEPERUNIT</dbName>
        </fieldMapping>
      </passThrough>
    </tableMapping>
  </tableMapping>
</mapping>
```

3) Mapeamento relacionado ao DTD “sales_information”:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapping SYSTEM "mapping.dtd">
<mapping>
  <header>
    <xmlSpecification type="DTD">sales_information.dtd</xmlSpecification>
    <dbInformation>
      <location>
        jdbc:firebirdsql:150.162.27.185:D:/Working/Mestrado/Database/MyFashionDM.GDB
      </location>
      <driver>org.firebirdsql.jdbc.FBDriver</driver>
      <user>SYSDBA</user>
      <pass>masterkey</pass>
    </dbInformation>
  </header>
  <tableMapping xmlName="sales_order" dbName="TABLECUSTOMERORDER">
    <foreignKey mapped="true">
```

```
      <from>SC_CODE</from>
      <to table="TABLESUPPLYCHAIN">SC_CODE</to>
    </foreignKey>
    <fieldMapping id="f1" isElement="true" isPK="true">
      <xmlName>supply_chain_id</xmlName>
      <dbName>SC_CODE</dbName>
    </fieldMapping>
    <fieldMapping id="f2" isElement="true" isPK="true">
      <xmlName>order_number</xmlName>
      <dbName>CO_CODE</dbName>
    </fieldMapping>
    <fieldMapping id="f3" isElement="true">
      <xmlName>order_date</xmlName>
      <dbName>DATE_</dbName>
    </fieldMapping>
    <fieldMapping id="f4" isElement="true">
      <xmlName>delivery_date</xmlName>
      <dbName>DELIVERY_DATE</dbName>
    </fieldMapping>
    <fieldMapping id="f5" isElement="true">
      <xmlName>sales_agent</xmlName>
      <dbName></dbName>
    </fieldMapping>
    <fieldMapping id="f6" isElement="true">
      <xmlName>client</xmlName>
      <dbName>CLIENT_CODE</dbName>
    </fieldMapping>
    <fieldMapping id="f7" isElement="true">
      <xmlName>season</xmlName>
      <dbName></dbName>
    </fieldMapping>
    <fieldMapping id="f8" isElement="true">
      <xmlName>status</xmlName>
      <dbName>STATUS</dbName>
    </fieldMapping>
    <fieldMapping id="f9" isElement="true">
      <xmlName>total_amount</xmlName>
      <dbName></dbName>
    </fieldMapping>
    <tableMapping xmlName="sales_detail" dbName="TABLECUSTOMERORDERDETAIL"
maxOccurs="unbounded">
      <foreignKey>
        <from>SC_CODE</from>
        <to table="TABLECUSTOMERORDER">SC_CODE</to>
      </foreignKey>
      <foreignKey>
        <from>CO_CODE</from>
        <to table="TABLECUSTOMERORDER">CO_CODE</to>
      </foreignKey>
      <fieldMapping id="f10" isElement="true" isPK="true">
        <xmlName>line_number</xmlName>
        <dbName>CO_DETAIL_CODE</dbName>
      </fieldMapping>
      <fieldMapping id="f11" isElement="true">
        <xmlName>cod_prod</xmlName>
        <dbName>PRODUCT_CODE</dbName>
      </fieldMapping>
      <fieldMapping id="f12" isElement="true">
        <xmlName>size</xmlName>
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f13" isElement="true">
        <xmlName>colour</xmlName>
        <dbName>COLOR</dbName>
      </fieldMapping>
      <fieldMapping id="f14" isElement="true">
        <xmlName>price</xmlName>
        <dbName>UNIT_PRICE</dbName>
      </fieldMapping>
      <fieldMapping id="f15" isElement="true">
        <xmlName>qty</xmlName>
        <dbName>QUANTITY</dbName>
      </fieldMapping>
      <fieldMapping id="f16" isElement="true">
        <xmlName>class</xmlName>
```

```
        <dbName></dbName>
      </fieldMapping>
      <fieldMapping id="f17" isElement="true">
        <xmlName>segment</xmlName>
        <dbName>SEGMENT_</dbName>
      </fieldMapping>
    </tableMapping>
  </tableMapping>
</mapping>
```

B.4 – Armazenamento de Documentos XML

A seguir são mostrados, em seqüência, os registros (*log*) das operações realizadas para o armazenamento de alguns dos documentos XML utilizados no teste (seção 8.2.2). Como o processo é análogo em todas as situações, serão mostrados apenas três documentos, relacionados a cada um dos DTDs que definem o MR adotado.

1) eSupplier-Cordolona.xml (enterprise_information.dtd):

```
Connected to ' jdbc:firebirdsql:150.162.27.185:D:/Working/Mestrado/Database/MyFashionDM.GDB'

------------------------------------------------------------------------------------------
Saving XML Document:

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE enterprise_info SYSTEM "enterprise_information.dtd">
<enterprise_info>
  <generic_info>
    <code_ent>E_00264</code_ent>
    <name_ent>Cordolona</name_ent>
    <type>textile</type>
    <contact>
      <contact_name>Roberto Carlos Caldas</contact_name>
      <contact_phone>+351222097200</contact_phone>
      <contact_fax>+351222097300</contact_fax>
      <email>caldas@cordolona.co.pt</email>
      <function>DepartmentChef</function>
      <department>sales</department>
    </contact>
    <address>
      <street>Gen. Humberto Delgado</street>
      <number>3520</number>
      <aditional>0</aditional>
      <zipcode>4049-001</zipcode>
      <POBox>3655</POBox>
      <city>Porto</city>
      <region/>
      <country>Portugal</country>
      <web>www.cordolona.co.pt</web>
    </address>
  </generic_info>
  <product_info>
    <code_prod>P10266</code_prod>
    <name_prod>canvas</name_prod>
    <max_capacity>1000</max_capacity>
    <allocated_capacity>9</allocated_capacity>
    <production_price>1000000</production_price>
    <color>brown</color>
    <volume unit="cm3">1</volume>
    <height unit="cm">1</height>
    <width unit="cm">1</width>
    <weight unit="kg">1</weight>
    <material>canvas</material>
```

```
    <productType>textile</productType>
    <productionLot>656</productionLot>
    <productAcquisition>total production</productAcquisition>
    <components_list/>
  </product_info>
  <product_info>
    <code_prod>P10277</code_prod>
    <name_prod>shoe blue cord</name_prod>
    <max_capacity>1000</max_capacity>
    <allocated_capacity>9</allocated_capacity>
    <production_price>1000000</production_price>
    <color>blue</color>
    <volume unit="cm3">1</volume>
    <height unit="cm">1</height>
    <width unit="cm">1</width>
    <weight unit="kg">1</weight>
    <material>cord</material>
    <productType>textile</productType>
    <productionLot>657</productionLot>
    <productAcquisition>total production</productAcquisition>
    <components_list/>
  </product_info>
  <product_info>
    <code_prod>P10278</code_prod>
    <name_prod>shoe black cord</name_prod>
    <max_capacity>1000</max_capacity>
    <allocated_capacity>9</allocated_capacity>
    <production_price>1000000</production_price>
    <color>black</color>
    <volume unit="cm3">1</volume>
    <height unit="cm">1</height>
    <width unit="cm">1</width>
    <weight unit="kg">1</weight>
    <material>cord</material>
    <productType>textile</productType>
    <productionLot>658</productionLot>
    <productAcquisition>total production</productAcquisition>
    <components_list/>
  </product_info>
</enterprise_info>
-----------------------------------------------------------------------------------------
Setting initial values for possible incremental primary keys (keys that does not have a
related XML element/attribute).

SELECT MAX(CONTACT_CODE) FROM TABLECONTACT

Max value for CONTACT_CODE: 3

-----------------------------------------------------------------------------------------
SELECT * FROM TABLEENTERPRISE
WHERE
ENTERPRISE_CODE = 'E_00264'

INSERT INTO TABLEENTERPRISE
(STATE, COUNTRY, CITY, ADDITIONAL, STREET, NAME, NUMBER, ENTERPRISE_CODE, PO_BOX, ZIP_CODE,
BRANCH)
VALUES
('', 'Portugal', 'Porto', '0', 'Gen. Humberto Delgado', 'Cordolona', '3520', 'E_00264',
'3655', '4049-001', 'textile')

-----------------------------------------------------------------------------------------
SELECT * FROM TABLEPRODUCT
WHERE
PRODUCT_CODE = 'P10266' AND ENTERPRISE_CODE = 'E_00264'

INSERT INTO TABLEPRODUCT
(PRODUCT_CODE, PRICE, MATERIAL, NAME, ENTERPRISE_CODE)
VALUES
('P10266', '1000000', 'canvas', 'canvas', 'E_00264')

-----------------------------------------------------------------------------------------
SELECT * FROM TABLEPRODUCT
WHERE
PRODUCT_CODE = 'P10277' AND ENTERPRISE_CODE = 'E_00264'
```

```
INSERT INTO TABLEPRODUCT
(PRODUCT_CODE, PRICE, MATERIAL, NAME, ENTERPRISE_CODE)
VALUES
('P10277', '1000000', 'cord', 'shoe blue cord', 'E_00264')

--------------------------------------------------------------------------------------------
SELECT * FROM TABLEPRODUCT
WHERE
PRODUCT_CODE = 'P10278' AND ENTERPRISE_CODE = 'E_00264'

INSERT INTO TABLEPRODUCT
(PRODUCT_CODE, PRICE, MATERIAL, NAME, ENTERPRISE_CODE)
VALUES
('P10278', '1000000', 'cord', 'shoe black cord', 'E_00264')

--------------------------------------------------------------------------------------------
SELECT * FROM TABLECONTACT
WHERE
ENTERPRISE_CODE = 'E_00264' AND CONTACT_CODE = '4'

INSERT INTO TABLECONTACT
(FUNCTIONS, DEPARTMENT, FAX, NAME, ENTERPRISE_CODE, EMAIL, PHONE, CONTACT_CODE)
VALUES
('DepartmentChef', 'sales', '+351222097300', 'Roberto Carlos Caldas', 'E_00264',
'caldas@cordolona.co.pt', '+351222097200', '4')

--------------------------------------------------------------------------------------------
```

2) ProductionCordolona.xml (production_information.dtd):

```
Connected to ' jdbc:firebirdsql:150.162.27.185:D:/Working/Mestrado/Database/MyFashionDM.GDB'

--------------------------------------------------------------------------------------------
Saving XML Document:

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE production_order SYSTEM "production_information.dtd">
<production_order>
  <supply_chain_id>73</supply_chain_id>
  <order_number>RO3888</order_number>
  <order_date>2002-02-19</order_date>
  <deliveryDate>2002-02-27</deliveryDate>
  <supplier>E_00264</supplier>
  <status>In Progress</status>
  <plannedStartDate>2002-02-23</plannedStartDate>
  <plannedEndDate>2002-02-27</plannedEndDate>
  <startDate>2002-02-23</startDate>
  <endDate>2002-02-25</endDate>
  <slackOverrun>960</slackOverrun>
  <progress>0.8</progress>
  <lastFeedback>2002-02-23</lastFeedback>
  <leadTime unit="hours">36</leadTime>
  <production_detail>
    <order_number>RO3888</order_number>
    <detail_number>RI10266</detail_number>
    <plannedStartDate>2002-02-23</plannedStartDate>
    <plannedEndDate>2002-02-25</plannedEndDate>
    <startDate>2002-02-23</startDate>
    <endDate>2002-02-25</endDate>
    <deliveryDate>2002-02-27</deliveryDate>
    <priceTotal>1500</priceTotal>
    <slackOverrun>240</slackOverrun>
    <status>Executed</status>
    <progress>1</progress>
    <lastFeedback>2002-02-23</lastFeedback>
    <requestedQuantity>150</requestedQuantity>
    <producedQuantity>150</producedQuantity>
    <requestedLotSize>3</requestedLotSize>
    <producedLotSize>3</producedLotSize>
    <lotValue>50</lotValue>
    <cycleTime unit="min">120</cycleTime>
    <earliestStartDate>2002-02-23</earliestStartDate>
    <latestStartDate>2002-02-25</latestStartDate>
```

```
    <earliestEndDate>2002-02-25</earliestEndDate>
    <latestEndDate>2002-02-25</latestEndDate>
    <related_sales_order>
      <sales_number>Sa_O_0153</sales_number>
      <sales_line>1</sales_line>
    </related_sales_order>
    <productSpecification>
      <cod_prod>P10266</cod_prod>
      <size>50</size>
      <colour>50 Brown</colour>
      <price>500</price>
    </productSpecification>
  </production_detail>
  <production_detail>
    <order_number>RO3888</order_number>
    <detail_number>RI10277</detail_number>
    <plannedStartDate>2002-02-25</plannedStartDate>
    <plannedEndDate>2002-02-26</plannedEndDate>
    <startDate>2002-02-25</startDate>
    <endDate>2002-02-26</endDate>
    <deliveryDate>2002-02-27</deliveryDate>
    <priceTotal>2000</priceTotal>
    <slackOverrun>30</slackOverrun>
    <status>Executed</status>
    <progress>1</progress>
    <lastFeedback>2002-02-25</lastFeedback>
    <requestedQuantity>200</requestedQuantity>
    <producedQuantity>200</producedQuantity>
    <requestedLotSize>4</requestedLotSize>
    <producedLotSize>4</producedLotSize>
    <lotValue>50</lotValue>
    <cycleTime unit="min">120</cycleTime>
    <earliestStartDate>2002-02-23</earliestStartDate>
    <latestStartDate>2002-02-26</latestStartDate>
    <earliestEndDate>2002-02-26</earliestEndDate>
    <latestEndDate>2002-02-25</latestEndDate>
    <related_sales_order>
      <sales_number>Sa_O_0152</sales_number>
      <sales_line>1</sales_line>
    </related_sales_order>
    <related_sales_order>
      <sales_number>Sa_O_0152</sales_number>
      <sales_line>2</sales_line>
    </related_sales_order>
    <productSpecification>
      <cod_prod>P10277</cod_prod>
      <size>50</size>
      <colour>20 Blue</colour>
      <price>500</price>
    </productSpecification>
  </production_detail>
  <production_detail>
    <order_number>RO3888</order_number>
    <detail_number>RI10278</detail_number>
    <plannedStartDate>2002-02-25</plannedStartDate>
    <plannedEndDate>2002-02-27</plannedEndDate>
    <startDate>2002-02-23</startDate>
    <endDate>2002-02-25</endDate>
    <deliveryDate>2002-02-27</deliveryDate>
    <priceTotal>3000</priceTotal>
    <slackOverrun>150</slackOverrun>
    <status>In Progress</status>
    <progress>0.75</progress>
    <lastFeedback>2002-02-23</lastFeedback>
    <requestedQuantity>300</requestedQuantity>
    <producedQuantity>100</producedQuantity>
    <requestedLotSize>6</requestedLotSize>
    <producedLotSize>4</producedLotSize>
    <lotValue>50</lotValue>
    <cycleTime unit="min">120</cycleTime>
    <earliestStartDate>2002-02-23</earliestStartDate>
    <latestStartDate>2002-02-25</latestStartDate>
    <earliestEndDate>2002-02-25</earliestEndDate>
    <latestEndDate>2002-02-27</latestEndDate>
    <related_sales_order>
```

```
      <sales_number>Sa_O_0152</sales_number>
      <sales_line>3</sales_line>
    </related_sales_order>
    <productSpecification>
      <cod_prod>P10278</cod_prod>
      <size>50</size>
      <colour>31 Black</colour>
      <price>500</price>
    </productSpecification>
  </production_detail>
</production_order>
-----------------------------------------------------------------------------------------
Setting initial values for possible incremental primary keys (keys that does not have a
related XML element/attribute).

-----------------------------------------------------------------------------------------
SELECT * FROM TABLEPRODUCTIONORDER
WHERE
PO_CODE = 'RO3888' AND ENTERPRISE_CODE = 'E_00264'

INSERT INTO TABLEPRODUCTIONORDER
(LEASTFEEDBACK, DELIVERY_DATE, PO_CODE, STARTDATEOFPROD, LEADTIME, PLANNEDENDDATEOFPROD,
SC_CODE, STATUS, SLACKOVERRUN, PLANNEDSTARTDATEOFPROD, ENDDATEOFPROD, LEADTIMEUNIT,
ENTERPRISE_CODE, DATE_, PROGRESS)
VALUES
('2002-02-23', '2002-02-27', 'RO3888', '2002-02-23', '36', '2002-02-27', '73', 'In
Progress', '960', '2002-02-23', '2002-02-25', 'hours', 'E_00264', '2002-02-19', '0.8')

-----------------------------------------------------------------------------------------
SELECT * FROM TABLEPRODUCTIONORDERDETAIL
WHERE
PO_CODE = 'RO3888' AND PO_DETAIL_CODE = 'RI10266' AND ENTERPRISE_CODE = 'E_00264'

INSERT INTO TABLEPRODUCTIONORDERDETAIL
(CYCLETIME, RELATEDCUSTOMERROW, PO_CODE, RELATEDCUSTOMERORDER, PRICEPERUNIT,
REQUESTEDQUANTITY, EARLIESTSTARTDATEOFPROD, PRODUCEDQUANTITY, CYCLETIMEUNIT, PRODUCT,
LATESTSTARTDATEOFPROD, EARLIESTENDDATEOFPROD, LATESTENDDATEOFPROD, PO_DETAIL_CODE,
ENTERPRISE_CODE)
VALUES
('120', '1', 'RO3888', 'Sa_O_0153', '500', '150', '2002-02-23', '150', 'min', 'P10266',
'2002-02-25', '2002-02-25', '2002-02-25', 'RI10266', 'E_00264')

-----------------------------------------------------------------------------------------
SELECT * FROM TABLEPRODUCTIONORDERDETAIL
WHERE
PO_CODE = 'RO3888' AND PO_DETAIL_CODE = 'RI10277' AND ENTERPRISE_CODE = 'E_00264'

INSERT INTO TABLEPRODUCTIONORDERDETAIL
(CYCLETIME, RELATEDCUSTOMERROW, PO_CODE, RELATEDCUSTOMERORDER, PRICEPERUNIT,
REQUESTEDQUANTITY, EARLIESTSTARTDATEOFPROD, PRODUCEDQUANTITY, CYCLETIMEUNIT, PRODUCT,
LATESTSTARTDATEOFPROD, EARLIESTENDDATEOFPROD, LATESTENDDATEOFPROD, PO_DETAIL_CODE,
ENTERPRISE_CODE)
VALUES
('120', '2', 'RO3888', 'Sa_O_0152', '500', '200', '2002-02-23', '200', 'min', 'P10277',
'2002-02-26', '2002-02-26', '2002-02-25', 'RI10277', 'E_00264')

-----------------------------------------------------------------------------------------
SELECT * FROM TABLEPRODUCTIONORDERDETAIL
WHERE
PO_CODE = 'RO3888' AND PO_DETAIL_CODE = 'RI10278' AND ENTERPRISE_CODE = 'E_00264'

INSERT INTO TABLEPRODUCTIONORDERDETAIL
(CYCLETIME, RELATEDCUSTOMERROW, PO_CODE, RELATEDCUSTOMERORDER, PRICEPERUNIT,
REQUESTEDQUANTITY, EARLIESTSTARTDATEOFPROD, PRODUCEDQUANTITY, CYCLETIMEUNIT, PRODUCT,
LATESTSTARTDATEOFPROD, EARLIESTENDDATEOFPROD, LATESTENDDATEOFPROD, PO_DETAIL_CODE,
ENTERPRISE_CODE)
VALUES
('120', '3', 'RO3888', 'Sa_O_0152', '500', '300', '2002-02-23', '100', 'min', 'P10278',
'2002-02-25', '2002-02-25', '2002-02-27', 'RI10278', 'E_00264')

-----------------------------------------------------------------------------------------
```

3) salesStyleModas.xml (sales_information.dtd):

```
Connected to ' jdbc:firebirdsql:150.162.27.185:D:/Working/Mestrado/Database/MyFashionDM.GDB'

--------------------------------------------------------------------------------------------
Saving XML Document:

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sales_order SYSTEM "sales_information.dtd">
<sales_order>
  <supply_chain_id>73</supply_chain_id>
  <order_number>4</order_number>
  <order_date>2002-03-19</order_date>
  <delivery_date>2002-04-05</delivery_date>
  <sales_agent>11</sales_agent>
  <client>8</client>
  <season>I02</season>
  <status>Sent</status>
  <total_amount>161</total_amount>
  <sales_detail>
    <line_number>1</line_number>
    <cod_prod>1</cod_prod>
    <size>39</size>
    <colour>36 Dune</colour>
    <price>26.00</price>
    <qty>60</qty>
    <class>Suede B93</class>
    <segment>Under 19</segment>
  </sales_detail>
  <sales_detail>
    <line_number>2</line_number>
    <cod_prod>3</cod_prod>
    <size>40</size>
    <colour>50 Brown</colour>
    <price>26.50</price>
    <qty>60</qty>
    <class>Shiny Leat A68</class>
    <segment>Under 20</segment>
  </sales_detail>
  <sales_detail>
    <line_number>3</line_number>
    <cod_prod>4</cod_prod>
    <size>41</size>
    <colour>94 Aged Blue</colour>
    <price>27.00</price>
    <qty>60</qty>
    <class>Leat. V25/V69</class>
    <segment>Under 21</segment>
  </sales_detail>
</sales_order>
--------------------------------------------------------------------------------------------
Setting initial values for possible incremental primary keys (keys that does not have a
related XML element/attribute).

--------------------------------------------------------------------------------------------
SELECT * FROM TABLECUSTOMERORDER
WHERE
SC_CODE = '73' AND CO_CODE = '4'

INSERT INTO TABLECUSTOMERORDER
(SC_CODE, CO_CODE, STATUS, DELIVERY_DATE, CLIENT_CODE, DATE_)
VALUES
('73', '4', 'Sent', '2002-04-05', '8', '2002-03-19')

--------------------------------------------------------------------------------------------
SELECT * FROM TABLECUSTOMERORDERDETAIL
WHERE
SC_CODE = '73' AND CO_CODE = '4' AND CO_DETAIL_CODE = '1'

INSERT INTO TABLECUSTOMERORDERDETAIL
(SEGMENT_, SC_CODE, CO_CODE, PRODUCT_CODE, QUANTITY, CO_DETAIL_CODE, UNIT_PRICE, COLOR)
VALUES
('Under 19', '73', '4', '1', '60', '1', '26.00', '36 Dune')

--------------------------------------------------------------------------------------------
SELECT * FROM TABLECUSTOMERORDERDETAIL
WHERE
```

```
SC_CODE = '73' AND CO_CODE = '4' AND CO_DETAIL_CODE = '2'

INSERT INTO TABLECUSTOMERORDERDETAIL
(SEGMENT_, SC_CODE, CO_CODE, PRODUCT_CODE, QUANTITY, CO_DETAIL_CODE, UNIT_PRICE, COLOR)
VALUES
('Under 20', '73', '4', '3', '60', '2', '26.50', '50 Brown')

--------------------------------------------------------------------------------------------
SELECT * FROM TABLECUSTOMERORDERDETAIL
WHERE
SC_CODE = '73' AND CO_CODE = '4' AND CO_DETAIL_CODE = '3'

INSERT INTO TABLECUSTOMERORDERDETAIL
(SEGMENT_, SC_CODE, CO_CODE, PRODUCT_CODE, QUANTITY, CO_DETAIL_CODE, UNIT_PRICE, COLOR)
VALUES
('Under 21', '73', '4', '4', '60', '3', '27.00', '94 Aged Blue')

--------------------------------------------------------------------------------------------
```

B.5 – Carregamento de Documentos XML

A seguir são mostrados, em seqüência, os registros (*log*) das operações realizadas para o carregamento de dados, a partir de algumas consultas XML utilizadas no teste (seção 8.2.2). Será mostrada apenas uma consulta para cada um dos DTDs que definem o MR adotado.

1) queryEnterprise2.xml (enterprise_information.dtd)

```
Connected to ' jdbc:firebirdsql:150.162.27.185:D:/Working/Mestrado/Database/MyFashionDM.GDB'

--------------------------------------------------------------------------------------------
XML Query:

<?xml version="1.0"?>

<!DOCTYPE select SYSTEM "XMLQuery.dtd">

<select>
  <element>enterprise_info</element>
  <from>enterprise_information.dtd</from>
  <where>
    <condition op1="enterprise_info/generic_info/address/country" operator="="
op2="Portugal" />
  </where>
</select>

--------------------------------------------------------------------------------------------
SQL Query Generated:

SELECT
TABLEPRODUCT.NAME AS f20,
TABLEPRODUCT.MATERIAL AS f33,
TABLEPRODUCT.PRICE AS f23,
TABLEPRODUCT.PRODUCT_CODE AS f19,
TABLEENTERPRISE.ENTERPRISE_CODE AS f1,
TABLEENTERPRISE.BRANCH AS f3,
TABLEENTERPRISE.NAME AS f2,
TABLECONTACT.NAME AS f4,
TABLECONTACT.EMAIL AS f7,
TABLECONTACT.FUNCTIONS AS f8,
TABLECONTACT.PHONE AS f5,
TABLECONTACT.FAX AS f6,
TABLECONTACT.DEPARTMENT AS f9,
```

```
TABLEENTERPRISE.PO_BOX AS f14,
TABLEENTERPRISE.COUNTRY AS f17,
TABLEENTERPRISE.STREET AS f10,
TABLEENTERPRISE.STATE AS f16,
TABLEENTERPRISE.ADDITIONAL AS f12,
TABLEENTERPRISE.CITY AS f15,
TABLEENTERPRISE.ZIP_CODE AS f13,
TABLEENTERPRISE.NUMBER AS f11
FROM
TABLEPRODUCT,
TABLECONTACT,
TABLEENTERPRISE
WHERE
TABLEENTERPRISE.COUNTRY = 'Portugal' AND
TABLEPRODUCT.ENTERPRISE_CODE = TABLEENTERPRISE.ENTERPRISE_CODE AND
TABLECONTACT.ENTERPRISE_CODE = TABLEENTERPRISE.ENTERPRISE_CODE

--------------------------------------------------------------------------------------------
XML Document(s) generated:

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE enterprise_info SYSTEM "enterprise_information.dtd">
<enterprise_info>
<generic_info>
<code_ent>{D83B8043-5F81-414D-A748-792B42D09781}</code_ent>
<name_ent>Bruno Belloni</name_ent>
<type>Textile</type>
<contact>
<contact_name>José</contact_name>
<contact_phone/>
<contact_fax/>
<email/>
<function/>
<department/>
</contact>
<address>
<street/>
<number/>
<aditional/>
<zipcode/>
<POBox/>
<city/>
<region/>
<country>Portugal</country>
<web/>
</address>
</generic_info>
<product_info>
<code_prod>880522</code_prod>
<name_prod>Fabric 880522</name_prod>
<max_capacity/>
<allocated_capacity/>
<production_price>52.25</production_price>
<color/>
<volume/>
<height/>
<width/>
<weight/>
<material>High quality</material>
<productType/>
<productionLot/>
<productAcquisition/>
<min_stock_limit/>
<max_stock_limit/>
<components_list>
<component>
<component_cod/>
<supplier_cod/>
<qty/>
<name/>
</component>
</components_list>
</product_info>
</enterprise_info>
```

```
--------------------------------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE enterprise_info SYSTEM "enterprise_information.dtd">
<enterprise_info>
<generic_info>
<code_ent>d5a13841-4a69-4f29-a893-aaa784efea8a</code_ent>
<name_ent>Silva&amp;amp;Sistelo</name_ent>
<type>Manufacturer</type>
<contact>
<contact_name>Victor Hugo Morais</contact_name>
<contact_phone>+351 22.485.30.00</contact_phone>
<contact_fax>+351 22.489.27.62</contact_fax>
<email>Victor.Morais@silvasistelo.pt</email>
<function/>
<department/>
</contact>
<address>
<street>Rua Sistelo</street>
<number/>
<aditional>Manufacturer suites</aditional>
<zipcode>4435-452</zipcode>
<POBox/>
<city>Rio tinto</city>
<region>Porto</region>
<country>Portugal</country>
<web/>
</address>
</generic_info>
<product_info>
<code_prod>33600</code_prod>
<name_prod>Suit Boss</name_prod>
<max_capacity/>
<allocated_capacity/>
<production_price>54.36</production_price>
<color/>
<volume/>
<height/>
<width/>
<weight/>
<material>High quality</material>
<productType/>
<productionLot/>
<productAcquisition/>
<min_stock_limit/>
<max_stock_limit/>
<components_list>
<component>
<component_cod/>
<supplier_cod/>
<qty/>
<name/>
</component>
</components_list>
</product_info>
</enterprise_info>

--------------------------------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE enterprise_info SYSTEM "enterprise_information.dtd">
<enterprise_info>
<generic_info>
<code_ent>E_00264</code_ent>
<name_ent>Cordolona</name_ent>
<type>textile</type>
<contact>
<contact_name>Roberto Carlos Caldas</contact_name>
<contact_phone>+351222097200</contact_phone>
<contact_fax>+351222097300</contact_fax>
<email>caldas@cordolona.co.pt</email>
<function>DepartmentChef</function>
<department>sales</department>
</contact>
<contact>
<contact_name>Roberto Carlos Caldas</contact_name>
<contact_phone>+351222097200</contact_phone>
```

```
<contact_fax>+351222097300</contact_fax>
<email>caldas@cordolona.co.pt</email>
<function>DepartmentChef</function>
<department>sales</department>
</contact>
<contact>
<contact_name>Roberto Carlos Caldas</contact_name>
<contact_phone>+351222097200</contact_phone>
<contact_fax>+351222097300</contact_fax>
<email>caldas@cordolona.co.pt</email>
<function>DepartmentChef</function>
<department>sales</department>
</contact>
<address>
<street>Gen. Humberto Delgado</street>
<number>3520</number>
<aditional>0</aditional>
<zipcode>4049-001</zipcode>
<POBox>3655</POBox>
<city>Porto</city>
<region/>
<country>Portugal</country>
<web/>
</address>
</generic_info>
<product_info>
<code_prod>P10266</code_prod>
<name_prod>canvas</name_prod>
<max_capacity/>
<allocated_capacity/>
<production_price>1000000.0</production_price>
<color/>
<volume/>
<height/>
<width/>
<weight/>
<material>canvas</material>
<productType/>
<productionLot/>
<productAcquisition/>
<min_stock_limit/>
<max_stock_limit/>
<components_list>
<component>
<component_cod/>
<supplier_cod/>
<qty/>
<name/>
</component>
</components_list>
</product_info>
<product_info>
<code_prod>P10277</code_prod>
<name_prod>shoe blue cord</name_prod>
<max_capacity/>
<allocated_capacity/>
<production_price>1000000.0</production_price>
<color/>
<volume/>
<height/>
<width/>
<weight/>
<material>cord</material>
<productType/>
<productionLot/>
<productAcquisition/>
<min_stock_limit/>
<max_stock_limit/>
<components_list>
<component>
<component_cod/>
<supplier_cod/>
<qty/>
<name/>
</component>
```

```
</components_list>
</product_info>
<product_info>
<code_prod>P10278</code_prod>
<name_prod>shoe black cord</name_prod>
<max_capacity/>
<allocated_capacity/>
<production_price>1000000.0</production_price>
<color/>
<volume/>
<height/>
<width/>
<weight/>
<material>cord</material>
<productType/>
<productionLot/>
<productAcquisition/>
<min_stock_limit/>
<max_stock_limit/>
<components_list>
<component>
<component_cod/>
<supplier_cod/>
<qty/>
<name/>
</component>
</components_list>
</product_info>
</enterprise_info>

--------------------------------------------------------------------------------------------
```

2) queryProduction1.xml (production_information.dtd)

```
Connected to ' jdbc:firebirdsql:150.162.27.185:D:/Working/Mestrado/Database/MyFashionDM.GDB'

--------------------------------------------------------------------------------------------
XML Query:

<?xml version="1.0"?>

<!DOCTYPE select SYSTEM "XMLQuery.dtd">

<select>
  <element>production_order</element>
  <from>production_information.dtd</from>
  <where>
    <and>
      <condition op1="production_order/supplier" operator="=" op2="E112AA85-7803-4203-8FBB-
4C7C51C57005" />
      <condition op1="production_order/order_number" operator="=" op2="P3" />
    </and>
  </where>
</select>

--------------------------------------------------------------------------------------------
SQL Query Generated:

SELECT
TABLEPRODUCTIONORDER.STARTDATEOFPROD AS f9,
TABLEPRODUCTIONORDER.SLACKOVERRUN AS f11,
TABLEPRODUCTIONORDER.PLANNEDENDDATEOFPROD AS f8,
TABLEPRODUCTIONORDER.DATE_ AS f3,
TABLEPRODUCTIONORDER.STATUS AS f6,
TABLEPRODUCTIONORDER.PLANNEDSTARTDATEOFPROD AS f7,
TABLEPRODUCTIONORDER.PROGRESS AS f12,
TABLEPRODUCTIONORDER.ENTERPRISE_CODE AS f5,
TABLEPRODUCTIONORDER.LEASTFEEDBACK AS f13,
TABLEPRODUCTIONORDER.DELIVERY_DATE AS f4,
TABLEPRODUCTIONORDER.PO_CODE AS f2,
TABLEPRODUCTIONORDER.ENDDATEOFPROD AS f10,
TABLEPRODUCTIONORDER.SC_CODE AS f1,
TABLEPRODUCTIONORDERDETAIL.EARLIESTENDDATEOFPROD AS f37,
```

```
TABLEPRODUCTIONORDERDETAIL.LATESTSTARTDATEOFPROD AS f36,
TABLEPRODUCTIONORDERDETAIL.EARLIESTSTARTDATEOFPROD AS f35,
TABLEPRODUCTIONORDERDETAIL.PO_DETAIL_CODE AS f17,
TABLEPRODUCTIONORDERDETAIL.PRODUCEDQUANTITY AS f29,
TABLEPRODUCTIONORDERDETAIL.LATESTENDDATEOFPROD AS f38,
TABLEPRODUCTIONORDERDETAIL.PO_CODE AS f16,
TABLEPRODUCTIONORDERDETAIL.REQUESTEDQUANTITY AS f28,
TABLEPRODUCTIONORDERDETAIL.RELATEDCUSTOMERORDER AS f39,
TABLEPRODUCTIONORDERDETAIL.RELATEDCUSTOMERROW AS f40,
TABLEPRODUCTIONORDERDETAIL.CYCLETIME AS f34,
TABLEPRODUCTIONORDERDETAIL.CYCLETIMEUNIT AS f33,
TABLEPRODUCTIONORDERDETAIL.PRICEPERUNIT AS f44,
TABLEPRODUCTIONORDERDETAIL.PRODUCT AS f41,
TABLEPRODUCTIONORDER.LEADTIME AS f15,
TABLEPRODUCTIONORDER.LEADTIMEUNIT AS f14
FROM
TABLEPRODUCTIONORDER,
TABLEENTERPRISE,
TABLEPRODUCTIONORDERDETAIL
WHERE
TABLEPRODUCTIONORDER.ENTERPRISE_CODE = 'E112AA85-7803-4203-8FBB-4C7C51C57005' AND
TABLEPRODUCTIONORDER.PO_CODE = 'P3'  AND
TABLEPRODUCTIONORDER.ENTERPRISE_CODE = TABLEENTERPRISE.ENTERPRISE_CODE AND
TABLEPRODUCTIONORDERDETAIL.PO_CODE = TABLEPRODUCTIONORDER.PO_CODE AND
TABLEPRODUCTIONORDERDETAIL.ENTERPRISE_CODE = TABLEPRODUCTIONORDER.ENTERPRISE_CODE

--------------------------------------------------------------------------------------------
XML Document(s) generated:

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE production_order SYSTEM "production_information.dtd">
<production_order>
<supply_chain_id>toto</supply_chain_id>
<order_number>P3</order_number>
<order_date>2004-06-21 00:00:00.0</order_date>
<deliveryDate>2004-06-25 00:00:00.0</deliveryDate>
<supplier>E112AA85-7803-4203-8FBB-4C7C51C57005</supplier>
<status>executed</status>
<plannedStartDate>2004-06-22 00:00:00.0</plannedStartDate>
<plannedEndDate>2004-06-25 00:00:00.0</plannedEndDate>
<startDate>2004-06-22 00:00:00.0</startDate>
<endDate>2004-06-25 00:00:00.0</endDate>
<slackOverrun>0</slackOverrun>
<progress>100.0</progress>
<lastFeedback>2004-06-23 00:00:00.0</lastFeedback>
<leadTime>0</leadTime>
<production_detail>
<order_number>P3</order_number>
<detail_number>0</detail_number>
<plannedStartDate/>
<plannedEndDate/>
<startDate/>
<endDate/>
<deliveryDate/>
<priceTotal/>
<slackOverrun/>
<status/>
<progress/>
<lastFeedback/>
<requestedQuantity>1.0</requestedQuantity>
<producedQuantity>1.0</producedQuantity>
<requestedLotSize/>
<producedLotSize/>
<lotValue/>
<cycleTime>0</cycleTime>
<earliestStartDate>2004-06-22 00:00:00.0</earliestStartDate>
<latestStartDate>2004-06-22 00:00:00.0</latestStartDate>
<earliestEndDate>2004-06-25 00:00:00.0</earliestEndDate>
<latestEndDate>2004-06-25 00:00:00.0</latestEndDate>
<related_sales_order>
<sales_number>toto</sales_number>
<sales_line>0</sales_line>
</related_sales_order>
<productSpecification>
<cod_prod>311101</cod_prod>
```

```
<size/>
<colour/>
<price>0.0</price>
</productSpecification>
</production_detail>
<production_detail>
<order_number>P3</order_number>
<detail_number>1</detail_number>
<plannedStartDate/>
<plannedEndDate/>
<startDate/>
<endDate/>
<deliveryDate/>
<priceTotal/>
<slackOverrun/>
<status/>
<progress/>
<lastFeedback/>
<requestedQuantity>1.0</requestedQuantity>
<producedQuantity>1.0</producedQuantity>
<requestedLotSize/>
<producedLotSize/>
<lotValue/>
<cycleTime>0</cycleTime>
<earliestStartDate>2004-06-23 00:00:00.0</earliestStartDate>
<latestStartDate>2004-06-23 00:00:00.0</latestStartDate>
<earliestEndDate>2004-06-25 00:00:00.0</earliestEndDate>
<latestEndDate>2004-06-25 00:00:00.0</latestEndDate>
<related_sales_order>
<sales_number>toto</sales_number>
<sales_line>2</sales_line>
</related_sales_order>
<productSpecification>
<cod_prod>370005</cod_prod>
<size/>
<colour/>
<price>0.0</price>
</productSpecification>
</production_detail>
</production_order>

--------------------------------------------------------------------------------
```

3) querySales1.xml (sales_information.dtd)

```
Connected to ' jdbc:firebirdsql:150.162.27.185:D:/Working/Mestrado/Database/MyFashionDM.GDB'

--------------------------------------------------------------------------------
XML Query:

<?xml version="1.0"?>

<!DOCTYPE select SYSTEM "XMLQuery.dtd">

<select>
  <element>sales_order</element>
  <from>sales_information.dtd</from>
  <where>
    <condition op1="sales_order/supply_chain_id" operator="=" op2="4547773876547509290-
4543771344097426121" />
  </where>
</select>

--------------------------------------------------------------------------------
SQL Query Generated:

SELECT
TABLECUSTOMERORDER.STATUS AS f8,
TABLECUSTOMERORDER.CLIENT_CODE AS f6,
TABLECUSTOMERORDER.DELIVERY_DATE AS f4,
TABLECUSTOMERORDER.DATE_ AS f3,
TABLECUSTOMERORDER.CO_CODE AS f2,
TABLECUSTOMERORDER.SC_CODE AS f1,
```

```
TABLECUSTOMERORDERDETAIL.SEGMENT_ AS f17,
TABLECUSTOMERORDERDETAIL.UNIT_PRICE AS f14,
TABLECUSTOMERORDERDETAIL.QUANTITY AS f15,
TABLECUSTOMERORDERDETAIL.COLOR AS f13,
TABLECUSTOMERORDERDETAIL.PRODUCT_CODE AS f11,
TABLECUSTOMERORDERDETAIL.CO_DETAIL_CODE AS f10
FROM
TABLECUSTOMERORDERDETAIL,
TABLESUPPLYCHAIN,
TABLECUSTOMERORDER
WHERE
TABLECUSTOMERORDER.SC_CODE = '4547773876547509290-4543771344097426121' AND
TABLECUSTOMERORDER.SC_CODE = TABLESUPPLYCHAIN.SC_CODE AND
TABLECUSTOMERORDERDETAIL.SC_CODE = TABLECUSTOMERORDER.SC_CODE AND
TABLECUSTOMERORDERDETAIL.CO_CODE = TABLECUSTOMERORDER.CO_CODE

-------------------------------------------------------------------------------------------
XML Document(s) generated:

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sales_order SYSTEM "sales_information.dtd">
<sales_order>
<supply_chain_id>4547773876547509290-4543771344097426121</supply_chain_id>
<order_number>4547773876547509290-4543771344097426121</order_number>
<order_date>2004-06-21 00:00:00.0</order_date>
<delivery_date>2004-07-06 00:00:00.0</delivery_date>
<sales_agent/>
<client>857FB5CC-210B-48D9-A695-F15FD884B4A0</client>
<season/>
<status>Created</status>
<total_amount/>
<sales_detail>
<line_number>0</line_number>
<cod_prod>4656</cod_prod>
<size/>
<colour>001</colour>
<price>1.0</price>
<qty>1.0</qty>
<class/>
<segment/>
</sales_detail>
</sales_order>

-------------------------------------------------------------------------------------------
```

Glossário

A
ANSI – *American Nationan Standards Institute*
API – *Application Programming Interface*

C
CIM – *Computer Integrated Manufacturing*
COCOMO – *COnstructive COst MOdel* – Modelo Construtivo de Custo
CSS – *Cascading Style Sheets*

D
DAMASCOS – *Dynamic Forecast for Master Production Planning with Stock and Capacity Constraints*
DAML – *DARPA Agent Markup Language*
DOM – *Document Object Model* – Modelo de Objeto de Documento
DSML – *Directory Services Markup Language*
DTD – *Document Type Definition* – Definição do Tipo de Documento

E
EAI – *Enterprise Application Integration*
ebXML – *Electronic Business using Extensible Markup Language*
EDI – *Eletronic Data Interchange* – Intercâmbio Eletrônico de Dados
EBNF – *Extended Bakus-Naur Form*
ERP – *Enterprise Resource Planning*
EV – *Empresa Virtual*

G
GTDI – *Guidelines for Trade Data Interchange*
GML – *Generalized Markup Language* – Linguagem de Marcação Generalizada

H
HTML – *Hyper Text Markup Language* – Linguagem de Marcação para Hipertexto
HTTP – *Hypertext Transfer Protocol* – Protocolo para Transferência de Hipertexto

J
JDBC – *Java Data Base Connectivity*
JSP – *Java Server Pages*

M
MOF – *Meta Object Facility*
MOM – *Message Oriented Middleware*

O
OASIS – *Organization for the Advancement of Structured Information Standards*
ODBC – *Open Data Base Connectivity*
ODETTE – *Organization for Data Exchange Through Tele-transmission in Europe*

OIL – *Ontology Inference Layer*
OMG – *Object Management Group*
OWL – *Web Ontology Language*

P
PDML – *Product Data Markup Language*

R
RDF – *Resource Description Format*

S
SAX – *Simple API for XML*
SGML – *Standard Generalized Markup Language* – Linguagem de Marcação Generalizada Padrão
SLOC – *Source Lines of Code* – Linhas do Código-Fonte
SOAP – *Simple Object Access Protocol*
SQL – *Structured Query Language* – Linguagem de Consulta Estruturada
STEP – *STandard for the Exchange of Product Model Data*

U
UDDI – *Universal Description, Discovery, and Integration*
UML – *Unified Modeling Language* – Linguagem de Modelagem Unificada
UN/CEFACT – *United Nations Centre for Trade Facilitation and Electronic Business*
UN/EDIFACT – *United Nations/Eletronic Data Interchange For Administration Commerce and Transport*
URI – *Uniform Resource Identifier*
URL – *Uniform Resource Locator*

W
W3C – *World Wide Web Consortium*
WML – *Wireless Markup Language*
WSDL – *Web Service Definition Language*
WWW – *World Wide Web*

V
VAN – *Value Added Network*

X
XMI – *XML Metada Interchange* – Intercâmbio de Metadados XML
XML – *Extensible Markup Language* – Linguagem de Marcação Extensível
XSL – *Extensible Stylesheet Language*

Referências Bibliográficas

AFSARMANESH, H.; GARITA, C.; UGUR, Y.; *et al.* 1999. Federated Information Management Requirements for Virtual Enterprises. In: WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'99: 27-28 Oct. 1999: Porto, Portugal). *Proceedings*. Porto, Portugal. p. 31-48.

AHO, A. V.; SETHI, R; ULLMAN, J. D.; 1986. Compilers: Principles, Techniques and Tools. Addison-Wesley.

APEDI, 1998. EDI. http://www.apedi.pt/edi.htm, acessado em 11 de julho de 2003.

ÁVILA P.; PUTNIK, G. D.; CUNHA, M. M.; 2002. Brokerage Function in Agile Virtual Enterprise Integration – A Literature Review. In: THIRD IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'02). (1-3.: May. 2002: Sesimbra, Portugal). *Proceedings*. Sesimbra, Portugal. p. 65-72.

BALDO, F.; 2003. *Configuração Semi-Automática de Empresas Virtuais – Uma Abordagem Multiagente*. Florianópolis. Dissertação (Mestrado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.

BOEHM, B. W.; 1981. *Software Engineering Economics*. Prentice-Hall. ISBN 0138221227.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I.; 2000. *The Unified Modeling Language User Guide*. Addison-Wesley, ISBN 0-2014-57168-4.

BORLAND; 2003. http://www.borland.com, acessado em 14 de agosto de 2003.

BOURRET, R.; BORNHÖVD, C.; BUCHMANN, A.; 2000. A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases. In P. You, editor, Proc. of the 2nd Intl. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, pages 135-143. IEEE Comp. Soc., 2000.

BOURRET, R.; 2001a. Mapping W3C Schemas to Object Schemas to Relational Schemas. http://www.rpbourret.com/xml/SchemaMap.htm, acessado em 20 de março de 2003.

BOURRET, R.; 2001b. Mapping DTDs to Databases. http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html, acessado em 20 de março de 2003.

BOURRET, R.; 2002. DTD Parser, Version 2.0. http://www.rpbourret.com/dtdparser/index.htm, acessado em 25 de julho de 2003.

BOURRET, R.; 2003a. XML and Databases. http://www.rpbourret.com/xml/XMLAndDatabases.htm, acessado em 20 de março de 2003.

BOURRET, R.; 2003b. XML Database Products. http://www.rpbourret.com/xml/XMLDatabaseProds.htm, acessado em 15 de abril de 2003.

BOURRET, R.; 2003c. XML Data Binding Resources. http://www.rpbourret.com/xml/XMLDataBinding.htm, acessado em 15 de abril de 2003.

BOURRET, R.; 2003d. XML Database Products: Middleware. http://www.rpbourret.com/xml/ProdsMiddleware.htm, acessado em 30 de setembro de 2003.

BREEZE; 2003. http://www.breezefactor.com, acessado em 11 de agosto de 2003.

BURKETT, W. C.; 2001. Product data markup language: a new paradigm for product data exchange and integration. *Computer-Aided Design*, v. 33, i. 7 (Jun.), p. 489-500.

CAMARA-E; 2003. www.camara-e.net, acessado em 13 de março de 2003.

CAMARINHA-MATOS, L. M.; AFSARMANESH, H.; 1999a. The Virtual Enterprise Concept. In: WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'99: 27-28 Oct. 1999: Porto, Portugal). *Proceedings*. Porto, Portugal. p. 3-14.

CAMARINHA-MATOS, L. M.; AFSARMANESH, H.; 1999b. Tendencies and General Requirements for Virtual Enterprises. In: WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'99: 27-28 Oct. 1999: Porto, Portugal). *Proceedings*. Porto, Portugal. p. 15-30.

CAMARINHA-MATOS, L. M.; LIMA, C.P.; 1999c. Coordination and Configuration Requirements in a Virtual Enterprise. In: WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'99: 27-28 Oct. 1999: Porto, Portugal). *Proceedings*. Porto, Portugal. p. 49-64.

CAMARINHA-MATOS, L. M.; AFSARMANESH, H.; 1999d. The PRODNET Goals and Approach. In: WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'99: 27-28 Oct. 1999: Porto, Portugal). *Proceedings*. Porto, Portugal. p. 97-108.

CAMARINHA-MATOS, L. M.; AFSARMANESH, H.; 1999e. The PRODNET Architecture. In: WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'99: 27-28 Oct. 1999: Porto, Portugal). *Proceedings*. Porto, Portugal. p. 109-126.

CAMARINHA-MATOS, L. M.; AFSARMANESH, H.;RABELO, R.; 2000. Supporting Agility in Virtual Entreprises. In: SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brasil). *Proceedings*. Florianópolis, Brasil. p. 89-104.

CAMARINHA-MATOS, L. M.; 2004. ICT Infrastructures for Virtual Organizations. In: a ser publicado em International Journal of Networking and Virtual Organizations, Inderscience, Inglaterra, 2004.

CASTOR; 2003. http://www.castor.org, acessado em 11 de agosto de 2003.

CERAMI, E.; 2002. *Web Services Essentials*. 1st edition. O'Reilly & Associates. ISBN 0-596-00224-6.

CIMOSA; 2003. CIM Open System Architecture. http://cimosa.cnt.pl/, acessado em 22 de outubro de 2003.

COCOMO; 2004. CSE Center for Software Engineering – COCOMO. http://sunset.usc.edu/research/COCOMOII/, acessado em 8 de dezembro de 2004.

COCOMO81; 2004. COCOMO 81 Intermediate Model Implementation. http://sunset.usc.edu/research/COCOMOII/cocomo81_pgm/cocomo81.html, acessado em 8 de dezembro de 2004.

CORDIS; 2004. *Community Research & Development Information Service*. www.cordis.lu, acessado em 19 de fevereiro de 2004.

COVE; 2003. *VE Portal*. http://www.uninova.pt/~cove/projects.htm, acessado em 18 de novembro de 2003.

CXML; 2003. http://www.cxml.org/home/, acessado em 22 de outubro de 2003.

DAMASCOS; 2003. *Dynamic Forecast for Master Production Planning with Stock and Capacity Constraints*. http://www.inescporto.pt/~damascos/, acessado em 18 de novembro de 2003.

DAML; 2004. www.daml.org, acessado em 9 de maio de 2004.

DB2XML; 2003. http://www.informatik.fh-wiesbaden.de/~turau/DB2XML/index.html, acessado em 12 de agosto de 2003.

DSML; 2003. http://www.dsml.org/, acessado em 22 de outubro de 2003.

EBXML; 2003. About ebXML. http://www.ebxml.org/geninfo.htm, acessado em 24 de setembro de 2003.

ECLIPSE; 2003. www.eclipse.org, acessado em 18 de fevereiro de 2003.

EDI; 1999. the EDI Overview, excerpt from the "Report of the New York EDI Collaborative – Electronic Data Interchange (EDI) Proceeding – Public Service Commission Case 98-M-0667" filed on June 30, 1999.

ELEGAL; 2003. http://cic.vtt.fi/projects/elegal, acessado em 13 de março de 2003.

EXPIDE; 2003. http://www.biba.uni-bremen.de/projects/expide/, acessado em 18 de novembro de 2003.

FILOS, E.; BANAHAN, E. P.; 2000. Will The Organisation Disappear? The Challenges of the new Economy and Future Perspectives. In: SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brasil). *Proceedings*. Florianópolis, Brasil. p. 3-20.

FINXML; 2003. http://www.finxml.org/, acessado em 22 de outubro de 2003.

FIREBIRD; 2003. http://firebird.sourceforge.net, acessado em 18 de fevereiro de 2003.

FREMANTLE, P.; WEERAWARANA S.; KHALAF R.; 2002. Enterprise Services. In: Communications of the ACM, v. 45, n. 10 (Oct.), p. 77-82.

GENTLEWARE; 2003. www.gentleware.com, acessado em 7 de abril de 2003.

GIBON, P.; CLAVIER, J. F.; LOISON, S.; 1999. Suport for Eletronic Data Interchange. In: WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'99: 27-28 Oct. 1999: Porto, Portugal). *Proceedings*. Porto, Portugal. p. 187-208.

HAROLD, E. R.; 2001. *XML Bible*. Second Edition. John Wiley & Sons, ISBN 0-764-54760-7.

HEDIN, Mette; 2003.Comparing Java Data Binding Tools. [Sep. 3, 2003] http://www.xml.com/pub/a/2003/09/03/binding.html, acessado em 30 de setembro de 2003.

IDRIS, N.; 1999a. SAX Tutorial 1. http://developerlife.com/saxtutorial1/default.htm, acessado em 25 de julho de 2003.

IDRIS, N.; 1999b. Introduction to DOM. http://developerlife.com/domintro/default.htm, acessado em 25 de julho de 2003.

IDRIS, N.; 1999c. Benefits of using XML. http://developerlife.com/xmlbenefits/default.htm, acessado em 25 de julho de 2003.

INTERBASE; 2003. www.borland.com/interbase/, acessado em 18 de fevereiro de 2003.

JAXB; 2003. http://java.sun.com/xml/jaxb/, acessado em 12 de agosto de 2003.

KANET, J. J.; FAISST, W.; MERTENS, P.; 1999. Application of information technology to a virtual enterprise broker: The case of Bill Epstein. In: International Journal on Production Economics, n. 62, p. 23-32.

KONTIO, J.; 2003. Enterprise Data Model: a cornerstone for intra-application communication facilitating electronic business. In: The 3rd IFIP Conference on e-Commerce, e-Business, and e-Government (21-24 Sep. 2003: Guarujá, Brasil). *Proceedings*. Guarujá, Brasil. p. 177-184.

LEAR, A.C.; 1999. XML Seen as Integral to Application Integration. In: IT Professional, (Sep./Oct.), p. 12-16.

MCLAUGHLIN, B.; 2002. *Java and XML Data Binding*. 1st edition. O'Reilly & Associates. ISBN 0-596-00278-5.

MICROSOFT; 2003. http://msdn.microsoft.com, acessado em 13 de agosto de 2003.

MORRISON, M.; 2000. *XML Unleashed*. Sams Publishing, ISBN 0-672-31514-9.

MYFASHION; 2003. www.myfashion.org, acessado em 18 de novembro de 2003.

OASIS; 2003a. www.oasis-open.org, acessado em 22 de outubro de 2003.

OASIS; 2003b. XML: Proposed Applications and Industry Initiatives. http://www.oasis-open.org/cover/xml.html#applications, acessado em 22 de outubro de 2003.

OASIS; 2003c. http://www.oasisopen.org/cover/wap-wml.html, acessado em 22 de outubro de 2003.

ODBC2XML; 2003. http://www.intsysr.com/odbc2xml.htm, acessado em 12 de agosto de 2003.

OIL; 2004. www.ontoknowledge.org/oil/, acessado em 9 de maio de 2004.

OMG; 2002. XML Metadata Interchange (XMI) Specification, Version 1.2.

ORACLE; 2003a. http://otn.oracle.com/tech/xml/index.html, acessado em 13 de agosto de 2003.

ORACLE; 2003b. http://otn.oracle.com/tech/xml/xmldb/index.html, acessado em 13 de agosto de 2003.

OSÓRIO, A. L.; BARATA, M. M.; GIBON, P.; 1999. Communication Infrastructure Requirements in a VE. In: WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'99: 27-28 Oct. 1999: Porto, Portugal). *Proceedings*. Porto, Portugal. p. 65-76.

OSÓRIO, A. L.; GIBON, P.; BARATA, M . M.; 2000. Electronic Commerce With XML/EDI in Virtual Enterprises. In: SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brasil). *Proceedings*. Florianópolis, Brasil. p. 311-324.

OUZOUNIS, Evangelos K.; 2001. An Agent-Based Platform for the Management of Dynamic Virtual Enterprises. Berlin. Tese (Doutorado em Engenharia) – Departamento de Eletrônica e Informática – Universidade de Berlin.

PARK, K. H.; FAVREL, J.; 1999. Virtual Enterprise – Information System and Networking Solution. Computers & Industrial Engineering, n. 37, p. 441-444.

PDML; 2003. http://pdit.com/pdml/, acessado em 17 de junho de 2003.

PRESSMAN, R. S.; 2002. *Engenharia de Software*. 5ª Edição. Rio de Janeiro: McGraw-Hill.

PRODAEC; 2003. *European Network for Product and Project Data Exchange, e-Work and e-Business in Architecture, Engineering and Construction*. http://cic.vtt.fi/projects/prodaec/, acessado em 18 de novembro de 2003.

PRODNET; 2003. *Production Planning and Management in an Extended Enterprise*. http://www.uninova.pt/~prodnet/, acessado em 18 de novembro de 2003.

RABELO, R. J.; PEREIRA-KLEN, A.; SPINOSA, L. M.; *et al.*; 1999. Agile Supply-Chain Coordination in the Virtual Enterprise Environment. In: FORTH BRAZILIAN SIMPOSIUM OD INTELLIGENT AUTOMATION (4th SBAI: São Paulo, Brasil). *Proceedings*.

RABELO, R. J.; CAMARINHA-MATOS, L. M.; VALLEJOS, R. V.; 2000. Agent-based Brokerage for Virtual Enterprise Creation in the Moulds Industry. In: SECOND IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ORGANIZATIONS: MANAGING COOPERATION IN VIRTUAL ORGANIZATIONS AND ELECTRONIC BUSINESS TOWARDS SMART ORGANIZATIONS. (4-6.: Dec. 2000: Florianópolis, Brasil). *Proceedings*. Florianópolis, Brasil. p. 281-290.

RABELO, R. J.; PEREIRA-KLEN, A.; KLEN, E. R.; 2002. A Multi-Agent System for Smart Coordination of Dynamic Supply Chains. In: THIRD IFIP WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'02). (1-3.: May. 2002: Sesimbra, Portugal). *Proceedings*. Sesimbra, Portugal. p. 379-386.

RABELO, R. J.; PEREIRA-KLEN, A.; KLEN, E. R.; 2003. Effective Management of Dynamic and Multiple Supply Chains. In: International Journal of Networking and V.O., Inglaterra, v2, n2, ISSN 14709503.

RABELO, R. J.; 2003. Interoperating Standards in Multiagent Manufacturing Scheduling Systems. International Journal Of Computer Applications In Technology, Inglaterra, v. 18, n. 1/4, p. 146-159, 2003. ISSN/ISBN: 09528091.

RABELO, R. J.; BALDO, F.; TRAMONTIN JR, R. J.; *et al.*; 2004. Smart Configuration of Dynamic Virtual Enterprises. In: a ser apresentado na 5th IFIP Working Conference on VIRTUAL ENTERPRISES (PROVE 2004) que será realizada de 23 a 26 de agosto de 2004 em Toulouse, França.

RATIONAL; 2003. www.rational.com, acessado em 7 de abril de 2003.

ROCHA A. P.; OLIVEIRA, E.; 1999. An Electronic Market Architecture for the Formation of Virtual Enterprises. In: WORKING CONFERENCE ON INFRASTRUCTURES FOR VIRTUAL ENTERPRISES (PRO-VE'99: 27-28 Oct. 1999: Porto, Portugal). *Proceedings*. Porto, Portugal. p. 421-432.

ROSETTANET; 2003. www.rosettanet.org, acessado em 22 de outubro de 2003.

SCHERER, R. J.; 1997. Legal Framework for a Virtual Enterprise in Building Construction. In: 4th International Conference on Concurrent Enterprising (ICE '97: 8-10 Oct 1997: Nottingham, Inglaterra) *Proceedings*. Nottingham, Inglaterra. P. 373-383.

SCHMIDT, R.; 2003. *Busca e Seleção de Parceiros para Empresas Virtuais: Uma Abordagem Baseada em Agentes Móveis*. Florianópolis. Dissertação (Mestrado em Engenharia Elétrica) – Centro Tecnológico, Universidade Federal de Santa Catarina.

SHANMUGASUNDARAN, J.; TUFTE, K.; HE G.; *et al.*; 1999. Relational Databases for Querying XML Documents: Limitations and Opportunities. In:, 25th International Conference on Very Large Databases (7-10 Sep. 1999: Edinburgh, Scotland). *Proceedings*. Edinburgh, Scotland.

SLOC; 2004. http://sourceforge.net/projects/slocc/, acessado em 25 de junho de 2004.

SOUSA, L. C. D. M.; SOUZA, F. F.; 2003. A Generic Middleware to Migrate Data between Relational Database and XML Documents. In: The 3rd IFIP Conference on e-Commerce, e-Business, and e-Government (21-24 Sep. 2003: Guarujá, São Paulo, Brasil). *Proceedings*. Guarujá, Brasil. P. 185-192.

SPINOSA, L. M.; RABELO, R. J.; PEREIRA-KLEN, A.; 1998. High-Level Coordination of Business Processes in a Virtual Enterprise. In: THE TENTH INTERNATIONAL IFIP TC5 WG-5.2 WG-5.3 CONFERENCE (PROLAMAT'98). *Proceedings*. P. 55-67.

STEP; 2003. http://www.npd-solutions.com/step.html, acessado em 17 de junho de 2003.

SUN, 2003a. Java™ 2, Standard Edition (J2SE™) Specification Version: 1.4.2: Class Writer. http://java.sun.com/j2se/1.4.2/docs/api/java/io/Writer.html

SUN, 2003b. Java™ 2, Standard Edition (J2SE™) Specification Version: 1.4.2: Class InputSource. http://java.sun.com/j2se/1.4.2/docs/api/org/xml/sax/InputSource.html

SUN, 2003c. Java™ 2, Standard Edition (J2SE™) Specification Version: 1.4.2: Package "java.sql". http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html

SUN, 2003d. Java™ 2, Standard Edition (J2SE™) Specification Version: 1.4.2: Interface DatabaseMetaData. http://java.sun.com/j2se/1.4.2/docs/api/java/sql/DatabaseMetaData.html

SXQL; 2003. http://www.hatop.de/, acessado em 11 de agosto de 2003.

TAMINO; 2003. http://www2.softwareag.com/corporate/products/tamino/default.asp, acessado em 13 de agosto de 2003.

UN/EDIFACT; 2003. www.unece.org/trade/untdid/welcome.htm, acessado em 20 de outubro de 2003.

W3C; 1999a. Namespaces in XML. W3C Recommendation 14 January 1999. http://www.w3.org/TR/REC-xml-names/

W3C; 1999b. XML Path Language (XPath) Version 1.0. W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath

W3C; 2000. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6 October 2000. http://www.w3.org/TR/REC-xml

W3C; 2001a. XML Schema Part 0: Primer. W3C Recommendation, 2 May 2001. http://www.w3.org/TR/xmlschema-0/

W3C; 2001b. XML Linking Language (XLink) Version 1.0. W3C Recommendation 27 June 2001. http://www.w3.org/TR/xlink/

W3C; 2001c. Extensible Stylesheet Language (XSL) Version 1.0. W3C Recommendation 15 October 2001. http://www.w3.org/TR/xsl/

W3C; 2002a. XML Pointer Language (XPointer). W3C Working Draft 16 August 2002. http://www.w3.org/TR/xptr/

W3C; 2003a. Document Object Model (DOM) Technical Reports. http://www.w3.org/DOM/DOMTR

W3C; 2003b. XML Query. http://www.w3.org/XML/Query

W3C; 2004. Semantic Web. http://www.w3.org/2001/sw/

WALSH, N.; 1998. A Technical Introduction to XML. ArborText Inc. http://www.xml.com/pub/a/98/10/guide0.html, acessado em 20 de março de 2003.

WALTON, J.; WHICKER, L; 1996.Virtual Enterprise: Myth & Reality. J. Control.

WEBER, D. R. R.; DUTTON, A.; 2000. Understanding ebXML, UDDI and XML/edi. XML Global Technologies, Inc.

WIKIPEDIA; 2004a. http://en.wikipedia.org/wiki/Enterprise_Application_Integration

WIKIPEDIA; 2004b. http://en.wikipedia.org/wiki/Ontology_%28computer_science%29

WIKIPEDIA; 2004c. http://en.wikipedia.org/wiki/SLOC

VLIST, E. V. D; 2002. *XML Schema*. First Edition. O'Reilly & Associates. ISBN 0-596-00252-1.

VOMAP; 2003. http://www.vomap.org/, acessado em 18 de novembro de 2003.

VOSTER; 2003. Projects in VOSTER. http://cic.vtt.fi/projects/voster/projects.html, acessado em 18 de novembro de 2003.

XGEN; 2003. http://www.commerceone.com/developers/docsoapxdk/xgen.html, acessado em 11 de agosto de 2003.

XML-DBMS; 2003. Middleware for Transferring Data between XML Documents and Relational Databases. http://www.rpbourret.com/xmldbms/index.htm, acessado em 12 de agosto de 2003.

XMLEDI. 2003. XML/EDI Group. www.xmledi-group.org, acessado em 22 de outubro de 2003.

XMLGLOBAL; 2003. XML and EDI Tutorial. www.xmlglobal.com/consult/xmledi/index.html, acessado em 11 de julho de 2003

ZAPTHINK; 2001. The Pros and Cons of XML, Technical Report. www.zapthink.com/reports/proscons-view.html, acessado em 13 de janeiro de 2003.