



**Pós-Graduação em Ciência da Computação**

**SIDATA – Sistema de Integração de Dados  
Apoiado no TAmينو**

***Marcelo da Silveira Siedler***

**Dissertação de Mestrado**



Universidade Federal de Pernambuco  
posgraduacao@cin.ufpe.br  
[www.cin.ufpe.br/~posgraduacao](http://www.cin.ufpe.br/~posgraduacao)

RECIFE, SETEMBRO/2004



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

Marcelo da Silveira Siedler

**SIDATA – Sistema de Integração de Dados Apoiado no TAmino**

*ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM  
CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA  
UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO  
PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA  
COMPUTAÇÃO.*

ORIENTADOR: PROF. DR FERNANDO DA FONSECA DE SOUZA

RECIFE, SETEMBRO/2004

**Siedler, Marcelo da Silveira**

**SIDATA – Sistema de Integração de Dados  
Apoiado no TAmينو / Marcelo da Silveira Siedler. –  
Recife : O Autor, 2004.**

**140 folhas : il., fig., tab., quadros.**

**Dissertação (mestrado) – Universidade Federal de  
Pernambuco. Cln. Ciência da Computação, 2004.**

**Inclui bibliografia e anexos.**

**1. Ciência da computação – Base de dados. 2.  
Integração de dados – SIDATA. 3. Técnicas Web –  
Programação XML (eXtended Markup Language) –  
Web semântica - Ontologias. I. Título.**

**004.652  
005.74**

**CDU (2.ed.)  
CDD (22.ed.)**

**UFPE  
BC2006-110**

# Resumo

---

O presente trabalho apresenta o SIDATA – Sistema de Integração de Dados Apoiado no TAmينو. O projeto consiste em uma proposta de arquitetura e desenvolvimento de um protótipo de um sistema de integração de dados utilizando técnicas relacionadas à *web*. As principais características do sistema consistem na utilização de XML tanto no armazenamento quanto na representação dos dados e, na utilização de ontologias, criadas a partir das premissas relacionadas à Web Semântica. Documentos XML são armazenados em sua forma nativa utilizando o SGBD XML Nativo Tamino, enquanto que as ontologias visam eliminar heterogeneidades sintáticas e/ou semânticas existentes entre as fontes de dados que estão sendo integradas. Por fim, cria-se uma opção aos sistemas de integração convencionais, buscando um menor esforço de administração e uma maior capacidade de integração de dados em sistemas legados.

**Palavras-chave:** Integração de Dados, XML, Ontologia.

# Abstract

---

This work introduces a System for Integrating Data based on Tamino (SIDATA). It highlights the developed architecture and a system prototype for data integration using concepts related to semantic web. The main characteristics of the system are the usage of XML for data representation and storage as well as ontology generated from semantic web premises. XML documents are stored in their native form through Tamino, a XML Native DBMS whereas the ontology is aimed at eliminating syntactic and semantic conflicts between the data sources to be integrated. Finally, it is expected to produce an alternative to conventional integration systems seeking for less effort for administrative purposes and better data integration from legacy systems

**Key Words:** Data Integration, XML, Ontology.

# Sumário

---

<b>1. INTRODUÇÃO.....</b>	<b>13</b>
1.1. TRABALHO DESENVOLVIDO.....	14
1.2. ESTRUTURA DA DISSERTAÇÃO.....	15
<b>2. INTEGRAÇÃO DE DADOS .....</b>	<b>16</b>
2.1. ABORDAGENS PARA INTEGRAÇÃO DE DADOS .....	17
2.2. ARQUITETURAS PARA INTEGRAÇÃO DE DADOS .....	18
2.2.1. <i>Mediadores</i> .....	20
2.2.2. <i>Data Warehouse</i> .....	21
2.3. INTEGRAÇÃO DE DADOS NA WEB .....	24
2.4. INTEGRAÇÃO E DADOS SEMI-ESTRUTURADOS.....	25
2.4.1. <i>XML</i> .....	28
2.5. BANCO DE DADOS E DOCUMENTOS XML .....	39
2.5.1. <i>Sistema de Gerenciamento de Banco de Dados XML</i> .....	40
2.6. WEB SEMÂNTICA .....	41
2.6.1. <i>Ontologia</i> .....	42
2.7. TRABALHOS RELACIONADOS .....	46
2.8. CONSIDERAÇÕES FINAIS .....	48
<b>3. TAMINO XML SERVER .....</b>	<b>49</b>
3.1. ARQUITETURA .....	49
3.1.1. <i>X-Machine</i> .....	50
3.1.2. <i>X-Node</i> .....	52
3.1.3. <i>Data Map</i> .....	52
3.1.4. <i>Server Extensions</i> .....	53
3.2. DEFININDO E ARMAZENANDO DADOS XML NO TAMINO.....	55
3.2.1. <i>Descrevendo os dados</i> .....	58
3.2.2. <i>Definindo o Esquema Tamino</i> .....	61
3.2.3. <i>Mapeando para Bases Relacionais</i> .....	64
3.2.4. <i>Recuperando Informações no Tamino XML Server</i> .....	66
3.3. FERRAMENTAS .....	68
3.3.1. <i>Tamino Manager</i> .....	68
3.3.2. <i>Tamino Schema Editor</i> .....	73
3.3.3. <i>Tamino Interactive Interface</i> .....	74
3.3.4. <i>Tamino X-Plorer</i> .....	76
3.3.5. <i>X-Tension Builder</i> .....	76
3.4. CONSIDERAÇÕES FINAIS .....	78
<b>4. DESENVOLVIMENTO DO SIDATA .....</b>	<b>79</b>
4.1. ARQUITETURA DO SISTEMA .....	79
4.2. IMPLEMENTAÇÃO .....	83

4.2.1.	<i>Busca dos dados nas bases</i> .....	83
4.2.2.	<i>O Papel do Tamino</i> .....	87
4.3.	ESTRUTURA DO SISTEMA.....	92
4.4.	ANÁLISE DO SIDATA.....	98
4.5.	CONSIDERAÇÕES FINAIS .....	99
<b>5.</b>	<b>UM EXEMPLO DE USO DO SIDATA.....</b>	<b>100</b>
5.1.	INTEGRANDO OS DADOS COM O SIDATA .....	105
5.2.	CONSIDERAÇÕES FINAIS .....	115
<b>6.</b>	<b>CONCLUSÃO .....</b>	<b>116</b>
<b>7.</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>118</b>
<b>8.</b>	<b>ANEXOS.....</b>	<b>125</b>
8.1.	ANEXO A – ONTOLOGIA USADA NO EXEMPLO DE USO .....	125
8.2.	ANEXO B – XML SCHEMAS REPRESENTANDO AS BASES DE DADOS .....	132
8.3.	ANEXO C – ESQUEMA GLOBAL GERADO NO EXEMPLO DE USO .....	140
8.4.	ANEXO D – XML RESULTANTE DA CONSULTA ÀS BASES INTEGRADAS .....	141

# Lista de Figuras

Figura 2.1 - Arquitetura Federada.....	19
Figura 2.2 - Arquitetura Multi-camada.....	20
Figura 2.3 - Arquitetura de Mediadores.....	21
Figura 2.4 - Arquitetura de Data Warehouse .....	22
Figura 3.1 - Arquitetura da X-Machine.....	51
Figura 3.2 - Comunicação entre o Tamino e as bases externas através do X-Node .	52
Figura 3.3 - Informações que são gerenciadas no Data Map.....	53
Figura 3.4 - Funcionamento da Server Extension no Tamino .....	53
Figura 3.5 – Esquema Tamino x SGBD Relacional .....	56
Figura 3.6 – Gerenciamento de servidores pelo <i>Tamino Manager</i> .....	69
Figura 3.7 - Interface de gerenciamento das bases do Tamino Manager .....	70
Figura 3.8 - Disposição dos dados da base no Tamino Manager.....	72
Figura 3.9 – Ambiente de desenvolvimento do Tamino Schema Editor.....	74
Figura 3.10 - Tela de consulta do Tamino Interactive Interface.....	75
Figura 3.11 - Ambiente de desenvolvimento do X-Tension Builder.....	77
Figura 4.1 - Modelo inicial da Arquitetura do SIDATA .....	80
Figura 4.2 – Arquitetura refinada.....	81
Figura 4.3 - Estrutura do Middleware.....	82
Figura 4.4 - Tela inicial do SIDATA .....	85
Figura 4.5 - Esquema gerado pelo SIDATA em sua versão inicial. ....	86
Figura 4.6 - Server Extensions Integra2004 .....	97
Figura 5.1 - Ontologia sobre o domínio futebol .....	104
Figura 5.2 - Passo 1 do sistema de integração .....	106
Figura 5.3 - Passo 2 do sistema de integração .....	107
Figura 5.4 - Passo 3 do sistema de integração .....	108
Figura 5.5 - Passo 4 do sistema de integração .....	109
Figura 5.6 – Representação das bases <i>serieA</i> e <i>serieB</i> como XML Schemas .....	110
Figura 5.7 – Representação do XML <i>Schema</i> global.....	113
Figura 5.8 - Representação da SXT Integra2004 no Tamino Manager.....	114



# Lista de Quadros

Quadro 2.1 – Exemplo de código XML.....	29
Quadro 2.2 – Exemplo de DTD.....	30
Quadro 2.3 – Exemplo de XML Schema .....	32
Quadro 2.4 – Exemplo de documento que utiliza namespaces .....	34
Quadro 2.5 – Lista de comandos XQL.....	37
Quadro 2.6 – Lista de comandos XQuery.....	38
Quadro 2.7 – Exemplo de documento RDF .....	44
Quadro 2.8 – Tabelas relacionais Jogador e Atleta .....	45
Quadro 3.1 – Estrutura de registro de paciente.....	58
Quadro 3.2 – Descrição da tabela de equipe médica .....	59
Quadro 3.3 – DTD representando as informações de médico e paciente .....	60
Quadro 3.4 – XML Schema representando as informações de Pacientes .....	63
Quadro 3.5 – Lista de atributos de mapeamento XML-Base Relacional .....	64
Quadro 3.6 – Representação e mapeamento das informações dos médicos .....	65
Quadro 3.7 – Documento XML com os dados do médico que atende o paciente. ....	68
Quadro 4.1 – Bases relacionadas ao domínio futebol .....	86
Quadro 4.2 - Representação do mapeamento inicial no XML Schema .....	88
Quadro 4.3 - Representação no XML Schema dos elementos que mapeiam campos de uma tabela.....	89
Quadro 4.4 – Documento XML resultante da consulta X-Query.....	90
Quadro 4.5 – Lista de parâmetros da função de mapeamento .....	95
Quadro 5.1 – Estrutura da base de dados <i>serieA</i> .....	101
Quadro 5.2 – Estrutura da base de dados <i>serieB</i> .....	102
Quadro 5.3 – Trecho de código do XML Schema <i>Base_serieA.tsd</i> .....	111
Quadro 5.4 – Trecho de código do XML Schema <i>Base_serieB.tsd</i> .....	112
Quadro 5.5 – Registros retornados da consulta ao esquema global.....	115

# Lista de Tabelas

---

Tabela 3.1 – Representação da tabela com os médicos cadastrados.....	59
--	----

# Dicionário de Siglas

---

ADM - Araneus Data Model  
API – Application Programming Interface  
BD – Banco de Dados  
DAML – DARPA Agent Markup Language  
DOM - Document Object Model  
DTD – Document Type Definition  
DW – Data Warehouse.  
ERP - Enterprise Resource Planning.  
HTML – HyperText Markup Language.  
IA – Inteligência Artificial  
JDBC – Java Database Connectivity  
NXD - Native XML Databases  
ODBC - Open Database Connectivity  
ODMG - Object Database Management Group  
OEM - Object Exchange Model  
OEM-QL – Object Exchange Model Query Language  
OIL - Ontology Inference Layer  
OQL - Object Query Language  
OWL - Ontology Web Language  
RDF - Resource Description Framework  
RDFS - Resource Description Framework Schema  
SAX - Simple API for XML  
SGBD – Sistema Gerenciador de Banco de Dados.  
SGML - Standard Generalized Markup Language  
SQL - Structured Query Language  
SXT – Server Extension  
XDMS - XML Data Management Server  
TSD - Tamino Schema Definition  
URL – Universal Resource Locator

VML - Vector Markup Language  
W3C – World Wide Web Consortium  
WHOWEDA- WareHouse Of WEb Data  
WSDL - Web Services Description Language  
WWW – Word Wide Web  
XML – eXtensible Markup Language.  
XQL – XML Query Language  
XQuery – XML Query  
XSD - XML Schema Definition  
XSL - Extensible Stylesheet Language Family  
XSLT - Extensible Stylesheet Language Transformations

# 1. INTRODUÇÃO

Empresas/instituições mantêm diversas bases de dados situadas em diferentes departamentos [Dw-institute 2002]. Apesar dessas bases conterem informações relativas a um mesmo domínio, elas podem possuir seus dados armazenados de maneiras totalmente distintas, dificultando o processo de análise das informações contidas na corporação de uma forma uniforme e coerente.

Gerenciar a heterogeneidade semântica e estrutural das informações, a fim de prover acesso integrado aos dados, é um dos principais problemas a serem solucionados por sistemas de integração de dados.

Atualmente, parte dos esforços de pesquisas e estudos desenvolvidos pela comunidade de Banco de Dados (BD) é dedicada aos problemas relacionados à integração de dados [Dw 2002]. Tal tarefa consiste na integração de diversas bases de dados, estruturalmente heterogêneas, numa base central que retrata o conteúdo existente em todas as demais de forma homogênea e integrada. Essa integração permite o processo de análise e compreensão das informações contidas nessas fontes de dados pela parte gerencial da instituição.

Dentro desse contexto, este trabalho apresenta uma proposta de arquitetura de sistema de integração, e o desenvolvimento do SIDATA (Sistema de Integração de Dados Apoiado no TAmينو), um protótipo implementado segundo a arquitetura desenvolvida. Um exemplo de uso do SIDATA também é apresentado.

O sistema de integração proposto utiliza diversas tecnologias relacionadas a *web* no auxílio ao processo de integração dos dados, dentre as quais pode-se destacar a linguagem de marcação XML (*eXtensible Markup Language*) [Mcgrath 1999], linguagem proposta pelo W3C (*World Wide Web Consortium*) como a forma padrão de representação e troca de dados na *web* [Bray 2000], sendo utilizada no presente trabalho como a estrutura de representação e consulta dos dados armazenados.

Outro conceito abordado neste trabalho que atualmente vem sendo difundido é a Web Semântica. Considerada por diversos estudos [Berners-Lee et al. 2001; Webont 2003; Chandrasekaran et al. 1999] como a evolução da *web* atual, a Web Semântica tem como objetivo fornecer estruturas e dar significado semântico ao conteúdo das páginas

*web*, visando tornar mais eficaz o processo de busca a informações, o que possibilitará acesso mais qualificado ao conteúdo que se deseja pesquisar.

Dentre os diversos conceitos que envolvem o desenvolvimento da Web Semântica o que será adotado no presente trabalho é o de ontologia. Esta, sob o paradigma de banco de dados, pode ser vista como uma especificação parcial de um domínio ou meta-domínio, descrevendo entidades, relações entre elas e regras de integridade dentro de um determinado projeto [Berners-Lee et al. 2001].

No presente trabalho, ontologia é utilizada para descrever metadados utilizados para corrigir problemas de integridade semântica dos dados provenientes das bases integradas.

### **1.1. Trabalho Desenvolvido**

A partir de estudos de diversas propostas de desenvolvimento de sistemas de integração de dados, desenvolveu-se um trabalho que visa ser uma alternativa aos sistemas de integração existentes.

Este trabalho tem por objetivo utilizar XML como forma de otimizar o processo de integração, estruturação e consulta das informações das diferentes bases de dados, propondo uma arquitetura de desenvolvimento, no qual os dados são representados e integrados em um NXD (*Native XML Database*). Portanto é utilizado o Tamino XML Server, o qual permite que esses dados possam ser extraídos por consultas realizadas no servidor.

O trabalho pretende analisar a viabilidade de utilizar NXD, através do Tamino XML Server, e quais as principais vantagens e limitações que o mesmo apresenta no suporte ao desenvolvimento de sistemas de integração, particularmente o proposto neste trabalho.

Um outro ponto importante a ser analisado é a forma de tratamento as inconsistências geradas no processo de integração entre os diversos bancos de dados heterogêneos e o repositório de dados (Tamino XML Server), objetivando corrigir as discrepâncias existentes entre as bases no que tange desde a forma adotada para representar os dados (por exemplo, forma de armazenamento do sexo de uma pessoa), até problemas de conversão dos dados da base original para o formato XML no Tamino.

Finalmente, o projeto visa a implementação do SIDATA, um protótipo construído segundo a arquitetura de desenvolvimento proposta.

## **1.2. Estrutura da Dissertação**

Além deste capítulo esta dissertação está organizada como segue.

O Capítulo 2 apresenta um estudo sobre os conceitos relacionados ao desenvolvimento de sistemas de integração de dados, abordando as diversas arquiteturas existentes, as particularidades da integração de dados na *web*, dados semi-estruturados, XML e padrões relacionados, como XML é utilizada em bancos de dados, definições relacionadas a Web Semântica e trabalhos relacionados.

No Capítulo 3 é abordado o Tamino XML Server e toda a estrutura de desenvolvimento oferecida pelo servidor. São também abordados conceitos e padrões relacionados ao armazenamento e à recuperação de dados no formato XML nativo, além de tópicos específicos da ferramenta, como sua arquitetura, forma de representação e recuperação de dados.

O Capítulo 4 apresenta uma proposta de arquitetura e o desenvolvimento do protótipo SIDATA, baseado na arquitetura definida, destacando as etapas de implementação e o papel do Tamino no referido desenvolvimento.

No Capítulo 5 é apresentado um exemplo de uso a partir do protótipo desenvolvido, demonstrando a utilização do SIDATA como sistema de integração aplicado a um determinado domínio de conhecimento.

O Capítulo 6 aborda a conclusão do trabalho, com a descrição dos objetivos alcançados, as dificuldades encontradas no processo de desenvolvimento do mesmo, e as contribuições atingidas. Por fim, são apresentadas as sugestões de trabalhos futuros que podem expandir o trabalho realizado.

Finalmente, após as referências bibliográficas citadas no trabalho, são apresentados os anexos. Estes anexos apresentam a representação completa das bases de dados, bem como os XML *Schemas* gerados e a ontologia desenvolvida.

## 2. INTEGRAÇÃO DE DADOS

Um dos principais desafios para os modernos sistemas de informação tem sido prover acesso integrado a informações armazenadas em múltiplas bases de dados distribuídas e heterogêneas. Os constantes avanços tecnológicos permitem que, cada vez mais, se tenha acesso a uma grande variedade de conteúdo que outrora não era possível, o que motivou a busca por sistemas de integração de dados que possam lidar com essa diversidade de informações.

Um sistema de integração de dados deve prover ao usuário uma *interface* uniforme de acesso às informações provenientes de diferentes bases de dados, permitindo que os mesmos possam realizar suas consultas de forma transparente. Assim, cabe ao sistema saber onde as consultas devem buscar os resultados, apresentando ao usuário as informações requeridas sem que os mesmos tenham que saber a localização física das fontes de informação [Dw 2002].

Em processos de integração mais convencionais, a estrutura dos dados integrados é composta, basicamente, de bases de dados que, quase sempre, são autônomas e dinâmicas, o que invariavelmente faz com que a atualizações no conteúdo das bases resultem em alterações nos esquemas das mesmas [Kalinichenko 1999]. Essas mudanças podem acarretar uma série de problemas ao sistema caso este não esteja apto a lidar com as alterações, ou seja, se ele não tiver a capacidade de ser estendido à medida que as bases de dados vão sendo alteradas no sistema.

Por serem bases de dados autônomas, supõe-se que elas tenham características diferentes entre si, com distintos modelos de dados e diversas variações na forma de definição dos conceitos relacionados ao domínio ao qual a base está associada. Essa heterogeneidade das bases deve ser sobreposta para que o sistema consiga apresentar ao usuário uma visão integrada das informações, corrigindo as diferenças semânticas e estruturais existentes entre as bases de dados.



## **2.1. Abordagens para Integração de Dados**

Para construir sistemas de integração de dados é importante que se defina qual abordagem será utilizada no desenvolvimento do mesmo. Neste sentido, existem duas abordagens principais de integração, a abordagem virtual e a abordagem materializada.

A abordagem virtual consiste na extração dos dados das fontes de dados somente quando as consultas são realizadas, ou seja, os dados não são armazenados fisicamente no sistema de integração. Na abordagem materializada, as informações são previamente recuperadas, integradas e armazenadas em um repositório de dados, de forma que as consultas são realizadas diretamente neste repositório, sem necessidade de acessar constantemente as fontes de informação.

Existem algumas vantagens e desvantagens que torna a abordagem utilizada uma decisão que pode ser fundamental para o sucesso ou não do sistema de integração. A principal vantagem apontada no sistema de abordagem virtual é que os dados estão constantemente atualizados, visto que estes sempre são recuperados das fontes de dados nas quais se encontram armazenados. Porém, pode-se ter problemas na consulta a dados se alguma fonte estiver inativa no momento da consulta, além de um tempo de resposta que pode ser maior, visto que as consultas são processadas independentemente nas diferentes fontes de dados, o que pode acarretar uma perda ou atraso na apresentação das informações requeridas. Sendo assim, a abordagem virtual é mais adequada para os casos em que as informações mudam rapidamente e para consultas que operam sobre uma grande quantidade de dados distribuídos em um grande número de fontes de dados [Holland 2000]. Porém, se as fontes de dados ficam constantemente indisponíveis ou o custo para fazer a tradução dos dados das fontes é muito alto, essa abordagem não é a mais adequada.

No caso da abordagem materializada, sua principal vantagem é o fato de que as informações integradas estão disponíveis imediatamente para consultas, visto que os dados estão dispostos em um repositório central, tornando desnecessário o acesso às fontes de dados locais. Entretanto, dependendo da aplicação, deve ser definida uma sistemática para manter a consistência entre os dados armazenados e os dados das fontes de dados de origem. Esta abordagem é mais adequada quando o usuário deseja ter um melhor desempenho no processamento de consulta, sem no entanto, requerer que os

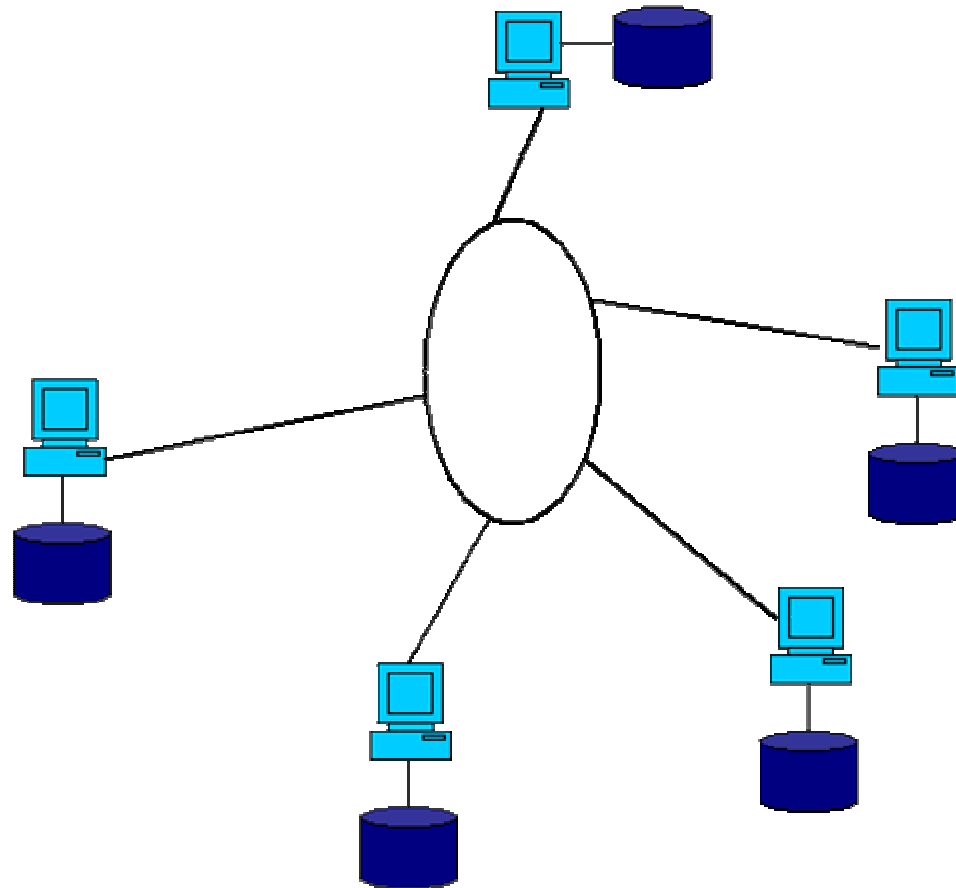
dados estejam no seu estado mais atualizado. Este tipo de abordagem é bastante útil quando temos sistemas nas quais os dados não devem ser atualizados nas fontes de dados, tais como informações históricas.

As duas abordagens apresentam diferenças bastante significativas, sendo cada uma apropriada para um determinado tipo de aplicação. Porém, muitas aplicações mais complexas, que tenham grande abrangência, podem exigir que ambas as abordagens sejam adotadas. Um estudo mais detalhado pode apontar que um determinado conjunto de dados deve ser previamente integrado, enquanto outras informações em atualização mais constante devem ser acessadas no momento da requisição da consulta.

## **2.2. Arquiteturas para Integração de Dados**

Existem diversas arquiteturas propostas para definição de sistemas de integração de dados. As soluções mais tradicionais para o problema são baseadas na construção de um esquema global a partir da integração dos esquemas das fontes de dados locais, de forma que as diversas fontes de dados distribuídas e heterogêneas possam ser acessadas de maneira uniforme e transparente. Porém, como o esquema global é estático e oferece uma visão única dos dados integrados, esta abordagem pode não ser muito adequada quando as fontes de dados a serem integradas são dinâmicas ou quando os consumidores da informação integrada têm diferentes requisitos.

Uma abordagem clássica para integração de dados distribuídos e heterogêneos é a abordagem federada [Kimball 1996]. Um sistema de banco de dados federado é uma coleção de sistemas de bancos de dados cooperantes e autônomos que participam da federação para permitir um compartilhamento parcial e controlado de seus dados. A característica chave de uma federação é a cooperação entre sistemas independentes. Nesta abordagem, ao invés de um único esquema global integrado, são oferecidos múltiplos esquemas, integrados de acordo com os requisitos das aplicações. Entretanto, estes esquemas integrados também são estáticos e definidos a priori, dessa forma, esta abordagem também não é adequada para os casos em que as fontes de dados são dinâmicas e precisam ser adicionadas ou removidas, ou quando os requisitos das aplicações precisam ser atualizados. A Figura 2.1 apresenta a arquitetura da abordagem federada.



**Figura 2.1 - Arquitetura Federada**

Considerando o ambiente *web* de desenvolvimento, pode-se dizer que existe um cenário que foge dos padrões tradicionais de integração de dados. Não existe um servidor central, e sim vários servidores que podem ser qualquer aplicação, mas que produz como resultado final dados que podem ser utilizados em um sistema de integração. Do lado cliente tem-se *interfaces* de usuários ou aplicações que consultam essas informações. Entre o usuário e as fontes de dados (servidores) existe *middleware*, que é utilizado para fazer com que os dados consultados nas fontes de informação cheguem ao usuário de forma precisa. Assim, pode-se afirmar que o processamento de dados no âmbito da *web* é desenvolvido segundo uma abordagem multi-camada, na qual a camada de mais baixo nível é constituída das fontes de dados, a camada intermediária do *middleware* e a camada de mais alto nível é a *interface* com o usuário [Abiteboul et al. 2000b]. A Figura 2.2 [Abiteboul et al. 2000b] apresenta o esquema da arquitetura multi-camada.

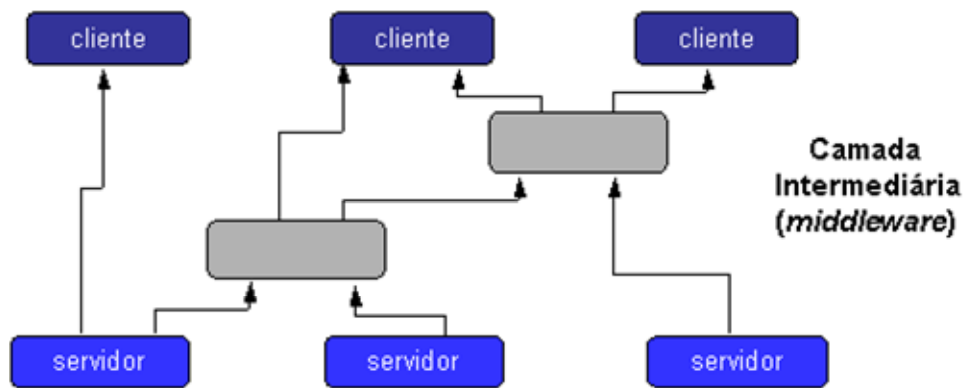


Figura 2.2 - Arquitetura Multi-camada

O desenvolvimento de *middleware*, portanto, vem sendo o grande alvo dos pesquisadores que trabalham com sistemas de integração. Neste sentido, duas arquiteturas se destacam: a arquitetura de mediadores, que implementa a abordagem virtual de integração de dados, e a arquitetura de *data warehouse*, que está associada à abordagem materializada de integração. A seguir, estas arquiteturas serão abordadas em mais profundidade.

### 2.2.1. Mediadores

Os mediadores são módulos de *software* que exploram o conhecimento representado em um conjunto ou subconjunto de dados para gerar informações para aplicações residentes em uma camada superior [Wiederhold 1983], oferecendo uma visão integrada sobre os dados das diversas fontes pertencentes ao sistema, disponibilizando um esquema para esta visão.

Para acessar os dados contidos nas diferentes bases de dados, o usuário deve submeter as consultas desejadas ao mediador através do esquema apresentado pelo mesmo, cabendo ao mediador a tarefa de transformar essa consulta global em subconsultas de forma que possibilite o envio da requisição a cada fonte de dados. Cada subconsulta deverá ser transformada para a linguagem de consulta correspondente a uma fonte de dados do sistema. Como retorno, as informações das fontes são novamente traduzidas para o modelo comum, de forma que o mediador possa interpretar os dados da consulta e enviar o resultado ao usuário.

Assim, além do mediador e das fontes de dados, existe um outro componente na arquitetura de mediadores de suma importância no processo de integração, os tradutores

(*wrappers*), já que estes são os responsáveis pela conversão dos dados, tanto das fontes de dados para o modelo comum, como o processo inverso, ou seja, do modelo para as fontes correspondentes. A Figura 2.3 [Holland 2000] apresenta a arquitetura de mediadores.

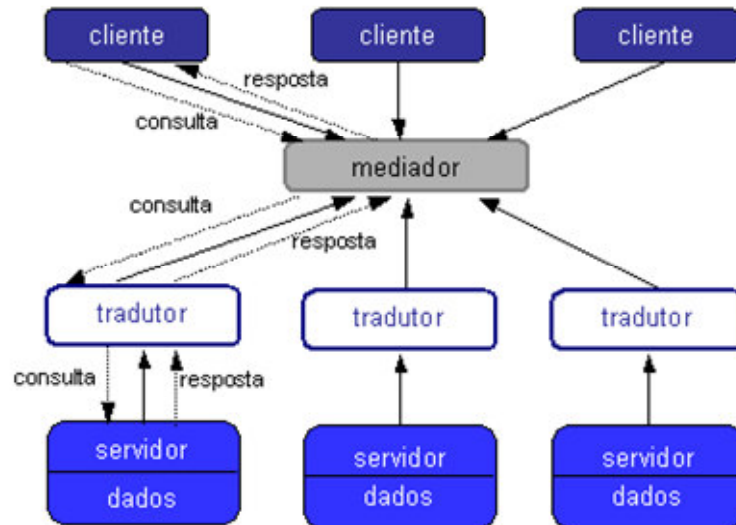


Figura 2.3 - Arquitetura de Mediadores

Originalmente, a arquitetura de mediadores foi proposta para integração de dados distribuídos em múltiplos bancos de dados. Posteriormente, pesquisou-se a possibilidade de se adaptar esta arquitetura a fim de possibilitar que ela possa ser utilizada em sistemas de integração de dados na *web* [Baru 1999; Holland 2000]. Como será apresentado posteriormente, este trabalho baseia-se na arquitetura de mediadores.

### 2.2.2. Data Warehouse

Outra arquitetura que tem sido largamente utilizada para integrar dados distribuídos em múltiplas fontes de dados é a arquitetura de *Data Warehouse* [Dw-institute 2002; Dw 2002]. O DW (*Data Warehouse*) surgiu a partir da valorização da informação nas organizações. Esta arquitetura foi criada para trabalhar em um ambiente em que existe uma grande quantidade de dados, sendo que estes precisam ser integrados para que as empresas e/ou instituições possam fazer uma análise mais detalhada desses dados. Diferentemente do ambiente de mediadores (também conhecido como *Virtual Data Warehouse*), os dados no *Data Warehouse* são lógica e fisicamente transformados, atualizados e mantidos pelo tempo que for conveniente pela organização. O grande

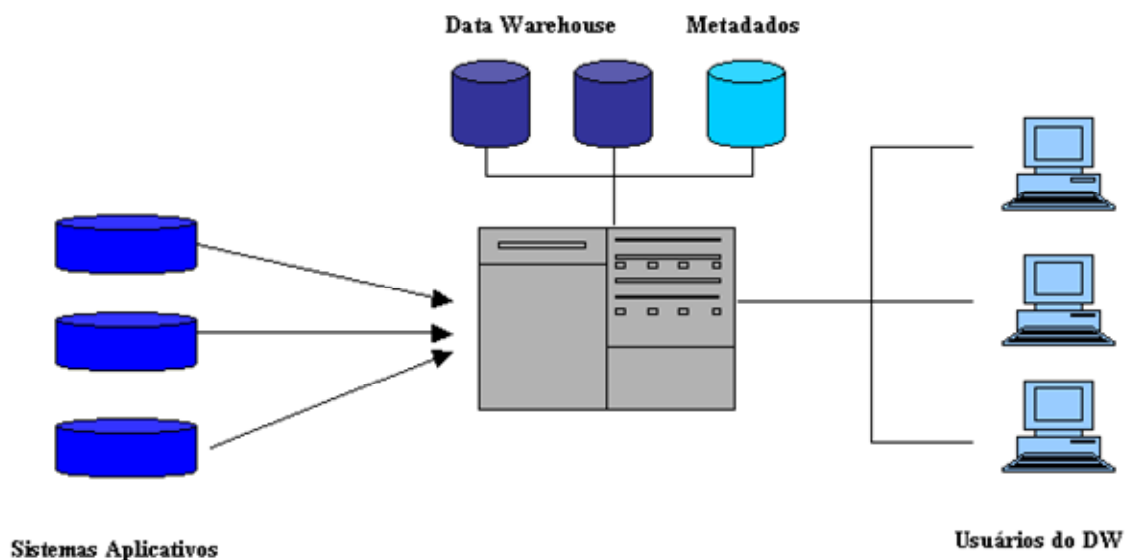
objetivo de um *Data Warehouse* é ser uma plataforma que ofereça dados integrados com qualidade, possibilitando que sejam feitas inferências no mesmo para que possa auxiliar no desenvolvimento de sistema de apoio à decisão.

Em termos tecnológicos, um *Data Warehouse* é uma combinação de várias tecnologias, tendo como primeiro objetivo a integração efetiva de bases de dados operacionais em um ambiente que habilita o uso estratégico dos dados. Estas tecnologias incluem sistemas gerenciadores de banco de dados relacional e multidimensional, arquitetura cliente/servidor, modelagem de metadados e repositórios, interfaces gráficas para usuário, etc.

A seguir são apresentados os componentes que compõem a arquitetura de um *Data Warehouse*:

- Sistemas, que alimentam o DW;
- Usuários do DW;
- O DW propriamente dito;
- Metadados, para que seja possível gerenciar e acessar os dados armazenados.

A Figura 2.4 [Dw 2002] apresenta os componentes do Data Warehouse dispostos na arquitetura do mesmo.



**Figura 2.4 - Arquitetura de Data Warehouse**

Desde o início da década de 90, o conceito e a operação de um *Data Warehouse* saíram do âmbito estritamente acadêmico para a área empresarial, concretizando a tendência natural da necessidade de integrar dados dos diversos sistemas existentes nas grandes corporações.

Antes do surgimento e popularização dos *Data Warehouses* e das ferramentas ERP (*Enterprise Resource Planning*) [ERP 2003a; ERP 2003b], que são desenvolvidas objetivando dar à empresa suporte à tomada de decisão, não existia concretamente a possibilidade de se criar sistemas verdadeiramente de integração de dados. Até então, sistemas trocavam dados na forma que atendesse às necessidades de cada um deles, sem que essa integração sequer se aproximasse do que se vê hoje nos ERP, cujas empresas desenvolvedoras têm, sistematicamente, dado a seus produtos características que os tornam facilmente fornecedores de dados aos *Data Warehouses* [ERP 2003b].

Cada aplicativo tinha sua própria visão do que era o cliente, uma operação ou um determinado produto, o que tornava a idéia de uma visão corporativa das informações dos diversos segmentos da empresa praticamente ficção, conseqüentemente não era possível obter dados históricos de forma organizada, e os dados sintéticos disponíveis eram restritos a apenas uma pequena parte da realidade da empresa.

ERP e *Data Warehouse* são capazes de suprir esta carência de informações, integrando dados, provendo dados históricos, permitindo que as informações possam ser recuperadas na forma mais adequada ao usuário do sistema [ERP 2003a].

Através do processo de integração de dados, o executivo tem uma visão corporativa dos dados de sua empresa, podendo migrar dados mantidos em sistemas anteriores e obter informações que podem auxiliá-lo na tomada de decisões futuras.

Porém, quando se discute o processo de integração dos dados, deve-se considerar que este não é um processo simples e tampouco barato. Construir esses sistemas exige muito planejamento, e estima-se que o custo gira em torno de 75% do investimento necessário à implementação do *warehouse* [Inmon 2001].

Uma vez que o *Warehouse* já esteja construído, o próximo passo é a sua exploração, no sentido de buscar e utilizar as informações nele contidas. Essa etapa do trabalho, chamada *Data Mining*, permite descobrir padrões importantes da empresa, tais

como relações de causa e efeito que vinham passando despercebidas e tendências a longo prazo.

No caso do presente escopo, não é implementado um *Data Warehouse*, e sim, um sistema de integração que fornece informações que podem vir a ser armazenadas em um *Data Warehouse*.

### **2.3. Integração de dados na Web**

Com a popularização da *web*, é crescente a quantidade de informações distribuídas pela rede, as quais podem estar dispostas nas mais diferentes fontes de dados. Essa massificação da informação e a natural facilidade de acesso a ela revolucionou o desenvolvimento de sistemas de informação e motivou o crescimento de aplicações que integram dados heterogêneos distribuídos[Eyal 2001].

Da mesma forma que cresce a possibilidade de obter-se acesso a uma vasta quantidade de dados, aumenta significativamente também a probabilidade de dados relativos ao mesmo domínio de conhecimento estarem dispostos de formas distintas, visto que tais bases são projetadas por pessoas que possuem diferentes concepções do mundo. Particularmente na *web*, temos as informações dispostas em fontes de dados autônomas e dinâmicas, com seus dados sendo semi-estruturados ou até mesmo não estruturados, dificultando ainda mais a integração dos diversos esquemas existentes [Abiteboul et al. 2000].

Muitos dos problemas encontrados na construção dos sistemas de integração de dados na *web* são similares aos problemas encontrados em sistemas de integração de bancos de dados tradicionais. Gerenciar a heterogeneidade semântica e estrutural das informações é, também, um dos principais desafios de sistemas de integração para *web*. Porém, em sistemas de integração de dados na *web* existe uma série de outros problemas adicionais que devem ser solucionados, dentre os quais podemos destacar os seguintes [Abiteboul et al. 2000]:

- O grande número de fontes de informação disponível e a constante mudança de localização das mesmas;
- A convenção heterogênea de valores, onde as fontes podem usar diferentes tipos de dados para representar a mesma informação, ou mesmo representar o mesmo tipo de dados com nomes diferentes;



- A falta de metadados sobre os dados;
- O alto grau de autonomia das fontes de informação;
- A estrutura de dados extremamente irregular.

Os dados disponíveis na *web* podem ser provenientes tanto de banco de dados tradicionais como podem ser totalmente não estruturados, como vídeos, imagens e textos. Porém, grande parte das informações dispostas no universo *web* apresenta-se de forma semi-estruturada, ou seja, apresenta uma estrutura em seus documentos, como a estrutura HTML(*HyperText Markup Language*) por exemplo.

O surgimento da linguagem XML, proposta da W3C [W3C/XML 2002] como sendo o padrão para troca de informações na *web* visa trazer algumas facilidades aos sistemas de integração de dados, visto que, sendo padrão, esta linguagem oferece uma sintaxe comum de compartilhamento de dados. O uso de XML no presente trabalho será discutido posteriormente neste capítulo.

## **2.4. Integração e dados semi-estruturados**

Quando se deseja desenvolver sistemas de integração de dados, especialmente no universo *web*, é fundamental considerar a presença de dados semi-estruturados, visto que a grande massa de informações dispostas na *web* atualmente está definida de forma semi-estruturada. A seguir é apresentada uma breve definição do que são os dados semi-estruturados e porque a linguagem XML é utilizada no processo de representação dos mesmos.

Dados semi-estruturados são aqueles nos quais há um sistema de representação presente, explícita ou implicitamente. Pode-se dizer que dados semi-estruturados são auto-descritivos [Abiteboul et al. 2000].

Mesmo em sistemas de integração nos quais as fontes de dados são estruturadas, é comum o aparecimento de informações semi-estruturadas. Porém, o fator *web* tem sido o grande motivador para novos estudos nesta área. A seguir são listadas algumas das razões que contribuíram para esse crescimento de interesse na pesquisa de dados semi-estruturados [Atzeni et al. 1997]:

- A necessidade de se tratar algumas fontes de dados *web* como se fossem bancos de dados, porém sem estarem restritas a um esquema;

- É aconselhável ter um formato flexível para troca de dados entre bancos de dados;
- Pode ser necessário, mesmo em caso de fontes estruturadas, ter uma visão semi-estrutura das informações para navegação.

As principais características dos dados semi-estruturados são destacadas abaixo [Abiteboul 1997]:

- Estrutura irregular - a falta de padrão possibilita que diferentes instâncias, mesmo sendo classificadas como de mesmo tipo, tenham estruturas diferentes. Um exemplo seria um preço de uma determinada mercadoria ser representada em dólar em uma determinada base e em reais em outra, ou em outro exemplo, o endereço ser tratado como string ou como uma tupla;
- Estrutura implícita - em muitas aplicações, embora exista uma estrutura precisa, esta se encontra implícita. Para se obter a estrutura das informações é necessário algum esforço computacional, tal como um *parser* que identifique a estrutura dos dados, por exemplo;
- Estrutura parcial - nem todos os tipos de dados possuem uma estrutura definida. Por exemplo, um componente de objeto, que são arquivos *bitmaps* é caracterizado como não estruturado, já um catálogo de produtos pode apresentar uma estrutura, sendo ela implícita ou explícita. Assim, o modelo para esse tipo de dados pode não estar sempre completo semanticamente, não representando completamente todas as informações presentes;
- Esquema à *posteriori* - os modelos (esquemas) de dados semi-estruturados, normalmente, são definidos em função da análise de informações já existentes, sendo o processo de construção de seus esquemas baseado na investigação das estruturas particulares existentes e na busca de similaridades e diferenças entre as fontes de dados;
- Esquema muito extenso - consequência da grande heterogeneidade existente entre as fontes. Uma grande quantidade de fontes de dados diferentes sobre um determinado domínio pode gerar um esquema demasiadamente extenso em relação à quantidade de informação;

- Esquema ignorado - é comum ignorar um esquema de algumas consultas que têm mais de uma resposta natural. Assim, tais consultas tornam-se eficientes quando se busca apenas uma resposta qualquer sem que se tenha que precisar onde a busca foi efetuada;
- Evolução rápida do esquema - diferentemente dos SGBD (Sistema Gerenciador de Banco de Dados), nos quais os esquemas são vistos quase sempre como imutáveis e as atualizações nos mesmos são raras, os dados semi-estruturados apresentam um esquema que deve estar apto a ser expandido com frequência. Alterações nas fontes de dados, ou inserção de novas fontes de dados ao esquema invariavelmente alteram a estrutura existente do esquema. É importante, portanto, tornar esse processo o mais simples possível;
- Distinção confusa entre os dados e o esquema - geralmente a distinção entre o esquema e os dados é complicada, visto que a estrutura está embutida na representação dos dados. Por exemplo, a representação de um telefone pode ser feita através de um inteiro em uma determinada fonte e em outra a mesma informação pode ser representada como um tipo derivado de uma classe chamada telefone;
- Dados do esquema com mais de um sentido - outro aspecto importante em dados semi-estruturados é a possibilidade da forma de representação dos dados depender do ponto de vista ou de uma fase particular no processo de aquisição dos mesmos.

Existem alguns modelos definidos para a representação de dados semi-estruturados. Dentre os mais difundidos podemos destacar o OEM (*Object Exchange Model*), que foi inicialmente definido para modelar dados de diversos projetos do grupo de bases de dados da Universidade de *Stanford* [Goldman 2003; OEM 2003]. Outro modelo existente é o ADM (*Araneus Data Model*) [Mecca 1997], que é um modelo de dados baseado no padrão ODMG (*Object Database Management Group*) [ODMG 2000] de representação de objetos. Pode-se dizer que ADM é um modelo para dados semi-estruturados *orientado a páginas web*. O ADM é o resultado da extração da estrutura de um conjunto de página homogêneas na *web* [Mecca 1997].

### 2.4.1. XML

A linguagem XML vem sendo largamente utilizada para representação de dados semi-estruturados. Sua estrutura auto-descritiva, com suporte em um modelo hierárquico, o qual separa a informação da formatação dos dados, é uma das razões que fazem desta linguagem alvo de diversos estudos buscando uma maior flexibilidade na representação de dados na *web*, obtendo assim uma maior facilidade de consulta aos mesmos, bem como uma maior eficácia no resultado das consultas.

No presente escopo, XML é utilizada em diversas etapas do desenvolvimento do protótipo. A seguir serão discutidos os principais conceitos relacionados a esta linguagem.

XML [W3C/XML 2002] é uma metalinguagem derivada da SGML (*Standard Generalized Markup Language*) que permite que uma marcação específica seja criada para especificar idéias e compartilhá-las na rede. Ela tem as virtudes de SGML e de HTML, sem qualquer das limitações inerentes a essas duas linguagens [Mcgrath 1999]. XML foi desenvolvida pelo consórcio W3C e se tornou um padrão para a publicação, a combinação e o intercâmbio de documentos, além de ser utilizada na representação de dados estruturados e semi-estruturados [W3C/XML 2002]. Dentre os fatores que tornaram XML uma linguagem cada vez mais difundida no universo *web* pode-se destacar os seguintes:

- Adequação - XML é adequada para qualquer nível de complexidade ou abstração. A marcação pode ser definida tanto para uma representação mais geral como <NOME> João </NOME> quanto para uma mais específica, como <ENDERECO> <NUMERO> 1233 </NUMERO> </ENDERECO>. Desta forma, é possível definir as *tags* de marcação de acordo com a complexidade ou nível de abstração de seu conteúdo;
- Adaptação - XML é uma linguagem-mãe para outras linguagens. Assim, linguagens como VML (*Vector Markup Language*) [W3C/VML 2002] e

MathML [W3C/Math 2002] tornaram-se possíveis, permitindo que marcações especializadas possam ser criadas para qualquer necessidade;

- **Manutenção:** a XML é fácil de manter. Ela contém somente idéias e marcações. Folhas de estilos e links vêm em separado e são escondidas no documento. Cada um pode ser alterado separadamente quando preciso;
- **Ligação** - XML possui uma maneira de ligar documentos que inclui todas as formas possíveis de ligação. Ela é capaz de ligar documentos de maneira que HTML não pode, visto que XML permite ligar dois ou mais pontos a uma idéia;
- **Suporte em grandes corporações** - grandes empresas estão adotando XML em marcas registradas, tais como Microsoft, IBM, Sun, Software AG, Netscape e Oracle. Conseqüentemente, abre-se um grande mercado para o desenvolvimento de ferramentas XML.

Os documentos XML possuem a mesma estrutura básica, sendo formados, basicamente, por elementos e atributos como mostra o exemplo no Quadro 2.1:

**Quadro 2.1 – Exemplo de código XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<brasileirao xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">

<jogador colecao="futebol" caminho="esporte/inter">
    <nome>Lúcio</nome>
    <posicao>Lateral</posicao>
    <equipe>Palemiras</equipe>
    <idade>22</idade>
</jogador>
<jogador colecao="futebol" caminho="esporte/inter">
    <nome>Leandrinho</nome>
    <posicao>Volante</posicao>
    <equipe>Juventude</equipe>
    <idade>23</idade>
</jogador>
<jogador colecao="futebol" caminho="esporte/inter">
    <nome>André Guerreiro</nome>
    <posicao>Ala</posicao>
    <equipe>Internacional</equipe>
    <idade>23</idade>
</jogador>

</brasileirao>
```

Não será detalhada a estrutura básica dos documentos XML, pois esta já é bastante conhecida e anteriormente apresentada em diversos trabalhos na área.

Um conceito relevante ao presente trabalho é a linguagem a partir da qual os esquemas para XML serão definidos. DTD (*Document Type Definition*) e XSD (*XML Schema Definition*) são as duas mais difundidas [Vlist 2001; W3Schools/DTD 2002].

### DTD

DTD é uma linguagem que permite que se estabeleça um conjunto de restrições para um documento XML. Ela define a forma como um documento XML deve ser construído, através da definição hierárquica de seus elementos.

O processo de validação de documentos XML em uma DTD é realizado através de um *parser* XML que verifica se o documento XML está estruturado de acordo com o que foi definido na DTD. No Quadro 2.2 é apresentado um exemplo de uma DTD para representar o arquivo XML relacionado à futebol representado anteriormente.

**Quadro 2.2 – Exemplo de DTD**

```
<!ELEMENT brasileiro (jogador)+>
<!ELEMENT jogador (nome, posicao, equipe, idade)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT posicao (#PCDATA)>
<!ELEMENT equipe (#PCDATA)>
<!ELEMENT idade (#PCDATA)>
<!ATTLIST jogador colecao CDATA #REQUIRED>
<!ATTLIST jogador caminho CDATA #IMPLIED>
<!ELEMENT brasileiro (jogador)+>
<!ELEMENT jogador (nome, posicao, equipe, idade)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT posicao (#PCDATA)>
<!ELEMENT equipe (#PCDATA)>
<!ELEMENT idade (#PCDATA)>
<!ATTLIST jogador colecao CDATA #REQUIRED>
<!ATTLIST jogador caminho CDATA #IMPLIED>
```

Até recentemente, a DTD era a forma mais adotada para a classificação de documentos XML. Porém, a linguagem apresenta algumas deficiências que estimularam a busca por novas alternativas para a representação dos dados. A seguir são listados os principais problemas.

A linguagem DTD:

- Não é escrita em XML;

- Não apresenta suporte para *namespaces* (que serão abordados posteriormente);
- Oferece poucas possibilidades de definição de tipos de dados, não permitindo representar dados como moeda, data, entre outros;
- É um mecanismo de extensão frágil e complexo, baseado basicamente em substituição de cadeia de caracteres.

Uma alternativa que visa suprir essas limitações da DTD é o uso de XSD.

### **XSD**

XSD é um modelo desenvolvido pelo W3C com o objetivo de auxiliar e, posteriormente, substituir as DTD como forma de definição de dados XML [W3C/XSD 2003]. Este modelo apresenta diversas características ausentes nas DTD, tais como tipagem de estrutura, criação de restrições, herança, unicidades, chaves, além de dar suporte à definição de *namespaces* e ser escrita em XML, o que facilita a compreensão do desenvolvedor no momento de criar a definição dos dados

No presente trabalho, os XML *Schemas* são o formato utilizado para a representação dos dados, visto que, além de serem mais completos que a DTD, esta é a forma de representação dos dados utilizado pelo Tamino XML Server (como será abordado posteriormente no capítulo 3). Um exemplo de XML *Schema* sobre futebol é apresentado no Quadro 2.3.

### Quadro 2.3 – Exemplo de XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="brasileirao_schema">
        <tsd:collection name="futebol"/>
        <tsd:doctype name="brasileirao">
          <tsd:logical>
            <tsd:content>closed</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="brasileirao">
    <xs:complexType>
      <xs:all maxOccurs="unbounded">
        <xs:element name="jogador">
          <xs:complexType>
            <xs:all>
              <xs:element name="nome" type="xs:string"/>
              <xs:element name="posicao" type="xs:string"/>
              <xs:element name="equipe" type="xs:string"/>
              <xs:element name="idade" type="xs:integer"/>
            </xs:all>
            <xs:attribute name="colecacao" type="xs:string" use="required"/>
            <xs:attribute name="caminho" type="xs:string" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

O XML *Schema* apresentado acima é uma outra forma de representação do documento XML anteriormente apresentado. Pode-se observar que como toda a sua definição é baseada em XML, fica mais clara para os que já estão familiarizados com esse tipo de estrutura identificar os elementos, seus filhos e assim por diante. Comparando as estruturas das representações da estrutura por DTD e por XSD, pode-se observar algumas diferenças na forma com que os dados são representados. Mesmo se tratando de um exemplo simples, o tipo definido no elemento *idade* é definido como texto



(PCDATA) na DTD, enquanto que no XSD foi possível definí-lo com o tipo inteiro (*integer*) mais de acordo com o valor real do elemento.

Outro ponto relevante é o suporte a *namespaces* (apresentado a seguir), que embora tenha sido referenciado no exemplo, não tem suas especificações utilizadas diretamente no mesmo, porém já mostra a maior capacidade desse modelo de documento. Com a importação do *namespace* do Tamino, o XML *Schema* poderia utilizar as tags próprias da TSD (*Tamino Schema Definition*) [Tamino 2003] para, por exemplo, mapear elementos para uma determinada base de dados, como será abordado em maiores detalhes no Capítulo 3.

A seguir são apresentadas algumas definições relacionadas a XML que são abordadas durante o desenvolvimento do projeto.

### **XML Namespaces**

*Namespaces* são uma maneira simples e direta de distinguir nomes usados em documentos XML, não importando de onde eles venham. Entretanto, o conceito é um pouco abstrato, e a especificação pode apresentar algumas questões relacionadas a como os *namespaces* são lidos [Bray 1999]. Assim, a maneira mais prática de se entender o conceito de *namespaces* e o quanto eles são úteis e devem ser utilizados nas mais diferentes aplicações XML é através de um exemplo.

Seja um cenário no qual se deseja utilizar marcações para fazer a revisão de livros segundo um modelo proposto pelo Centro de Informática da UFPE. Mesmo querendo marcar as informações, deseja-se utilizar HTML como linguagem auxiliar para definir a forma de exibição das mesmas. Assim, poderíamos ter um documento da forma mostrada no Quadro 2.4:

**Quadro 2.4 – Exemplo de documento que utiliza namespaces**

```
<h:html xmlns:xdc="http://www.cin.ufpe.br/books"
        xmlns:h="http://www.w3.org/HTML/1998/html4">
  <h:head><h:title>Revisão de Livros</h:title></h:head>
  <h:body>
    <xdc:bookreview>
      <xdc:title>XML: A Primer</xdc:title>
      <h:table>
        <h:tr align="center">
          <h:td>Author</h:td><h:td>Price</h:td>
          <h:td>Pages</h:td><h:td>Date</h:td></h:tr>
        <h:tr align="left">
          <h:td><xdc:author>Simon St. Laurent</xdc:author></h:td>
          <h:td><xdc:price>31.98</xdc:price></h:td>
          <h:td><xdc:pages>352</xdc:pages></h:td>
          <h:td><xdc:date>1998/01</xdc:date></h:td>
        </h:tr>
      </h:table>
    </xdc:bookreview>
  </h:body>
</h:html>
```

Nesse exemplo, os elementos que utilizam o prefixo *xdc* são associados com o *namespace* definido pelo CI, no caso *http://www.cin.ufpe.br/books*, enquanto que os com o prefixo *h* são associados à especificação de HTML na W3C.

Os prefixos são associados ao nome completo dos *namespaces* através das inserções dos atributos no elemento raiz do documento, pelos nomes iniciados por *xmlns:*. A definição dos *namespaces* é realizada através de uma URL (*Universal Resource Locator*) onde o mesmo deve estar localizado.

### **Interfaces para Programação de Aplicação (API)**

Outro ponto importante é a forma de trabalhar com documentos XML. Existe uma série de linguagens e meios de acesso a tais documentos, acessando e alterando seus elementos e atributos. A seguir serão abordadas algumas das principais API (*Application Programming Interface*) utilizadas no acesso a documentos XML.

### **DOM**

DOM (*Document Object Model*) trata as informações armazenadas nos documentos XML como um modelo de objetos hierárquicos, criando uma árvore de nós (baseado na estrutura e na informação do documento XML) possibilitando o acesso à

informação do documento através de interações nessa árvore [W3C/DOM 2003]. DOM preserva a sequência dos elementos lidos a partir dos documentos XML, visto que ele trata toda a estrutura como se fosse um documento. Isto justifica seu nome: modelo de objeto do documento.

Uma questão a ser considerada é que, através do DOM, a árvore hierárquica é gerada representando todo o documento XML, portanto, todo o documento é colocado na memória, o que por um lado é positivo pois o acesso aos elementos do mesmo é rápido. Em contrapartida, dependendo do tamanho e complexidade do documento que se está manipulando, pode acarretar uma série de problemas, tal como a sobrecarga de recursos resultando na falta de memória vindo até a inviabilizar a aplicação.

### **SAX**

SAX (*Simple API for XML*), outra API útil para processar documentos XML em qualquer linguagem de programação. Processa a informação em documento XML como uma sequência de eventos [SAX 2003]. Isso torna SAX mais rápido que DOM, embora exija:

- Criação de um modelo personalizado de objetos próprio;
- Criação de uma classe que capture os eventos SAX (gerados pelo *parser* SAX conforme ele obtém o documento XML) e que crie adequadamente o modelo de objetos.

A utilização das API para acesso aos documentos XML no presente escopo é discutida no Capítulo 4.

### **Linguagens de consulta para XML**

Considerando consultas efetuadas sobre SGBD relacionais tradicionais, temos o domínio das consultas bem estruturado, sendo realizado sobre tabelas, e índices que tornam mais padronizado o processo de busca das informações. No domínio de dados semi-estruturados, no qual se encaixam os documentos XML, a linguagem de consulta deve ter a habilidade de trabalhar com estruturas pouco regulares, o que inclui flexibilidade de tipos e estruturas complexas, entre outras, além de permitir uma sintaxe ao menos semelhante às linguagens de consulta convencionais [Abiteboul 1997]. Assim, várias pesquisas vêm sendo realizadas com o intuito de criar e aprimorar as linguagens de consulta XML.

Dentro desse universo, foram definidas duas linhas diferentes de desenvolvimento. A primeira, nas quais as linguagens são inspiradas em linguagens tradicionais de SGBD, como SQL (*Structured Query Language*) [W3Schools/SQL 2003] e OQL (*Object Query Language*) [Cisco 2002], por exemplo. A outra linha é baseada em alguma definição formal de dados semi-estruturados, mais voltada à estrutura da XML, sem se preocupar com nenhuma relação com o modelo das bases de dados tradicionais.

Dentre as linguagens existentes, pode-se citar a LOREL [Abiteboul et al. 1997], XQuery (*XML Query*) [W3C/XQuery 2003], XPath [W3C/XPath 2003], XML-QL [W3C/XML-QL 1998] e XQL (*XML Query Language*) [Robie 1999], entre outras.

A seguir são apresentadas algumas características desejáveis em uma linguagem de consulta para XML [W3C/XML\_Query 2003; Abiteboul 1997].

- O resultado da consulta deve gerar um documento XML bem formado;
- Prover facilidades para a criação e a manipulação de consultas pelos programas;
- Preservar a estrutura dos documentos (ordenação e associação dos elementos);
- Manter as operações primitivas de consultas encontradas em bancos de dados, tais como seleção e extração;
- Consultas constituindo de três partes: cláusula, filtro e construtor;
- Conversão de tipos.

No desenvolvimento do projeto são utilizadas as linguagens XQL e XQuery, visto que estas têm suporte no Tamino XML Server. No entanto, é importante ressaltar que no Tamino são implementadas linguagens baseadas no modelo proposto pela W3C, porém já há uma grande parte da especificação das linguagens implementada e todos os exemplos de consultas apresentados neste trabalho têm suporte no Tamino XML Server versão 4.1.4.1.

## **XQL**

Tratada com o nome X-Query no Tamino, esta é uma linguagem do paradigma funcional proposta com o intuito de estender os padrões da XSL (*Extensible Stylesheet Language Family*) [W3C/XSL 2003]. Esta linguagem é relativamente simples, bastando conhecer os elementos que se deseja consultar, respeitando a sua ordem hierárquica, que é possível acessar o seu conteúdo navegando através de sua estrutura. A sintaxe para

acessar o documento é realizada de forma semelhante à utilizada quando se navega na estrutura de diretórios de um sistema operacional. Tomando como exemplo o documento XML que representa o cadastro de jogadores de futebol apresentado anteriormente neste capítulo, é possível consultar os nomes dos jogadores através da chamada *brasileirao/jogador/nome*, tendo como resultado todos os nomes dos jogadores do documento. No caso do Tamino, todos os nomes de jogadores em todas as instâncias do modelo serão retornadas, porém, isso será discutido posteriormente no capítulo 3.

O Quadro 2.5 apresenta alguns exemplos de consulta XQL, considerando a estrutura XML para representação de jogadores de futebol apresentada anteriormente.

**Quadro 2.5 – Lista de comandos XQL**

<b>Comando</b>	<b>Resultado Retornado</b>
<code>/brasileirao/jogador/equipe</code>	Todas as equipes que tem algum jogador.
<code>/brasileirao/jogador[@colecacao='futebol']</code>	Todos os jogadores cujo atributo coleção é 'futebol'
<code>/brasileirao/jogador/*</code>	Todos os elementos filhos de jogador.
<code>/brasileirao/jogador[idade &lt; 23 and equipe='Internacional']/nome</code>	O nome de todos os jogadores cuja idade é menor que 23 e a equipe é 'Internacional'
<code>/brasileirao/jogador [nome~='A*']</code>	Todos os jogadores cujo nome inicia com 'A'

### **XQuery**

A linguagem XQuery é a outra linguagem de consulta XML implementada pelo Tamino XML Server. Ela foi especificada e desenvolvida pela W3C [W3C/XQuery 2003] como linguagem para realizar consultas em coleções de dados XML, ou seja, não apenas arquivos XML, mas qualquer informação que possa aparecer como XML, incluindo bases de dados relacionais [Hunter 2004].

A XQuery provê um mecanismo para facilitar a extração de informações de NXD [W3C/NXD 2003], que será abordado posteriormente neste trabalho, além de consultas a bases de dados relacionais. Através da XQuery é possível visualizar tabelas relacionais como se fora uma fonte de dados XML qualquer.

A linguagem XQuery é funcional, permitindo uma série de tipos de expressões aninhadas e combinadas, além de ser fortemente tipada, tendo os tipos oferecidos baseados na definição do XML *Schema*. Outra característica é a implementação de expressões FLWR que contém cláusulas que permitem que sejam realizadas consultas de forma semelhante a linguagem SQL. Essas cláusulas oferecidas são as seguintes [XQuery 2004] :

- **FOR:** é um construtor de iterações que liga uma variável a uma sequência de valores retornados por uma consulta;
- **LET:** relaciona variáveis a valores de maneira análoga à atribuição em linguagens de programação;
- **WHERE:** contém predicados que são utilizados nos valores retornados pela consulta;
- **RETURN:** gera a saída para a expressão.

O Quadro 2.6 apresenta algumas consultas XQuery realizadas segundo o modelo de dados sobre futebol.

**Quadro 2.6 – Lista de comandos XQuery**

<b>Comando</b>	<b>Resultado Retornado</b>
For \$b in input()/brasileirao/jogador return \$b/nome	Os nomes de todos os jogadores.
Let \$a := input()/brasileirao/jogador return <index type="jogador"> { \$a/equipe } { \$a/posicao } </index>	O nome de todas as equipes dos jogadores, seguido das posições dos mesmos. Essas informações indexadas pelo campo “jogador”.
For \$a in input()/brasileirao/jogador return \$a/@*	Todos os atributos existentes em “jogador”.
Let \$a := input()/brasileirao/jogador return <p>Existem atualmente { count(\$a) } jogadores cadastrados.</p>	A consulta retorna quantos jogadores existem cadastrados. É importante frisar a possibilidade de personalizar a saída, no caso, é mandado um texto com a tag <p> que poderia perfeitamente ser inserido em uma página HTML.

Como se pode observar nos exemplos acima, existem muitas possibilidades de consultas XQuery e sua sintaxe é sintaticamente semelhante a linguagens de programação e de consulta de banco de dados. Esta linguagem é bastante útil para geração de consultas complexas a bases de dados de armazenamento nativo. No caso do Tamino, ela tem suporte desde que não seja usado nenhum componente no esquema, como será detalhado no Capítulo 3.

## **2.5. BANCO DE DADOS E DOCUMENTOS XML**

A linguagem XML foi criada com a finalidade de descrever informações. Porém, com a expansão da utilização de XML como meio de troca de informações na *web*, surgiram os bancos de dados XML, que consiste de coleções de documentos XML que persistem e podem ser manipulados. Considerando-se que XML está cada vez mais difundida no que tange ao transporte de informações, é importante considerar a forma como os dados transportados são armazenados no SGBD.

Analizando a linguagem XML em relação aos bancos de dados relacionais, pode-se afirmar que a estrutura de dados XML é mais expansiva, visto que essa estrutura permite a representação de múltiplos elementos do mesmo tipo e de diversos atributos para cada elemento, facilitando a representação de dados de estrutura mais complexa [Gomes 2002].

Várias são as formas de se incluir um documento XML em uma base de dados. A inclusão pode ser feita como um documento ou como partes menores, sendo incluídas em fragmentos ou até mesmo como elementos individuais. Esses elementos podem ser inseridos em banco de dados XML, estrutura XML, ou passar por um *parser* e serem incluídos como instâncias de relações (SGBD Relacionais). A seguir é apresentado um resumo sobre a utilização de XML em SGBD.

### **Sistema Gerenciador de Banco de Dados Relacional**

Os principais fabricantes de Sistemas de Banco de Dados Relacionais estão inserindo na estrutura de seus SGBD suporte ao armazenamento de documentos XML. A forma adotada por muitos fabricantes é a criação de uma interface XML entre os documentos XML e o armazenamento físico (linhas /colunas, no caso dos SGBD relacionais) [Tamino 2003].

Esta forma de armazenamento de arquivos XML apresenta uma série de vantagens e desvantagens, dependendo do cenário em que se esteja trabalhando. Uma das principais vantagens dessa forma de armazenamento XML é que esta permite que se tenha acesso a consagrada e robusta tecnologia de banco de dados relacionais. Caso já exista uma base de dados relacional na organização, esta alternativa é largamente adotada.

Com a estrutura relacional de armazenamento, mantém-se a possibilidade de realizar consultas complexas através da linguagem SQL [Browne 2002], o que garante eficiência na busca de informações. Nessa forma de armazenamento, as aplicações herdam outras características importantes dos bancos relacionais, como sofisticado sistema de gerenciamento de transações, bom mecanismo de recuperação, e diversas ferramentas que dão suporte a este tipo de tecnologia.

Existe, porém, uma série de desvantagens em se utilizar essa forma de armazenamento. Dentre as quais, pode-se destacar a dificuldade em tratar documentos XML de estruturas complexas com dados multivalorados e com diversos atributos. Esses problemas podem gerar erros de tradução causando um custo muito grande para os administradores do BD, além da possibilidade de provocar inconsistências nos dados.

### **2.5.1. Sistema de Gerenciamento de Banco de Dados XML**

Um SGBD XML provê acesso direto a um documento XML ou a fragmentos do documento e habilita consultas através desses documentos e fragmentos. Dentro desse contexto, pode-se destacar os Sistemas Gerenciadores de Banco de Dados XML Nativo, que são projetado especificamente para armazenar e manipular dados XML. O acesso aos dados é através de XML e padrões relacionados, como XSLT (*Extensible Stylesheet Language Transformations*) [XSLT 2002], DOM ou SAX [Bray 2000]. Nesta categoria são relacionados todos os bancos de dados nos quais a representação dos dados mantém a estrutura do documento XML, bem como os metadados associados.

A unidade fundamental de armazenamento dos NXD é um documento XML. Dentro dessa categoria se enquadra o Tamino XML Server, que foi desenvolvido para *e-business*, sendo a primeira base de dados nativa XML permitindo o armazenamento de dados tanto estruturados como não-estruturados. Diversos estudos apontam para as vantagens do armazenamento de XML de forma nativa [Tamino 2003; Carr 2003], no



qual não é realizada conversão para nenhum outro formato de dados. Através da manipulação de dados de forma nativa, espera-se um menor tempo de resposta, menos esforço de administração e maior capacidade de integração de dados de sistemas legados [Bourret 2003].

No Capítulo 3 será abordado em mais detalhes o funcionamento dos sistemas de gerenciamento de bancos de dados de armazenamento XML nativo, apresentando exemplos de funcionamento através do Tamino XML Server.

## **2.6. Web Semântica**

Em meados da década de 90 surgiram os primeiros estudos e esforços direcionados à criação da aplicação mais popular da Internet, a WWW (*World Wide Web*) [Berners-Lee et al. 1994]. A *web* se consolidou como o meio de distribuição de informação com o crescimento mais rápido da história contemporânea. Entretanto, esse rápido desenvolvimento e expansão fizeram com que seu conteúdo se tornasse um gigantesco depósito de informação tão grande quanto desorganizado. Tarefas como localização e acesso à informações distribuídas na rede tornam-se a cada dia mais complicadas, pois esse crescente universo de documentos é, além de vasto, desprovido de qualquer forma de padronização.

Passada uma década do surgimento da *web* e, detectados seus maiores problemas, surge um novo desafio: dar mais inteligência a *www*, visando com isso eliminar ou amenizar os problemas advindos de suas deficiências. A idéia principal consiste em permitir que a *web*, além de interligar documentos, possa reconhecer o significado da informação contida neles. Esta tarefa, simples para os seres humanos, representa um enorme desafio para os computadores. Esta nova *web* inteligente foi batizada de Web Semântica. Segundo [Berners-Lee et al. 2001], a Web Semântica representa a evolução da *web* atual. Ela objetiva fornecer estruturas e dar significado semântico ao conteúdo das páginas *web*, criando um ambiente no qual agentes de software e usuários possam trabalhar de forma cooperativa.

A Web Semântica é composta por vários elementos que, trabalhando juntos, proverão a estrutura necessária para que a visão idealizada em [Berners-Lee et al. 2001], seja concretizada. Tais elementos incluem desde linguagens de marcação e anotação

semântica, até agentes de software, responsáveis por consumir e processar essas informações. No presente trabalho, a parte que inicialmente interessa é a relacionada ao uso de ontologia, a qual é descrita com maiores detalhes na subseção a seguir.

### **2.6.1. Ontologia**

O termo ontologia advém primordialmente das áreas de Filosofia e Epistemologia significando, respectivamente, um “sujeito da existência” e um “conhecimento e saber”. Na computação, o termo ontologia tem sido usado sobretudo pela área de IA (Inteligência Artificial), visando descrever e relacionar conceitos utilizados por um agente ou por uma comunidade dos mesmos, sendo aplicada, por exemplo, no processamento de linguagens naturais e representação do conhecimento [Bézivin 1998].

Mais recentemente, a noção de ontologia também está sendo difundida em áreas que pesquisam a cooperação de sistemas de informação, comércio eletrônico, recuperação de informação e gerência de conhecimento. Do ponto de vista de BD, uma ontologia é uma “especificação parcial de um domínio ou meta-domínio, descrevendo entidades, relações entre elas e regras de integridade”. Ontologia pode ser definida como um modelo conceitual de dados. Segundo [Daum 2002], uma ontologia pode ser vista como uma taxonomia de conjunto de termos estruturados sob a forma de árvore. Ela é usada para descrever um domínio específico através de seus termos, propriedades e relacionamentos entre eles.

A definição que melhor caracteriza a essência do que é uma ontologia é: “uma ontologia é uma especificação formal e explícita de uma conceitualização compartilhada”, sendo que conceitualização refere-se a um modelo abstrato de algum fenômeno, e formal refere-se ao fato de que a ontologia deve permitir sua leitura por uma máquina [Bézivin 1998].

Para entender um pouco o uso de ontologias, podemos exemplificar seu uso da seguinte maneira: suponha duas lojas especializadas na venda de artigos esportivos, sendo que ambas mantêm esquemas distintos, os quais descrevem a estrutura na qual as informações de seus produtos são armazenadas. Para que duas lojas A e B possam trocar informações de forma correta, uma ontologia pode ser utilizada para explicitar formalmente que a propriedade chamada “Valor” na loja A é totalmente equivalente à propriedade “PreçoFinal” na loja B. Devido a isso, pode-se afirmar que ontologias

realizam definições comuns e compartilhadas sobre domínios de conhecimento, pois a partir dela, um agente “entenderá” que “Valor” é o mesmo que “PreçoFinal” [Chandrasekaran et al. 1999].

Atualmente, com os estudos da comunidade *web* sendo voltados para o desenvolvimento da Web Semântica, diversas linguagens para definição e editoração de ontologias têm sido propostas. RDF/RDFS (*Resource Description Framework/ Resource Description Framework Schema*) [Decker et al. 2000], OWL (*Ontology Web Language* e DAML (*DARPA Agent Markup Language*)+OIL (*Ontology Inference Layer*) [McGuinness et al. 2002] são alguns exemplos. A combinação de DAML e OIL, denominada DAML+OIL [OIL 2003], foi desenvolvida a partir do DAML-Ont combinando muitos componentes da linguagem OIL, resultando em uma nova linguagem de semântica bem definida. A DAML+OIL foi escrita em RDF e, sendo assim, declarações em DAML+OIL são também declarações RDF, ou seja, a DAML+OIL é na verdade, um vocabulário adicionado a RDF, com o objetivo de prover uma linguagem mais poderosa para a criação de ontologias.

O Quadro 2.7 abaixo apresenta um exemplo de documento RDF no qual informações sobre o domínio futebol são representadas.

### Quadro 2.7 – Exemplo de documento RDF

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
  <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY a 'http://protege.stanford.edu/system#'>
  <!ENTITY futebol 'http://protege.stanford.edu/futebol#'>
  <!ENTITY rdfs 'http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#'>
]>
<rdf:RDF xmlns:rdf="&rdf;"
  xmlns:a="&a;"
  xmlns:futebol="&futebol;"
  xmlns:rdfs="&rdfs;">
  <rdf:Property rdf:about="&futebol;KB_1676_00006"
    a:maxCardinality="1"
    rdfs:label="KB_1676_00006">
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdf:Property>
  <rdfs:Class rdf:about="&futebol;atleta"
    rdfs:label="atleta">
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&futebol;equipe"
    rdfs:label="equipe">
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
  </rdfs:Class>
  <rdf:Property rdf:about="&futebol;idade"
    a:maxCardinality="1"
    a:minCardinality="1"
    rdfs:label="idade">
    <rdfs:domain rdf:resource="&futebol;atleta"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdf:Property>
  <rdf:Property rdf:about="&futebol;nome_jogador"
    a:maxCardinality="1"
    rdfs:label="nome_jogador">
    <rdfs:domain rdf:resource="&futebol;atleta"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdf:Property>
  <rdf:Property rdf:about="&futebol;posicao"
    a:maxCardinality="1"
    rdfs:label="posicao">
    <rdfs:domain rdf:resource="&futebol;atleta"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdf:Property>
</rdf:RDF>
```

No esquema em RDF do Quadro 2.7, basicamente são representadas as classes *equipe* e *atleta*, sendo que na classe *atleta* são vinculadas as propriedades *idade*, *nome\_jogador* e *posição*.

No presente escopo as ontologias são utilizadas como meio de integrar semanticamente diversos bancos de dados que estejam sobre o mesmo domínio, de forma a tratar conceitos descritos de maneira distinta, porém com o mesmo significado. Para tanto, é definida uma ontologia sobre um determinado *domínio* (hospitais, *e-business*, eventos culturais, entre outros) e aplicar essa ontologia às bases de dados a serem integradas, visando obter uma visão única dos dados nos quais os conflitos semânticos sejam eliminados ou minimizados. A linguagem utilizada para o desenvolvimento da ontologia foi a DAML+OIL.

O Quadro 2.8 considera duas tabelas relacionais, uma em cada base, que apresentam estrutura semelhante ao exemplo de uso apresentado no Capítulo 5. Com elas será mais fácil entender como a ontologia irá auxiliar o sistema de integração.

**Quadro 2.8 – Tabelas relacionais Jogador e Atleta**

<b>Base A</b>	
<b>Tabela</b>	<b>Campos</b>
<i>Jogador</i>	<i>nome_jogador, idade, salario, posicao, time.</i>
<b>Base B</b>	
<b>Tabela</b>	<b>Campos</b>
<i>Atleta</i>	<i>nome, data_nasc, salario, posicao, equipe.</i>

Pode-se observar que os dados visam descrever dados relacionados a jogadores de futebol, porém, os mesmos foram elaborados de forma distinta:

- No caso mais geral pode-se identificar que o nome das tabelas, apesar de representar a mesma informação, são diferentes, visto que na base A é representada com o nome de *Jogador* e na Base B com o nome de *Atleta*;
- Da mesma forma, os campos *nome* e *nome\_jogador* e *time* e *equipe* também entram no caso de campos com o mesmo significado, mas representados de forma diferente;

- Os campos *idade* e *data\_nasc* também representam a mesma informação, só que neste caso, além do nome do campo a forma como a informação é apresentada também é diferente. Enquanto na Base A se representa a idade do jogador com o campo *idade*, os valores armazenados serão números, na Base B o que teremos serão campos do tipo *Data*.

O campo *posição* presente nas duas tabelas pode representar também um caso de conflito semântico. Na tabela *Jogador* da Base A podemos ter “*ala-direito*” para representar a posição de um determinado jogador enquanto que na base B essa informação pode ser representada por “*lateral-direito*”.

O Anexo A apresenta um exemplo mais significativo e completo de representação de uma ontologia através da linguagem DAML+OIL, além de representar uma quantidade maior de informação, apresenta aspectos de equivalência entre diferentes termos. É esse documento que será utilizado no exemplo de uso do presente trabalho.

A utilização de ontologia no sistema tem o objetivo de deixar o mesmo apto a “entender” que dois termos sintaticamente distintos podem ter o mesmo valor semântico.

## 2.7. Trabalhos Relacionados

Há uma série de projetos e estudos relacionados à integração de dados, dentre os quais pode-se destacar:

TSIMMIS – TSIMMIS (*The Stanford-IBM Manager of Multiple Information Sources*) [Tsimmis 2002] é um projeto constituído de uma série de ferramentas desenvolvidas para facilitar e agilizar o processo de integração de fontes de dados heterogêneas, o que pode incluir tanto dados estruturados quanto semi-estruturados. Seus componentes têm as seguintes funções:

- Traduzir consultas e informações (tradutores - *wrappers*);
- Extrair informações de *web sites*;
- Combinar informações de diversas fontes de dados;
- Permitir busca de dados na *web*.

O TSIMMIS suporta consultas através de sua linguagem de consulta OEM-QL (*Object Exchange Model Query Language*) [Molina 1995] e retorna objetos OEM [Molina 1995]. Evidente que é possível fazer inferências ao sistema de integração através das ferramentas oferecidas no projeto para este fim.

DEByE - ambiente DEByE desenvolvido no Laboratório de Bancos de Dados da Universidade Federal de Minas Gerais com o objetivo de prover ferramentas para permitir a extração de dados semi-estruturados [Debye 2003]. Essa extração é realizada com base na percepção do usuário para reconhecimento da estrutura implícita dos dados a serem extraídos a partir de exemplos fornecidos pelo próprio usuário. Estes exemplos são obtidos das fontes de dados semi-estruturadas das quais se deseja extrair objetos, através de uma interface gráfica que auxilia o usuário a descrever a estrutura implícita desses objetos. Os dados extraídos são armazenados em arquivos que constituem representações textuais de objetos semi-estruturados. Esses arquivos seguem o formato XML [Arantes et al. 2003], a partir do qual podem ser consultados através das ferramentas desenvolvidas para o ambiente.

ARANEUS - é um projeto combinado dos grupos de Banco de Dados da *Università di Roma Ter* e da *Università della Basilicata*. O projeto consiste no desenvolvimento de ferramentas de gerenciamento de dados provenientes da *web*. As técnicas propostas são baseadas em tecnologias de bancos de dados da seguinte forma [Araneus 2003; Mecca 1999]:

- Os *web sites* são descritos usando um modelo de dados formal;
- Baseadas no modelo, são desenvolvidas ferramentas e metodologias para tradução, consulta, integração, projeto e implementação dos *web sites*.

Dentre as contribuições deste projeto, pode-se destacar a linguagem *Ulixes* [Mecca 1999], proposta para criação de visões relacionais da *web*.

WHOWEDA (*WareHouse Of WEb DAta*) – Projeto desenvolvido pelo grupo *Web Warehousing & Mining Group* da *Nanyang Technological University* de Singapura com o objetivo de projetar e implementar *warehousing* para organizações que materializem e gerenciem suas informações a partir da *web*, desenvolvendo sistemas de apoio a decisão [Whoweda 2003]. Assim, o projeto prevê a construção de um *data warehouse* contendo informações estratégicas derivadas da *web*, permitindo que esses dados possam vir a ser incorporadas em *data warehouses* convencionais.

Xsync-ML – É um projeto desenvolvido ao longo de um trabalho de mestrado desenvolvido no centro de informática da UFPE que consiste no desenvolvimento de um *middleware* que visa permitir que documentos XML válidos e bem-formados possam ser

gerados a partir de bancos de dados relacionais, possibilitando que documentos XML possam ser gerados a partir de consultas SQL submetidas a um SGBD relacional via JDBC (*Java Database Connectivity*) [Sousa 2003].

Além dos trabalhos apresentados acima, existe também uma série de outros projetos e ferramentas comerciais que visam prover mecanismos eficientes de integração de dados, tais como Nimble [Nimble 2003], MetaMatrix [Metamatrix 2003] e Enosys [Enosys 2003].

## **2.8. Considerações Finais**

O presente trabalho utiliza diversos conceitos abordados neste capítulo no processo de integração de dados. As arquiteturas para integração, como data warehouse e mediadores serviram como base para a proposta apresentada no Capítulo 4, assim como técnicas atualmente em expansão na *web* também foram utilizadas, dentre as quais pode-se destacar ontologia e XML, esta última usada para a representação dos dados integrados.



### 3. TAMINO XML SERVER

Tamino XML Server [Dawis 2003] é uma plataforma desenvolvida pela Software AG cuja finalidade consiste em gerenciar diferentes tipos de dados com alto desempenho. Tamino é baseado em XML, usando esta linguagem para buscar, gerenciar e armazenar os dados, além de representar como documentos XML todas as informações retornadas ao usuário (mensagens de alerta, resultado de consultas, entre outras).

No âmbito do armazenamento de dados, Tamino pode armazenar tanto dados no modelo relacional quanto XML nativo, esse último o foco principal do servidor. O Tamino é o primeiro servidor de dados capaz de armazenar diretamente documentos XML em sua forma nativa, sem a necessidade de transformação para outro formato [Softwareag 2003].

Uma vez escolhido o Tamino, passou-se a buscar os componentes que seriam necessários para viabilizar a integração entre as bases e o NXD. Através de estudos aprofundados das funcionalidades oferecidas pelo Tamino XML Server concluiu-se que o mesmo oferece as seguintes características relevantes ao desenvolvimento de sistemas de integração:

- Possibilidade de armazenamento de dados, tanto na forma nativa quanto relacional;
- Mapeamento para bases externas;
- Representação da estrutura de dados através de XML *Schemas*;
- Recuperação das informações através do formato XML;
- Possibilidade de estender as funcionalidades oferecidas pelo servidor através das *Server Extensions*.

Dentre as características apresentadas acima, a única que não foi considerada para o desenvolvimento do trabalho é a que diz respeito ao armazenamento relacional, visto que no Tamino este tipo de armazenamento é extremamente limitado se comparado a outros SGBD relacionais mais robustos existentes no mercado.

#### 3.1. ARQUITETURA

A arquitetura do Tamio apresenta uma série de componentes responsáveis por tarefas que vão desde o armazenamento de dados em formato XML, até o mapeamento com bases externas. Entender como funciona a sua arquitetura é relevante para o desenvolvimento do presente trabalho, pois alguns dos componentes descritos serão utilizados posteriormente na implementação do protótipo do sistema, enquanto que outros serão responsáveis por funções mais internas do Tamino.

Os principais componentes relevantes ao trabalho desenvolvido que fazem parte da arquitetura do Tamino são:

- *X-Machine*;
- *X-Node*;
- *Data Map*;
- *Tamino Server Extensions*.

A seguir são descritas as funcionalidades principais de cada componente.

### **3.1.1. X-Machine**

O componente *X-Machine*, mostrado na Figura 3.1, pode ser considerado o motor que dirige o Tamino [Softwareag 2003]. Sua função básica é armazenar objetos XML e retorná-los a partir de suas respectivas fontes de dados. Isto é feito baseado no esquema definido pelo administrador no *Data Map* (componente no qual os XML *Schemas* são armazenados, como apresentado a seguir).

Para armazenar e recuperar objetos XML, a *X-Machine* provê uma série de sub componentes brevemente apresentados a seguir [Tamino 2003].

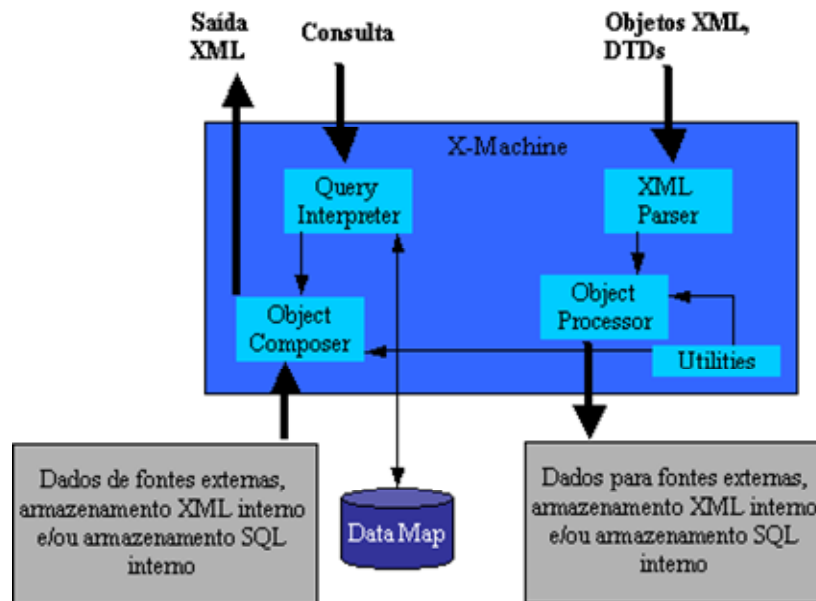


Figura 3.1 - Arquitetura da X-Machine

### XML Parser

Componente responsável por verificar se os objetos XML que estão sendo inseridos são sintaticamente compatíveis com o esquema do qual eles serão instâncias, bem como verificar se o mesmo é um documento XML bem formado.

### Object Processor

O *Object Processor* é utilizado quando os dados são armazenados usando o Tamino. É responsável por enviar os dados que devem ser armazenados para seus respectivos repositórios, ressaltando-se que o Tamino permite tanto armazenamento no formato XML como no modelo relacional.

### Object Composer

O *Object Composer* é acionado quando alguma informação deve ser retornada. Usando as regras de armazenamento e retorno definidas no *Data Map*, o *Objeto Composer* monta as informações requeridas, retornando-as como um documento XML. Como exemplo simples, ele retorna um objeto XML a partir de dados já armazenados como XML no Tamino, porém, em casos mais complexos, é necessário comunicar-se com outros componentes (como *X-Node*, que será apresentado a seguir) para construir o objeto XML a partir de fontes não XML.

### XML Query Interpreter

Responsável por resolver as consultas que são realizadas no Tamino, interagindo com o *Object Composer* e retornando o objeto XML de acordo com o que está armazenado na base consultada.

### 3.1.2. X-Node

Um outro componente, extremamente importante na comunicação do Tamino com bases externas, é o *X-Node*. Ele é o componente responsável por integrar os dados que estejam armazenados fora do Tamino, através do mapeamento dos mesmos em XML *Schemas* específicos. Ele permite o mapeamento de diversos formatos de dados, tais como bases heterogêneas e sistema de arquivos. A Figura 3.2 representa a comunicação do *X-Node* com as fontes externas e sua comunicação com o componente *X-Machine*.

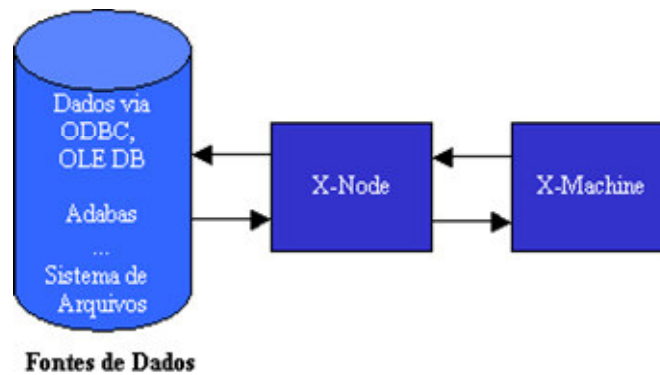


Figura 3.2 - Comunicação entre o Tamino e as bases externas através do X-Node

### 3.1.3. Data Map

Componente considerado a base de conhecimento do Tamino. Ele contém os XML *Schemas* e as informações referentes ao armazenamento e à indexação dos objetos XML. Visto de um nível mais alto do desenvolvimento da aplicação, é no *Data Map* onde ficam as coleções com seus respectivos esquemas criados, como é destacado na Figura 3.3.

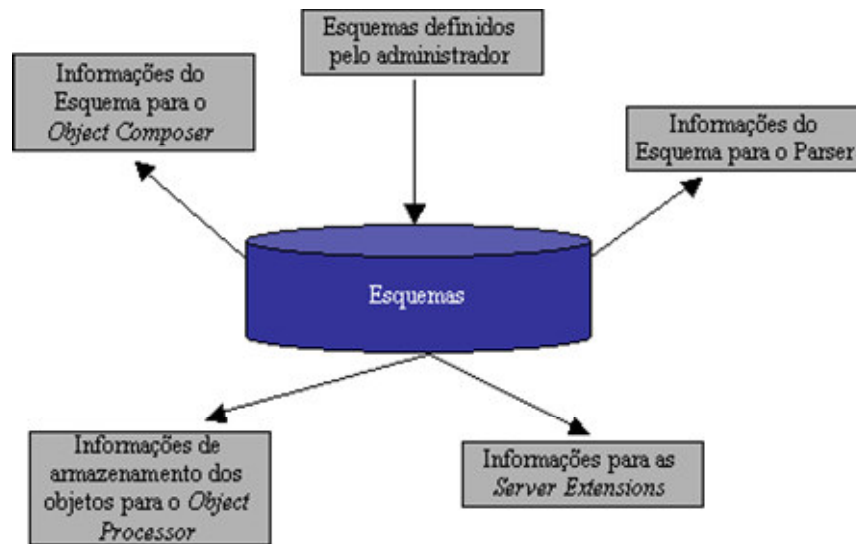


Figura 3.3 - Informações que são gerenciadas no Data Map

### 3.1.4. Server Extensions

SXT (*Server Extensions*) [Tamino 2003] são utilizadas para estender as funcionalidades oferecidas pelo Tamino XML Server (Figura 3.4), adicionando uma lógica definida pelo usuário [Softwareag 2003] à aplicação. Estas podem ser chamadas em linguagens de consultas, gatilhos e funções de mapeamento.

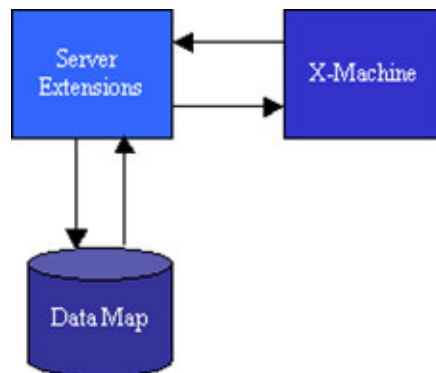


Figura 3.4 - Funcionamento da Server Extension no Tamino

As *Server Extensions* podem ser desenvolvidas em diversas linguagens, tais como C++, Visual Basic e Java, sendo esta última a linguagem adotada no presente trabalho, visto que o restante do desenvolvimento do protótipo também é realizado nesta linguagem.

O desenvolvimento de *Server Extensions* envolve algumas particularidades que devem ser consideradas quando da implementação das mesmas. A sua estrutura consiste, além do arquivo com o código fonte da *Server Extension* desenvolvida, de um arquivo de instalação que será abordado posteriormente nesta seção.

Antes de iniciar a implementação das *Server Extensions*, deve-se analisar a melhor forma a partir da qual elas podem ser úteis na resolução do sistema proposto. As *Server Extensions* apresentam uma série de tipos de funções que devem ser executadas de acordo com o contexto nos quais elas são utilizadas.

A seguir são listados os tipos existentes de funções oferecidas de acordo com o contexto da execução:

- Funções de Consulta (*Query Functions*) – são explicitamente chamadas a partir de consultas *X-Query*;
- Funções de Mapeamento (*Mapping Functions*) – são usadas para armazenar, retornar ou excluir documentos, sendo chamadas pelo Tamino quando um documento relacionado ao *XML Schema* que utiliza a *Server Extension* é inserido, consultado ou excluído;
- Funções de Gatilho (*Trigger Functions*) – utilizada para executar funções onde o armazenamento ou a exclusão de um documento relacionado independe da forma de mapeamento;
- Funções Iniciais (*Initial Functions*) – permite executar funções de inicialização antes de qualquer função de mapeamento, gatilho ou consulta;
- Funções de Evento do Servidor (*Server Event Functions*) – usadas para garantir a consistências de transações.

No caso do presente trabalho, são utilizadas as funções de mapeamento da *Server Extension*, mais precisamente as funções do tipo *Map-in*, executadas quando ocorre uma inserção de um objeto XML em um determinado esquema, e *Map-out*, executada quando uma consulta é efetuada pelo usuário.

Quando criada, a *Server Extension* gera um arquivo chamado *install.xml*. Este arquivo é responsável por guardar informações sobre os parâmetros da *Server Extension*, dentre os quais pode-se destacar:

- *Name* – o nome atribuído à *Server Extension*, a partir do qual esta pode ser referenciada pela aplicação;
- *Infrastructure* – linguagem na qual a *Server Extension* foi desenvolvida;
- *Function* – descrição de cada função criada, referenciando também o nome e tipo de mapeamento da mesma;
- *Parameter* – definição do parâmetro da função, referenciando seu nome e o tipo (baseado na definição de tipos do *Tamino Schema Definition*) [Tamino 2003].

Para que a *Server Extension* funcione corretamente, qualquer mudança estrutural feita no código fonte deve ser refletida no arquivo de instalação, como por exemplo, uma nova função de mapeamento, ou alguma alteração no nome da classe.

No Capítulo 4 será detalhada a implementação da *Server Extension* desenvolvida no presente trabalho.

### **3.2. Definindo e armazenando dados XML no Tamino**

Esta seção aborda a utilização do Tamino no desenvolvimento de uma aplicação com acesso a SGBD, mostrando o processo desde a representação dos dados até o armazenamento físico dos mesmos na base.

Alguns conceitos que divergem dos SGBD tradicionais serão abordados à medida que as etapas de desenvolvimento da aplicação forem sendo apresentadas.

O conceito inicial, básico para o armazenamento dos XML *Schemas* no Tamino, é a definição de coleções. Uma coleção é a estrutura a partir da qual os XML *Schemas* relativos a um mesmo domínio de conhecimento devem ser referenciados na base de dados. Ou seja, dentro de uma mesma base é possível inserir esquemas relacionados aos mais diversos assuntos, porém, é interessante manter cada esquema armazenado na sua estrutura correspondente, no caso, na respectiva coleção. Assim, já é possível perceber a primeira mudança em relação ao modelo relacional, no qual temos tabelas e bases de dados. No modelo nativo, além da base, temos mais uma estrutura (coleção) que é representada antes de se acessar a fonte de dados.

O primeiro passo necessário para trabalhar com o Tamino é criar ou conectar-se à base onde os dados serão dispostos. A segunda etapa é vincular o XML *Schema* criado a uma coleção existente na base de dados à qual o usuário está conectado. No caso da coleção não estar previamente criada na base, o Tamino automaticamente cria a nova coleção e vincula a mesma ao XML *Schema* correspondente. É através das coleções e das bases que os usuários poderão acessar o conteúdo dos XML *Schemas* desenvolvidos.

Comparativamente, pode-se dizer que uma coleção seria o equivalente a uma base de dados no modelo relacional (Figura 3.5), assim como os XML *Schemas* (*Doctypes*) representariam as tabelas no mesmo [Tamino 2003]. Evidentemente que é uma comparação com o intuito apenas de relacionar os dois modelos, pois na prática, a forma como os dados são armazenados e recuperados apresentam diferenças consideráveis.

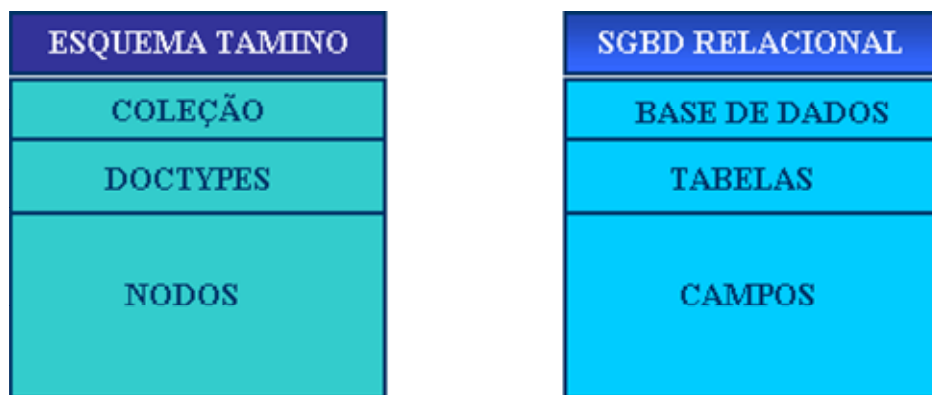


Figura 3.5 – Esquema Tamino x SGBD Relacional

Cada elemento do XML *Schema* será tratado como *nodo* dentro da estrutura do Tamino. Essa notação segue a notação XML de tratar sua estrutura como uma árvore hierárquica, chamando seus elementos como “raiz” e “nodos”.

Para identificar os esquemas em sua coleção, o Tamino utiliza um elemento chamado *Doctype*, sendo que este deve, obrigatoriamente, ter o mesmo nome do elemento raiz do XML *Schema*. É através dos *Doctypes* que será possível consultar, atualizar e excluir instâncias de um determinado esquema.

A sintaxe do XML *Schema* armazenado no Tamino segue os padrões recomendados pela W3C, bem como as instâncias dos esquemas, que são documentos



XML que precisam ser, além de bem formados, válidos em relação ao XML *Schema* ao qual serão inseridos como instâncias.

O Tamino suporta uma série de possibilidades diferentes de consulta às instâncias de seus esquemas. Os dados armazenados podem ser consultados via alguma linguagem de consulta, como a *XQuery*, que se baseia na especificação da linguagem *XQuery* definida pela W3C ou a *X-Query*, que é semelhante à linguagem XQL e ao *XPath* [Softwareag 2003]. Além dessas duas linguagens, é possível acessar os dados através da utilização de alguma das API oferecidas pelo Tamino, permitindo que se manipule o conteúdo XML via DOM ou SAX, dependendo da biblioteca utilizada.

Para criar aplicações no Tamino deve-se considerar, inicialmente, que o desenvolvimento de aplicações cujo armazenamento dos dados é XML Nativo envolve alguns passos que diferem das aplicações em SGBD mais tradicionais. Considerando organizações nas quais as informações são armazenadas de forma mista, ou seja, mantendo dados em suas bases relacionais, definindo apenas uma parte das informações armazenadas de forma nativa, temos uma estrutura mais complexa ainda.

Neste cenário, é necessário seguir algumas regras para evitar falhas no projeto do sistema. A seguir são apresentados os passos para o desenvolvimento de uma aplicação em ambiente de armazenamento nativo, seguindo de um exemplo de aplicação do mesmo [Tamino 2003]:

1. Inicialmente, é recomendável que os dados sejam descritos usando XML *Schemas*. Até que os XML *Schemas* sejam finalizados, o Tamino aceita DTDs;
2. Definido o XML *Schema*, que representa a estrutura dos dados a serem armazenados, o próximo passo é definir as informações relevantes ao armazenamento no SGBD Nativo. O Tamino provê uma linguagem de esquema que deve ser utilizada para descrever as informações relacionadas ao armazenamento do XML *Schema* no NXD. Neste ponto são definidas informações como o nome do esquema, a coleção na qual o mesmo será inserido, além da forma de armazenamento e recuperação dos elementos do documento;

3. Incorporar o XML *Schema* (com as especificações do Tamino incluídas) a uma base de dados;
4. Uma vez que o esquema é inserido no Tamino, o próximo passo é armazenar instâncias do esquema. O Tamino armazena e recupera as informações de acordo com as regras especificadas no esquema.

### 3.2.1. Descrevendo os dados

Em termos gerais, o processo de descrição dos dados pode ser visto como um exercício de modelagem [XMLDB 2004]. Entretanto, a descrição dos dados é uma meta-estrutura que descreve estruturas existentes em formato XML. A definição dos dados que irá resultar no XML *Schema* ocorre fora do Tamino e é pré-requisito para usar o *Tamino Schema Editor* (que será apresentado na seção 3.3).

Por exemplo, assumindo que se deseja usar o Tamino para armazenar registros de pacientes de um hospital, deve-se criar uma nova base de dados de pacientes em formato XML. Além dos dados do paciente, é necessário saber as informações referentes aos médicos e funcionários envolvidos no tratamento médico do mesmo.

Neste exemplo será considerado que já existem informações a respeito da equipe médica e de funcionários do hospital disponíveis em uma base de dados relacional externa e que é conveniente mantê-la desta forma.

Um típico registro de pacientes pode ter a estrutura descrita no Quadro 3.1:

**Quadro 3.1 – Estrutura de registro de paciente**

Name: Jones, Kevin Martin
Sex: Male
Born: 28.Feb.1950
Address: 16 Mill Lane, Bradford, BRA160ML
Diagnosis: Prolaps L4-L5
Doctor: Dr. Creedy
Pager: 404

A tabela na base relacional contém as informações da equipe médica de funcionários do hospital, e foi criada segundo o formato mostrado no Quadro 3.2.

**Quadro 3.2 – Descrição da tabela de equipe médica**

```
create table "Employees"
(
  "Employee ID"   INTEGER PRIMARY KEY,
  "Last Name"     VARCHAR(50),
  "First Name"    VARCHAR(30),
  "City"          VARCHAR(40),
  "Home Phone"    CHAR(16),
  "Mobile"        CHAR(16),
  "Pager"         CHAR(7),
)
```

Sendo que a tabela foi alimentada com as seguintes informações:

**Tabela 3.1 – Representação da tabela com os médicos cadastrados**

Employee ID	Last Name	First Name	City	Home Phone	Mobile	Pager
1	Allan	Peter	London	01454678	120957483	101
2	Atkins	Roland	London	01567383	120756348	202
3	Bader	Alfred	London	01274634	120645893	303
4	Creedy	Simon	London	01346738	120729645	404

E finalmente, é apresentada a representação da DTD correspondente ao XML anteriormente apresentado no Quadro 3.3.

**Quadro 3.3 – DTD representando as informações de médico e paciente**

```

<!ELEMENT patient (p-name?, sex, born?, p-address?,
                    diagnosis, doctor)>
<!ELEMENT p-name (p-surname, p-firstname, p-middlename*)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT born (#PCDATA)>
<!ELEMENT p-address (p-street?, p-housenumber?, p-city?,
                     country?, p-postcode?)>
<!ELEMENT diagnosis (#PCDATA)>
<!ELEMENT doctor (employee-id, s-surname, s-firstname, s-city?,
                  s-phonehome?, s-phonemobile?, pager)>
<!ELEMENT employee-id (#PCDATA)>
<!ELEMENT p-surname (#PCDATA)>
<!ELEMENT p-firstname (#PCDATA)>
<!ELEMENT p-middlename (#PCDATA)>
<!ELEMENT p-street (#PCDATA)>
<!ELEMENT p-housenumber (#PCDATA)>
<!ELEMENT p-city (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT p-postcode (#PCDATA)>
<!ELEMENT s-surname (#PCDATA)>
<!ELEMENT s-firstname (#PCDATA)>
<!ELEMENT s-city (#PCDATA)>
<!ELEMENT s-phonehome (#PCDATA)>
<!ELEMENT s-phonemobile (#PCDATA)>
<!ELEMENT pager (#PCDATA)>
<!ATTLIST patient
    id      ID      #IMPLIED>

```

É importante observar os indicadores de ocorrência dos elementos, bem como a extensibilidade dos modelos satisfeitos. Através dessa definição, é possível “imaginar” o modelo de pacientes sendo estendido pelas sub-árvores, descrevendo as terapias pré-escritas, ou os resultados das medicações. Outro ponto relevante é a distinção entre a DTD dos detalhes do nome do paciente e os doutores (por exemplo, “*p-surname*” com “*s-surname*”), no qual o mesmo modelo é satisfeito, porém um é armazenado como XML no Tamino e o outro numa base relacional externa. Assim, a solução adotada foi representar com diferentes nomes as informações que representam os mesmos dados por ser mais simples do que introduzir *namespaces*. Porém, dependendo do caso, o uso de *namespaces* pode ser aconselhável.

### 3.2.2. Definindo o Esquema Tamino

Definir um esquema Tamino significa usar a linguagem de definição de esquema do Tamino para expressar a estrutura dos dados, especificando onde os dados são armazenados e como podem ser recuperados. Se os dados já estão escritos em um XML *Schema* ou DTD é possível utilizar as vantagens do editor de esquemas do Tamino.

Neste exemplo, os detalhes pessoais do pacientes são armazenados no Tamino de forma nativa, enquanto que os detalhes pessoais dos médicos em uma tabela externa existente.

#### A linguagem de esquemas do Tamino

O Tamino *Schema* é implementado como um esquema XML (no presente exemplo, uma DTD). Ele usa os tipos coleções de elementos, *Doctypes* e Nodo cuja representatividade no modelo é apresentada a seguir:

- Coleção representa uma base de dados e pode conter múltiplos *Doctypes*;
- *Doctype* representa um repositório de itens de informação e pode ser comparado a uma tabela em uma base de dados relacional;
- Nodo representa um item de informação. Elementos nodo podem ser aninhados, além de permitir descrever mais profundamente estruturas de dados em uma sub-árvore do esquema.

Todas as definições relevantes para o armazenamento e recuperação das informações são expressas como atributos desses elementos. Algumas regras na definição dos esquemas são importantes para que as bases sejam criadas de forma a representar corretamente os dados desejados:

- Cada coleção, *Doctype* e nodo são identificados unicamente pelos atributos nome e chave.
- Relacionamentos pai-filho são armazenados (gravados) em um atributo Pai, sendo a referência no elemento filho feita através do valor do atributo chave (*key*) do elemento Pai;
- Os índices dos nodos são indicados pelo atributo *Obj-Type*.

Quando um item de informação é armazenado, este está especificado pelo atributo *Map-Type*. Por exemplo, dados armazenados no Tamino de forma nativa são indicados

pelo valor “*Native*”, enquanto que os armazenados em uma base de dados relacional são indicados pelo valor “*Map SubTreeSQL*”.

A conexão com as bases de dados externas é efetuada através de uma série de atributos, os quais permitem definir diversos parâmetros relevantes ao processo de integração, como por exemplo, definir o nome da tabela a ser mapeada, a fonte de dados via ODBC (*Open Database Connectivity*), o nome da chave primária a partir da qual os dados são indexados, entre outros.

No presente exemplo todas as informações, com exceção dos registros dos médicos, são armazenadas de forma nativa no Tamino. No Quadro 3.4 é apresentada uma parte do que seria a representação dos dados dos pacientes em um XML *Schema*, já com as definições necessárias para armazenamento no Tamino. Não é apresentada a representação completa do XML *Schema*, pois esta seria muito extensa e o objetivo do exemplo é apresentar apenas como os dados são armazenados e recuperados no Tamino.

**Quadro 3.4 – XML Schema representando as informações de Pacientes**

```
<Collection Name="Hospital" Key="HealthC01">
  <Doctype Name="patient" Key="Patient001"
    Options="READ INSERT UPDATE DELETE">

    <Node Name="patient"
      Key="Patient002"
      Obj-Type="SEQ"
      Parent="Patient001"
      Search-Type="No"
      Map-Type="Native"/>

    <Node Name="p-name"
      Key="Patient003"
      Obj-Type="SEQ"
      Parent="Patient002"
      Search-Type="No"
      Map-Type="No"/>

    <Node Name="p-surname"
      Key="Patient004"
      Obj-Type="PCDATA"
      Parent="Patient003"
      Search-Type="Text"
      Map-Type="No"/>

    <Node Name="p-firstname"
      Key="Patient005"
      Obj-Type="PCDATA"
      Parent="Patient003"
      Search-Type="Text"
      Map-Type="No"/>

    ....
  </Doctype>
</Collection>
```

Para facilitar a visualização dos elementos, foi omitido da definição do XML *Schema* o prefixo “*ino:*”, porém o mesmo deve ser usado sempre que um *Schema* for criado. A primeira definição do XML *Schema* apresentado é a coleção definida como *Hospital*, enquanto que o *Doctype* é chamado de *patient*. Assim, instâncias de *patient* podem ser lidas, inseridas, atualizadas ou excluídas.

As definições dos nodos seguem a árvore *patient*, lembrando que o relacionamento pai-filho no atributo pai é referenciado pelo atributo *Key* do elemento Pai. O atributo *Obj-Type="Seq"* indica que o nodo é pai de outros nodos.

As configurações dos atributos *Map-Type* e *Search-Type* podem surtir os seguintes efeitos sobre os dados:

- Definir *Map-Type* com o valor *Native* no nodo *Patient* indica que toda a sub-árvore será armazenada de forma nativa no Tamino. Este valor é herdado pelos nodos de níveis abaixo, sendo assim desnecessário que cada nodo tenha que definir explicitamente o valor *Native* para o atributo *Map-Type*;
- O valor *No* atribuído para *Search-Type* no nodo *Patient* indica que nenhuma indexação será processada. Assim como o *Map-Type*, os nodos filhos herdam a definição do pai para o atributo *Search-Type*, porém os nodos que contêm dados no formato texto (*p-surname* e *p-firstname*) apresentam o valor *Text*. Isto permite a recuperação do texto desses nodos [Tamino 2003].

### 3.2.3. Mapeando para Bases Relacionais

Para fazer o mapeamento de dados a serem armazenados e recuperados em uma base relacional é necessário definir o valor do atributo *Map-Type* atribuindo o valor “*SQLTable*” ou “*SqlColumn*” ao elemento (nodo) que se deseja mapear. Bases de dados relacionais fora do Tamino são acessadas via *X-Node* usando ODBC, e estas definições devem ser representadas nos atributos do elemento mapeado.

O Quadro 3.5 mostra os atributos que são usados no presente exemplo para realizar o mapeamento com a base relacional:

**Quadro 3.5 – Lista de atributos de mapeamento XML-Base Relacional**

<i>DataSource</i>	Nome da base de dados como definido nas configurações ODBC.
<i>SqlTable</i>	Nome da tabela no modelo Relacional
<i>SqlColumn</i>	Nome do campo (coluna) no modelo relacional
<i>SQLPrimaryKeys</i>	Lista de chaves utilizada para descrever as chaves primárias da tabela.

O modelo de conteúdo para representar as informações dos médicos no exemplo atual, já com as definições de mapeamento, é mostrado no Quadro 3.6.



**Quadro 3.6 – Representação e mapeamento das informações dos médicos**

```
<Node Name="doctor"
  Key="Patient016"
  Obj-Type="SEQ"
  Parent="Patient002"
  Search-Type="No"
  Map-Type="SqlTable"
  DataSource="NorthWind"
  SqlTable="Employees"
  SqlPrimaryKeys="\"Last Name\" \"First Name\"
                  \"City\" \"Home Phone\"
                  \"Móbile\" \"Pager\""/>

<Node Name="employee-id"
  Key="Patient017"
  Obj-Type="PCDATA"
  Parent="Patient016"
  Search-Type="standard"
  Map-Type="SqlColumn"
  Data-Type="INTEGER"
  SqlColumn="Employee ID"/>

<Node Name="s-surname"
  Key="Patient018"
  Obj-Type="PCDATA"
  Parent="Patient016"
  Search-Type="text"
  Map-Type="SqlColumn"
  Data-Type="VARCHAR"
  SqlColumn="Last Name"/>

<Node Name="s-firstname"
  ID="Patient019"
  Obj-Type="PCDATA"
  Parent="Patient016"
  Search-Type="text"
  Map-Type="SqlColumn"
  Data-Type="VARCHAR"
  SqlColumn="First Name"/>

<Node Name="city"
  ID="Patient019"
  Obj-Type="PCDATA"
  Parent="Patient016"
  Search-Type="text"
  Map-Type="SqlColumn"
  Data-Type="VARCHAR"
  SqlColumn="City"/>

<Node Name="s-phonehome"
  Key="Patient020"
```

```

        Obj-Type="PCDATA"
        Parent="Patient016"
        Search-Type="standard"
        Map-Type="SqlColumn"
        Data-Type="CHAR"
        SqlColumn="Home Phone"/>

<Node Name="s-phonemobile"
      Key="Patient021"
      Obj-Type="PCDATA"
      Parent="Patient016"
      Search-Type="standard"
      Map-Type="SqlColumn"
      Data-Type="CHAR"
      SqlColumn="Mobile"/>

<Node Name="pager"
      Key="Patient022"
      Obj-Type="PCDATA"
      Parent="Patient016"
      Search-Type="text"
      Map-Type="SqlColumn"
      Data-Type="VCHAR"
      SqlColumn="Pager"/>
....
</Doctype>
</Collection>

```

É importante salientar que os valores dos atributos *Map-Type*, *Search-Type* e *Data-Type* devem estar relacionados, pois o valor do atributo tem que ser compatível com o tipo de dado que será retornado da base, o mesmo acontecendo com o tipo da busca em relação ao campo mapeado.

### 3.2.4. Recuperando Informações no Tamino XML Server

O Tamino provê acesso ao conteúdo nativo nele armazenado através de diversas formas de consulta. Algumas interfaces e ferramentas são disponibilizadas para fazer essas consultas (seção 3.3), sendo que em grande parte delas é oferecido um ambiente para consultas através das linguagens XQuery e XQL (seção 2.4.1).

Através dessas linguagens, o usuário pode fazer inferências sobre o modelo de dados inserido, ficando a cargo do Tamino retornar o resultado desejado como um documento XML.

Algumas considerações devem ser feitas quanto à consulta em XML *Schema* que possui mapeamento para bases externas:

- Não é necessário inserir instâncias do XML *Schema*. As informações contidas nas bases mapeadas são buscadas no momento da consulta, retornado para o usuário o resultado da busca como se a mesma tivesse sido efetuada em uma base de dados XML, ou seja, em formato XML;
- Em caso de instâncias inseridas no Tamino, as mesmas são dispostas tanto no Tamino quanto na base de dados externa;
- Não há replicação no resultado da consulta. Os dados existentes como instâncias do Tamino e registros na base de dados externa são retornados ao usuário como uma instância única;
- Na versão atual do Tamino XML Server, versão 4.1.4.1, não é possível fazer consultas XQuery a *Schemas* com mapeamento para bases externas;
- O suporte aos SGBD que podem ser mapeados depende da licença do Tamino.

Tomando como base o exemplo anterior, é apresentada a seguir uma consulta simples, na qual se deseja retornar os dados do médico do paciente cujo sobrenome é Jones.

Como os dados do médico estão armazenados em uma base de dados relacional, a consulta não pode ser realizada através da XQuery. Assim sendo, a consulta será efetuada através da linguagem XQL da seguinte forma:

```
patient[/p-surname="Jones"]/doctor
```

Reiterando-se que a consulta pode ser processada utilizando uma ferramenta de consulta, uma API (seção 2.4.1), ou através de uma chamada direta através do protocolo HTTP:

```
http://servidor/tamino/nomeBase/coleção?_XQL=consulta
```

Considerando *xavante* como nome do servidor e *DBPatient* o nome da base, a consulta seria efetuada da seguinte forma:

```
http://xavante/tamino/DBPatient/Hospital?_XQL=patient[/p-surname="Jones"]/doctor
```

O Quadro 3.7 apresenta o resultado da consulta em formato XML.

**Quadro 3.7 – Documento XML com os dados do médico que atende o paciente.**

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ino:response
xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
xmlns:xql="http://metalab.unc.edu/xql/">
<xql:query>bascAc/jogador/nome</xql:query>
<ino:message ino:returnValue="0">
  <ino:messageline>XQL Request processing</ino:messageline>
</ino:message>
<xql:result>
  <doctor>
    <employee-id>3</employee-id>
    <s-surname>Bader</s-surname>
    <s-firstname>Alfred</s-firstname>
    <pager>303</pager>
  </doctor>
</xql:result>
<ino:message ino:returnValue="0">
  <ino:messageline>XQL Request processed</ino:messageline>
</ino:message>
</ino:response>
```

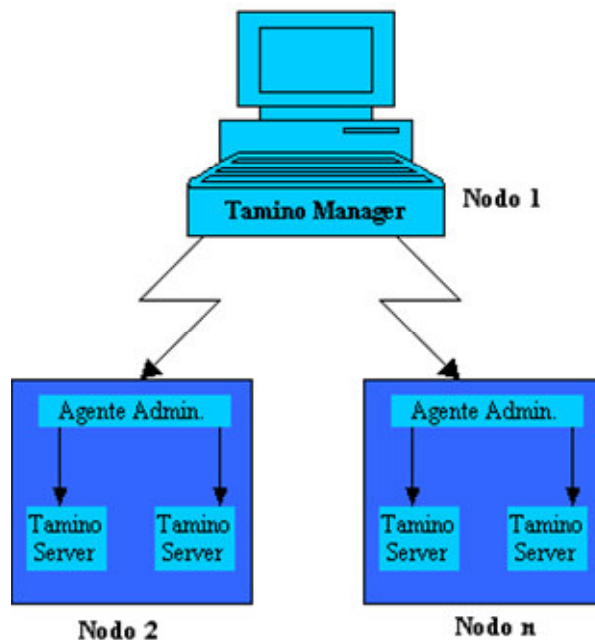
### 3.3. FERRAMENTAS

O desenvolvimento de sistemas utilizando o Tamino XML Server envolve uma série de ferramentas que auxiliam desde a construção de XML *Schemas* até o gerenciamento das diversas bases de dados criadas.

Nesta seção é feito um breve resumo das principais ferramentas utilizadas no presente projeto.

#### 3.3.1. Tamino Manager

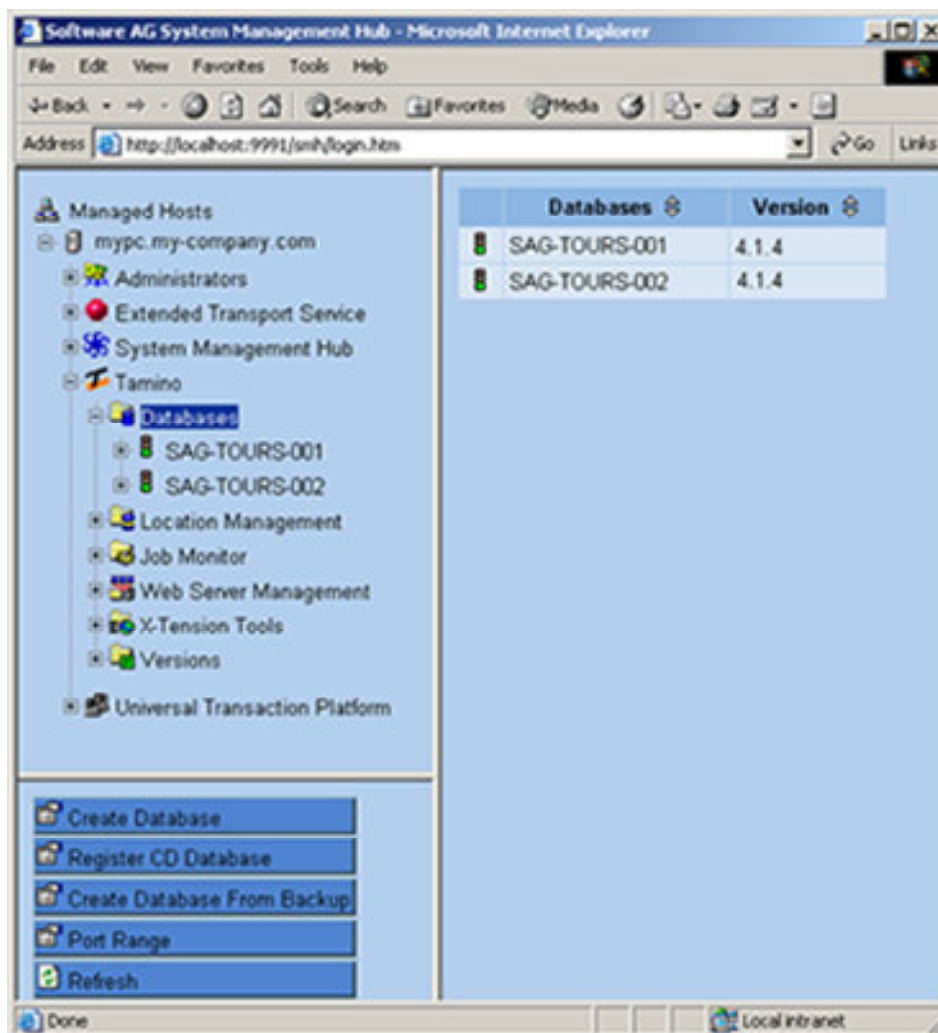
O *Tamino Manager* é a ferramenta central de administração do Tamino para aplicações cliente-servidor. Esta ferramenta roda no navegador padrão do cliente sendo a ferramenta na qual se centraliza grande parte das operações necessárias para a utilização do Tamino XML Server. Funções básicas de administração, tais como criar, iniciar, parar e excluir bases de dados são disponibilizadas pela ferramenta, além de permitir o gerenciamento completo dos recursos utilizados pelas bases, entre outras funcionalidades de administração. É possível, através do *Tamino Manager*, gerenciar uma série de servidores diferentes como mostra a Figura 3.6.



**Figura 3.6 – Gerenciamento de servidores pelo *Tamino Manager***

Como o objetivo é mostrar as funcionalidades relevantes ao trabalho, o foco será na administração do *Data Map* e no controle das *Server Extensions*.

Para criar e gerenciar as diferentes bases de dados é utilizada a interface “*System Management Hub Window*” do *Tamino Manager*, mostrado na Figura 3.7. Através dela o usuário pode gerenciar cada base de dados criada, suas coleções e respectivos *Schemas* associados a cada coleção.



**Figura 3.7 - Interface de gerenciamento das bases do Tamino Manager**

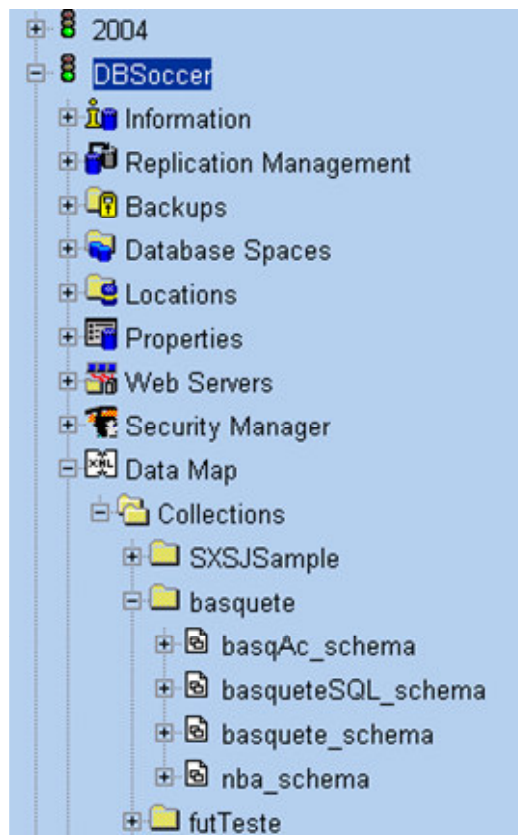
A interface é dividida em três partes, a principal é a superior esquerda, onde estão os diversos itens que podem ser acessados através do *Tamino Manager*. Uma vez selecionado um determinado item nesta parte, as outras duas passam a se comportar de acordo com o item selecionado. Na figura acima é selecionada a pasta *Databases*. Assim sendo, a parte direita da interface apresenta a lista de bases de dados existentes no servidor com o seu estado atual (o semáforo verde indica que a base está “no ar”, amarelo indica que a mesma está sendo iniciada e vermelho a base está desativada), além da versão do Tamino na qual foram criadas as bases, no caso, Tamino 4.1.4. A parte inferior esquerda da interface apresenta operações que podem ser realizadas no item selecionado. No caso das bases de dados, é possível criar, registrar e configurar porta, entre outras funcionalidades.

Um item básico, porém extremamente importante do Tamino, quando se deseja trabalhar com armazenamento nativo, é entender como o *Tamino Manager* organiza as coleções e esquemas dentro da sua estrutura.

Como apresentado na seção 3.1.3, o Tamino dispõe seus dados no componente *Data Map*, sendo que este é criado para cada nova base de dados inserida. Cada *Data Map* apresenta uma pasta chamada *Collections*, a partir da qual todas as coleções existentes na base são dispostas.

Algumas coleções são criadas automaticamente pelo Tamino sempre que uma nova base de dados é definida. Estas coleções são precedidas do prefixo *ino:* e são utilizadas no funcionamento interno do Tamino, portanto não devem ser utilizadas para referenciar novos esquemas.

Conforme os XML *Schemas* são criados e associados às bases de dados, suas coleções correspondentes são automaticamente criadas, passando a se tornar acessíveis através do *Tamino Manager*. Como uma coleção pode ter uma série de esquemas associados a ela, o *Tamino Manager* cria uma pasta para representar cada coleção, sendo que esta contém todos os XML *Schemas* a ela associados. A Figura 3.8 apresenta a forma como a estrutura de armazenamento apresentada anteriormente é exibida na interface de gerenciamento do *Tamino Manager*.



**Figura 3.8 - Disposição dos dados da base no Tamino Manager**

Especificamente no que tange ao gerenciamento das bases criadas, o *Tamino Manager* permite ao usuário gerenciar diversas definições relevantes tais como o tamanho de cada base, o número de sessões que já foram abertas na base de dados, a data de criação, controlar o sistema de replicação dos dados, efetuar backup do conteúdo das bases entre outras definições.

O gerenciamento de *Server Extensions* é outra tarefa executada pelo *Tamino Manager*. Através dele é possível instalar novas *Server Extensions*, além de permitir configurar diversos parâmetros relacionados às mesmas. É através do *Tamino Manager* que o desenvolvedor pode configurar o sistema de “debug” das *Server Extensions*. Essa funcionalidade é inicializada quando se aciona a opção “X-Tension trace switch” que habilita a função interna “SXS-Trace” para gravar uma mensagem no SXS-Trace do Tamino, que é a função oferecida para viabilizar o acompanhamento dos passos de execução da *Server Extension*. A SXS-Trace gera um arquivo de log no qual é possível gravar informações a partir da *Server Extension* desenvolvida.



Através do *Tamino Manager* é possível verificar todas as *Server Extensions* com suas respectivas funções, *classpath*, data de instalação, entre outros parâmetros, além de permitir alterar e desinstalar as mesmas.

Existe uma série de outras funcionalidades oferecidas pelo *Tamino Manager* para gerenciamento dos dados provenientes do Tamino. Maiores informações sobre o uso dessa ferramenta podem ser encontradas na documentação do Tamino [Tamino 2003].

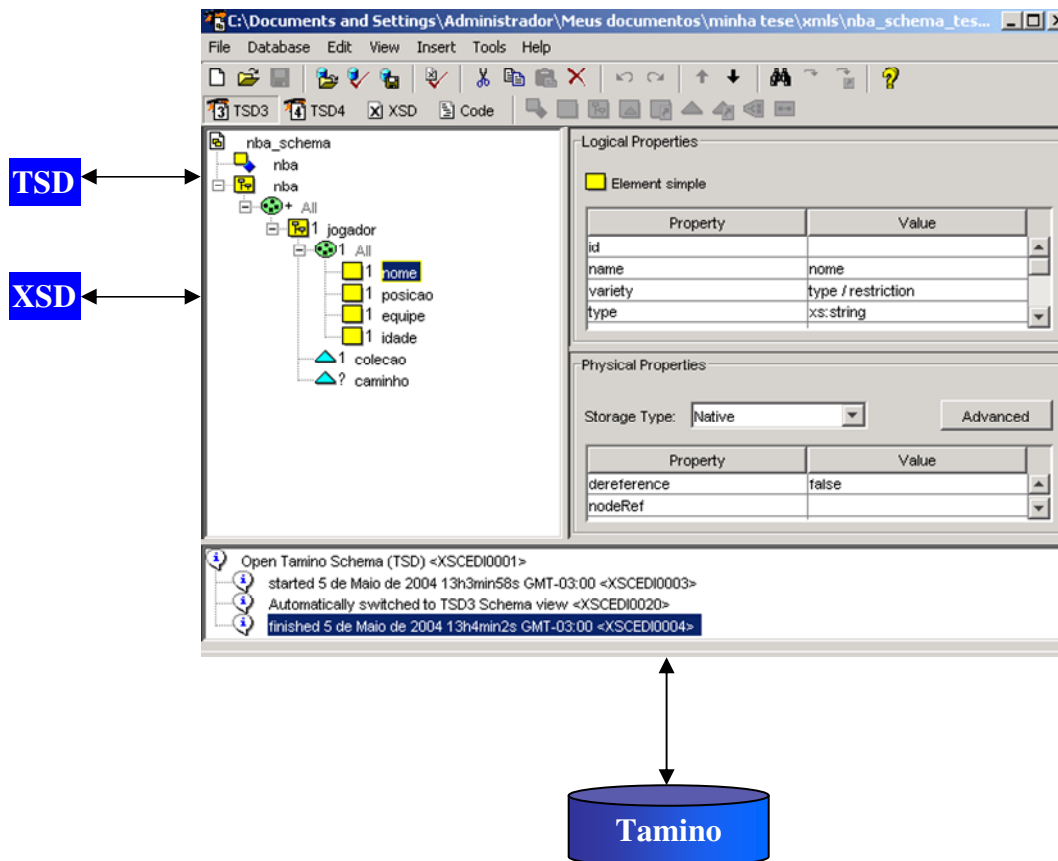
### **3.3.2. Tamino Schema Editor**

*Tamino Schema Editor* é uma ferramenta oferecida no pacote Tamino XML Server com o objetivo de auxiliar o desenvolvimento de XML *Schemas* que venham a ser armazenados no Tamino.

Usando o *Tamino Schema Editor*, o desenvolvedor pode ter as seguintes vantagens:

- Uma interface gráfica que auxilia o usuário a montar o XML *Schema* sem que seja necessário ter conhecimento da sintaxe de desenvolvimento do mesmo (embora seja recomendável);
- Oferece caixas de diálogo para especificar alguns parâmetros mais complexos na montagem do esquema;
- Possibilidade de alterar o código, verificando sua consistência em tempo de desenvolvimento;
- Interface e mapeamento com as bases externas, oferecendo um ambiente gráfico que permite a validação das conexões no próprio editor;
- Conexão com o Tamino, onde os esquemas desenvolvidos podem ser associados à base de dados correspondente na própria ferramenta;
- Apresentação de erros de formatação e de validação dos esquemas, bem como de falhas no armazenamento, como duplicação de esquemas, problemas na comunicação com determinada base externa, entre outras.

A Figura 3.9 apresenta o ambiente de desenvolvimento do *Tamino Schema Editor* [Tamino 2003].



**Figura 3.9 – Ambiente de desenvolvimento do Tamino Schema Editor**

Além do *Tamino Schema Editor*, outras ferramentas podem ser utilizadas no desenvolvimento de XML Schemas para aplicações Tamino, dentre os quais pode-se destacar a ferramenta *xmlspy*, um editor XML desenvolvido pela Antova que permite trabalhar com diversos padrões relacionados, tais como XSL/XSLT [XSLT 2002], SOAP [W3C/SOAP 2003], WSDL (*Web Services Description Language*) [W3Schools/WSDL 2002] e tem suporte ao Tamino, permitindo que XML Schemas definidos no editor sejam incorporados a uma base no Tamino usando a própria ferramenta [Xmlspy 2003].

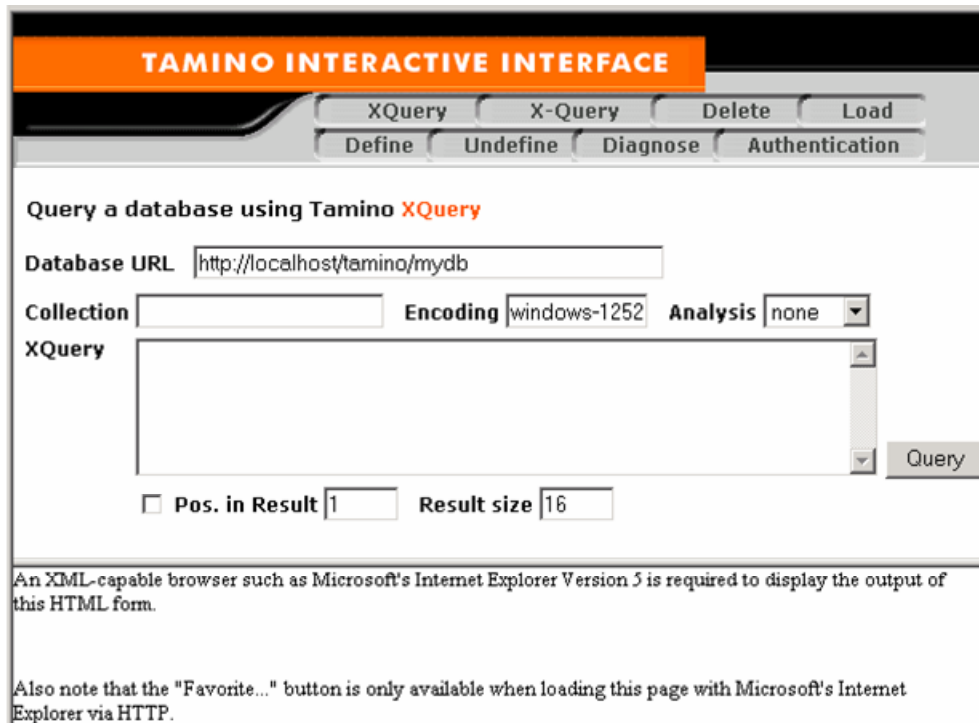
### 3.3.3. Tamino Interactive Interface

*Tamino Interactive Interface* é uma interface de programação oferecida pelo Tamino que pode ser executada a partir de um navegador *web*. Esta ferramenta tem como objetivo permitir que sejam executadas as seguintes funções [Tamino 2003]:

- Consultar uma base de dados usando a linguagem XQuery, como proposto pela W3C;
- Realizar consultas utilizando a Tamino X-Query;

- Excluir instâncias XML de uma base de dados;
- Ler documentos XML e não-XML a partir de uma determinada base de dados;
- Definir ou atualizar coleções e esquemas;
- Desfazer a definição de um esquema, coleção ou *doctype*;
- Consultar informações sobre uma determinada base;
- Autenticar usuários.

Todas as requisições feitas ao servidor são retornadas em formato de documentos XML. Para exibir esses documentos, o *Tamino Interactive Interface* deve ser executado a partir de um navegador que tenha suporte a XML. A Figura 3.10 apresenta a *interface* oferecida para consultas através da linguagem XQuery.



**Figura 3.10 - Tela de consulta do Tamino Interactive Interface**

Esta ferramenta é bastante útil, pois oferece através de uma interface simples, todas as funcionalidades básicas desejáveis para a definição e o gerenciamento de dados no Tamino.

### **3.3.4. Tamino X-Plorer**

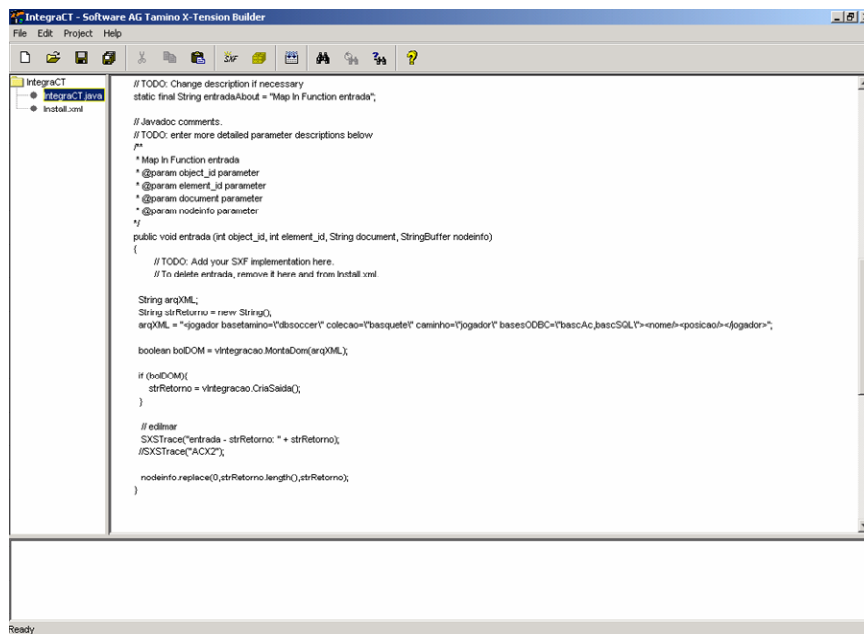
*Tamino X-Plorer* é uma interface gráfica para visualizar o conteúdo das bases de dados em uma árvore de navegação, permitindo assim, que se acesse e manipule o conteúdo das mesmas.

Através do *Tamino X-Plorer* é possível manipular as coleções, esquemas e suas instâncias sem que seja necessário fazer nenhum esforço de programação. É uma alternativa ao *Tamino Iterative Interface* anteriormente apresentado. O *X-Plorer* permite analisar a estrutura dos esquemas armazenados, inserir de forma interativa novas instâncias de esquemas, visualizar os documentos XML armazenados, entre outras opções.

### **3.3.5. X-Tension Builder**

*X-Tension Builder* é uma ferramenta que provê suporte ao desenvolvimento de *Server Extensions* [Tamino 2003], mostrado na Figura 3.11. Ela trabalha através de projetos que engloba toda a estrutura de desenvolvimento de *Server Extensions* apresentada anteriormente (seção 3.1.4), ou seja, o arquivo de instalação e o código-fonte da *Server Extension*, além da referência a outras classes que possam vir a ser utilizadas no desenvolvimento do trabalho.

O *X-Tension Builder* permite ao desenvolvedor criar, abrir e alterar o conteúdo das *Server Extensions*. Além destas tarefas mais básicas, a ferramenta oferece uma interface para inserir de forma automatizada as funções de acordo com os tipos oferecidos pela *Server Extension*. Esta interface facilita o trabalho do desenvolvedor, pois os parâmetros obrigatórios de cada tipo de função são inseridos automaticamente no código fonte, da mesma forma que o arquivo XML de instalação é atualizado para que referencie a função inserida. Além destas funcionalidades, outras duas extremamente relevantes são a possibilidade de compilar e gerar as *Server Extensions*.



**Figura 3.11 - Ambiente de desenvolvimento do X-Tension Builder**

Apesar de todas as funcionalidades citadas acima, o *X-Tension Builder* apresenta algumas dificuldades ao usuário. A principal delas é o ambiente limitado no qual o código fonte é disposto. Devido a essa limitação, é recomendável que se utilize o *X-Tension Builder* para criar a base da aplicação e para gerar a *Server Extension*, utilizando um editor próprio da linguagem no desenvolvimento da lógica do programa.

Maiores informações sobre as ferramentas e funcionalidades oferecidas pelo Tamino podem ser encontradas na documentação do mesmo [Tamino 2003].

Além do Tamino existe uma série de outros SGBD de armazenamento nativo. Grande parte deles seguindo estrutura de armazenamento e recuperação de dados semelhante à apresentada neste capítulo. A utilização de coleções para o armazenamento dos XML *Schemas*, bem como a linguagem XQuery para consultas, são algumas dessas características.

A seguir são listados alguns dos principais NXD do mercado:

- SQL/XML-IMDB [Quilogic 2003];
- *Sonic XML Server* [Sonic 2003];
- *TigerLogic XML Data Management Server (XDMS)* [Rainingdata 2003];
- *XStreamDB Native XML Database* [Bluestram 2003];

- Xindice [Apache 2003].

### **3.4. Considerações finais**

O Tamino XML Server tem um papel de grande importância no desenvolvimento do presente trabalho. É nele que a estrutura de representação dos dados integrados é armazenada, além das consultas serem realizadas diretamente no ambiente oferecido pelo mesmo.

## **4. DESENVOLVIMENTO DO SIDATA**

Este capítulo apresenta uma arquitetura para integração de dados utilizando bases de dados nativas. O desenvolvimento de um protótipo que implementa a arquitetura proposta é também abordado.

### **4.1. Arquitetura do Sistema**

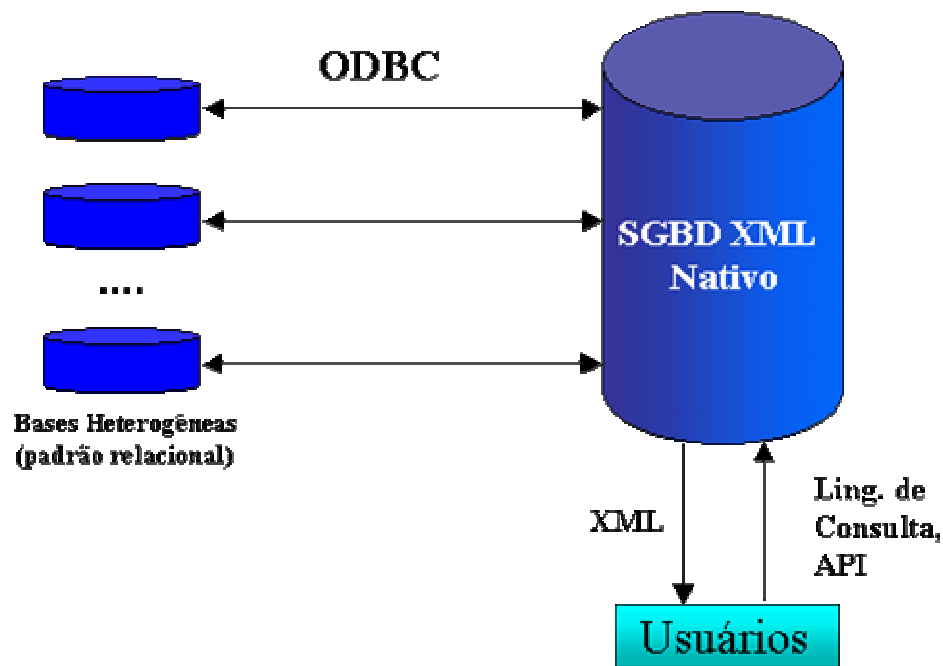
A presente proposta de arquitetura visa integrar diversas bases de dados que apresentam características heterogêneas, tornando-as acessíveis através de uma interface única, por meio da qual é possível consulta-las como se fossem uma única fonte de dados global.

A arquitetura desenvolvida considera o cenário no qual todas bases de dados seguem o padrão relacional, porém a mesma pode ser estendida para incorporar outros SGBD, tal como Orientado a Objeto.

Para armazenar a representação dos dados integrados é utilizado um NXD, no caso o Tamino XML Server. O objetivo é aproveitar as facilidades oferecidas pelo Tamino para automatizar algumas etapas do processo de integração das informações. Além de trabalhar com a estrutura oferecida pela XML, que é uma linguagem em expansão na *web* e tem um grande número de linguagens derivadas utilizadas em diversas propostas de sistemas de integração [Arantes et al. 2003; Sousa 2003].

O Tamino oferece a possibilidade de trabalhar com XML de uma forma abrangente, visto que toda a estrutura de armazenamento e recuperação das informações pode ser efetuada através de padrões relacionados a XML [Tamino 2003].

Com a estrutura oferecida pelo Tamino, analisando as funcionalidades oferecidas pelo mesmo, chegou-se a um modelo inicial que consistiu na estrutura mostrada na Figura 4.1.



**Figura 4.1 - Modelo inicial da Arquitetura do SIDATA**

Este modelo representa, de um nível mais alto de abstração, como ocorre a interação entre bases de dados, Tamino e o usuário.

A partir da requisição feita pelo usuário (através de uma API ou linguagem de consulta), o Tamino busca os dados que satisfaçam aquela consulta em cada uma das bases. Para isso, ele utiliza a sua funcionalidade de mapeamento que permite relacionar os XML *Schemas* armazenados pelo Tamino com bases de dados externas via ODBC.

Por fim, as informações são retornadas ao usuário como um documento XML. A partir deste cenário, deve-se analisar como é possível montar uma arquitetura na qual este sistema possa funcionar na prática, conhecendo as limitações do Tamino em viabilizá-lo e o que deve ser definido para suprir essas carências.

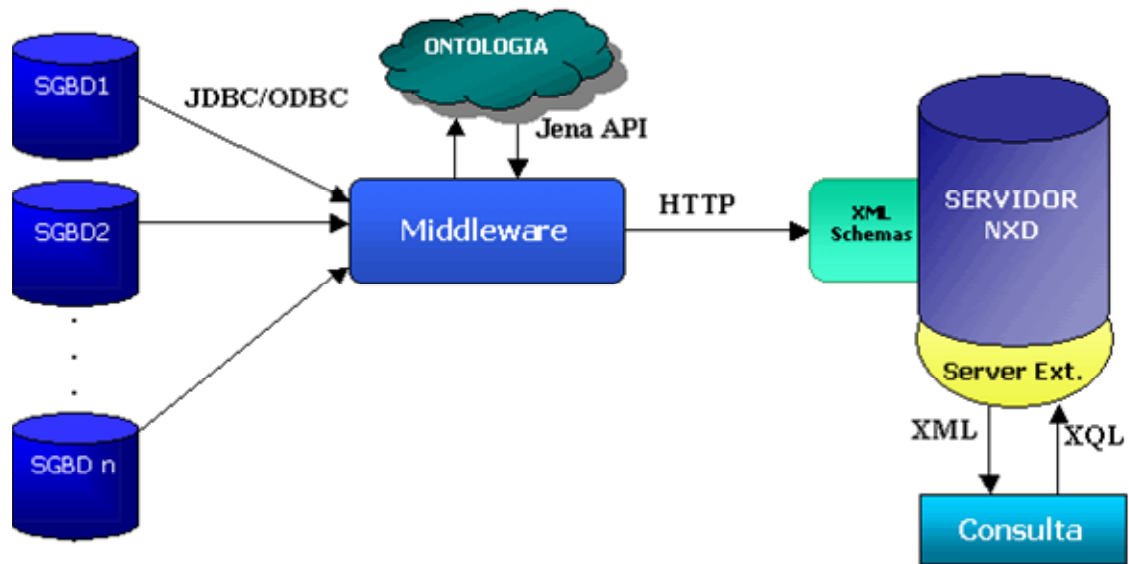
A principal limitação do Tamino neste processo é que ele permite que um elemento no XML *Schema* seja associado a apenas um campo de uma tabela em uma base relacional. Assim sendo, é necessário buscar uma alternativa que possibilite a busca em todas as bases de dados relacionadas a partir de um esquema global.

Neste ponto surge a necessidade de ter algum componente que interprete a consulta simples de forma que a mesma seja processada em cada uma das bases de dados inseridas no sistema de integração. Além disso, é necessário um outro componente que



funcione como um *Middleware* entre as bases de dados e o Tamino, possibilitando ao usuário determinar quais tabelas e campos deseja integrar.

A correção das discrepâncias semânticas e estruturais entre as diferentes bases também deve ser um ponto do sistema a ser considerado. Com base nessas necessidades e conhecendo as possibilidades do Tamino, a arquitetura proposta foi refinada, como mostrado na Figura 4.2.



**Figura 4.2 – Arquitetura refinada**

A arquitetura refinada é formada por diversas bases de dados distintas, as quais representam informações relacionadas a um determinado domínio de conhecimento. Como será abordado posteriormente neste capítulo, o acesso às bases é realizado através da combinação JDBC/ODBC, sendo que o acesso ocorre em dois momentos do processo:

- Quando do acesso à estrutura das bases;
- Na comunicação entre as bases de dados e o Tamino.

A primeira diferença significativa, comparando o modelo inicialmente apresentado para este, é que as bases de dados, no modelo refinado interagem com o *Middleware*. Este possui alguns módulos responsáveis por tarefas pertinentes ao sistema de integração, como mostrado na Figura 4.3, destacados a seguir.

- Módulo Estrutura: responsável por extrair das fontes de dados suas tabelas e campos, permitindo que o usuário defina quais campos de quais tabelas devem ser integrados;

- Módulo Integridade: a partir dos campos selecionados pelo usuário, tenta eliminar, ou ao menos minimizar os problemas de integridade semântica e estrutural das fontes de dados através do uso de uma ontologia;
- Módulo XSD: com as informações a serem integradas disponíveis, este módulo é o responsável por automatizar o processo de criação dos XML *Schemas* gerados, se comunicando com o NXD (no caso, o Tamino) e enviando os esquemas ao mesmo, passando as informações necessárias para que os XML *Schemas* sejam armazenados como o usuário determinou.

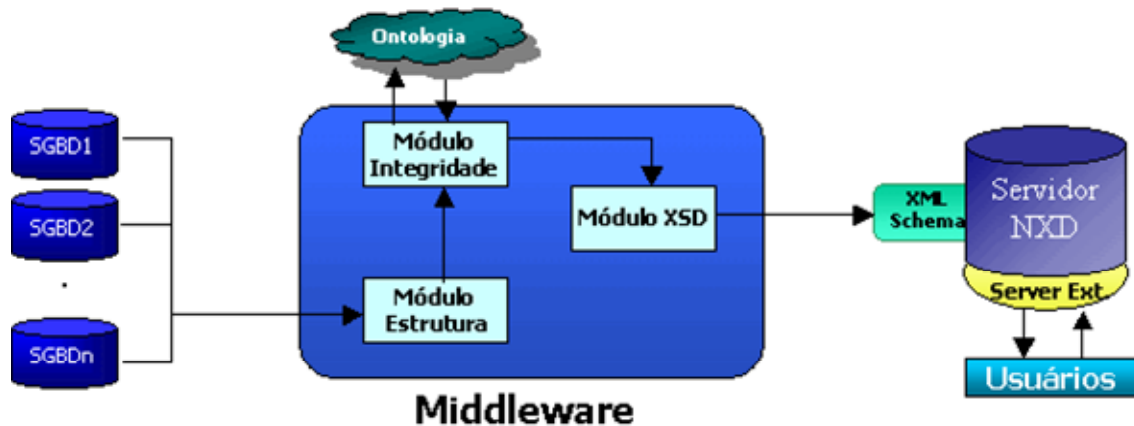


Figura 4.3 - Estrutura do Middleware

Outro ponto importante na arquitetura é a consulta aos dados integrados. Como foi apresentado anteriormente, o mapeamento entre os elementos e as bases é de um para um, o que força o sistema a encontrar uma alternativa para contornar essa limitação. Assim, toda vez que uma informação é consultada através do XML *Schema*, um componente do tipo *Server Extension* (apresentado no capítulo anterior) deve ser acessado para possibilitar a busca em todas as bases referenciadas.

Neste componente devem ser inseridas duas funções, uma de entrada, que é acessada quando o documento XML com os parâmetros de conexão é inserido no Tamino, e a função de saída, responsável por retornar o resultado da consulta ao usuário.

Esta arquitetura foi desenvolvida visando possibilitar que a integração dos dados seja realizada apoiada em um NXD. Assim sendo, com ajustes na forma de acesso e consulta aos dados é possível migrar o modelo proposto para outros NXD existentes no mercado.

## 4.2. Implementação

Definida a proposta de arquitetura, tendo como base o Tamino XML Server, passou-se à fase de implementação do SIDATA, um protótipo que visa a integração de dados de diversas fontes em um XML *Schema* global, a partir do qual as informações possam ser consultadas.

O protótipo é composto de um *Middleware* e uma *Server Extension*, sendo ambos desenvolvidos na plataforma Java, visto que esta é uma linguagem bastante difundida, multi-plataforma e que provê suporte a todos os requisitos fundamentais para o desenvolvimento da aplicação desejada.

A lista de requisitos necessários para que uma linguagem possa ser adotada no presente trabalho é apresentada como segue:

- Acesso à estrutura das bases;
- Manipulação de XML;
- API para o Tamino XML Server;
- Manipulação de documentos RDF e DAML (ontologias).

A seguir são abordadas as diversas etapas de desenvolvimento do protótipo, iniciando com as etapas do *Middleware*, responsável pela integração dados, e terminado com o desenvolvimento da *Server Extension*. O papel do Tamino no processo de integração também é destacado.

### 4.2.1. Busca dos dados nas bases

A etapa inicial da implementação consiste na forma como as informações das bases de dados podem ser acessadas e manipuladas pelo SIDATA. Como o protótipo foi desenvolvido em Java, o acesso às bases de dados é realizado via JDBC/ODBC.

Para ter acesso à estrutura das bases, utilizou-se a *interface* `DatabaseMetaData`. Esta *interface* visa oferecer as capacidades do SGBD combinado com um *driver* baseado na tecnologia ODBC, no caso o JDBC [JDBC 2003], que também é utilizado no projeto.

No presente trabalho, o `DatabaseMetaData` é responsável por buscar as informações estruturais das bases de dados, tais como tabelas, campos, versão, entre outras.

Por se tratar de uma forma genérica de acesso a bases de dados, é importante analisar as particularidades dos principais SGBD (Oracle, SQL Server, MySQL, entre outros) que se deseja integrar, visto que alguns deles apresentam algumas limitações que devem ser contornadas com o objetivo de evitar que as informações não sejam analisadas e integradas de forma adequada.

Dentre os problemas já detectados quando do processo de integração, pode-se citar a dificuldade do SQL Server 2000 em permitir via ODBC duas conexões simultâneas acessando o banco. Problema solucionado com o *Service Pack 3* [MS SQL Server 2003], porém o sistema deve estar apto a lidar com essas e outras particularidades no acesso à estrutura das bases.

Definida a forma como os dados serão acessados, passou-se à execução da primeira etapa da implementação. Como o Tamino permite o mapeamento apenas de bases de dados via ODBC, o protótipo foi focado nesse tipo de acesso. Assim sendo, é necessário permitir que o usuário determine quais bases de dados ele deseja integrar para que o sistema possa buscar a sua estrutura e manipular seu conteúdo. Como o acesso é via ODBC, é necessário apenas que o usuário indique o nome da fonte de dados, porém, é importante que o sistema saiba também o SGBD ao qual a fonte de dados pertence com o objetivo de tratar os problemas de acesso à estrutura das bases (como apresentado anteriormente).

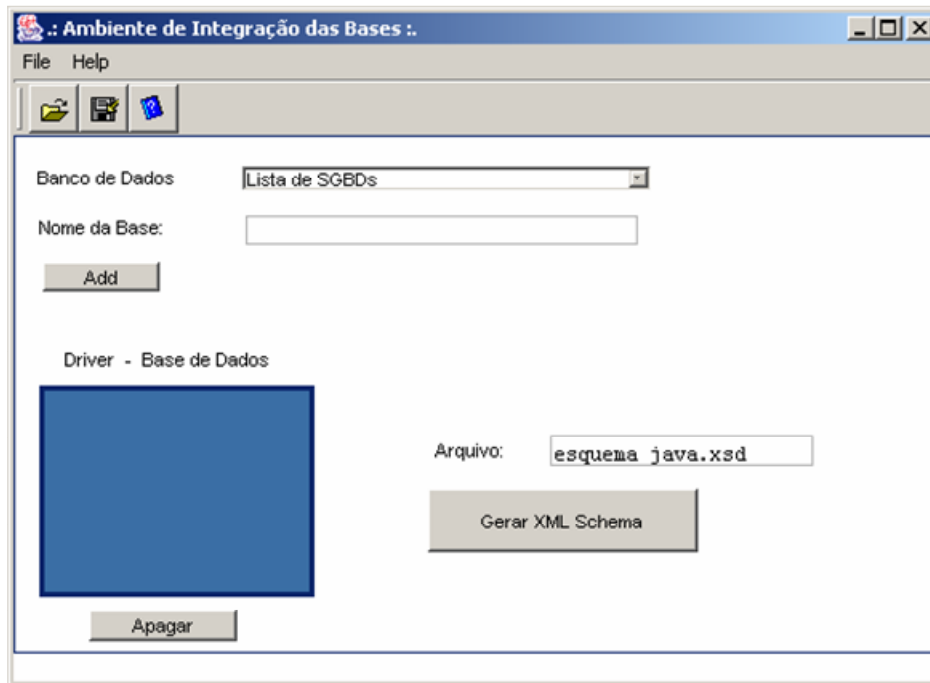
Nesta primeira etapa, o objetivo foi buscar as informações na base e gerar um XML *Schema* com os dados integrados sem se preocupar ainda com a integridade semântica dos mesmos, porém, já são tratados os tipos de dados que são encontrados nas bases, buscado um correspondente dentro do que é oferecido no XML *Schema*.

Mesmo sabendo que os tipos de dados com suporte em XML *Schemas* são uma evolução em relação à representação DTD [Vlist 2001] (que tratava todos os elementos com o tipo CDATA), estes apresentam limitações se comparados aos oferecidos nos SGBD.

Existem algumas adaptações que devem ser feitas para que os campos definidos nos SGBD possam ser mapeados para o XML *Schema* sem que a integridade da informação contida no campo seja perdida. Para isso, os tipos existentes nos principais SGBD do mercado são mapeados para um tipo correspondente em XML *Schema*. Esse

mapeamento visa minimizar tal problema. Caso o tipo definido no SGBD seja desconhecido pelo protótipo ele será tratado como *String* no *Schema* integrado.

A Figura 4.4 apresenta a tela do SIDATA em sua versão inicial.



**Figura 4.4 - Tela inicial do SIDATA**

Na tela apresentada acima, o usuário pode adicionar as bases de dados que deseja inserir no sistema selecionando o SGBD seguido do nome que representa a base de dados desejada. Por fim, pode-se gerar o XML *Schema* com a estrutura das bases selecionadas.

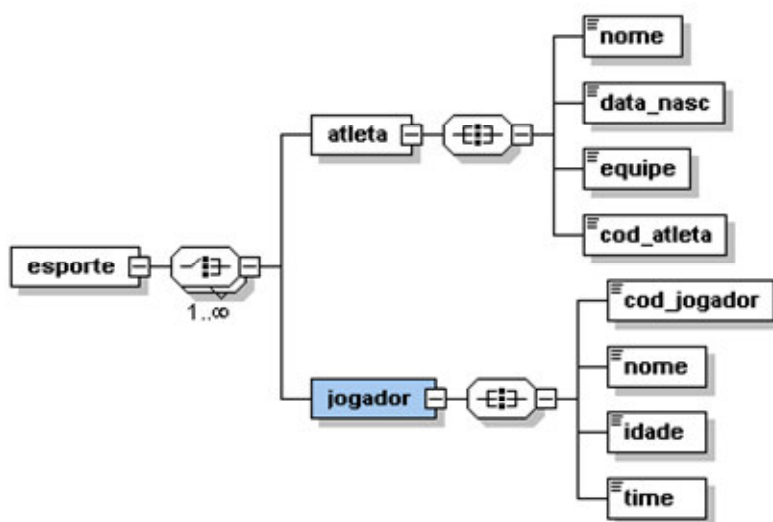
As bases foram integradas sem que os conflitos semânticos sejam eliminados, porém, os dados já podem ser incorporados ao NXD e serem consultados em um único XML *Schema* por uma linguagem de consulta XML com suporte no mesmo.

Como exemplo serão integradas as bases do domínio futebol como descritas no Quadro 4.1:

**Quadro 4.1 – Bases relacionadas ao domínio futebol**

<b>Tabela</b>	<b>Campos</b>
atleta	cod_atleta, nome, data_nasc, equipe
<b>(a) Base futSQL (SQL Server)</b>	
<b>Tabela</b>	<b>Campos</b>
jogador	cod_jogador, nome, idade, time
<b>(b) Base futAc (Access)</b>	

São duas bases cujas tabelas representam dados relativos a jogador, porém, os campos são ainda representados de forma diferente. Como os conflitos semânticos não estão sendo tratados, o XML *Schema* gerado seria representado como mostrado na Figura 4.5:



**Figura 4.5 - Esquema gerado pelo SIDATA em sua versão inicial.**

Mesmo não sendo o cenário ideal, já é possível fazer consultas às bases de dados através do mapeamento de cada elemento do XML *Schema* para seu correspondente na base de dados. A próxima seção apresenta como o Tamino é utilizado no processo de implementação do sistema.

#### **4.2.2. O Papel do Tamino**

Como apresentado no Capítulo 3, o Tamino permite, além do armazenamento de dados XML de forma nativa, mapear bases de dados em XML *Schemas* de forma que, para o usuário, a consulta seja efetuada como se esta estivesse sendo efetuada em uma base Tamino, independente do local onde os dados estão armazenados fisicamente, através do componente *X-Node*.

O *X-Node* permite que o conteúdo das bases externas seja acessado e alterado através do Tamino via ODBC. Apesar de ODBC fazer parte do sistema operacional Windows, o Tamino XML Server provê um software adicional que permite que sejam mapeados também dados de bases de dados em sistemas UNIX.

##### **Mapeamento SGBD – XML *Schema***

Para que seja possível consultar o mapeamento de dados de bases externas no Tamino é necessário explicitar quais elementos dentro do XML *Schema* serão mapeados e a forma como os dados do mesmo devem ser acessados.

Quando o usuário seleciona os dados que deseja integrar, estes devem ser representados em XML *Schemas* com os correspondentes mapeamentos para as bases de origem. O próximo processamento a ser realizado pelo *Middleware* é gerar os XML *Schemas* necessários no sistema de integração, mapeando o conteúdo das bases de origem para os XML *Schemas* correspondentes.

A definição das características básicas do elemento é feita de forma padrão, ou seja, de acordo com a especificação da W3C. Porém, as informações de mapeamento são incorporadas ao XML *Schema* através da TSD, que é a linguagem definida no Tamino para que seja possível incorporar aos XML *Schemas* convencionais as funcionalidades oferecidas pelo Tamino, neste caso, o acesso a bases externas.

Considerando o exemplo no qual os dados são mapeados de uma base de dados chamada de *futSQL*, que apresenta uma tabela *atleta*, a seguir é apresentado como estes dados podem ser representados e acessados através de um XML *Schema* no Tamino.

Primeiramente, é necessário definir o elemento que irá mapear a tabela na base relacional. Nele deve ser alterada a forma como seu conteúdo deve ser tratado pelo Tamino, ao invés de ele ser um elemento de armazenamento Nativo (*Native*) ele passará a ser uma sub-árvore que irá mapear o conteúdo de uma tabela externa (*Map SubTreeSQL*). Posteriormente é necessário definir os atributos correspondes ao nome da tabela, à fonte de dados ODBC e à(s) chave(s) primária(s) da tabela mapeada, além de login e senha, caso seja requerido pela base acessada.

Um exemplo de código para fazer o mapeamento inicial é mostrado no Quadro 4.2.

**Quadro 4.2 - Representação do mapeamento inicial no XML Schema**

```
(...)
<xs:element name = "atleta">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:subTreeSQL table = "atleta" datasource = "futSQL">
              <tsd:primaryKeyColumn>cod_atleta</tsd:primaryKeyColumn>
            </tsd:subTreeSQL>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
(...)
```

O primeiro círculo feito no Quadro 4.2 mostra a definição do nome do elemento, que é a forma padrão de definição de nomes da W3C. Em seguida, é definido como este será mapeado, com as informações de tabela e fonte de dados definida em *tsd:subTreeSQL* através dos atributos *table* e *datasource* respectivamente. Pode-se observar também a chave primária definida através do elemento *tsd:primaryKeyColumn*.

Dentro do elemento *atleta* pode-se definir os campos que serão mapeados. No caso, o mesmo irá representar o nome da tabela cabendo a seus elementos filhos representarem cada campo que se deseja mapear.

Os elementos filhos devem ser mapeados como mostrado no Quadro 4.3:



**Quadro 4.3 - Representação no XML Schema dos elementos que mapeiam campos de uma tabela**

```
(...)
<xs:complexType>
  <xs:all>
    <xs:element name = "cod" type = "xs:int">
      <xs:annotation>
        <xs:appinfo>
          <tsd:elementInfo>
            <tsd:physical>
              <tsd:map>
                <tsd:nodeSQL column = "cod_jogador" /> </tsd:nodeSQL>
              </tsd:map>
            </tsd:physical>
          </tsd:elementInfo>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name = "nome" type = "xs:string">
      <xs:annotation>
        <xs:appinfo>
          <tsd:elementInfo>
            <tsd:physical>
              <tsd:map>
                <tsd:nodeSQL column = "nome" /> </tsd:nodeSQL>
              </tsd:map>
            </tsd:physical>
          </tsd:elementInfo>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:all>
</xs:complexType>
(...)
```

Nesse mapeamento a atribuição de nome e tipo dos elementos é realizada seguindo a notação padrão de definição de XML *Schemas*. No caso do SIDATA, o tipo definido em cada elemento do esquema é correspondente ao campo existente na base de dados.

Não será detalhado todo o código que representa o XML *Schema*, tal como a especificação de atributos e como é feito cada tipo de mapeamento, pois um exemplo completo de aplicação Tamino foi apresentado no capítulo anterior.

Com o XML *Schema* criado, pode-se, após o mesmo ser inserido no Tamino, fazer consultas acessando o conteúdo das bases de dados como se estas estivessem armazenadas no próprio Tamino, tendo seu resultado retornado no formato XML.

Para exemplificar é apresentado o resultado de uma consulta *X-Query* que retorna o primeiro registro encontrado em atleta. A sintaxe da consulta seria a seguinte:

*http://localhost/tamino/DBSoccer/futebol?\_XQL(1,1)=esporte/atleta*

Resultando no documento XML apresentado no Quadro 4.4:

**Quadro 4.4 – Documento XML resultante da consulta X-Query**

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ino:response
xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
xmlns:xql="http://metalab.unc.edu/xql/">
  <xql:query>esporte/atleta</xql:query>
  <ino:message ino:returnValue="0">
    <ino:messageline>XQL Request processing</ino:messageline>
  </ino:message>
  <xql:result>
    <atleta>
      <cod_atleta>1</cod_atleta>
      <nome>Leandrinho</nome>
      <data_nasc>23/11/1981</data_nasc>
      <equipe>Juventude</equipe>
    </atleta>
  </xql:result>
  <ino:cursor ino:count="3">
    <ino:first ino:href="?_XQL(1,1)= esporte/atleta" />
    <ino:next ino:href="?_XQL(2,1)= esporte/atleta" />
    <ino:last ino:href="?_XQL(3,1)= esporte/atleta" />
  </ino:cursor>
  <ino:message ino:returnValue="0">
    <ino:messageline>XQL Request processed</ino:messageline>
  </ino:message>
</ino:response>
```

O documento XML acima representa o resultado da consulta feita no XML Schema solicitando o primeiro registro da tabela *atleta* na base *futSQL* na qual, além do item desejado, o documento apresenta outras informações relevantes, como por exemplo, uma referência para o elemento seguinte na estrutura.

### Mapeando múltiplas bases de dados

Até o presente estágio da implementação, a estrutura é composta de duas bases de dados dispostas em um único XML Schema, sendo que sua representação é aquém da desejada, pois na verdade pouco há de integração entre as bases. Elas são dispostas na

estrutura, porém cada elemento é inserido de forma separada, tendo ele o mesmo significado semântico ou não.

A limitação do componente *X-Node*, que permite o mapeamento de apenas um elemento para um campo em uma tabela, trouxe uma nova questão a ser trabalhada no SIDATA, o mapeamento de múltiplas bases utilizando um único elemento no XML *Schema*.

Primeiramente, analisou-se a possibilidade de criar um XML *Schema* no qual os elementos receberiam o nome da tabela ou campo que se deseja mapear. Este então seria associado com o nome da fonte de dados de onde as informações são consultadas, algo como *jogador\_futSQL* e *jogador\_futAc*. Esta solução resolveria o problema da limitação do *X-Node*, mas eliminaria a transparência que se deseja ter no sistema, quando da consulta aos dados integrados. Neste modelo, o usuário do sistema acabaria por fazer consultas nas bases específicas e muito pouco ganharia em termos de transparência das informações.

Um complemento a essa solução seria criar um ambiente de consulta, no qual o usuário realizaria as buscas e caberia ao ambiente localizar as fontes de dados apropriadas.

Neste cenário, o Tamino deixaria de funcionar como um mediador entre as bases, criando uma nova camada entre o usuário e o Tamino. Assim, o SIDATA não conseguiria aproveitar de forma significativa as capacidades oferecidas pelo Tamino, além de tornar o sistema mais complexo para o usuário que teria que manipular mais uma ferramenta para consulta dos dados.

Objetivando evitar que se crie mais uma camada no processo de integração, passou-se a buscar soluções que viabilizasse a consulta dos dados integrados diretamente no ambiente Tamino.

A primeira proposta de solução foi através da utilização de múltiplos *Namespaces*. Pretendia-se que o processo de integração gerasse dois XML *Schemas*, no qual o primeiro teria a referência dos dados para as suas bases (no formato *jogador\_futSQL*, *jogador\_fitAc* apresentado anteriormente) e um XML *Schema* global, no qual seus elementos fariam referências aos do *Namespace* que tem as definições das bases, através

da herança de *Namespaces*. Porém, como o Tamino não implementa herança de *Namespaces* utilizando o componente *X-Node*, essa proposta acabou sendo descartada.

Com a impossibilidade do uso de herança de *Namespaces* passou-se a buscar uma alternativa que viabilizasse o mapeamento de múltiplas bases de dados.

Uma alternativa encontrada foi analisar a viabilidade de se resolver o problema de mapeamento dos dados desenvolvendo uma extensão para o Tamino, através da implementação de um componente do tipo Tamino *Server Extension*. Esta acabou sendo a forma encontrada para se minimizar o problema de mapeamento entre o XML *Schema* e as bases de dados.

As *Server Extensions* [Softwareag 2003] podem ser desenvolvidas em diversas linguagens, como C++, Visual Basic, e Java, sendo que esta última foi a linguagem escolhida, visto que a outra parte do trabalho também é desenvolvida nesta linguagem.

O desenvolvimento de *Server Extensions* envolve algumas particularidades que devem ser consideradas quando da implementação das mesmas. Posteriormente neste capítulo será abordado em detalhes a *Server Extension* Integra2004, desenvolvida no presente trabalho.

Um ponto importante é o processo de correção da integridade semântica dos dados através da definição de uma ontologia. O SIDATA utiliza o conceito de ontologias para corrigir problemas de integridade semântica dos dados.

Neste trabalho são abordadas as características da ontologia, a partir da qual uma é desenvolvida baseada em um determinado domínio de conhecimento, porém a incorporação das mesmas no sistema é parcialmente implementada, pois apenas a integridade da estrutura das bases é analisada.

A ontologia definida visa representar os termos referentes ao domínio das bases que estão sendo integradas. Por exemplo, considerando futebol como domínio, essa ontologia deve ser capaz de encontrar campos diferentes, porém semanticamente equivalentes, definindo a forma global de representação dos mesmos. Após a correção semântica, é que os respectivos XML *Schemas* devem ser criados, para só então serem incorporados ao Tamino.

### **4.3. Estrutura do sistema**

O protótipo desenvolvido apresenta uma proposta de integração em um dado domínio de conhecimento, nas quais as diversas bases podem ser integradas e consultadas de forma centralizada através do Tamino.

Para o desenvolvimento do *Middleware*, foram desenvolvidas três classes que servem como base para o desenvolvimento da aplicação e serão apresentados a seguir.

A classe `InfoBase` apresenta a estrutura básica para descrever o conteúdo das tabelas, contendo as seguintes propriedades:

- `sgbd` – descrição do SGBD onde a base está armazenada;
- `driver` – forma de acesso à base de dados. Atualmente, o mapeamento às bases no Tamino é feito apenas via ODBC, porém é importante a estrutura da classe estar apta a lidar com outras formas de acesso, como JDBC;
- `desc_base` – descrição da base de dados. A forma como ela pode ser acessada a partir do *driver* especificado.

A classe `Estrutura` apresenta as principais funções que tratam da conversão de tabelas para XML *Schemas*, assim como a estrutura na qual as informações das bases serão armazenadas e acessadas no *Middleware*. A seguir são descritos os principais métodos desenvolvidos na classe:

- `Conecta` – método responsável por criar a conexão com uma determinada base de dados. Recebe dois parâmetros do tipo `String`, sendo um representando o tipo de acesso à base e o outro a *string* de conexão com o banco. Retorna um objeto do tipo `Connection` contendo a conexão com a base, ou no caso de a conexão não ter sido efetuada corretamente, retorna o objeto vazio;
- `MontaEstrutura` – acessa cada uma das bases a ser integradas e recupera sua estrutura, ou seja, monta as informações relativas a cada tabela com a base à qual ela pertence, os campos existentes na mesma, bem como os seus respectivos tipos. Para armazenar a estrutura das bases, é utilizada a classe `Tabela`, desenvolvida no presente projeto para este fim, visando facilitar armazenamento e o acesso às propriedades das tabelas pelos métodos do sistema;

- `GeraSchema` – a partir da definição do conteúdo que será integrado, este método é o responsável pela criação dos XML *Schemas* que serão armazenados no Tamino. A partir deste método, outros dois são chamados, o método `GeraSchemaGlobal`, responsável por criar o esquema global de consulta, e o `GeraSchemaFragmentado`, que cria os XML *Schemas* que irão mapear cada base de dados.

A classe `Ontologia` é responsável por carregar o arquivo DAML selecionado pelo usuário no SIDATA. Para percorrer a ontologia, a classe utiliza a API Jena, permitindo percorrer arquivos RDF e DAML [Jena 2004].

Esta classe oferece o método `BuscaCampo`, que recebe como parâmetro uma `String` com o termo que se deseja verificar na ontologia, retornando uma `String` com o termo equivalente no esquema global, ou com o próprio termo se este fizer parte do esquema global e não for encontrado na ontologia.

Com o *Middleware* desenvolvido, o próximo passo é permitir que consultas ao sistema sejam feitas através do Tamino XML Server, visto que é nele que os XML Schemas gerados são armazenados.

### **Server Extension**

Uma outra etapa do processo de integração desenvolvida no presente trabalho é o desenvolvimento da *Server Extension* Integra2004. Como abordado no capítulo anterior, quando da apresentação dos conceitos relacionados as *Server Extensions*, deve-se analisar com cuidado de que forma estas podem ser úteis dentro do problema da integração de múltiplas bases de dados.

No presente problema, o objetivo é que, quando os dados forem consultados em um XML *Schema*, estes sejam retornados de todas as bases que tenham ocorrências do mesmo. Assim sendo, a *Server Extension* deve ser executada quando os dados fossem recuperados em uma consulta. Ela deve buscar nas bases os dados de acordo com a consulta efetuada e retornará, obrigatoriamente no formato XML, o resultado da consulta efetuada.

Para realizar tal tarefa, são usadas as funções de mapeamento do tipo *Map-in*, que são executadas quando ocorre uma inserção de uma instância do documento no Tamino, e *Map-out*, que é executada quando uma consulta é efetuada pelo usuário.

A seguir é apresentado de forma mais detalhada como foi implementada a *Server Extension*, destacando a definição das funções implementadas para satisfazer o problema proposto.

Inicialmente, foram implementadas as funções *Entrada* e *Saida*, a serem executadas quando da inserção do documento XML na base e na consulta ao mesmo, respectivamente.

Estas funções de mapeamento possuem três ou quatro parâmetros, como apresentados no Quadro 4.5.

**Quadro 4.5 – Lista de parâmetros da função de mapeamento**

<b>Nomenclatura</b>	<b>Posição/Direção</b>	<b>Obrigatório</b>	<b>Tipo XML</b>	<b>Tipo Java</b>
<i>Object ID</i>	1 / entrada	SIM	xs:int	Int
<i>Element ID</i>	2 / entrada	SIM	xs:int	Int
<i>XML Object</i>	3 / entrada	SIM	ino:XML-OBJ	String
<i>Node Info</i>	4 / saída	NÃO	xs:string	StringBuffer

Os parâmetros *Object ID* e *Element ID* são usados para identificar de forma única os dados que são passados para a função da *Server Extension*.

O *XML Object* representa o documento ou fragmento XML a partir do qual a função foi chamada, ou seja, considera-se que um elemento *medico* foi associado à função de mapeamento, este objeto irá representar como `string` toda a estrutura XML a partir do elemento mapeado.

*Node Info* é o quarto parâmetro oferecido pela função. Ele é opcional, e permite que sejam armazenadas informações que possam ser acessadas pelas funções *Map-out* e *Delete* correspondentes.

No caso deste trabalho, os parâmetros *Node Info* e *XML Object* serão os mais importantes dentro da *Server Extension* desenvolvida.

A partir das funções de mapeamento, todo o processamento necessário para a extração e a exibição dos dados consultados é realizado. Para isso, foi desenvolvida uma classe *Integracao*, responsável pela recuperação dos dados das bases mapeadas, além de montar o documento XML que será retornado ao usuário.

A classe utiliza API de manipulação de documentos XML, mais precisamente o *parser* da *javax* e o DOM da W3C.

Esta classe contém os métodos descritos a seguir:

- `MontaDom` – recebe como parâmetro uma string contendo o documento XML consultado. Verifica se o documento é bem formado. Caso seja, monta o DOM do mesmo, retornando um valor booleano que indica se o DOM foi gerado corretamente ou não. Este método utiliza o *parser javax* para gerar o DOM a partir de um documento XML representado como `string`;
- `CriaSaida` – recupera os parâmetros do documento XML consultado. Esses parâmetros são as informações necessárias para buscar nas bases de dados o conteúdo integrado via os XML *Schemas* mapeados. Dentre os parâmetros passados, pode-se destacar o nome da coleção onde estão os esquemas, a base Tamino onde os mesmos se encontram e a lista com nomes dos XML *Schemas* locais que devem ser consultados. Com esses parâmetros é possível chamar o procedimento que busca as informações relativas a cada base mapeada. Por fim, o método retorna uma `string` com o conteúdo das bases de dados em formato XML;
- `BuscaBase` – método definido para buscar, através do Tamino, o conteúdo de uma determinada base de dados mapeada. Recebe como parâmetro uma `string` contendo o nome do *doctype* do esquema a ser consultado, buscando através de uma consulta XQL os dados requeridos pelo usuário. Este método utiliza o DOM da W3C para montar a árvore hierárquica de elementos e processar cada todas as ocorrências de elementos que forem retornados. O método retorna uma `string` contendo um documento XML com todos os elementos retornados no resultado da consulta;
- `MontaFilhos` – é um método recursivo que percorre a lista de filhos dos elementos resultantes da consulta à base de dados. Quando a consulta é processada pelo método `BuscaBase`, as informações retornadas ficam dispostas em uma lista de elementos do tipo `NodeList`. Essa lista é percorrida pelo método `MontaFilhos`, que monta o documento XML



resultante da consulta varrendo toda a árvore de elementos, retornando como resposta uma `string` com o resultado da consulta em formato XML;

- `GeraSaida` – método chamado para montar a saída que será exibida pelo usuário. Este é o único método chamado na função `Map-out Saida`. Ele recebe como parâmetro uma variável do tipo `StringBuffer` contendo o resultado da consulta a todas as bases, retornando uma `string` com o documento XML requerido pelo usuário.

A Figura 4.6 apresenta a estrutura da *Server Extension* com a comunicação entre os métodos da classe `Integracao`.

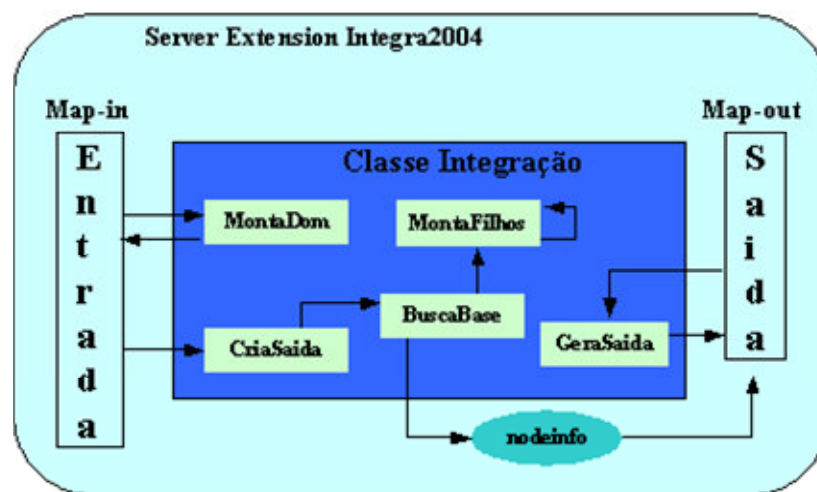


Figura 4.6 - Server Extensions Integra2004

Como pode-se observar na figura acima, a função `Entrada` interage com os métodos `MontaDom` e `CriaSaida`, o primeiro fazendo a verificação do documento XML e o segundo desencadeando todos os métodos de busca de dados e geração do documento XML a ser retornado para o usuário.

Na estrutura de desenvolvimento da *Server Extension*, o atributo `nodeinfo` é a variável que pode ser acessada e manipulado nas funções do tipo *Map-in* e *Map-out*, sendo a ligação entre duas funções de mapeamento desenvolvidas. Na função `Entrada` o atributo `nodeinfo` recebe o resultado das consultas feitas nas bases enquanto que na função `saida` `nodeinfo` recebe o documento XML que será retornado ao usuário.

Com a *Server Extension* Integra2004 desenvolvida tornou-se viável o mapeamento de informações do esquema global para os esquemas que mapeiam as informações das bases de dados.

#### **4.4. Análise do SIDATA**

O SIDATA foi implementado baseado na arquitetura proposta na qual se tem um NXD (no caso o Tamino) utilizado para simplificar o processo de integração, oferecendo *interfaces* de mapeamento e consulta visando possibilitar que os maiores esforços sejam concentrados nas demais etapas de desenvolvimento de sistemas de integração.

Durante o processo de implementação do SIDATA, seu desenvolvimento foi dividido em duas partes, um *Middleware* de integração e uma *Server Extension*, cujas funções foram descritas ao longo do presente capítulo.

Tais funções foram implementadas de acordo com as especificações da arquitetura definida, objetivando demonstrar como a mesma pode ser aplicada em um sistema de integração de dados.

É no *Middleware* desenvolvido que são implementadas as etapas de busca e representação dos dados de acordo com a especificação da arquitetura. Porém, não são desenvolvidas todas as funcionalidades necessárias para que o processo de integração seja realizado de forma completa.

Dentre as limitações existentes no *Middleware*, pode-se destacar a restrição à integração de bases de dados relacionais e a análise de integridade semântica (apenas a estrutura das tabelas é analisada).

Considerando a *Server Extension* Integra2004, responsável por permitir que as consultas efetuadas no esquema global sejam mapeadas corretamente para os respectivos esquemas locais, pode-se dizer que ela foi a alternativa encontrada para que não fosse necessário o desenvolvimento de uma *interface* entre o Tamino e o usuário. Seu desenvolvimento visa permitir que os dados integrados sejam corretamente consultados, cabendo ao usuário realizar a consulta ao esquema global sem se preocupar em como os dados estão fisicamente armazenados.

A grande dificuldade encontrada nesta etapa é a limitação de consultas, pois esquemas mapeados restringem as consultas à linguagem X-Query.

Considerando o objetivo definido quando do desenvolvimento do SIDATA, ou seja, do mesmo ser um protótipo que demonstre de forma simplificada o funcionamento de um sistema de integração de dados baseado na arquitetura proposta, considera-se que o objetivo do mesmo foi alcançado, pois o SIDATA define em sua estrutura todos os passos definidos na arquitetura proposta.

## **4.5. Considerações Finais**

Este capítulo apresentou o SIDATA, uma arquitetura para integração de dados utilizando NXD, com o desenvolvimento de um protótipo que implementa a arquitetura. O objetivo do capítulo foi descrever como foi definida a arquitetura proposta e a implementação do SIDATA, destacando o papel do NXD, no caso o Tamino, no desenvolvimento do mesmo.

## 5. UM EXEMPLO DE USO DO SIDATA

Com o protótipo desenvolvido é apresentado um exemplo de uso, onde o mesmo é aplicado a um determinado domínio de conhecimento, no caso serão bases de dados relacionadas a futebol.

São integradas duas bases de dados, uma que apresenta dados relacionados aos times da série A do campeonato brasileiro e outra com informações de equipes da série B.

O objetivo deste exemplo de uso é apresentar como o SIDATA é utilizado em uma aplicação, exibindo as informações geradas e fornecidas por ele e como o usuário deve proceder para utilizá-lo corretamente.

O Quadro 5.1 apresenta a estrutura de uma das bases de dados utilizadas nesse exemplo de uso. Esta base, chamada *serieA*, representa dados relacionados aos times da primeira divisão do campeonato brasileiro e é armazenada no SGBD Oracle.

**Quadro 5.1 – Estrutura da base de dados serieA**

<b>BASE SÉRIE A</b>	
Tabela: Jogador	
<b>Campo</b>	<b>Tipo</b>
cod_jogador	inteiro
nome_jogador	string
data_nasc	data
posicao	string
naturalidade	string
clube anterior	string
cod_time	inteiro
Tabela: Time	
<b>Campo</b>	<b>Tipo</b>
cod_time	inteiro
nome_time	string
uf	string
cidade	string
cod_comissao	inteiro
Tabela: Comissao_tecnica	
<b>Campo</b>	<b>Tipo</b>
cod_comissao	inteiro
tecnico	string
assistente	string
preparador_fisico	string
medico	string
fisioterapeuta	string

A outra base de dados utilizada neste exemplo é chamada *serieB*, representa dados relacionados às equipes da segunda divisão do campeonato brasileiro e é

armazenada no Microsoft SQL Server. A estrutura desta base de dados é apresentada no Quadro 5.2.

**Quadro 5.2 – Estrutura da base de dados serieB**

<b>BASE - SÉRIE B</b>	
Tabela: Atleta	
<b>Campo</b>	<b>Tipo</b>
cod_atleta	inteiro
nome	string
idade	inteiro
posicao	string
cidade_natal	string
ultimo_clube	string
cod_equipe	inteiro
Tabela: Equipe	
<b>Campo</b>	<b>Tipo</b>
cod_equipe	inteiro
nome_equipe	string
Estado	string
Municipio	string
cod_comissao	inteiro
Tabela: Corpo_tecnico	
<b>Campo</b>	<b>Tipo</b>
cod_corpo_tecnico	inteiro
Treinador	string
Auxiliar	string
Fisicultor	string
Doutor	string

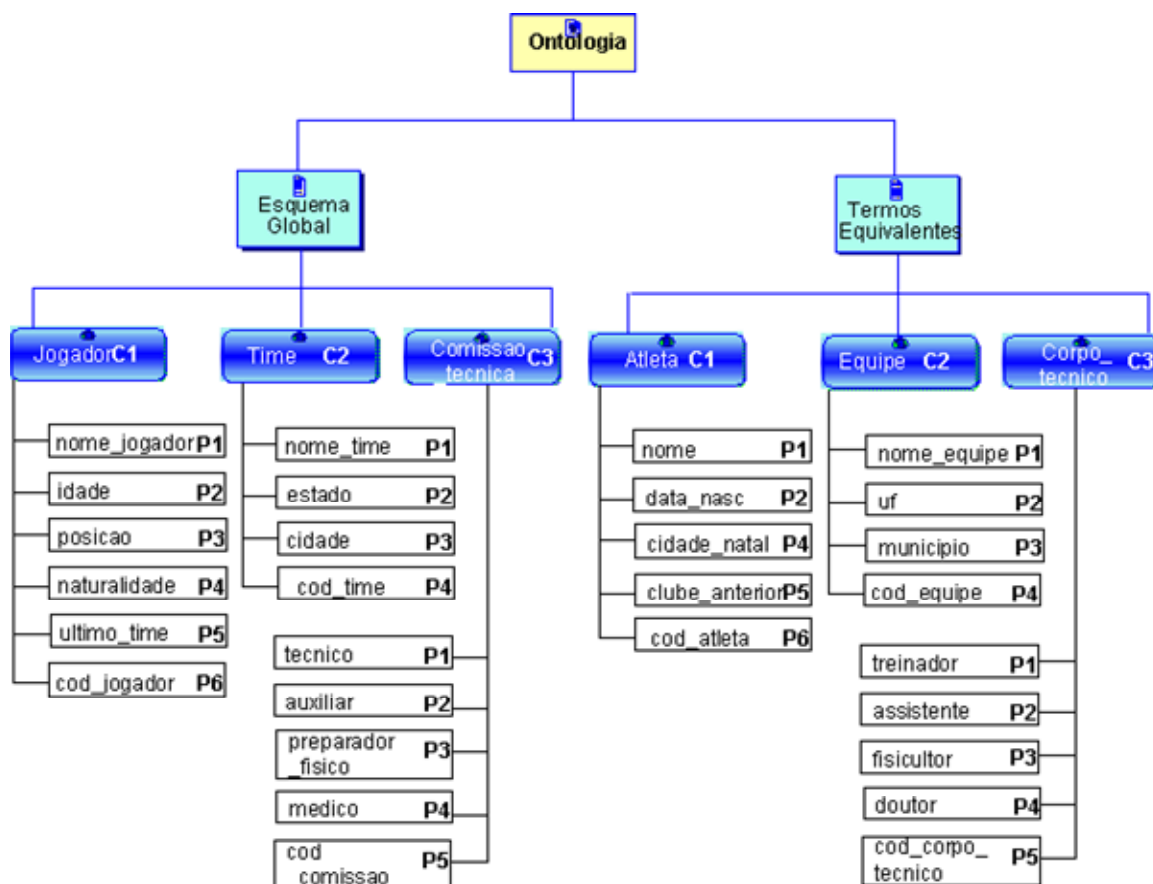
As duas bases integradas são relacionadas ao domínio futebol, porém apresentam algumas diferenças que devem ser tratadas pelo SIDATA.

Analisando a estrutura das tabelas relacionadas a jogadores nas bases `serieA` e `serieB`, pode-se perceber algumas diferenças na forma de representação dos dados armazenados. A seguir são discutidas algumas diferenças encontradas nas bases integradas.

- Diversos nomes de campos e tabela representando a mesma informação porém dispostos com nomes diferentes;
- Campos nos quais, além dos nomes diferentes, tal como os campos `idade` e `data_nasc` que, apesar de representar a mesma informação, armazenam de forma distinta também os dados na base de dados, sendo o tipo de dados uma diferença a ser considerada, visto que o campo `idade` é armazenado como inteiro e `data_nasc` com o tipo `data`;
- Alguns campos existem em apenas uma das tabelas.

Como apresentado no decorrer deste trabalho, são inúmeras as questões que um sistema de integração deve tratar quando deseja integrar de forma satisfatória dados de bases de dados heterogêneas. O SIDATA visa, também, corrigir os problemas de integridade semântica existente na estrutura das bases de dados, oferecendo uma visão global a partir da qual os dados poderão ser consultados.

Para o exemplo de uso apresentado foi desenvolvida uma ontologia na linguagem DAML visando representar as informações relacionadas ao domínio `futebol`. A Figura 5.1 apresenta a estrutura da ontologia.



**Figura 5.1 - Ontologia sobre o domínio futebol**

Essa figura visa representar de forma gráfica a ontologia desenvolvida. São apresentadas as classes que formam a mesma. Estas são representadas em dois esquemas, um com os dados definidos como esquema global, e outro com as classes que representam os *termos equivalentes* aos que compõem o modelo global.

No código da ontologia, a definição do esquema global não é feita de forma tão explícita, pois não existe nenhuma definição direta que cite o esquema global. O objetivo da figura é auxiliar na visualização das classes e propriedades das ontologias, e que tipo de informação elas estão representando.

A ontologia definida apresenta seis classes, cada qual com uma série de propriedades que representam as informações relacionadas a cada classe. A numeração P1 .. Pn é realizada para relacionar as propriedades representadas no esquema global com as definidas nos demais esquemas. Assim pode-se observar, por exemplo, que a propriedade *posicao* não apresenta correspondente em *termos equivalentes*.



O Anexo A apresenta o código definido em DAML para representar a ontologia da Figura 5.1.

Com a definição das bases de dados e da ontologia sobre o domínio de conhecimento desejado, pode-se iniciar o processo de integração através do *Middleware* do SIDATA.

### **5.1. Integrando os dados com o SIDATA**

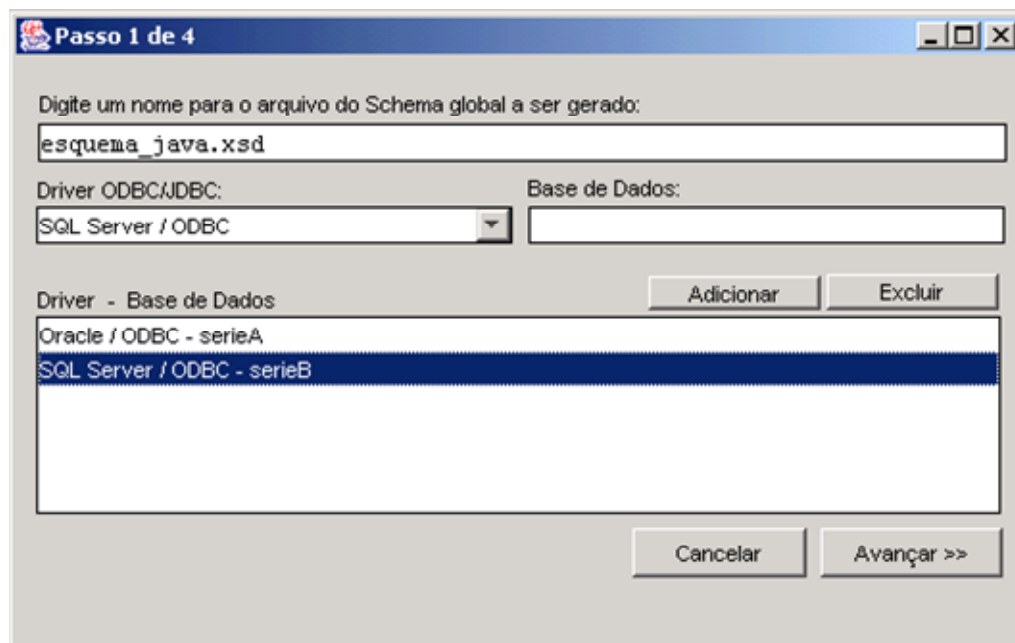
Partindo das especificações das bases de dados acima apresentadas, serão apresentadas a seguir as etapas do processo de integração através do *Middleware* do SIDATA.

Inicialmente, o sistema deve receber do usuário as bases de dados que se deseja integrar. Nessa etapa deve-se referenciar o SGBD onde a base de dados foi criada, além do nome da fonte de dados ODBC que representa a mesma.

Será considerado o mesmo nome das bases para as conexões ODBC, necessárias para a comunicação com as bases. Assim os parâmetros iniciais definidos no *Middleware* serão os seguintes:

- Oracle/ODBC – serieA;
- SQL Server/ODBC – serieB.

A Figura 5.2 apresenta a tela com o passo 1 do sistema de integração, com a *interface* provida para a etapa descrita acima.

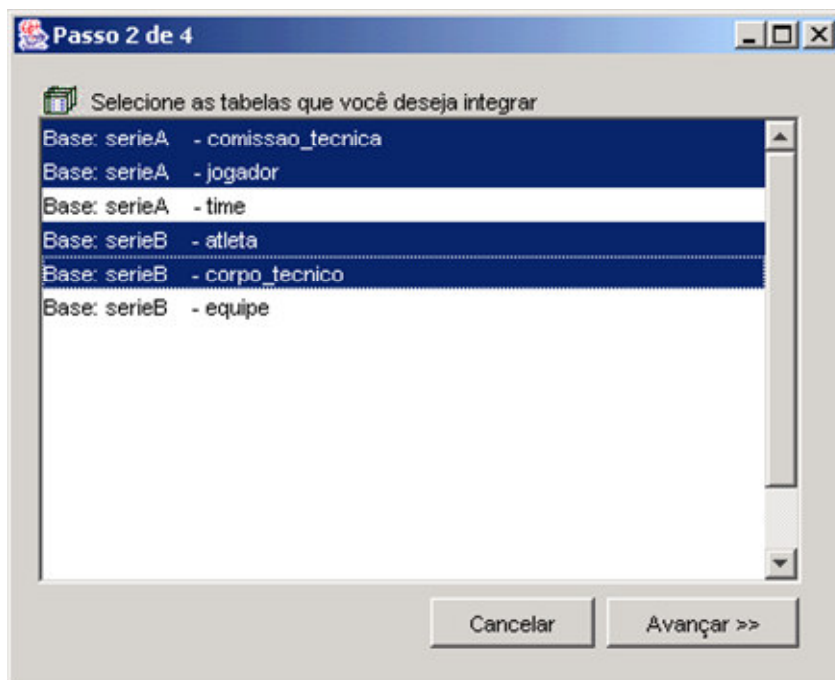


**Figura 5.2 - Passo 1 do sistema de integração**

Quando o usuário seleciona a opção *Avançar*, as bases selecionadas são montadas de acordo com a definição da classe `InfoBase`, possibilitando a posterior chamada ao método `MontaEstrutura` para que a estrutura das bases seja recuperada e esteja disponível no próximo passo do processo de integração.

Com as bases definidas, o usuário passa a escolher as tabelas que deseja integrar. No caso, considera-se tabelas relacionadas a jogador e comissão técnica como sendo as que se tem interesse em integrar. Assim, as tabelas representando as equipes de ambas as bases não serão selecionadas.

A Figura 5.3 apresenta a tela do sistema com a seleção das tabelas que serão integradas.



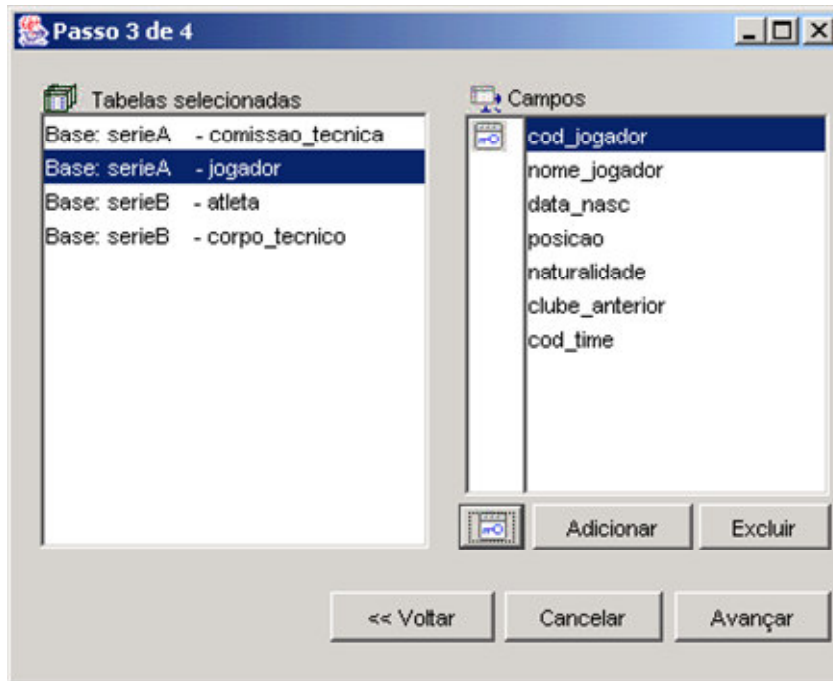
**Figura 5.3 - Passo 2 do sistema de integração**

Com as tabelas selecionadas, o usuário passa à definição de quais campos dentro das mesmas devem ser integrados. Nesse ponto, deve-se definir quais campos serão chaves-primárias, ou seja, os índices para recuperação de dados no sistema, pois é a partir da definição das chaves-primárias que o Tamino busca nas fontes de dados o resultado da consulta requisitada pelo usuário.

Cada tabela pode ser referenciada por uma ou mais chaves-primárias, considerando que o sistema irá gerar um XML *Schema* para mapear o conteúdo de cada base de dados. Sendo assim, pode-se ter uma série de chaves distintas, porém, na definição do esquema elas são relacionadas às respectivas tabelas, evitando que haja problemas na recuperação dos campos.

No exemplo, será considerado que as tabelas serão integradas com todos os campos, com exceção do que se refere ao código do time que não é necessário, visto que as informações relacionadas às equipes foram excluídas do processo de integração.

A Figura 5.4 apresenta essa etapa do processo de integração.



**Figura 5.4 - Passo 3 do sistema de integração**

Com a estrutura completa de todos os dados a serem integrados, a próxima etapa é a geração dos *XML Schemas*, tanto os que representam cada base de dados, quanto o esquema global.

Para tanto, é necessário, além dos dados relativos às estruturas das bases, obter os parâmetros que indicam como os *XML Schemas* devem ser armazenados no Tamino. O usuário deve determinar a coleção na qual os *XML Schemas* serão armazenados, além do nome do *Doctype* do esquema global, a partir do qual as consultas devem ser efetuadas.

O usuário deve determinar também, em qual base do Tamino os dados devem ser inseridos, bem como qual o caminho no qual o Tamino se encontra no servidor. Além disso, é preciso referenciar o arquivo que contém a ontologia utilizada pelo sistema para solucionar as discrepâncias semânticas existentes nas bases.

Com essas informações, todos os *XML Schemas* podem ser gerados pelo sistema, ficando aptos a serem incorporados ao Tamino de acordo com as especificações do usuário.

Para o presente exemplo os parâmetros são definidos da seguinte forma:

- Caminho Tamino – *http://xavante/tamino*;

- Base de Dados – *brasileirao2004*;
- Coleção – *futebol*;
- Doctype – *brasileiro*;
- Ontologia – *integracao\_divisao12.daml*.

A Figura 5.5 apresenta a tela do último passo de interação com o usuário, no qual os mesmos entram com os parâmetros do Tamino e o arquivo representando a ontologia.

Passo 4 de 4

Informe os dados abaixo para integração dos Schemas com o Tamino.

Caminho Tamino

Base de Dados

Coleção

DocType

\* Atenção: todos os campos são obrigatórios.

Caminho do arquivo da Ontologia  
 ...

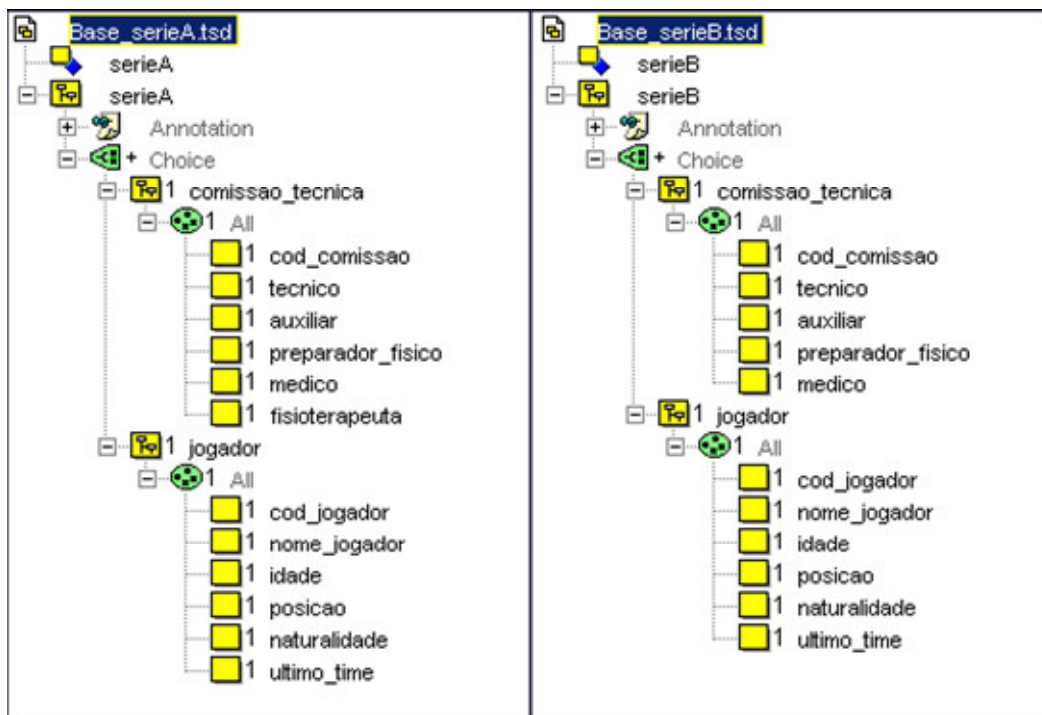
<< Voltar    Cancelar    Finalizar

**Figura 5.5 - Passo 4 do sistema de integração**

Antes da geração dos XML *Schemas*, cada base de dados terá sua estrutura analisada na ontologia, visando buscar uma visão única dos dados. Para isso, é utilizada a classe *Ontologia*, que permitirá através do método *BuscaCampo* que os problemas de integridade semântica da estrutura das bases seja solucionado.

Por fim, o método *GeraSchema* é chamado, uma vez que todas as informações relevantes à definição dos esquemas já foram atribuídas pelo usuário. Este método irá gerar os XML *Schemas* locais e o XML *Schema* global que serão inseridos no Tamino.

A Figura 5.6 mostra a estrutura dos dois XML *Schemas* locais que foram gerados a partir das bases de dados integradas.



**Figura 5.6 – Representação das bases *serieA* e *serieB* como XML Schemas**

Analisando a figura acima, pode-se observar que aparentemente são poucas as diferenças entre os dois XML Schemas. Basicamente, as diferenças consistem no nome do esquema, o *Doctype*, o elemento raiz e o elemento *fisioterapia*, presente apenas no esquema *Base\_serieA.tsd*.

Essa aparente igualdade entre os esquemas se deve às correções de integridade realizadas através da ontologia. As diferenças mais significativas entre os dois esquemas estão no mapeamento de seus campos para a base externa, e isso não está representado na Figura 5.6.

Para representar esses diferentes mapeamentos, é apresentado nos Quadro 5.3 e 5.4 um trecho do código dos esquemas correspondentes aos esquemas *Base\_serieA.tsd* e *Base\_serieB.tsd* respectivamente. O código completo dos esquemas é apresentado no Anexo B.

Quadro 5.3 – Trecho de código do XML *Schema Base\_serieA.tsd*

```

<xs:element name = "comissao_tecnica">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:subTreeSQL table = "comissao_tecnica" datasource = "serieA">
              <tsd:primaryKeyColumn>cod_comissao</tsd:primaryKeyColumn>
            </tsd:subTreeSQL>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:all>
      <xs:element name = "cod_comissao" type = "xs:integer">
        <xs:annotation>
          <xs:appinfo>
            <tsd:elementInfo>
              <tsd:physical>
                <tsd:map>
                  <tsd:nodeSQL column = "cod_comissao"></tsd:nodeSQL>
                </tsd:map>
              </tsd:physical>
            </tsd:elementInfo>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>

```

O código acima representa o mapeamento da tabela *comissao\_tecnica* e do campo *cod\_comissão*. Como a nomenclatura usada na tabela e no campo apresentado corresponde ao que foi definido no esquema global na ontologia, o mapeamento esquema-base não apresenta diferenças, ou seja, o elemento chamado *comissao\_tecnica* é mapeado em uma tabela chamada *comissao\_tecnica*, o mesmo ocorrendo em relação ao campo *cod\_comissao*.

O mapeamento do esquema *Base\_serieB.tsd* apresenta diferenças em relação ao *Base\_serieA.tsd*, como pode-se observar no Quadro 5.4.

Quadro 5.4 – Trecho de código do XML Schema *Base\_serieB.tsd*

```

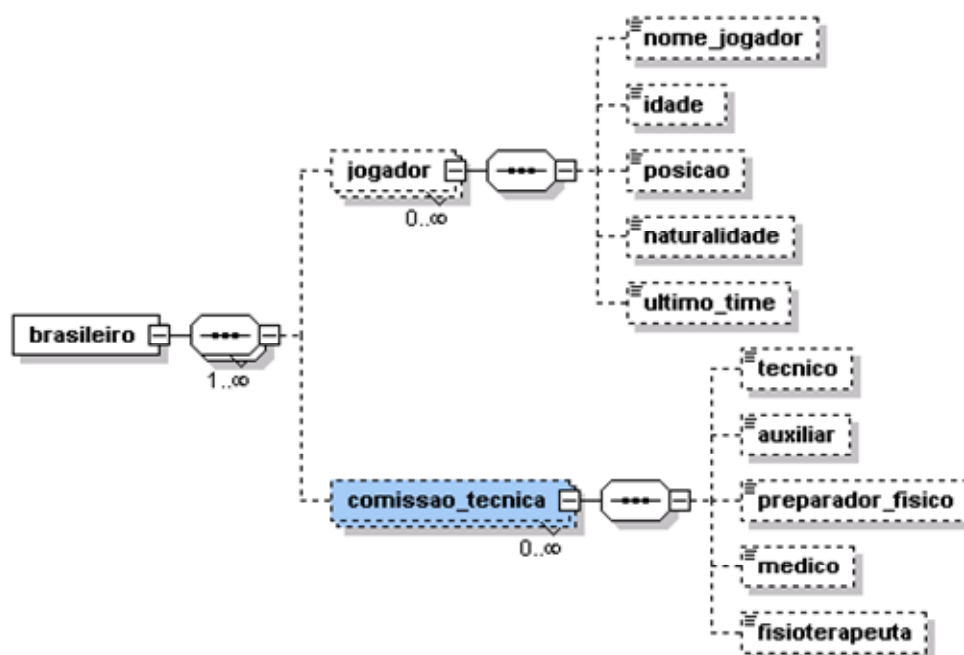
<xs:element name = "comissao_tecnica">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:subTreeSQL table = "corpo_tecnico" datasource = "serieB">
              <tsd:primaryKeyColumn>cod_comissao</tsd:primaryKeyColumn>
            </tsd:subTreeSQL>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:all>
      <xs:element name = "cod_comissao" type = "xs:integer">
        <xs:annotation>
          <xs:appinfo>
            <tsd:elementInfo>
              <tsd:physical>
                <tsd:map>
                  <tsd:nodeSQL column = "cod_corpo_tecnico"></tsd:nodeSQL>
                </tsd:map>
              </tsd:physical>
            </tsd:elementInfo>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>

```

As diferenças de mapeamento entre os dois esquemas definidos neste exemplo podem ser constatadas comparando praticamente qualquer trecho do código que represente o mapeamento de um mesmo elemento. Cada base de dados, mesmo estando relacionada ao mesmo domínio de conhecimento, apresenta nomes de tabelas e campos distintos e, conseqüentemente, mapeamentos distintos.

Outro esquema gerado pelo *Middleware* é o esquema global, que é a visão dos dados a partir da qual o usuário poderá fazer consultas às bases integradas. Basicamente, este esquema é composto dos elementos que formam as bases locais, desconsiderando as informações de mapeamento das mesmas, como mostra a Figura 5.7.





**Figura 5.7 – Representação do XML Schema global**

Pode-se observar que os campos *cod\_jogador* e *cod\_comissao* foram excluídos do esquema global. Sendo estes códigos seqüências, foi considerado que as informações representadas nestes campos não são relevantes para este exemplo de uso. O Anexo C apresenta o código do esquema global gerado pelo SIDATA.

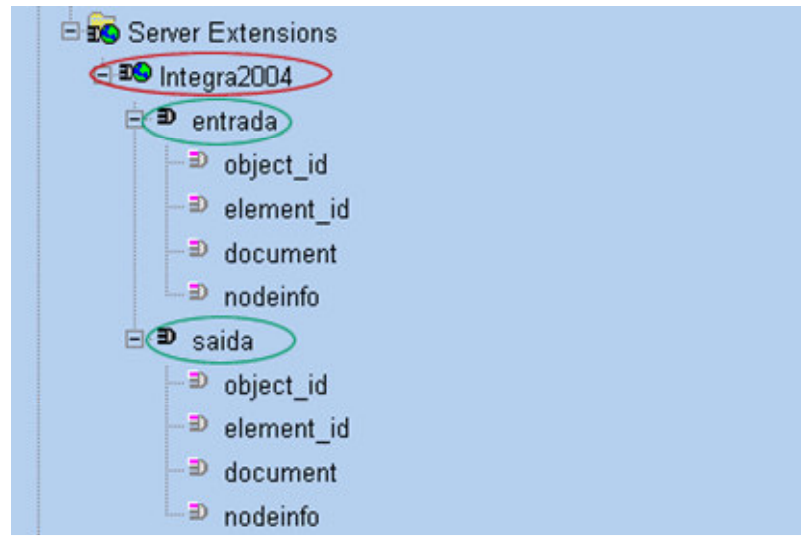
Com os esquemas criados, é chamado o método `EnviaTamino`, este tendo a função de se comunicar com o Tamino e enviar os esquemas ao mesmo de acordo com os parâmetros estabelecidos pelo usuário.

A próxima etapa do exemplo de uso é o processo de consulta aos dados integrados no Tamino. Como apresentado no Capítulo 3, existe uma série de formas de acesso ao Tamino porém, as consultas devem ser feitas através da linguagem X-Query, visto que esta é a única linguagem que o Tamino provê suporte para consultas a *XML Schemas* com elementos mapeados.

O objetivo do SIDATA é que o Tamino XML Server seja utilizado para consultas, sem que o usuário tenha que se preocupar em carregar os XML Schemas gerados (são automaticamente inseridos pelo Tamino), cabendo apenas realizar as consultas ao Tamino segundo os parâmetros estabelecidos quando da integração no *Middleware*. A única instalação que o usuário terá de fazer é a *Server Extension Integra2004*.

Este componente deve ser manualmente instalado em todas as bases de dados Tamino que forem utilizadas em consulta a XML Schemas gerados a partir do SIDATA, visto que é através dele que os dados do esquema global são passados aos esquemas locais e vice-versa.

A Figura 5.8 exibe a *Server Extension* Integra2004 instalada com suas funções entrada e saída.



**Figura 5.8 - Representação da SXT Integra2004 no Tamino Manager**

Com a *Server Extensions* instalada e considerando que os XML Schemas já estão integrados ao sistema, pode-se processar alguma consulta ao XML Schema global utilizando algum ambiente de consulta do Tamino, ou através de uma aplicação via API.

Para o presente exemplo de uso será realizado uma consulta aos dados dos jogadores utilizando o protocolo *http* da seguinte forma:

```
http://xavante/tamino/brasileirao2004/futebol?_XQL=brasileir  
o/jogador
```

O Quadro 5.5 exibe o resultado da consulta, destacando registros que foram retornados pela mesma. O Anexo D apresenta o código XML completo resultante da consulta.

**Quadro 5.5 – Registros retornados da consulta ao esquema global**

	⌘ nome_jogador	⌘ idade	⌘ posicao	⌘ naturalidade	⌘ ultimo_time
1	Marcelo Ramos	11/11/1972	Atacante	Belo Horizonte	Cruzeiro
2	Nilmar	12/07/1982	Atacante	Porto Alegre	Prata da casa
3	Rogério Ceni	01/06/1973	Goleiro	São Paulo	Prata da casa
4	Leonardo	09/09/1976	Zagueiro	Campinas	Ponte Preta
5	Simão	11/01/1973	Meio-Campo	São Leopoldo	Kashima
6	Ataliba	03/08/1973	Meio-Campo	Coritiba	Sport
7	Kuki	31	Atacante	São Luis	Tanaka
8	Robson	34	Atacante	Salvador	Santos
9	Valença	29	Zagueiro	Blumenau	Brusque
10	Lucio	30	Meia	Rio de Janeiro	Goiás

Assim como jogadores, consultas sobre comissão técnica também podem ser efetuadas, porém vale ressaltar que o Tamino ainda apresenta muitas limitações no que tange a consulta aos dados representados em XML *Schemas* que utilizam *Server Extensions*.

## 5.2. Considerações Finais

Este capítulo mostrou um exemplo de uso do SIDATA, apresentando os passos do processo de integração e destacando como o Tamino é utilizado neste processo. Apresenta também como os dados armazenados em bases relacionais são representados como XML *Schemas* e posteriormente consultados, sendo retornados ao usuário como documentos XML.

## 6. CONCLUSÃO

A massificação de informação, com a grande quantidade de dados disponíveis tanto na *web* quanto em redes internas de empresas/instituições permite que se tenha acesso a uma quantidade cada vez mais rica de informações, porém distribuídas de uma forma pouco organizada. Nessa realidade, o desafio de integrar diversas fontes de dados em um sistema único, nos quais essas informações possam ser acessadas e visualizadas de forma uniforme, tem sido amplamente estudado pela comunidade de Banco de Dados.

Este trabalho apresentou o SIDATA, um protótipo de um sistema de integração de dados que utiliza técnicas relacionadas à *web*, tendo XML como forma de representação e armazenamento dos dados integrados, e as ontologias como forma de remover as discrepâncias estruturais existentes entre as diferentes bases.

Tais características fazem com que o sistema provenha, além de facilidades de manipulação, outras características tais como flexibilidade, legibilidade e grande aplicabilidade em sistemas *web*. O armazenamento de documentos XML em sua forma nativa faz com que seja desnecessário qualquer tipo de conversão para outros formatos de dados. Através da manipulação de dados de forma nativa ganha-se tempo de resposta, menor esforço de administração e maior capacidade de integração de dados de sistemas legados [Tamino 2003]. Neste trabalho, utilizou-se o Tamino XML Server como plataforma de gerenciamento de dados, sendo que este utiliza XML e padrões relacionados para realizar o armazenamento e recuperação das informações.

Um dos objetivos do trabalho foi a análise da adequação de NXD como suporte a sistemas de integração. Com o desenvolvimento do SIDATA apoiado em um NXD, pôde-se observar uma série de pontos a serem considerados sobre a ferramenta e seu apoio ao processo de integração.

Dentre os pontos positivos pode-se destacar o suporte para o mapeamento de bases externas, a representação do resultado das consultas como documentos XML, o suporte às principais linguagens de programação existentes no mercado e o grande número de ferramentas oferecidas para a manipulação dos dados, além da representação eficiente de diversos tipos de dados, tais como textos, imagens, e-mail e sons.

Como pontos negativos encontrados no Tamino pode-se destacar a dificuldade no desenvolvimento das *Server Extensions*, que apresentam algumas particularidades que dificultam o seu desenvolvimento, bem como a consulta a *XML Schemas* que utilizam mapeamento ou mesmo *Server Extension*, que não pode ser realizado através de consultas XQuery.

Por fim, o SIDATA foi desenvolvido como uma alternativa aos sistemas de integração existentes, realizando essa tarefa da forma mais transparente possível para o usuário.

Como proposta de trabalhos futuros está o refinamento do módulo de integridade, tratando de maneira abrangente os problemas de integridade das bases, tanto de sua estrutura quanto do conteúdo das mesmas, além de um ambiente interativo no qual o usuário pode ajustar algum campo que não tenha seu significado semântico encontrado na ontologia.

Outra funcionalidade que pode ser adicionada à ferramenta é permitir ao usuário determinar a forma como os dados provenientes das bases serão tratados. No modelo atual todas as bases são mapeadas no Tamino, mas por uma questão de disponibilidade da fonte de dados ou por essa não ser alterada com frequência o usuário pode desejar que seu conteúdo fique armazenado fisicamente no Tamino, portanto é interessante que essa possibilidade seja oferecida no sistema.

Implementar a integração de outros tipos de SGBD, como Orientado a Objetos, por exemplo, visando tratar a heterogeneidade das bases é outra proposta de trabalho futuro, visto que será necessário adequar o mapeamento entre a base e o Tamino para que dê suporte à integração de outros tipos de SGBD.

## 7. REFERÊNCIAS BIBLIOGRÁFICAS

[Abiteboul 1997] Abiteboul, S. Querying Semistructured Data. Conference on Database Theory, 1997.

[Abiteboul et al. 1997] Abiteboul, S.; Quass, D.; Mchugh, J.; Widom, J.; Wiener, J., The Lorel query Language for Semistructured Data. International Journal on Digital Libraries, 1997.

[Abiteboul et al. 2000] Abiteboul S.; Buneman, P.; Suciu, D., Data on the Web: from relations to semistructured data and XML. San Francisco : Morgan Kaufman Publishers, 2000.

[Apache 2003] Apache Xindice [sd]. Disponível por [www](http://xml.apache.org/xindice/) em <http://xml.apache.org/xindice/>. Último acesso: abril/2003.

[Araneus 2003] The ARANEUS Project [sd]. Disponível por [www](http://www.dia.uniroma3.it/Araneus/) em <http://www.dia.uniroma3.it/Araneus/>. Último acesso: julho/2003.

[Arantes et al. 2003] Arantes, A., Laender, A., Silva, A. Materialização e Manutenção de Visões de Fontes de Dados Web [2000]. Disponível por [www](http://www.dcc.ufmg.br/pos/html/spg2000/anais/alissonr/alissonr.html) em <http://www.dcc.ufmg.br/pos/html/spg2000/anais/alissonr/alissonr.html>. Último acesso: abril/2003.

[Atzeni et al. 1997] Paolo Atzeni, Giansalvatore Mecca, Paolo Merlaldo. Semistructured e Structured Data in the Web: Going Back and Forth [1997]. Disponível por [www](http://citeseer.ist.psu.edu/cache/papers/cs/10278/http%3A%2F%2Fwww.dia.uniroma3.it%2FAraneus%2Fpublications%2Fsigrec97.pdf/atzeni97semistructured.pdf) em <http://citeseer.ist.psu.edu/cache/papers/cs/10278/http%3A%2F%2Fwww.dia.uniroma3.it%2FAraneus%2Fpublications%2Fsigrec97.pdf/atzeni97semistructured.pdf>. Último acesso: setembro/2003.

[Baru et al. 1998] Baru, C. et al. Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation. W3C Workshop on Query Language [1998]. Disponível em <http://www.w3.org/TandS/QL/QL98/pp/xmas.html>. Último acesso: novembro/2003.

[Berners-Lee et al. 1994] Berners-Lee, T.; Cailliau, A.; Luotinen, A.; Nielsen, H. F.; Secret, A.. The World Wide Web. Communication of the ACM, 37(8): 76-82, 1994.

[Berners-Lee et al. 2001] Berners-Lee, T.; Hendler, J.; Lassila, O., The Semantic Web. [2001]. Scientific American, New York, USA. Maio/2001.

[Bézivin 1998] Bézivin, J., Who's Afraid of Ontologies [1998]. Disponível por [WWW](http://www.metamodel.com/oopsla98-cdif-workshop/bezivin1) em <http://www.metamodel.com/oopsla98-cdif-workshop/bezivin1>. Último acesso: julho/2003.

[Bluestram 2003] Bluestram: Xstram 3.0, Native XML Database [sd]. Disponível por www em <http://www.bluestream.com/xstreamdb/>. Último acesso: outubro/2003.

[Bourret 2003] Bourret ,R. XML and databases [sd]. Disponível em <http://www.rpbourret.com/xml/XMLAndDatabases.htm>. Último acesso: junho/2003.

[Bray 1999 ] Bray, T. XML Namespaces by Example [1999]. Disponível em <http://www.xml.com/pub/a/1999/01/namespaces.html>. Último acesso: outubro/2002.

[Bray 2000] BRAY, Tim. Extensible Markup Language (XML) [2000]. Disponível por www em <http://www.w3.org/TR/2000/REC-xml-20001006.html>. Último acesso: abril/2003.

[Browne 2002] Browne, C., SQL Databases [sd]. Disponível por www em <http://www.ntlug.org/~cbbrowne/rdbmssql.html>. Último acesso: dezembro/2002.

[Carr 2001] Carr, D. XML-Native Databases [2001]. Disponível por www em <http://www.internetworld.com/magazine.php?inc=071501/07.15.01technology3.html>. Último acesso: julho/2003.

[Catarci 1997] Tiziana Catarci, Luca Iocchi, Daniele Nardi, Giuseppe Santucci Dipartimento Conceptual Views over the Web [1997]. Knowledge Representation Meets Databases.

[Chandrasekaran et al. 1999] Chandrasekaran, B.; Josephson, J. R.; Benjamins, V. R., What Are Ontologies, and Why Do We Need Them? [1999]. In IEEE Intelligent Systems. 1999. Magazine. pp 20 - 26.

[Cisco 2002 ] Cisco Systems, The Object Query Language (OQL) [2002]. Disponível por www em: <http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cw2k4mw/mwfm/mwfm201/mwfmto33oql.htm>. Último acesso: dezembro/2002.

[Daum & Merten 2002] Daum, B.; Merten, U., Arquitetura de Sistemas com XML. Alameda: Editora Campus, 2002, 441p.

[Dawis 2003] Dawis Data Management Systems and Knowledge Represenation, Tamino Documentation Overview [sd]. Disponível por www em <http://www.cs.uni-essen.de/dawis/teaching/ss2000/nsdb/tamino/help/overview.htm>. Último acesso: fevereiro/2003.

[Debye 2003] DEByE – Data Extraction by Example [sd]. Disponível por www: <http://www.lbd.dcc.ufmg.br/~debye/debye.html>. Último acesso: agosto/2003.

[Decker et al. 2000] Decker, S.; Mitra, P.; Melnik, S. Framework for the Semantic Web: An RDF Tutorial. In IEEE Internet Computing. 2000. Magazine. p.p. 68 - 73.

[Dw 2002] DW. The Data Warehousing Information Center [sd]. Disponível por www em <http://www.dwinfocenter.org/>. Último acesso: dezembro/2002.

[Dw-institute 2002] DW-INSTITUTE. The Premier Association for Data Warehousing & Business Intelligence Professionals [sd]. Disponível por www em <http://www.dw-institute.com/>. Último acesso: setembro/2002.

[Enoy 2003] Enoy Software Home Page Oficial [sd]. Disponível por www em <http://www.enosyssoftware.com/>. Último acesso: março/2003.

[ERP 2003a] Enterprise Resource Planning Research Center [sd]. Disponível por www em <http://www.cio.com/research/erp/edit/erpbasics.html>. Último acesso: abril/2003.

[ERP 2003b] Sistema ERP, introdução e desenvolvimento [sd]. Disponível por www em <http://www.erp.rg3.net/>. Último acesso: abril/2003

[Eyal 2001] Eyal, A., Milo T. Integrating and Customizing Heterogeneous E-Commerce Applications VLDB Journal: Very Large Data Bases [2001]. Disponível por www em <http://math.tau.ac.il/~milo/ecommerce.ps>. Último acesso: maio/2003.

[Goldman 2003] Roy Goldman, Sudarshan Chawathe, Arturo Crespo, Jason McHugh, A Standard Textual Interchange Format for the Object Exchange Model [sd]. Disponível por www em <http://www-db.stanford.edu/~mchughj/oemsyntax/oemsyntax.html>. Último acesso: dezembro/2002.

[Holland 2000] Holland, P. Virtues of a virtual data warehouse [2000]. Disponível por www em <http://itmanagement.earthweb.com/datbus/article.php/621401>. Último acesso: outubro/2002.

[Hunter 2004] Hunter, J. X Is for Xquery [sd]. Disponível por www em: <http://otn.oracle.com/oramag/oracle/03-may/o33devxml.html>. Último acesso: fevereiro/2004.

[Infoworld 2001] Infoworld, Technology of the Year [2001]. Disponível por www em <http://www.aps.com.ve/noticiasaps/premioinfoworld.htm>. Último acesso: março/2003.

[Inmon 2001] William H. Inmon and Robert H. Terdeman [2001]. The Evolution of The Corporate Information Factory. Disponível por www em <http://www.billinmon.com/library/whiteprs/cifevol1.pdf>. Último acesso: março/2003.

[Jdbc 2003] JDBC. Accessing Databases Using Java and JDBC [sd]. Disponível por www em <http://www.edm2.com/0607/msql3.html>. Último acesso: junho/2003.

[Jena 2004] *Jena 2 Ontology API* [sd]. Disponível por WWW em <http://jena.sourceforge.net/ontology/>. Último acesso: janeiro/2004.

[Kalinichenko 1999] Kalinichenko, L. Integration of Heterogeneous Semistructured Data Models in the Canonical One [1999]. Disponível por www em <http://citeseer.ist.psu.edu/kalinichenko99integration.html>. Último acesso: junho/2003.

[Kimball 1996] Kimball, R. The Data Warehouse Toolkit. John & Sons, 1996.



[Mcgrath 1999] Mcgrath S., XML, Aplicações Práticas. Rio de Janeiro: Editora Campus, 1999, 368p.

[McGuinness et al. 2002] McGuinness, D. L.; Fikes, R.; Hendler, J.; Stein, L. A., DAML+OIL: An Ontology Language for the Semantic Web. [2002]. Disponível por www em <http://dsonline.computer.org/0211/f/x5mcg.pdf>. Último acesso: fevereiro/2003.

[Mecca 1997] Giansalvator Mecca, Paolo Merialdo, Paolo Atzeni. To Weave the Web [1997]. Disponível por www em <http://citeseer.ist.psu.edu/cache/papers/cs/17070/ftp:zSzzSzftp.informatik.uni-trier.dezSzpubzSzUsers-DBISzSzleyzSzvldbzSzAtzeniMM97zSzvldb97cr.pdf/atzeni97to.pdf/>. Último acesso: novembro/2002.

[Mecca 1999] Mecca, G., Merialdo, P., Atzeni, P. Araneus in the Era of XML [1999]. Disponível por www em <http://citeseer.ist.psu.edu/cache/papers/cs/11803/ftp:zSzzSzftp.research.microsoft.comzSzpubzSzdebullzSzsept99-a4draft.pdf/data-engineering.pdf/>. Último acesso: setembro/2003.

[Metamatrix 2003] Metamatrix, Home Page Oficial [sd] . Disponível por www em <http://www.metamatrix.com>. Último acesso: agosto/2003.

[Molina 1995] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and Jennifer Widom. "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS". In Proceedings of the AAAI Symposium on Information Gathering, pp. 61-64, Stanford, California, March 1995.

[MS SQL Server 2003] MS SQL Server. *Microsoft SQL Server*. [sd]. Disponível por WWW em <http://www.microsoft.com/sql>. Último acesso: agosto/2003.

[Nimble 2003] Nimble Tecnology, Information Integration for Web Services and Applications [sd]. Disponível por www em: <http://www.nimble.com/>. Último acesso: agosto/2003.

[OEM 2003] OEM: Object Exchange Model [sd]. Disponível por www em <http://www.cs.huji.ac.il/~sdbi/1998/galel/oem.html>. Último acesso: março/2003.

[OIL 2004] OIL, Description of [sd]. Disponível por www em <http://www.ontoknowledge.org/oil/>. Último acesso: março/2004.

[ODMG 2000] ODMG, Object data Management Group Home Page [2000]. Disponível por www em <http://www.odmg.org/>. Último acesso: fevereiro/2004.

[Quilogic 2003] Quilogic: In Memory SQL / XML Database Technology for Universal Data Management [sd]. Disponível por www em <http://www.quilogic.cc/>. Último acesso outubro de 2003.

[Rainingdata 2003] RainingData: What is TigerLogic XDMS [sd]. Disponível por www em <http://www.rainingdata.com/products/tl/abouttl.html>. Último acesso: julho/2003.

[Robie 1999] Robie, J. XQL Tutorial [1999]. Disponível por www em <http://www.ibiblio.org/xql/xql-tutorial.html>. Último acesso: agosto/2003.

[Roy 2003] Prasan Roy, Jinesh Vora, Krithi Ramamritham, S. Seshadri, S. Sudarshan. Query Result Caching in Data Warehouses and Data Marts[sd]. Disponível por www em [http://umass.edu/db/papers/iitb\\_caching.os](http://umass.edu/db/papers/iitb_caching.os). Último acesso: janeiro/2003.

[SAX 2003] About SAX [sd]. Disponível por www em <http://sax.sourceforge.net/>. Último acesso: outubro/2003.

[Softwareag 2003] Software AG, Tamino XML Server [sd]. Disponível por www em <http://www2.softwareag.com/Corporate/products/tamino/>. Último acesso: dezembro/2003.

[Sonic 2003] Sonic XML Server. Documentation Overview [sd]. Disponível por www em [http://www.sonicsoftware.com/products/sonic\\_xml\\_server/index.ssp](http://www.sonicsoftware.com/products/sonic_xml_server/index.ssp). Último acesso: julho/2003.

[Sousa, 2003] Sousa, L. C. D.M.; Souza, F. F., Xsync-ML - Um Middleware Genérico para Migrar dados entre Bancos de Dados Relacionais e Documentos XML, Dissertação de Mestrado, Departamento de Informática - Universidade Federal de Pernambuco, fevereiro/2003.

[Staken 2001] Staken, K. Introduction to Native XML Databases [2001]. Disponível por www em <http://www.xml.com/pub/a/2001/10/31/nativexmlldb.html>. Último acesso: janeiro/2003.

[Tamino 2003] Tamino XML Server Documentation [sd]. Disponível por www em <http://www.cs.uni-essen.de/dawis/teaching/ss2000/nsdb/tamino/help/overview.htm>. Último acesso: novembro/2003.

[Tsimmis 2002] TSIMMIS. The Stanford- IBM Manager of Multiple Information Sources [sd]. Disponível por www em <http://www-db.stanford.edu/tsimmis/tsimmis.html>. Último acesso: outubro/2002.

[Vlist 2001] Vlist, E. V., Using W3C XML Schema [2001]. Disponível por www em <http://www.xml.com/pub/a/2000/11/29/schemas/part1.html>. Último acesso: maio/2004.

[W3C/DOM 2003] W3C World Wide Web Consortium, “Document Object Model (DOM)” [sd]. Disponível por www em <http://www.w3.org/DOM/>. Último acesso: maio/2003.

[W3C/Math 2002] W3C World Wide Web Consortium, “MathML” [sd]. Disponível por www em <http://www.w3.org/Math/>. Último acesso: março/2002.

[W3C/NXD 2003] W3C World Wide Web Consortium, “Introduction to Native XML Databases (NXD)” [sd]. Disponível por www em <http://www.w3.org/Style/XSL/>. Último acesso: agosto/2003.

[W3C/SOAP 2003] W3C World Wide Web Consortium, “Latest SOAP versions” [sd], Disponível por www em <http://www.w3.org/TR/soap/> . Último acesso: junho/2003.

[W3C/VML 2002] W3C World Wide Web Consortium, “Vector Markup Language (VML)” [sd]. Disponível por www em <http://www.w3.org/TR/1998/NOTE-VML-19980513/>. Último acesso: maio/2002.

[W3C/XML 2002] W3C World Wide Web Consortium, “Extensible Markup Language (XML)” [sd]. Disponível por www em <http://www.w3.org/XML/>. Último acesso: maio/2002.

[W3C/XML\_Query 2003] W3C World Wide Web Consortium, “XML Query (XQuery)” [sd]. Disponível por www em <http://www.w3.org/TR/xquery-requirements/> . Último acesso: agosto/2003.

[W3C/XML-QL 2003] W3C World Wide Web Consortium, “A Query Language for XML (XML-QL)” [sd]. Disponível por www em <http://www.w3.org/TR/xpath/> . Último acesso: agosto/2003.

[W3C/XPath 2003] W3C World Wide Web Consortium, “XML Path Language (XPath)” [sd], Disponível por www em <http://www.w3.org/TR/xpath/>. Último acesso: agosto/2003.

[W3C/XQuery 2003] W3C World Wide Web Consortium, “XQuery”, Disponível por www em <http://www.w3.org/TR/xquery/> [sd] . Último acesso: novembro/2003.

[W3C/XSD 2003] W3C World Wide Web Consortium, “XML Schema (XSD)” [sd]. Disponível por www em <http://www.w3.org/XML/Schema>. Último acesso: maio/2003.

[W3C/XSL 2003] W3C World Wide Web Consortium, “The Extensible Stylesheet Language Family (XSL)” [sd]. Disponível por www em <http://www.w3.org/Style/XSL/>. Último acesso: agosto/2003.

[W3Schools/DTD 2002] W3Schools, “DTD Tutorial” [sd]. Disponível por www em <http://www.w3schools.com/dtd/default.asp>. Último acesso: novembro/2002.

[W3Schools/SQL 2003] W3Schools, “SQL Tutorial” [sd] . Disponível por www em <http://www.w3schools.com/sql/default.asp>. Último acesso: março/2003

[W3Schools/WSDL 2002] W3Schools, “WSDL Tutorial” [sd] . Disponível por www em <http://www.w3schools.com/wsdl/default.asp>. Último acesso: novembro/2002.

[Webont 2003] Web-Ontology (WebOnt) Working Group. Disponível por www em <http://www.w3.org/2001/sw/WebOnt/>. Último acesso: fevereiro/2003.

[Whoweda 2003] Whoweda - The Warehousing and Data Mining Group [sd]. Disponível em: <http://mandolin.cais.ntu.edu.sg/~whoweda/>. Último acesso agosto/2003.

[Wiederhold 1983] Wiederhold, Gio: Database Design; McGraw-Hill Book Company, New York, NY, na Computer Science Series, Maio 1977, 678 páginas; Segunda edição, Janeiro 1983, 768 paginas; republicado na ACM Digital Library CD ROM, março/ 2002.

[XML-Journal 2001] XML Jornal, Best Product [2001]. Disponível por www em [http://www.softwareagusa.com/news/releases/2001/xml\\_journal\\_award.asp?PARENT=news](http://www.softwareagusa.com/news/releases/2001/xml_journal_award.asp?PARENT=news). Último acesso: março/2003.

[XMLDB 2004] XML:DB Initiative for XML Databases [sd]. Disponível por www em <http://www.xmldb.org/>. Último acesso: março/2004.

[Xmlspy 2003] Xmlspy, Documentation Overview [sd]. Disponível por www em [http://www.altova.com/products\\_ide.html](http://www.altova.com/products_ide.html). Último acesso: setembro/2003.

[XQuery 2004] XQuery.com, Articles, Specifications, Mailing List and Vendors [sd]. Disponível por www em <http://www.xquery.com/>. Último acesso: março/2004.

[XSLT 2002] XSLT. Documentation Overview [sd]. Disponível por www em <http://www.xslt.com/>. Último acesso: novembro/2002.

## 8. ANEXOS

### 8.1. Anexo A – Ontologia usada no exemplo de uso

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#">

  <daml:Ontology rdf:about="">
    <dc:title>&quot; Ontologiasobre Futebol &quot;</dc:title>
    <dc:date></dc:date>
    <dc:creator>Marcelo da Silveira Sielder</dc:creator>
    <dc:description></dc:description>
    <dc:subject></dc:subject>
    <daml:versionInfo></daml:versionInfo>
  </daml:Ontology>

  <!-- ##### CLASSES ##### -->

  <daml:Class rdf:about="#jogador">
    <rdfs:label>jogador</rdfs:label>
    <rdfs:comment>Classe jogador, referente a tabela jogador</rdfs:comment>
    <rdfs:subClassOf>
      <daml:Class rdf:about="#root"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <daml:Restriction>
        <!-- Alguma coisa está na classe R se satisfaz as restrições anexadas, e vice-versa -->
        <daml:onProperty rdf:resource="#nome_jogador"/>
        <!-- daml:onProperty é usado na classe daml:Restriction para especificar a propriedade sobre a qual a
restrrição agirá -->
        <daml:hasClass>
          <!-- a propriedade daml:hasClass especifica que pelo menos um valor da propriedade de ser do
rdf:type para satisfazer uma restrição-->
          <daml:Class rdf:about="#jogador"/>
        </daml:hasClass>
      </daml:Restriction>
    </rdfs:subClassOf>
  </daml:Class>

  <daml:Class rdf:about="#atleta">
    <rdfs:label>atleta</rdfs:label>
    <rdfs:comment>Classe atleta referente a tabela atleta</rdfs:comment>
    <rdfs:subClassOf>
      <daml:Class rdf:about="#root"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <daml:Restriction>
        <daml:onProperty rdf:resource="#nome"/>
        <daml:hasClass>
          <daml:Thing>
        </daml:hasClass>
      </daml:Restriction>
    </rdfs:subClassOf>
    <daml:sameClassAs>
      <daml:Class rdf:about="#jogador"/>
    </daml:sameClassAs>
  </daml:Class>
```

```

<daml:Class rdf:about="#equipe">
  <rdfs:label>equipe</rdfs:label>
  <rdfs:comment>Classe equipe</rdfs:comment>
  <rdfs:subClassOf>
    <daml:Class rdf:about="#root"/>
  </rdfs:subClassOf>
  <daml:sameClassAs>
    <daml:Class rdf:about="#time"/>
  </daml:sameClassAs>
</daml:Class>

<daml:Class rdf:about="#time">
  <rdfs:label>time</rdfs:label>
  <rdfs:comment>Classe time</rdfs:comment>
  <rdfs:subClassOf>
    <daml:Class rdf:about="#root"/>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:about="#comissao_tecnica">
  <rdfs:label>equipe</rdfs:label>
  <rdfs:comment>Classe equipe</rdfs:comment>
  <rdfs:subClassOf>
    <daml:Class rdf:about="#root"/>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:about="#corpo_tecnico">
  <rdfs:label>equipe</rdfs:label>
  <rdfs:comment>Classe equipe</rdfs:comment>
  <rdfs:subClassOf>
    <daml:Class rdf:about="#root"/>
  </rdfs:subClassOf>
  <daml:sameClassAs>
    <daml:Class rdf:about="#comissao_tecnica"/>
  </daml:sameClassAs>
</daml:Class>

<daml:Class rdf:about="#root">
  <rdfs:label>root</rdfs:label>
  <rdfs:comment>Classe principal</rdfs:comment>
</daml:Class>

<!--##### PROPIEDADES #####-->

<!--## JOGADOR ## -->

<daml:ObjectProperty rdf:about="#posicao">
  <rdfs:label>idade</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:domain>
    <daml:Class rdf:about="#jogador"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#ultimo_time">
  <rdfs:label>idade</rdfs:label>

```

```

    <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:domain>
    <daml:Class rdf:about="#jogador"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#clube_anterior">
  <rdfs:label>data_nasc</rdfs:label>
  <daml:samePropertyAs rdf:resource="#ultimo_time"/>
  <rdfs:domain>
    <daml:Class rdf:about="#atleta"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

  <daml:ObjectProperty rdf:about="#naturalidade">
    <rdfs:label>idade</rdfs:label>
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <rdfs:domain>
      <daml:Class rdf:about="#jogador"/>
    </rdfs:domain>
    <rdfs:range>
      <xsd:literal/>
    </rdfs:range>
  </daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#cidade_natal">
  <rdfs:label>data_nasc</rdfs:label>
  <daml:samePropertyAs rdf:resource="#naturalidade"/>
  <rdfs:domain>
    <daml:Class rdf:about="#atleta"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#idade">
  <rdfs:label>idade</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:domain>
    <daml:Class rdf:about="#jogador"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#data_nasc">
  <rdfs:label>data_nasc</rdfs:label>
  <rdfs:comment>Propriedade data_nasc da tabela atleta, que deve ser equivalente a idade</rdfs:comment>
  <daml:samePropertyAs rdf:resource="#idade"/>
  <rdfs:domain>
    <daml:Class rdf:about="#atleta"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

```

```

<daml:ObjectProperty rdf:about="#nome_jogador">
  <rdfs:label>nome_jogador</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:domain>
    <daml:Class rdf:about="#jogador"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#nome">
  <rdfs:label>nome</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <daml:samePropertyAs rdf:resource="#nome_jogador"/>
  <rdfs:domain>
    <daml:Class rdf:about="#atleta"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#cod_jogador">
  <rdfs:label>nome_jogador</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:domain>
    <daml:Class rdf:about="#jogador"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#cod_atleta">
  <rdfs:label>nome</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <daml:samePropertyAs rdf:resource="#cod_jogador"/>
  <rdfs:domain>
    <daml:Class rdf:about="#atleta"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<!-- TIME -->

<daml:ObjectProperty rdf:about="#nome_time">
  <rdfs:label>estado</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>

  <rdfs:domain>
    <daml:Class rdf:about="#time"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#nome_equipe">
  <rdfs:label>UF</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <daml:samePropertyAs rdf:resource="#nome_time"/>

```



```

<rdfs:domain>
  <daml:Class rdf:about="#equipe"/>
</rdfs:domain>
<rdfs:range>
  <xsd:literal/>
</rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#UF">
  <rdfs:label>UF</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <daml:samePropertyAs rdf:resource="#estado"/>
  <rdfs:domain>
    <daml:Class rdf:about="#equipe"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#estado">
  <rdfs:label>estado</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:domain>
    <daml:Class rdf:about="#time"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#cidade">
  <rdfs:label>cidade</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:domain>
    <daml:Class rdf:about="#time"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#municipio">
  <rdfs:label>municipio</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <daml:samePropertyAs rdf:resource="#cidade"/>
  <rdfs:domain>
    <daml:Class rdf:about="#equipe"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#cod_time">
  <rdfs:label>estado</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>

  <rdfs:domain>
    <daml:Class rdf:about="#time"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>

```

```

    </rdfs:range>
  </daml:ObjectProperty>

  <daml:ObjectProperty rdf:about="#cod_equipe">
    <rdfs:label>UF</rdfs:label>
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <daml:samePropertyAs rdf:resource="#cod_time"/>
    <rdfs:domain>
      <daml:Class rdf:about="#equipe"/>
    </rdfs:domain>
    <rdfs:range>
      <xsd:literal/>
    </rdfs:range>
  </daml:ObjectProperty>

<!-- Comissão Técnica-->
  <daml:ObjectProperty rdf:about="#tecnico">
    <rdfs:label>cidade</rdfs:label>
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <rdfs:domain>
      <daml:Class rdf:about="#comissao_tecnica"/>
    </rdfs:domain>
    <rdfs:range>
      <xsd:literal/>
    </rdfs:range>
  </daml:ObjectProperty>

  <daml:ObjectProperty rdf:about="#treinador">
    <rdfs:label>municipio</rdfs:label>
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <daml:samePropertyAs rdf:resource="#tecnico"/>
    <rdfs:domain>
      <daml:Class rdf:about="#corpo_tecnico"/>
    </rdfs:domain>
    <rdfs:range>
      <xsd:literal/>
    </rdfs:range>
  </daml:ObjectProperty>

  <daml:ObjectProperty rdf:about="#auxiliar">
    <rdfs:label>cidade</rdfs:label>
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <rdfs:domain>
      <daml:Class rdf:about="#comissao_tecnica"/>
    </rdfs:domain>
    <rdfs:range>
      <xsd:literal/>
    </rdfs:range>
  </daml:ObjectProperty>

  <daml:ObjectProperty rdf:about="#assistente">
    <rdfs:label>municipio</rdfs:label>
    <rdfs:comment><![CDATA[]]></rdfs:comment>
    <daml:samePropertyAs rdf:resource="#auxiliar"/>
    <rdfs:domain>
      <daml:Class rdf:about="#corpo_tecnico"/>
    </rdfs:domain>
    <rdfs:range>
      <xsd:literal/>
    </rdfs:range>
  </daml:ObjectProperty>

  <daml:ObjectProperty rdf:about="#preparador_fisico">
    <rdfs:label>cidade</rdfs:label>
    <rdfs:comment><![CDATA[]]></rdfs:comment>

```

```

<rdfs:domain>
  <daml:Class rdf:about="#comissao_tecnica"/>
</rdfs:domain>
<rdfs:range>
  <xsd:literal/>
</rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#fisicultor">
  <rdfs:label>municipio</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <daml:samePropertyAs rdf:resource="#preparador_fisico"/>
  <rdfs:domain>
    <daml:Class rdf:about="#corpo_tecnico"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#medico">
  <rdfs:label>cidade</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:domain>
    <daml:Class rdf:about="#comissao_tecnica"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#doutor">
  <rdfs:label>municipio</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <daml:samePropertyAs rdf:resource="#medico"/>
  <rdfs:domain>
    <daml:Class rdf:about="#corpo_tecnico"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#cod_comissao">
  <rdfs:label>cidade</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <rdfs:domain>
    <daml:Class rdf:about="#comissao_tecnica"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:about="#cod_corpo_tecnico">
  <rdfs:label>municipio</rdfs:label>
  <rdfs:comment><![CDATA[]]></rdfs:comment>
  <daml:samePropertyAs rdf:resource="#cod_comissao"/>
  <rdfs:domain>
    <daml:Class rdf:about="#corpo_tecnico"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:literal/>
  </rdfs:range>
</daml:ObjectProperty>

</rdf:RDF>

```

## 8.2. Anexo B – XML Schemas representando as bases de dados

### (a) Base SerieA

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="Base_serieA.tsd">
        <tsd:collection name="futebol"/>
        <tsd:doctype name="serieA">
          <tsd:logical>
            <tsd:content>closed</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="serieA">
    <xs:annotation>
      <xs:documentation/>
    </xs:annotation>
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="comissao_tecnica">
          <xs:annotation>
            <xs:appinfo>
              <tsd:elementInfo>
                <tsd:physical>
                  <tsd:map>
                    <tsd:subTreeSQL table="comissao_tecnica" datasource="serieA">
                      <tsd:primaryKeyColumn>cod_comissao</tsd:primaryKeyColumn>
                    </tsd:subTreeSQL>
                  </tsd:map>
                </tsd:physical>
              </tsd:elementInfo>
            </xs:appinfo>
          </xs:annotation>
          <xs:complexType>
            <xs:all>
              <xs:element name="cod_comissao" type="xs:integer">
                <xs:annotation>
                  <xs:appinfo>
                    <tsd:elementInfo>
                      <tsd:physical>
                        <tsd:map>
                          <tsd:nodeSQL column="cod_comissao"/>
                        </tsd:map>
                      </tsd:physical>
                    </tsd:elementInfo>
                  </xs:appinfo>
                </xs:annotation>
              </xs:element>
              <xs:element name="tecnico" type="xs:string">
                <xs:annotation>
```

```

        <xs:appinfo>
          <tsd:elementInfo>
            <tsd:physical>
              <tsd:map>
                <tsd:nodeSQL column="tecnico"/>
              </tsd:map>
            </tsd:physical>
          </tsd:elementInfo>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="auxiliar" type="xs:string">
      <xs:annotation>
        <xs:appinfo>
          <tsd:elementInfo>
            <tsd:physical>
              <tsd:map>
                <tsd:nodeSQL column="assistente"/>
              </tsd:map>
            </tsd:physical>
          </tsd:elementInfo>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="preparador_fisico" type="xs:string">
      <xs:annotation>
        <xs:appinfo>
          <tsd:elementInfo>
            <tsd:physical>
              <tsd:map>
                <tsd:nodeSQL column="preparador_fisico"/>
              </tsd:map>
            </tsd:physical>
          </tsd:elementInfo>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="medico" type="xs:string">
      <xs:annotation>
        <xs:appinfo>
          <tsd:elementInfo>
            <tsd:physical>
              <tsd:map>
                <tsd:nodeSQL column="medico"/>
              </tsd:map>
            </tsd:physical>
          </tsd:elementInfo>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="fisioterapeuta" type="xs:string">
      <xs:annotation>
        <xs:appinfo>
          <tsd:elementInfo>
            <tsd:physical>
              <tsd:map>
                <tsd:nodeSQL column="fisioterapeuta"/>
              </tsd:map>
            </tsd:physical>
          </tsd:elementInfo>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:all>
</xs:complexType>
</xs:element>

```

```

<xs:element name="jogador">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:subTreeSQL table="jogador" datasource="serieA">
              <tsd:primaryKeyColumn>null</tsd:primaryKeyColumn>
            </tsd:subTreeSQL>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:all>
      <xs:element name="cod_jogador" type="xs:integer">
        <xs:annotation>
          <xs:appinfo>
            <tsd:elementInfo>
              <tsd:physical>
                <tsd:map>
                  <tsd:nodeSQL column="cod_jogador"/>
                </tsd:map>
              </tsd:physical>
            </tsd:elementInfo>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="nome_jogador" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <tsd:elementInfo>
              <tsd:physical>
                <tsd:map>
                  <tsd:nodeSQL column="nome_jogador"/>
                </tsd:map>
              </tsd:physical>
            </tsd:elementInfo>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="idade" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <tsd:elementInfo>
              <tsd:physical>
                <tsd:map>
                  <tsd:nodeSQL column="data_nasc"/>
                </tsd:map>
              </tsd:physical>
            </tsd:elementInfo>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="posicao" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <tsd:elementInfo>
              <tsd:physical>
                <tsd:map>
                  <tsd:nodeSQL column="posicao"/>
                </tsd:map>
              </tsd:physical>
            </tsd:elementInfo>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>

```

```

        </xs:annotation>
      </xs:element>
      <xs:element name="naturalidade" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <tsd:elementInfo>
              <tsd:physical>
                <tsd:map>
                  <tsd:nodeSQL column="naturalidade"/>
                </tsd:map>
              </tsd:physical>
            </tsd:elementInfo>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="ultimo_time" type="xs:string">
        <xs:annotation>
          <xs:appinfo>
            <tsd:elementInfo>
              <tsd:physical>
                <tsd:map>
                  <tsd:nodeSQL column="clube_anterior"/>
                </tsd:map>
              </tsd:physical>
            </tsd:elementInfo>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>

```

## (b) Base SerieB

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema elementFormDefault = "qualified" xmlns:tsd =
"http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition" xmlns:xs =
"http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name = "Base_serieB.tsd">
        <tsd:collection name = "futebol"></tsd:collection>
        <tsd:doctype name = "serieB">
          <tsd:logical>
            <tsd:content>closed</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name = "serieB">
    <xs:annotation>
      <xs:documentation></xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:choice maxOccurs = "unbounded">
        <xs:element name = "comissao_tecnica">
          <xs:annotation>
            <xs:appinfo>
              <tsd:elementInfo>
                <tsd:physical>
                  <tsd:map>
                    <tsd:subTreeSQL table = "corpo_tecnico" datasource = "serieB">
                      <tsd:primaryKeyColumn>cod_comissao</tsd:primaryKeyColumn>
                    </tsd:subTreeSQL>
                  </tsd:map>
                </tsd:physical>
              </tsd:elementInfo>
            </xs:appinfo>
          </xs:annotation>
        </xs:complexType>
      </xs:choice>
      <xs:all>
        <xs:element name = "cod_comissao" type = "xs:integer">
          <xs:annotation>
            <xs:appinfo>
              <tsd:elementInfo>
                <tsd:physical>
                  <tsd:map>
                    <tsd:nodeSQL column = "cod_corpo_tecnico"></tsd:nodeSQL>
                  </tsd:map>
                </tsd:physical>
              </tsd:elementInfo>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name = "tecnico" type = "xs:string">
          <xs:annotation>
            <xs:appinfo>
              <tsd:elementInfo>
                <tsd:physical>
                  <tsd:map>
                    <tsd:nodeSQL column = "treinador"></tsd:nodeSQL>
                  </tsd:map>
                </tsd:physical>
              </tsd:elementInfo>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



```

        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
<xs:element name = "auxiliar" type = "xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:nodeSQL column = "auxiliar"></tsd:nodeSQL>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
<xs:element name = "preparador_fisico" type = "xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:nodeSQL column = "fisicultor"></tsd:nodeSQL>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
<xs:element name = "medico" type = "xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:nodeSQL column = "doutor"></tsd:nodeSQL>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
</xs:all>
</xs:complexType>
</xs:element>
<xs:element name = "jogador">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
        <tsd:physical>
          <tsd:map>
            <tsd:subTreeSQL table = "atleta" datasource = "serieB">
              <tsd:primaryKeyColumn>cod_atleta</tsd:primaryKeyColumn>
            </tsd:subTreeSQL>
          </tsd:map>
        </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
<xs:complexType>
  <xs:all>
    <xs:element name = "cod_jogador" type = "xs:integer">
      <xs:annotation>
        <xs:appinfo>

```

```

    <tsd:elementInfo>
    <tsd:physical>
    <tsd:map>
    <tsd:nodeSQL column = "cod_atleta"></tsd:nodeSQL>
    </tsd:map>
    </tsd:physical>
    </tsd:elementInfo>
  </xs:appinfo>
</xs:annotation>
</xs:element>
<xs:element name = "nome_jogador" type = "xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
      <tsd:physical>
      <tsd:map>
      <tsd:nodeSQL column = "nome"></tsd:nodeSQL>
      </tsd:map>
      </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
<xs:element name = "idade" type = "xs:integer">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
      <tsd:physical>
      <tsd:map>
      <tsd:nodeSQL column = "idade"></tsd:nodeSQL>
      </tsd:map>
      </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
<xs:element name = "posicao" type = "xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
      <tsd:physical>
      <tsd:map>
      <tsd:nodeSQL column = "posicao"></tsd:nodeSQL>
      </tsd:map>
      </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
<xs:element name = "naturalidade" type = "xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>
      <tsd:physical>
      <tsd:map>
      <tsd:nodeSQL column = "cidade_natal"></tsd:nodeSQL>
      </tsd:map>
      </tsd:physical>
      </tsd:elementInfo>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
<xs:element name = "ultimo_time" type = "xs:string">
  <xs:annotation>
    <xs:appinfo>
      <tsd:elementInfo>

```

```
<tsd:physical>
  <tsd:map>
    <tsd:nodeSQL column = "ultimo_time"></tsd:nodeSQL>
  </tsd:map>
</tsd:physical>
</tsd:elementInfo>
</xs:appinfo>
</xs:annotation>
</xs:element>
</xs:all>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
```

## 8.3. Anexo C – Esquema global gerado no exemplo de uso

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="esquema_brasileirao">
        <tsd:collection name="basquete"/>
        <tsd:doctype name="brasileiro">
          <tsd:logical>
            <tsd:content>closed</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="brasileiro">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="jogador" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nome_jogador" type="xs:string" minOccurs="0"/>
              <xs:element name="idade" type="xs:string" minOccurs="0"/>
              <xs:element name="posicao" type="xs:string" minOccurs="0"/>
              <xs:element name="naturalidade" type="xs:string" minOccurs="0"/>
              <xs:element name="ultimo_time" type="xs:string" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="colecacao" type="xs:string"/>
            <xs:attribute name="basetamino" type="xs:string"/>
            <xs:attribute name="basesODBC" type="xs:string"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="comissao_tecnica" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="tecnico" type="xs:string" minOccurs="0"/>
              <xs:element name="auxiliar" type="xs:string" minOccurs="0"/>
              <xs:element name="preparador_fisico" type="xs:string" minOccurs="0"/>
              <xs:element name="medico" type="xs:string" minOccurs="0"/>
              <xs:element name="fisioterapeuta" type="xs:string" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="colecacao" type="xs:string"/>
            <xs:attribute name="basetamino" type="xs:string"/>
            <xs:attribute name="basesODBC" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## 8.4. Anexo D – XML resultante da consulta às bases integradas

```
<?xml version="1.0" encoding="windows-1252"?>
<ino:response xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
xmlns:xql="http://metalab.unc.edu/xql/">
  <xql:query>brasileiro/jogador</xql:query>
  <ino:message ino:returnValue="0">
    <ino:messageline>fetching cursor</ino:messageline>
  </ino:message>
  <xql:result>
    <jogador>
      <nome_jogador>Marcelo Ramos</nome_jogador>
      <idade>11/11/1972</idade>
      <posicao>Atacante</posicao>
      <naturalidade>Belo Horizonte</naturalidade>
      <ultimo_time>Cruzeiro</ultimo_time>
    </jogador>
    <jogador>
      <nome_jogador>Nilmar</nome_jogador>
      <idade>12/07/1982</idade>
      <posicao>Atacante</posicao>
      <naturalidade>Porto Alegre</naturalidade>
      <ultimo_time>Prata da casa</ultimo_time>
    </jogador>
    <jogador>
      <nome_jogador>Rogério Ceni</nome_jogador>
      <idade>01/06/1973</idade>
      <posicao>Goleiro</posicao>
      <naturalidade>São Paulo</naturalidade>
      <ultimo_time>Prata da casa</ultimo_time>
    </jogador>
    <jogador>
      <nome_jogador>Leonardo</nome_jogador>
      <idade>09/09/1976</idade>
      <posicao>Zagueiro</posicao>
      <naturalidade>Campinas</naturalidade>
      <ultimo_time>Ponte Preta</ultimo_time>
    </jogador>
    <jogador>
      <nome_jogador>Simão</nome_jogador>
      <idade>11/01/1973</idade>
      <posicao>Meio-Campo</posicao>
      <naturalidade>São Leopoldo</naturalidade>
      <ultimo_time>Kashima</ultimo_time>
    </jogador>
    <jogador>
      <nome_jogador>Ataliba</nome_jogador>
      <idade>03/08/1973</idade>
      <posicao>Meio-Campo</posicao>
      <naturalidade>Coritiba</naturalidade>
      <ultimo_time>Sport</ultimo_time>
    </jogador>
    <jogador>
      <nome_jogador>Kuki</nome_jogador>
      <idade>31</idade>
      <posicao>Atacante</posicao>
      <naturalidade>São Luis</naturalidade>
      <ultimo_time>Tanaka</ultimo_time>
    </jogador>
    <jogador>
      <nome_jogador>Robson</nome_jogador>
      <idade>34</idade>
    </jogador>
  </xql:result>
</ino:response>
```

```

        <posicao>Atacante</posicao>
        <naturalidade>Salvador</naturalidade>
        <ultimo_time>Santos</ultimo_time>
    </jogador>
    <jogador>
        <nome_jogador>Valença</nome_jogador>
        <idade>29</idade>
        <posicao>Zagueiro</posicao>
        <naturalidade>Blumenau</naturalidade>
        <ultimo_time>Brusque</ultimo_time>
    </jogador>
    <jogador>
        <nome_jogador>Lucio</nome_jogador>
        <idade>30</idade>
        <posicao>Meia</posicao>
        <naturalidade>Rio de Janeiro</naturalidade>
        <ultimo_time>Goiás</ultimo_time>
    </jogador>
</xql:result>
<ino:cursor ino:handle="1"/>
<ino:message ino:returnValue="0">
    <ino:messageline>cursor fetched</ino:messageline>
</ino:message>
</ino:response>

```



SERVIÇO PÚBLICO FEDERAL  
UNIVERSIDADE FEDERAL DE PERNAMBUCO  
CENTRO DE INFORMÁTICA  
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata de Defesa de Dissertação de Mestrado do  
Centro de Informática da Universidade Federal  
de Pernambuco, em 30 de agosto de 2004.

Ao trigesimo dia do mês de agosto do ano dois mil e quatro, no Centro de Informática da Universidade Federal de Pernambuco, às quinze horas, teve início a defesa de dissertação do Mestrado em Ciência da Computação intitulada **"Sistema de Integração de Esquemas Apoiado em SGBD XML Nativo"** do candidato **Marcelo da Silveira Siedler**, o qual já havia preenchido as demais condições exigidas para a obtenção do grau de mestre. A Banca Examinadora composta pelos professores Décio Fonseca, pertencente ao Centro de Informática desta Universidade, Cristina Dutra de Aguiar Ciferri, pertencente ao Departamento de Informática da Universidade Estadual de Maringá, e Fernando da Fonseca de Souza, pertencente ao Centro de Informática desta Universidade, sendo o primeiro presidente da Banca Examinadora e o último orientador do trabalho de dissertação, resolveu: **Aprovar por unanimidade e dar o prazo de trinta dias para as entrega da versão final do trabalho.** E para constar lavrei a presente ata que vai por mim assinada e pela Banca Examinadora. Recife, 30 de agosto de 2004.

Maria Lília Pinheiro de Freitas  
(secretária)

Prof. Décio Fonseca  
(primeiro examinador)

Profa. Cristina Dutra de Aguiar Ciferri  
(segunda examinadora)

Prof. Fernando da Fonseca de Souza  
(terceiro examinador)

Confere Com o Original  
Recife, 17/09/06  
  
Ivoneide da Silva Ribeiro  
Assistente em Administração  
Pós - Graduação em Ciência  
da Computação