# Integração de Dados na Web

**Ana Carolina Salgado**
**Bernadette Farias Lóscio**
Centro de Informática
Universidade Federal de Pernambuco

# Integração de Dados na Web

**Ana Carolina Salgado, Bernadette Farias Lóscio**

Centro de Informática – Universidade Federal de Pernambuco (UFPE)
CP. 7851, CEP 50732-970, Recife - PE - Brasil

{acs,bfl}@cin.ufpe.br

***Abstract.*** *Several Web applications require integrated access to various data sources ranging from traditional databases to semistructured data repositories. Many of the problems encountered in Web information integration systems are similar to those in database integration systems. However, Web data integration have some additional problems, including: large number of data sources, little metadata about the data sources, large degree of source autonomy and high level of irregularity in the data structure. This text presents an overview of research on data integration. Semistructured data models and query languages are also presented. XML is presented, being focused XML schemas and XML query languages.*

***Resumo.*** *Muitas aplicações para Web requerem acesso integrado a várias fontes de dados, que podem ser desde bases de dados convencionais até repositórios de dados semi-estruturados. Muitos dos problemas encontrados na integração de dados na Web são similares aos problemas encontrados na integração de bancos de dados tradicionais. Porém, a integração de dados na Web apresenta alguns problemas adicionais: o grande número de fontes de dados, a falta de metadados sobre estas fontes, o alto grau de autonomia das fontes de dados e a grande irregularidade na estrutura dos dados. Neste texto é apresentada uma visão geral sobre as pesquisas na área de integração de dados. Também são apresentados os modelos de dados e linguagens de consulta para dados semi-estruturados. A linguagem XML é apresentada, destacando-se os esquemas e linguagens de consulta para dados XML.*

## 1. Introdução

A Web é um dos meios mais utilizados para a disseminação de informações e abrange diversas aplicações, como: bibliotecas digitais, museus virtuais, catálogos de serviços e produtos, informações governamentais, entre outros. A Web facilita o acesso aos dados provenientes de diversas áreas, que incluem não apenas centros acadêmicos, mas também empresas públicas e privadas, instituições militares e governamentais. A facilidade de acesso aos dados e o grande volume de informações disponível na Web revolucionaram o desenvolvimento dos sistemas de informação e motivaram o crescimento de aplicações que freqüentemente necessitam integrar dados heterogêneos e distribuídos em diferentes fontes de dados.

A Web pode ser vista como um enorme Banco de Dados (BD) onde documentos são disponibilizados para leitura, sendo alguns destes documentos gerados a partir de consultas a um BD. Comparando as tecnologias Web e BD, constata-se que a Web proporciona uma infra-estrutura global e um conjunto de padrões para dar suporte à troca de documentos, um formato de apresentação para hipertexto (HTML), interfaces

com o usuário bem construídas para recuperação de documentos e um novo formato, XML, para troca de dados através de uma estrutura. Por outro lado, a tecnologia de BD oferece técnicas de armazenamento e linguagens de consulta a dados altamente estruturados, modelos e métodos para estruturar dados, mecanismos para a manutenção da integridade e consistência de dados e um novo modelo, de dados semi-estruturados, que abranda os rigores dos sistemas de BD altamente estruturados. O poder de abstração desenvolvido pela comunidade de banco de dados pode prover a chave para diminuir a complexidade da Web.

O problema de integração de dados já foi amplamente discutido no contexto de bancos de dados, especificamente quando foram analisados os aspectos relativos à interoperabilidade de sistemas heterogêneos. Mais recentemente, este problema foi ampliado abordando o contexto da Web [Ambite 1998], [Arens 1993], [Baru 1999]. 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. Entretanto, existem algumas diferenças que precisam ser consideradas quando fontes de dados Web são integradas. Primeiro, o número de fontes de dados pode ser muito grande, o que dificulta os processos de integração de esquemas e resolução de conflitos. Segundo, as fontes de dados são muito dinâmicas e assim a adição ou remoção de fontes de dados deve ser feita de maneira a minimizar o impacto na visão integrada. Terceiro, as fontes de dados são muito heterogêneas, podendo ser desde sistemas de gerenciamento de bancos de dados até simples arquivos. Finalmente, as fontes de dados podem ser não estruturadas ou semi-estruturadas e, na maioria das vezes, não fornecem informações suficientes para a integração de esquemas.

Neste texto é apresentada uma visão geral sobre as pesquisas na área de integração de dados. Na seção 2 são apresentadas as duas principais abordagens para integração de dados (virtual e materializada) e algumas arquiteturas para integração de dados, entre elas destacam-se as arquiteturas clássicas, como bancos de dados federados, e arquiteturas mais adequadas para integração de dados na Web, como a arquitetura de mediadores e a arquitetura de *data warehouse*. Na seção 2 também são discutidos os principais desafios enfrentados para integração de dados distribuídos em diferentes fontes de dados autônomas e heterogêneas, além de alguns projetos que têm como objetivo o desenvolvimento de sistemas para integração de dados na Web. Na seção 3 são definidos os conceitos associados a dados semi-estruturados, destacando os modelos de dados e linguagens de consulta. Na seção 4 a linguagem XML é apresentada, destacando-se os esquemas para documentos XML e linguagens de consulta para dados XML.

## 2. Integração de Dados

O objetivo de um sistema de integração de dados é oferecer aos usuários uma *interface* uniforme de acesso à diferentes fontes de dados, de forma que os usuários definam consultas especificando "o que" se deseja saber e o sistema determine "onde" a informação pode ser encontrada e, em seguida, apresente as respostas para as consultas do usuário.

Ao contrário do que ocorre nos sistemas de bancos de dados distribuídos, um sistema de integração de dados trabalha com fontes de dados que podem ser autônomas e dinâmicas, ou seja, as fontes de dados continuam a suportar aplicações locais,

podendo atualizar tanto os dados quanto o esquema que disponibilizam para o sistema de integração. Como conseqüência da natureza dinâmica das fontes de dados é necessário que o sistema de integração seja extensível, ou seja, o sistema deve ser capaz de adaptar a visão oferecida aos usuários, bem como gerenciar os novos tipos de informação oferecidos pelas fontes de dados.

Um sistema de integração de dados também precisa lidar com a heterogeneidade das fontes de dados, as quais, na maioria das vezes, foram projetadas independentemente utilizando diferentes modelos de dados e diferentes representações para os mesmos conceitos do mundo real. Desta forma, um sistema de integração de dados deve ser capaz de sobrepor a heterogeneidade de forma a oferecer uma visão integrada dos dados. É importante notar que um sistema de integração não precisa oferecer necessariamente uma única visão integrada, ou seja, o sistema de integração de dados pode oferecer diferentes visões integradas que são definidas de acordo com as necessidades dos usuários.

Nesta seção é apresentada uma visão geral sobre as pesquisas na área de integração de dados, enfatizando a integração de dados na Web. Na seção 2.1 as duas principais abordagens para integração de dados (virtual e materializada) são discutidas. Na seção 2.2 algumas arquiteturas para integração de dados são apresentadas, entre elas destacam-se as arquiteturas clássicas, como bancos de dados federados, e arquiteturas mais adequadas para integração de dados na Web, como a arquitetura de mediadores e a arquitetura de *data warehouse*. Na seção 2.3 são discutidos os principais desafios enfrentados para integração de dados distribuídos em diferentes fontes de dados autônomas e heterogêneas. Na seção 2.4 alguns projetos que têm como objetivo o desenvolvimento de sistemas para integração de dados na Web são apresentados.

## 2.1 Abordagens para Integração de Dados

Uma importante decisão na construção de um sistema de integração de dados na Web é a escolha da abordagem a ser utilizada: virtual ou materializada. Na abordagem virtual, as informações são extraídas das fontes somente quando consultas são requisitadas. Por outro lado, na abordagem materializada, as informações são recuperadas, integradas e armazenadas em um repositório, de maneira que as consultas podem ser avaliadas diretamente neste repositório, sem a necessidade de acessar as fontes de dados. A Tabela I apresenta os passos que descrevem mais detalhadamente o processo de extração de informações de cada uma dessas abordagens.

Tanto a abordagem virtual quanto a abordagem materializada possui vantagens e desvantagens, sendo adequadas para situações diferentes. Se por um lado uma das principais vantagens da abordagem virtual é que as informações recuperadas estão sempre atualizadas, por outro lado, esta abordagem torna-se ineficiente quando as fontes de dados estão temporariamente inacessíveis. 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. A abordagem virtual não é adequada quando os passos de tradução e integração das informações ficam muito caros ou quando as fontes de dados freqüentemente ficam inacessíveis.

No caso da abordagem materializada, sua principal vantagem é o fato de que as informações integradas estão disponíveis imediatamente para consultas, não sendo

necessário acessar diretamente as fontes de dados locais. Entretanto, quando a abordagem materializada é adotada é preciso manter a consistência entre os dados armazenados no repositório de dados e os dados das fontes de dados de origem. Esta abordagem é mais adequada quando: i) são requisitadas porções específicas e previsíveis da informação disponível, ii) usuários demandam alto desempenho de consulta, sem requerer que o estado da informação seja o mais atualizado, iii) é necessário acessar cópias privadas de informação, de forma que estas possam ser modificadas, anotadas e resumidas, iv) usuários querem guardar informações que não são mantidas nas fontes de dados, tais como informações históricas. Esta abordagem não é adequada quando as informações integradas precisam estar sempre atualizadas.

Certamente, existem situações onde uma abordagem é mais adequada do que a outra. Contudo, para futuras aplicações mais complexas e em grande escala, ambas as abordagens serão necessárias. Para manipular tais aplicações, o sistema de integração de informação ideal é aquele no qual algumas informações são recuperadas, processadas e integradas previamente e, em seguida, armazenadas em um repositório do sistema, enquanto que outras informações são recuperadas e processadas somente quando consultas são requisitadas.

| Abordagem virtual | Abordagem materializada |
|---|---|
| **Passo 1:** o sistema de integração recebe uma consulta, determina o conjunto de fontes de dados necessárias para respondê-la e gera as subconsultas apropriadas para cada fonte de dados. | **Passo 1:** as informações relevantes de cada fonte de dados são extraídas, traduzidas e filtradas, em seguida estas informações são integradas com as informações relevantes de outras fontes de dados e armazenadas em um repositório central. |
| **Passo 2:** o sistema coleta os resultados das subconsultas de cada fonte de dados, executa as traduções apropriadas, filtra e integra as informações, e em seguida, retorna o resultado final para o usuário. | **Passo 2:** quando uma consulta é requisitada ao sistema de integração, a consulta é avaliada diretamente no repositório, sem a necessidade de acessar as fontes de dados originais. |

**Tabela I - Processo de extração de informações das abordagens virtual e materializada.**

## 2.2 Arquiteturas para Integração de Dados

Diversas arquiteturas já foram propostas para solucionar o problema de integração de dados. Neste trabalho, as arquiteturas de integração são classificadas em: i) Arquiteturas Clássicas para Integração de Dados e ii) Arquiteturas para Integração de Dados na Web. É importante notar que as arquiteturas classificadas como arquiteturas para integração de dados na Web também podem ser utilizadas para integração de dados distribuídos em bancos de dados convencionais. A seguir, estas arquiteturas são apresentadas com mais detalhes.

### 2.2.1 Arquiteturas Clássicas para Integração de Dados

As soluções mais tradicionais para o problema de integração de dados 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 podem ser acessadas de maneira uniforme e transparente através deste esquema global. Como o esquema global é estático e oferece uma visão única dos dados integrados, esta abordagem não é muito adequada quando as fontes de dados a serem integradas são dinâmicas ou quando os consumidores da informação integrada têm diferentes requisitos. Além disso, como o esquema global deve suportar diferentes requisitos de um grande número de usuários, o esquema pode tornar-se muito grande, sendo difícil criá-lo e mantê-lo [Batini 1986].

Outra abordagem clássica para integração de dados distribuídos e heterogêneos é a abordagem federada [Kim 1995], [Sheth 1990]. 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.

### 2.2.2 Arquiteturas para Integração de Dados na Web

De forma alternativa aos tradicionais sistemas de gerenciamento de banco de dados, que utilizam a arquitetura cliente/servidor para a troca de informações, o processamento de dados no contexto da Web é baseado em uma abordagem "*multitier*", como ilustrado na Fig.1 [Abiteboul 2000]. A camada de mais baixo nível consiste nas fontes de dados, também chamadas de servidores, que podem ser servidores de bancos de dados convencionais, sistemas legados, servidores de arquivos ou qualquer aplicação que produz dados. A camada mais alta é a camada do cliente e consiste em interfaces de usuários ou aplicações que consomem dados. Entre estas duas camadas existe uma coleção de camadas intermediárias que oferecem o "*middleware*" - *software* para transformar e integrar dados.

Pesquisadores interessados na integração de dados têm trabalhado constantemente no desenvolvimento do *middleware*. Neste sentido, duas arquiteturas se destacam: a arquitetura de mediadores e a arquitetura de *data warehouse*. Na arquitetura de mediadores, que implementa a abordagem virtual de integração de dados, as consultas são submetidas ao sistema de integração através do mediador, que é responsável por decompor as consultas em subconsultas a serem enviadas às fontes de dados. Além disso, o mediador deve ser capaz de integrar os resultados das subconsultas e devolver os dados integrados ao cliente. Por outro lado, a arquitetura de *data warehouse* está associada à abordagem materializada onde os dados são recuperados, integrados e armazenados em um repositório de dados, de forma que as consultas podem ser avaliadas diretamente neste repositório. A seguir, estas arquiteturas são apresentadas com mais detalhes.

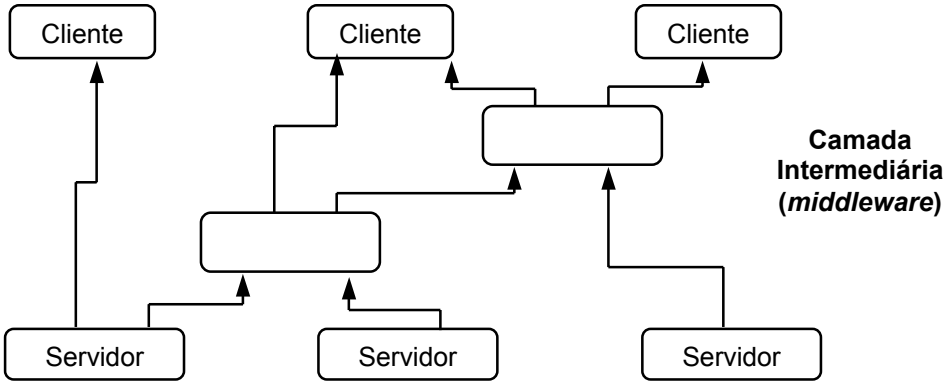


**Fig. 1 - Arquitetura Multi-camada.**

### 2.2.2.1. Arquitetura de 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 1992]. Os mediadores oferecem uma visão integrada sobre dados distribuídos em múltiplas fontes de dados e disponibilizam um esquema para esta visão, de forma que consultas e atualizações podem ser feitas através deste esquema.

Na arquitetura de mediadores (Fig. 2) [Abiteboul 2000], o acesso aos dados distribuídos em múltiplas fontes de dados é efetuado através de consultas que são submetidas ao sistema, via mediador, e este as transforma em subconsultas a serem enviadas às fontes de dados. As subconsultas geradas pelo mediador devem ser traduzidas para linguagens de consultas de cada fonte de dados componente. Ao final, os resultados das consultas são traduzidos e a resposta é devolvida ao usuário. Outros componentes desta arquitetura são os tradutores, que convertem tanto os dados das fontes de dados para um modelo de dados comum, como também consultas de aplicações em consultas específicas da fonte de dados correspondente.

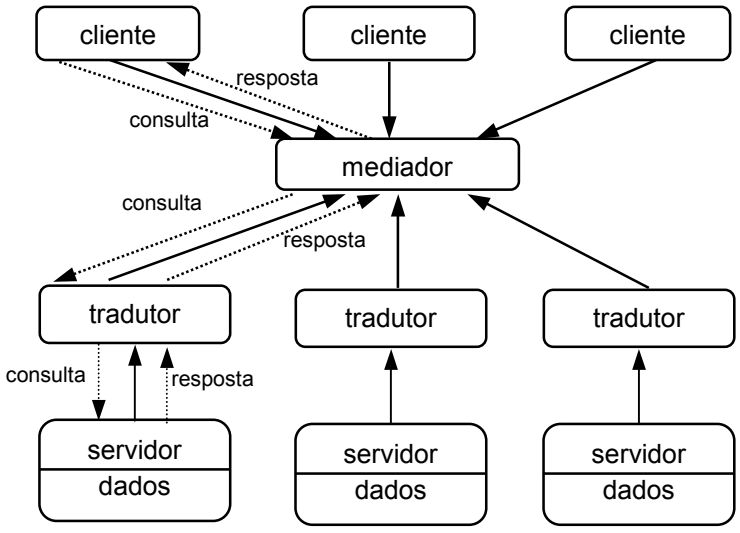


**Fig. 2 - Arquitetura de Mediadores.**

Inicialmente, o uso de mediadores era restrito apenas para responder consultas que requerem a extração e a combinação de dados em múltiplas fontes de dados. Em [Vidal 1999] também foi tratado o problema de atualização de múltiplas fontes de dados através de mediadores. Assim como as consultas, as atualizações submetidas ao sistema, via mediador, devem ser traduzidas em atualizações a serem executadas nas fontes de dados correspondentes. A arquitetura de mediadores foi proposta originalmente para integração de dados distribuídos em múltiplos bancos de dados. Atualmente, muitas pesquisas [Ambite 1998], [Arens 1993], [Baru 1999], [Castano 1999] estão sendo desenvolvidas para adaptar esta arquitetura a fim de possibilitar que ela possa ser utilizada em sistemas de integração de dados na Web.

#### 2.2.2.2 Arquitetura de Data Warehouse

Outra arquitetura que tem sido muito utilizada para integrar dados distribuídos em múltiplas fontes de dados é a arquitetura de *data warehouse* [Gupta 1996], [Widom 1995]. Nesta arquitetura, como mostra a Fig. 3 [Abiteboul 2000], os dados são recuperados, integrados e armazenados em um repositório de dados, obtendo-se assim uma visão materializada dos dados. Desta forma as consultas podem ser avaliadas diretamente no repositório, sem a necessidade de acessar as fontes de dados.

Um dos principais problemas a serem considerados nesta arquitetura consiste em manter o repositório de dados consistente com os dados das fontes de dados [Gupta 1995a], [Widom 1995]. Como as fontes de dados permanecem ativas e continuam a suportar aplicações locais, é possível que seus dados sofram atualizações. Basicamente, existem duas opções para a manutenção do *data warehouse*:

- *Rematerialização da visão*: o conteúdo do *data warehouse* é descartado e a visão é novamente materializada de acordo com os novos dados das fontes de dados.
- *Manutenção incremental*: mudanças nos dados das fontes de dados locais são propagadas incrementalmente para o data warehouse. A propagação das atualizações pode ser feita utilizando técnicas de banco de dados ativos [Zhou 1996], [Ceri 1991].

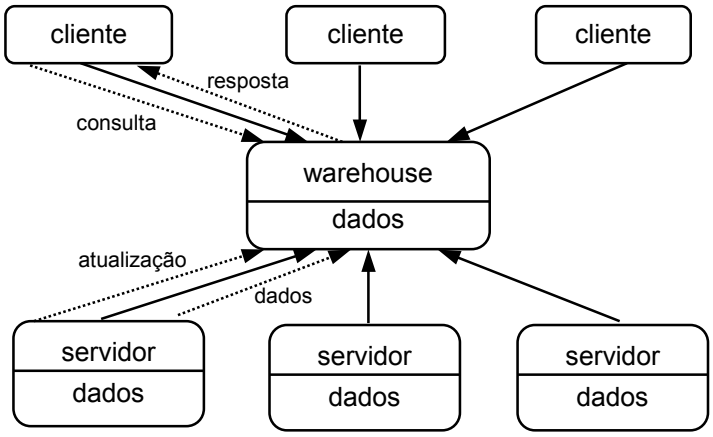


**Fig. 3 - Arquitetura de Data Warehouse.**

Um dos fatores que determina qual estratégia de manutenção deve ser utilizada é a capacidade de atualização das fontes de dados, ou seja, a habilidade das fontes de dados de origem enviarem suas modificações por meio de regras através de uma rede, de modo que as visões materializadas possam ser atualizadas apropriadamente. Assim,

as fontes de dados podem ser suficientemente ativas, possuírem atividade restrita ou não serem capazes de enviar qualquer notificação de mudança. As fontes são suficientemente ativas se puderem enviar qualquer modificação através de uma rede. A atividade restrita acontece quando apenas mensagens simples puderem ser transmitidas. Quando as fontes de dados não suportam atividade de manutenção, apenas a rematerialização da visão é possível.

Além do conjunto de visões materializadas, dados adicionais podem ser armazenados no repositório de dados, a fim de reduzir o custo de manutenção das visões inicialmente materializadas ou para otimizar o desempenho de alguma consulta [Gupta 1995b]. A presença de dados auxiliares no repositório de dados pode tornar as visões materializadas *self-maintainable* [Tompa 1988], [Huyn 1997], ou seja, visões capazes de serem mantidas sem precisar recuperar informações nas fontes de dados de origem.

Mesmo que a inclusão de dados adicionais não torne o conjunto de visões materializadas *self-maintainable*, os dados adicionais podem reduzir a necessidade de acesso às fontes de dados durante a manutenção das visões. Reduzir a necessidade de acesso às fontes de dados pode evitar anomalias de atualização. Estas anomalias podem ocorrer quando a seguinte seqüência de fatos acontece: i) Consultas são enviadas às fontes de dados em decorrência de alguma atualização que deve ser feita nas visões materializadas, como parte do processo de manutenção das visões; ii) Alguma relação de origem que possui dados relevantes a esta consulta é modificada antes de enviar a resposta para a consulta; iii) O resultado da consulta é recebido, porém corresponde a outro estado do banco de dados. Dessa forma, as visões materializadas podem ficar inconsistentes.

Muitos trabalhos já abordaram o problema de manutenção de visões materializadas no contexto de bancos de dados relacionais e bancos de dados orientados a objetos. Mais recentemente, este problema tem sido abordado no contexto dos dados semi-estruturados [Zhuge 1998], [Abiteboul 1998].

## 2.3 Desafios na Integração de Dados

Existem muitos desafios que precisam ser vencidos na construção de um sistema de integração de dados, entre eles destacam-se [Levy 1999]: a modelagem dos dados e a definição da visão integrada, a reformulação de consultas e a construção de tradutores. A seguir, são discutidos esses desafios, enfatizando os problemas encontrados no desenvolvimento de sistemas de integração de dados na Web. Ainda nesta seção, os problemas associados à manutenção dos sistemas de integração de dados são discutidos.

### 2.3.1 Modelagem de Dados

Um sistema de integração de dados provê acesso integrado a fontes de dados distribuídas e heterogêneas, que podem ser desde sistemas gerenciadores de banco de dados até simples arquivos. O primeiro passo no desenvolvimento de um sistema de integração de dados consiste em definir o esquema da visão integrada, o qual descreve os dados das fontes de dados locais que são relevantes para os usuários do sistema. Utiliza-se o termo visão integrada para designar um esquema de banco de dados, cujos elementos (por exemplo: relações ou classes) são obtidos a partir da integração de elementos de um ou mais esquemas de fontes de dados já especificados. É importante lembrar que a visão integrada pode ser virtual ou materializada.

Basicamente, existem dois enfoques para a definição do esquema da visão integrada: i) Visão Global: cada componente no esquema da visão integrada tem uma correspondência com o esquema da fonte de dados, tornando a reformulação de consultas mais simples. ii) Visão Local: cada informação na fonte de dados tem uma correspondência com um componente do esquema da visão integrada (por exemplo: relação ou classe), facilitando a manutenção das fontes de dados.

Além do esquema da visão integrada, o desenvolvimento de um sistema de integração de dados também requer que sejam fornecidas descrições das fontes de dados. Estas descrições (metadados) são cruciais para o sucesso de um sistema de integração de dados e devem, preferencialmente, ser extraídas automaticamente. Idealmente, os metadados de uma fonte de dados devem incluir: a identidade da fonte de dados (nome, tipo e localização); descrições do conteúdo (principais tópicos cobertos nas informações disponíveis) e informações sobre o esquema (estrutura dos dados, nomes das estruturas, nomes e tipos dos atributos). Quando o problema de integração de dados na Web é considerado, a tarefa de obter os metadados das fontes de dados locais torna-se mais complexa, porque na maioria das vezes as fontes de dados Web não apresentam um esquema definido. Para sobrepor este problema, mecanismos para extração e evolução de esquemas das fontes de dados Web devem ser desenvolvidos, a fim de possibilitar que essas fontes sejam integradas a outras.

Além das descrições das fontes de dados, também devem ser especificadas as correspondências entre os elementos do esquema integrado e os elementos dos esquemas locais. O processo de identificação de correspondências entre conceitos de esquemas de diferentes fontes de dados não é uma tarefa fácil, e torna-se mais complexa à medida que o número de fontes de dados a serem integradas aumenta. Para ajudar no processo de descoberta das correspondências, várias ferramentas já foram desenvolvidas [Navathe 1986], [Savasere 1991]. A identificação de correspondências pode ser enriquecida com o uso de um simples dicionário, para identificar palavras sinônimas, ou até mesmo, dicionários mais sofisticados e bases de conhecimento de conceitos e termos (*thesaurus* [Mirbel 1995] ou ontologias [Russel 1995]), para explicar os termos do domínio da aplicação e como eles estão relacionados. Para estabelecer as correspondências entre os componentes dos esquemas, podem ser utilizadas assertivas de correspondência [Spaccapietra 1994], [Vidal 2001] que são tipos especiais de restrições de integridade usadas para especificar que a semântica de algumas partes de um esquema está de alguma forma relacionada com a semântica de algumas partes de outro(s) esquema(s).

Outro problema que precisa ser solucionado diz respeito à heterogeneidade das fontes de dados a serem integradas. As fontes de dados são heterogêneas devido a diferenças a nível físico (diferentes plataformas de *hardware* e *software*), nível lógico (diferentes modelos de dados) e nível conceitual (diferentes esquemas e conceitos). Diferentes representações de um mesmo conceito podem acontecer, ou porque o modelo suporta diferentes representações ou porque projetistas têm diferentes percepções da realidade. A multiplicidade de possíveis representações de um conceito do mundo real é chamada de relativismo semântico [Hull 1997].

Para sobrepor a heterogeneidade estrutural das fontes de dados, os sistemas de integração usam um modelo de dados comum para representar o conteúdo e a estrutura das fontes de dados. Por exemplo, o projeto TSIMMIS [Chawathe 1994], que tem como

objetivo prover meios para a integração de dados heterogêneos e distribuídos, usa o modelo OEM [Papakontantinou 1995] baseado em grafos autodescritíveis. Mais recentemente, devido à flexibilidade de XML [Bray 1998] para representar dados estruturados e dados semi-estruturados, e à facilidade para converter os dados para XML, surgiu um grande interesse em usar XML como modelo de dados comum para troca e integração de dados.

### 2.3.2 Reformulação de consultas

Como discutido anteriormente, o usuário de um sistema de integração submete consultas baseadas no esquema integrado, ao invés de submeter consultas diretamente aos esquemas locais. Quando o enfoque virtual é adotado, as consultas submetidas ao sistema devem ser traduzidas em consultas a serem executadas diretamente nas fontes de dados locais. Sendo assim, um sistema de integração de dados deve ter mecanismos para reformular as consultas submetidas ao sistema de integração em consultas a serem executadas nas fontes de dados locais. Esta reformulação é feita com base nas descrições das fontes de dados e nas correspondências entre o esquema da visão integrada e os esquemas das fontes de dados locais. Além de garantir que a reformulação esteja semanticamente correta (i.e. as respostas obtidas das fontes correspondem à resposta correta para a consulta), também é importante garantir que fontes de dados irrelevantes não estejam sendo acessadas (i.e. fontes de dados que não são capazes de contribuir com respostas para a consulta), a fim de obter um melhor desempenho na execução das consultas.

### 2.3.3 Construção de Tradutores

Um tradutor é um *software* que tem como tarefa converter os dados das fontes de dados para o modelo de dados comum e converter consultas de aplicações em consultas específicas da fonte de dados correspondente. Uma das grandes dificuldades em gerar tradutores é que eles são específicos para cada tipo de fonte de dados. Além disso, desenvolver *software* para fazer traduções entre modelos de dados não é uma tarefa muito simples. As pesquisas nesta área buscam desenvolver técnicas e ferramentas para automatizar o processo de implementação de tradutores [Hammer 1998], [Kushmerick 1997].

### 2.3.4 Manutenção do sistema de integração

As fontes de dados que fornecem os dados para o sistema de integração, geralmente, são autônomas e dinâmicas, ou seja, continuam a suportar aplicações locais, podendo sofrer alterações tanto nos dados quanto no esquema que disponibilizam para o sistema de integração. Além das fontes de dados, as aplicações que fazem uso do sistema de integração também podem ser dinâmicas, ou seja, podem alterar seus requisitos em relação ao sistema.

A manutenção do sistema de integração pode envolver tarefas como a manutenção de tradutores e a manutenção das visões integradas quando novas fontes de dados são adicionadas ao sistema. Além disso, o sistema também precisa ser atualizado quando ocorrem atualizações nos esquemas das fontes de dados locais ou quando fontes de dados que participam do sistema forem removidas ou ficarem inacessíveis [Nica 1999]. Quando o sistema de integração de dados adota a abordagem materializada,

também deve ser levada em consideração a manutenção das visões integradas materializadas.

## 2.4 Sistemas de Integração de Dados

Um sistema de integração de dados provê acesso integrado a fontes de dados distribuídas e heterogêneas, oferecendo ao usuário a ilusão de um sistema de informação centralizado e homogêneo. Ao utilizar um sistema de integração de dados, o usuário não precisa ter conhecimento dos detalhes técnicos do sistema, ou seja, o usuário não precisa saber onde os dados estão fisicamente armazenados, os modelos de dados utilizados para representação dos dados e as interfaces de consulta utilizadas para recuperá-los. Idealmente, o sistema de integração disponibiliza uma *interface* uniforme através da qual o usuário pode expressar consultas sobre as diferentes fontes de dados. Internamente, um sistema de integração adota um modelo de dados e uma linguagem de consultas ricos o suficiente para suportar os modelos de dados e linguagens de consulta das diversas fontes de dados a serem integradas. Além disso, um sistema de integração de dados precisa ter mecanismos para: i) ocultar as diferenças entre as fontes de dados, ii) identificar as fontes que contêm informações relevantes e iii) combinar os resultados das consultas.

Um sistema de integração de dados deve ser capaz de adaptar-se às mudanças que ocorrem tanto no ambiente quanto nos requisitos dos usuários. Neste sentido, um sistema de integração deve ser extensível para poder adaptar-se a novas fontes de dados, novas interfaces, protocolos e formatos. No caso dos sistemas de integração de dados na Web, os sistemas devem ser capazes de lidar com um número exponencialmente crescente de fontes de dados. Capacidade de adaptação também é uma característica desejável para um sistema de integração de dados, porque um sistema genérico pode não ser capaz de manipular diferentes domínios de aplicação.

Vários sistemas de integração de dados são descritos na literatura, incluindo: TSIMMIS [Chawathe 1994], HERMES [Subrahmanian 1995], SIMS [Arens 1993], INFOMASTER [Genesereth 1997], ARIADNE [Ambite 1998] e MOMIS [Bergamaschi 1998].

Alguns desses sistemas, como o TSIMMIS e o HERMES, além de fornecer mecanismos para integração de dados também oferecem ferramentas para facilitar o processo de integração. O TSIMMIS não tem como objetivo executar uma integração de dados completamente automatizada que oculta todas as diversidades dos usuários mas, ao invés disso, fornecer ferramentas para auxiliar as atividades de integração e processamento de informações. O HERMES fornece um conjunto de ferramentas para auxiliar a construção de mediadores. Algumas ferramentas são úteis para a construção do código do mediador e outras são úteis para a construção do código de acesso às fontes de dados.

Outros sistemas, como o SIMS, foram propostos para integração de dados armazenados em diferentes bancos de dados, e posteriormente, suas idéias foram adaptadas para o contexto Web. A adaptação das idéias do SIMS para a Web deu origem ao sistema ARIADNE, que é um sistema para extração e integração de dados de fontes de dados Web semi-estruturadas. ARIADNE é um sistema de integração que permite que os usuários criem agentes de informação para a Web. Usando as ferramentas de modelagem oferecidas pelo ARIADNE, um desenvolvedor de aplicações

toma como ponto de partida um conjunto de fontes Web (páginas HTML semi-estruturadas, que podem estar localizadas em múltiplos sites Web) e cria uma visão unificada destas fontes de dados.

INFOMASTER é um sistema de integração de dados que tem como núcleo um facilitador que determina dinamicamente como responder às consultas submetidas pelos usuários, minimizando o número de fontes de dados utilizadas e solucionando a heterogeneidade entre as fontes de dados. O INFOMASTER é capaz de conectar-se a uma grande variedade de fontes de dados, incluindo bancos de dados relacionais e algumas fontes de dados Web.

MOMIS (*Mediator envirOnment for Multiple Information Sources*) é um *framework* para extração e integração de dados estruturados e semi-estruturados. Uma linguagem orientada a objetos chamada ODL-I3, derivada a partir do padrão ODMG, é introduzida para extração de informação. A integração de informações é executada de forma semi-automática, explorando o conhecimento de um *Thesaurus* (definido pelo *framework*) e descrições dos esquemas das fontes de dados definidas em ODL-I3. O processo de integração origina uma visão integrada virtual das fontes de dados locais (esquema global), para a qual regras de mapeamento de restrições de integridade são especificadas a fim de permitir a manipulação da heterogeneidade. O sistema MOMIS, baseado em uma arquitetura convencional tradutor/mediador, provê métodos e ferramentas para o gerenciamento de dados em sistemas de informação baseados na Web usando a interface CORBA-2.

## 3. Dados semi-estruturados

Os dados disponíveis na Web podem ter uma estrutura bem definida, como os dados provenientes de bancos de dados tradicionais (relacionais/orientados a objeto), mas também podem ser dados completamente não estruturados, como imagens e textos. Entre estes dois extremos estão os dados semi-estruturados caracterizados por sua estrutura irregular, algumas vezes implícita e capaz de evoluir de forma imprevisível, e por permitir a representação de dados incompletos [Abiteboul 1997a], [Buneman 1997]. Pesquisas recentes têm sido desenvolvidas com o objetivo de estender técnicas tradicionais de banco de dados para a manipulação de dados semi-estruturados. A maioria destas pesquisas tem focalizado no desenvolvimento de modelos de dados e linguagens de consulta para dados semi-estruturados, discutidos nesta seção.

Esta seção apresenta o estado da arte nas pesquisas sobre dados semi-estruturados. Na seção 3.1 as principais características dos dados semi-estruturados e a motivação para o seu uso são apresentadas. Na seção 3.2 é apresentada uma visão geral dos modelos de dados semi-estruturados e o modelo de dados OEM é discutido. Na seção 3.3 são discutidas as linguagens de consulta para dados semi-estruturados. Na seção 3.4 é apresentado o conceito de *DataGuide*.

### 3.1 Visão Geral

Dados armazenados em bancos de dados tradicionais, tais como relacionais ou orientados a objeto, são estruturados, i.e. um esquema é sempre especificado e nenhum dado pode ser armazenado no banco de dados se não estiver de acordo com a descrição especificada pelo esquema do banco de dados. Esta abordagem é adequada para aplicações tradicionais, onde um esquema de banco de dados é previamente definido, os

dados são regulares e se adaptam perfeitamente ao esquema. Contudo, esta abordagem apresenta diversas limitações na modelagem de aplicações avançadas, que requerem a representação e o gerenciamento de informações para as quais um esquema preciso não está disponível. Isto pode acontecer por várias razões: a estrutura existe, mas não é conhecida; o usuário decide não considerar a estrutura, por exemplo por motivos de navegação; a estrutura pode estar implícita (um texto formatado), mas não é tão rígida quanto em um banco de dados tradicional; o esquema freqüentemente muda, portanto ele é ignorado. Para atender a estes requisitos, modelos de dados semi-estruturados são mais adequados do que os modelos de dados adotados em BDs tradicionais.

Os dados semi-estruturados apresentam diversas características que os distinguem dos dados estruturados, entre elas se destacam:

- Os dados semi-estruturados apresentam uma estrutura irregular e implícita.
- A estrutura dos dados pode ser parcial, ou seja, uma parte dos dados pode não ter estrutura (ex: bitmaps) e outros podem ter uma estrutura "fraca" (ex: textos).
- Ao contrário do sistema de tipos rígido das aplicações de BD, os tipos são apenas indicativos.
- Os esquemas para dados semi-estruturados são definidos a posteriori, ao contrário do que ocorre com os bancos de dados tradicionais.
- Como conseqüência da heterogeneidade dos dados os esquemas são grandes e não se espera que o usuário conheça todos os detalhes do esquema.
- O esquema geralmente é ignorado nas consultas, que são feitas através de navegação ou busca por palavras.
- O esquema evolui rapidamente. O esquema deve ser bastante flexível e pode ser atualizado tão freqüentemente quanto os dados, causando um sério desafio à tecnologia de BD.
- Os tipos de dados são "ecléticos". O tipo de um objeto pode mudar durante as várias fases do processo. Os tipos de dados podem ser aplicados a objetos com vários papéis ou em várias visões.
- A diferença entre esquema e dado desaparece, pois as mesmas consultas/atualizações podem ser feitas aos dados e aos esquemas e a mesma informação pode ser vista como um dado em uma fonte (atributo sexo) ou como um tipo em outra (classe sexo).

Dados semi-estruturados surgiram como um importante tópico de pesquisa por várias razões. Uma delas é a necessidade de tratar fontes de dados, existentes na Web, de forma semelhante aos bancos de dados tradicionais. Deseja-se tratar a Web como um BD para que seja possível manter a integridade dos dados, fazer consultas de acordo com alguma estrutura (de forma oposta às consultas baseadas em conteúdo) e introduzir alguma organização. Além disso, também existe a necessidade de ter um formato extremamente flexível para troca de dados entre BDs distintos. A maioria dos dados do mundo real está representada em algum formato de dados. Os formatos de dados são definidos para a troca e armazenamento de dados mas, algumas vezes, a representação textual oferecida pelo formato de dados pode não ser imediatamente traduzida para uma representação padrão relacional/objeto-relacional.

Quando o objetivo é integrar diferentes tipos de dados, incluindo dados não estruturados, dados irregulares ou ausentes e dados com estrutura não conhecida completamente, também são necessários modelos de dados mais flexíveis, pois os modelos de dados e linguagens tradicionais são inadequados para acomodar conjuntos de dados heterogêneos (diferentes tipos e estruturas). Finalmente, existem situações onde dados estruturados devem ser visualizados como dados semi-estruturados com propósitos de navegação. Para consultar um BD é preciso entender seu esquema, o qual muitas vezes não tem uma terminologia muito clara. Entretanto, o usuário pode querer consultar os dados com pouco ou nenhum conhecimento do esquema.

Dados semi-estruturados requerem linguagens de consulta específicas, métodos de atualização, técnicas de otimização e indexação. A idéia em todos os modelos de dados, técnicas e linguagens de consulta para dados semi-estruturados consiste em lidar com os dados sem ter uma completa informação sobre eles, garantindo desempenho similar àquele alcançado em BDs tradicionais.

Dentre os modelos de dados e linguagens propostas para dados semi-estruturados, XML está assumindo grande importância. XML emergiu como um novo padrão para troca de dados na Web. Do ponto de vista de BDs, não apenas o uso de XML como formato para troca de dados mas também como representação interna de dados, garante um bom desempenho no suporte à execução eficiente de consultas sobre grandes quantidades de dados Web. Muitas abordagens que foram propostas para dados semi-estruturados agora estão sendo adaptadas para lidar com documentos XML. Ao mesmo tempo, muitos produtos de BDs estão sendo estendidos para obter sistemas compatíveis com XML

### 3.2 Modelos de dados semi-estruturados

Modelos de dados tradicionais, como relacional e orientado a objetos, não são adequados para dados semi-estruturados, por não serem flexíveis o suficiente para suportar a heterogeneidade da representação dos dados semi-estruturados.

Dados semi-estruturados são autodescritíveis, ou seja, o esquema de um banco de dados semi-estruturado está embutido nos próprios dados e nenhuma estrutura é previamente definida. Uma forma de modelar bancos de dados semi-estruturados é através de grafos, onde o esquema do banco de dados é representado através de um grafo, na forma de rótulos (*labels*) que são anexados aos nós ou aos arcos do grafo. Estes rótulos representam os atributos dos dados semi-estruturados.

Em um modelo de dados semi-estruturado, não existe restrição no conjunto de arcos que partem de um dado nó do grafo para outro nó, nem nos tipos dos valores dos atributos. Devido a estas características, inerentes aos modelos de dados semi-estruturados, um aspecto importante a ser considerado é a habilidade para consultar o esquema, isto é, os rótulos dos arcos ou dos nós do grafo.

Como exemplo de modelo de dados baseado em grafo pode-se destacar o OEM (*Object Exchange Model*) [Papakonstantinou 1995], que foi desenvolvido originalmente para o projeto de integração de dados TSIMMIS [Chawathe 1994]. Os dados no modelo OEM são representados através de grafos, onde os nós do grafo são objetos, os arcos são identificados com nomes de atributos e algumas folhas podem estar associadas a valores atômicos. Um objeto pode ser de dois tipos: atômico (tipo base) ou complexo (conjunto de referências de objetos). A Fig. 4 mostra um exemplo de representação de

dados semi-estruturados no modelo OEM. Neste exemplo, o objeto raiz do banco de dados é o objeto `biblio`, que é um objeto complexo composto por um objeto `artigo` e um objeto `livro` (cada objeto tem um identificador único (oid), tal como "`o1`"). Os objetos `artigo` e `livro` também são complexos. O objeto `artigo` é composto por um objeto `autor`, um objeto `título` e um objeto `referencia`. Os objetos `autor` e `titulo` são atômicos e o objeto `referencia` é um objeto complexo. O objeto `livro` é um objeto complexo composto por dois objetos `autor`, um objeto `título` e um objeto `editora`. Cada objeto `autor` é um objeto complexo composto por um objeto `nome` e um objeto `sobrenome`, que por sua vez são atômicos e contêm *strings*. Os objetos `título` e `editora` também são atômicos e contêm *strings*.

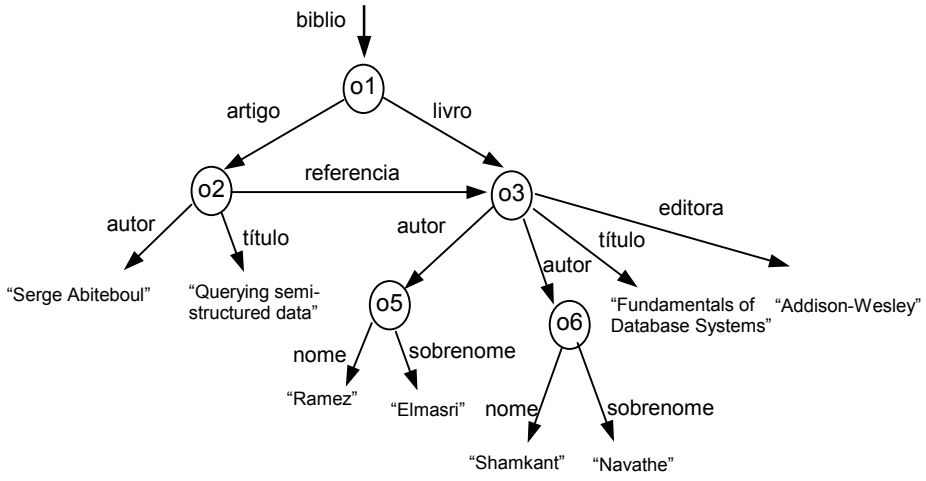


**Fig. 4 - Exemplo do Modelo OEM.**

Em um banco de dados OEM não existe a noção de um esquema fixo. Toda a informação sobre o esquema esta incluída nos rótulos, os quais podem mudar dinamicamente. Um banco de dados OEM é autodescritível e não existe regularidade nos dados. Este modelo permite a representação de informações incompletas, bem como informações com estruturas heterogêneas. Por exemplo, um objeto `artigo` ou um objeto `livro` pode ter um ou mais objetos `autor`, e um objeto `autor` pode ser uma simples *string* ou pode ter uma estrutura complexa composta pelos objetos `nome` e `sobrenome`.

### 3.3 Linguagens de Consulta para Dados Semi-estruturados

As consultas na Web são baseadas no conteúdo das páginas e na estrutura de links que conectam estas páginas. As primeiras ferramentas desenvolvidas para recuperar informações na Web foram os mecanismos de busca ou *Search Engines*. Para recuperar informações, estes mecanismos fazem buscas em bancos de índices que contêm apontadores para documentos na Web. Os bancos de índices são construídos a partir da navegação sistemática na Web e devem ser atualizados periodicamente. Posteriormente, foi incluída também a exploração da estrutura de links para aumentar a qualidade das respostas.

De acordo com [Florescu 1998], as linguagens de consulta para Web podem ser classificadas em duas gerações. As linguagens de consulta para Web pertencentes à

primeira geração visam combinar consultas baseadas no conteúdo dos dados, semelhantes às consultas realizadas nos mecanismos de busca, com consultas baseadas na estrutura dos dados, semelhantes às consultas realizadas nos bancos de dados tradicionais. Dentre as linguagens de consulta que pertencem a esta geração se destacam: WebSQL [Mendelzon 1997], W3QL [Konopnicki 1995] e WebLog [Lakshmanan 1996].

As linguagens de consulta para Web, pertencentes à segunda geração, são chamadas de linguagens de manipulação de dados Web. Estas linguagens diferenciam-se das linguagens da primeira geração porque permitem acessar tanto a estrutura interna dos documentos Web como os *links* que conectam documentos Web. Além disso, estas linguagens oferecem habilidade para criar estruturas complexas como resultados de consultas. Dentre as linguagens de consulta que pertencem a esta geração se destacam: WebOQL [Arocena 1998], Florid [Himmeröder 1997] e StruQL [Fernandez 1997].

As linguagens de consulta para a Web não permitem consultas sobre os dados implícitos na estrutura dos documentos. Por outro lado, as linguagens para dados semi-estruturados permitem este tipo de consulta e usam um modelo flexível de grafos rotulados para representar dados semi-estruturados. Como exemplo de linguagens para dados semi-estruturados se destacam Lorel [Abiteboul 1997b] e UnQL [Buneman 1996].

Algumas características das linguagens de consulta para dados semi-estruturados são apresentadas a seguir [Abiteboul 2000]:

- *Capacidade de expressão*: poder expressar tanto as operações convencionais quanto as de reestruturação de dados semi-estruturados.
- *Semântica*: uma semântica é necessária. Deve-se saber o que a semântica adiciona (ou suprime) à sintaxe utilizada.
- *Poder de composição*: a saída de uma consulta pode ser utilizada como entrada de outra (ex: construção de visões).
- *Esquema*: apesar dos dados semi-estruturados não terem um esquema definido, as linguagens de consulta devem ser "conscientes da estrutura" para checagem de tipos, otimização, etc.
- *Uso de coerção* (critérios de conversão de tipos) na comparação entre atributos de objetos, para lidar com as heterogeneidades de tipo e de estrutura dos dados.

### 3.3.1 Sintaxe Básica

As linguagens de consulta para dados semi-estruturados também caracterizam-se pela capacidade de navegação em grafos, e para isto usam o conceito de expressão de caminho [Abiteboul 2000]. Uma expressão de caminho percorre a estrutura de objetos semi-estruturados para obter informação ou formular uma condição sobre um atributo. A forma mais usual de especificação de expressões de caminho é a concatenação de nomes de rótulos.

De uma maneira geral, o resultado de uma expressão de caminho $l_1,l_2, .,l_n$, em um grafo de dados, é o conjunto de nós $v_n$, tais que existam arestas $(r, l_1, v_1)$, $(v_1, l_2 , v_2)$, . $(v_{n-1}, l_n, v_n)$ no grafo de dados, onde $r$ é a raiz. Assim, expressões de consultas resultam em um conjunto de nós e não (ainda) em porções de dados semi-estruturados.

Por exemplo, na aplicação da Fig. 4, a expressão de caminho `biblio.artigo.autor` tem como resultado {o1, o2, o4}.

As expressões de caminho também podem indicar padrões de busca, quando combinadas com certos caracteres especiais. Essa facilidade é útil quando se desconhece total ou parcialmente a estrutura dos objetos. Seguem alguns exemplos:

- `biblio.livro|artigo.autor:` autor de livro OU artigo
- `biblio._.autor:` qualquer caminho com uma aresta entre `biblio` e `autor`
- `biblio._*.autor:` qualquer caminho com qualquer número de arestas entre `biblio` e `autor`

O conceito de expressão de caminho é fundamental para as linguagens de consulta de dados semi-estruturados. Entretanto, como dito acima, uma expressão de caminho retorna apenas um conjunto de nós. As expressões de caminho não são capazes de construir novos nós, de expressar junções ou de comparar valores armazenados no banco. Para realizar estas operações são utilizadas outras funcionalidades das linguagens de consulta. Por exemplo, na aplicação da Fig. 4, a consulta `q1` apresentada a seguir recupera os autores e títulos de artigos cujo autor é "Serge Abiteboul", implementando uma forma generalizada de projeção e seleção. Nesta consulta, a variável `X` é instanciada com cada objeto `artigo` obtido a partir da expressão de caminho `biblio.artigo`. O predicado definido na cláusula `where` seleciona apenas as instâncias de `X`, ou seja, os artigos que possuem "Serge Abiteboul" como `autor`. Para cada objeto `artigo` que satisfaz a condição da cláusula `where` é criado um novo nó para representar o registro `row:{autor:Y, título T}`, onde `Y` corresponde ao autor de `X` e `T` corresponde ao título de `X`. O resultado desta consulta é o grafo de dados apresentado na Fig. 5.

```
q1: select row: (select autor: Y, título: T
                   from X.autor Y X.título T)
    from biblio.artigo X
    where "Serge Abiteboul" in X.autor
```

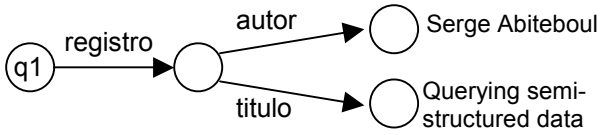


**Fig. 5 - Resultado da consulta q1.**

Além de seleções e projeções, as linguagens de consultas para dados semi-estruturados também permitem expressar junções. Considere, por exemplo, a consulta `q2` apresentada a seguir, que retorna os autores que foram referenciados pelo menos duas vezes em algum artigo com "Database" no título.

```
q2:   select row: W
      frombiblio.artigo X, X.referencia Y,
        Y.autor W, X.referencia Z
      whereNOT (Y = Z)
      and W in Z.autor
      and matches ("Database*", X.título)
```

### 3.3.2 DataGuide

Como discutido anteriormente, um banco de dados semi-estruturado não tem um esquema fixo para definir a estrutura do banco de dados. Sem algum conhecimento sobre a estrutura do banco de dados, escrever consultas pode se tornar bastante difícil. Além disso, sem maiores informações sobre a estrutura do banco de dados, o processador de consultas pode ser obrigado a trabalhar mais do que o necessário. Uma solução para conhecer melhor a estrutura de um banco de dados consiste em navegá-lo manualmente. Entretanto, esta solução torna-se inviável para grandes bancos de dados.

Para solucionar os problemas relacionados à falta de estrutura dos bancos de dados semi-estruturados, foram propostos os *DataGuides*. Um *DataGuide* [Goldman 1997] é um sumário conciso e preciso da estrutura de um banco de dados semi-estruturado. Cada expressão de caminho do banco de dados é codificada exatamente uma vez no *DataGuide*, assim como não existem expressões de caminho no *DataGuide* que não fazem parte do banco de dados. A Fig. 6 apresenta o *DataGuide* para o banco de dados OEM apresentado na Fig. 4. Um *DataGuide* tem um papel similar aos metadados dos bancos de dados tradicionais. Um *DataGuide* pode ser consultado ou navegado, permitindo aos usuários ou aplicações examinarem a estrutura do banco de dados.

Em sistemas de bancos de dados tradicionais (relacionais ou orientados a objetos) um esquema é explicitamente criado antes de qualquer dado ser carregado. Por outro lado, os *DataGuides* são gerados e mantidos dinamicamente. Um *Data Guide* pode ser utilizado com os seguintes objetivos: armazenar estatísticas para o otimizador de consultas, servir como índice de caminhos, oferecer suporte para a interface gráfica, facilitar a navegação na estrutura dos dados, visualizar valores atômicos e facilitar a formulação de consultas "*by example*". O conceito de *DataGuide* é utilizado no *Lore* (*Lightweight Object Repository*) [Goldman 1999], um sistema gerenciador de bancos de dados semi-estruturados, que tem OEM como modelo de dados e *Lorel* como linguagem de consulta.

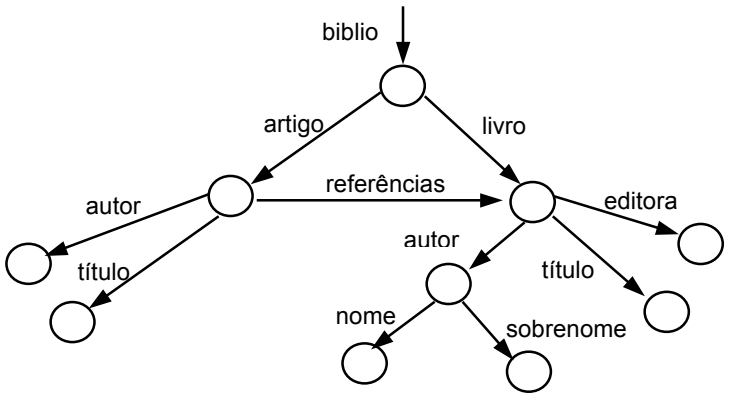


**Fig. 6 - Exemplo de DataGuide.**

## 4. XML

A linguagem XML (*Extensible Markup Language*) foi proposta pelo W3C (*World Wide Web Consortium*) como um padrão para representação e troca de dados na Web [Bray 1998]. Uma das aplicações beneficiadas com a utilização de XML é a integração de

dados, pois XML fornece uma plataforma para compartilhamento de dados através de uma sintaxe comum.

Uma das dificuldades em integrar informações de múltiplas fontes de dados é a estrutura heterogênea das fontes. Para sobrepor esta heterogeneidade, os sistemas de integração usam um modelo de dados comum para representar tanto o conteúdo quanto a estrutura das bases de dados. Devido à flexibilidade de XML para representar dados semi-estruturados e dados estruturados, e à facilidade para converter dados para o formato de XML, existe um grande interesse em usar XML como modelo de dados comum para representação e troca de dados.

Esta seção apresenta a linguagem XML e aborda tópicos importantes sobre esta linguagem. Na seção 4.1, uma visão geral sobre XML é apresentada. Na seção 4.2 é apresentada a sintaxe básica para construção de documentos XML. Na seção 4.3 são apresentadas as linguagens de definição de esquemas para XML, com destaque para DTD e XML Schema. Na seção 4.4 são discutidos alguns modelos de dados para XML. Na seção 4.5 algumas linguagens de consulta para XML são apresentadas.

## 4.1 Visão Geral

XML é uma linguagem de marcadores (*markup language*) para descrever informações. Um marcador é uma forma de tornar explícita a interpretação de um texto e uma linguagem de marcadores é um conjunto de convenções de marcadores utilizados para codificação de textos.

Ao contrário de HTML, que foi projetada exclusivamente para descrever a apresentação dos dados, XML foi projetada para descrever o conteúdo dos dados. A linguagem XML é mais poderosa do que HTML nos seguintes aspectos:

- XML é uma linguagem extensível, ou seja, permite que os usuários definam novos marcadores de acordo com o domínio que está sendo modelado;
- a estrutura dos documentos pode ser aninhada em qualquer nível;
- um documento XML pode conter uma descrição opcional da sua gramática, a qual pode ser usada pelas aplicações para validações na estrutura dos documentos;
- XML é uma linguagem flexível, pois permite a apresentação de um mesmo conteúdo em diferentes formatos.

Por ser uma linguagem extensível, XML também pode ser vista como uma metalinguagem, ou seja, uma linguagem que pode ser usada para criação de outras linguagens de marcadores. Estas linguagens de marcadores podem ser criadas para domínios específicos (ex: matemática, química, comércio eletrônico, entre outros) através da definição de marcadores próprios de cada domínio. Por exemplo, quando o domínio de uma livraria é considerado, os seguintes marcadores podem ser definidos: `<livro>`, `<autor>` e `<editora>`. A idéia é que os marcadores possam agregar algum significado ao texto associado a eles e, assim, capturar mais semântica sobre os dados que estão sendo modelados. Isto não ocorre em um documento HTML, pois os marcadores definidos em HTML têm função apenas de formatar o texto para apresentação. Considere, por exemplo, o documento HTML e o documento XML apresentados na Fig. 7. Qual destes documentos captura mais semântica? Claramente, o documento XML captura mais semântica, pois os marcadores `<livro>`, `<titulo>` e

<autor> têm significado para o domínio que está sendo representado através destes documentos, ao contrário dos marcadores <TR>, <TH> e <TD> que não têm significado algum para o domínio.

```
<----HTML ---->
<TABLE BORDER=1>
  <TR>
    <TH>Livro ID</TH>
    <TH>Titulo</TH>
    <TH>Autor</TH>
  </TR>
  <TR>
    <TD>L01</TD>
    <TD> Foundations of
         Data Bases </TD>
    <TD> Abiteboul </TD>
  </TR>
</TABLE>
```

```
<----XML ---->

<livro>
   <id>L01</id>
   <titulo> Foundations of
         Data Bases </titulo>
   <autor> Abiteboul </autor>
</livro>
```

**Fig. 7 - Exemplo de um documento HTML e um documento XML com dados sobre uma livraria.**

XML é um subconjunto de SGML (*Standard Generalized Markup Language*), um padrão internacional para definição de formatos de representação de textos em meio eletrônico. SGML é um padrão muito poderoso e bastante geral, o que torna complicada a sua implementação e restringe sua utilização a grandes empresas. Como mostrado na Fig. 8 [McGrath 1998], SGML e XML são metalinguagens utilizadas para definição de outras linguagens de marcadores aplicadas a domínios bastante distintos.

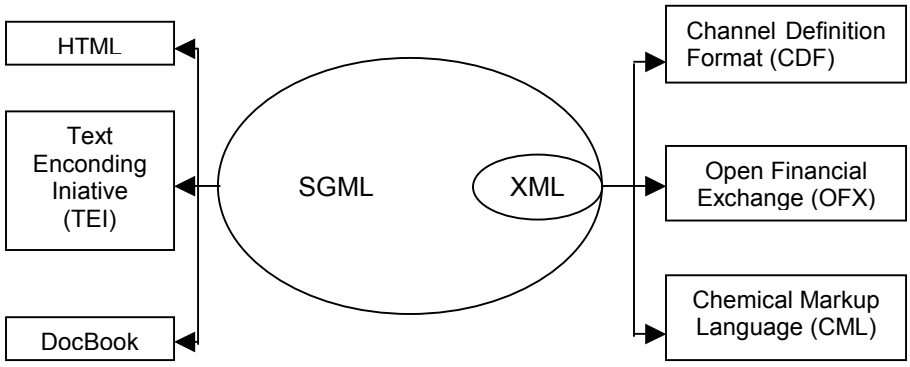


**Fig. 8 - Relacionamento entre XML e outras linguagens de marcadores.**

Uma das idéias principais de XML é tornar explícita a separação entre os componentes de um documento eletrônico: apresentação, conteúdo e estrutura. Um documento XML descreve as informações de acordo com o domínio que está sendo modelado, sem levar em consideração como estas informações serão apresentadas em um *browser*. Sendo assim, documentos XML podem ser usados e reusados de diferentes formas e em diferentes formatos. Como o padrão XML trata apenas do conteúdo dos documentos, devem existir outros padrões para descrever o formato de apresentação dos dados. Atualmente, existem duas propostas do W3C para criação de estilos de apresentação para documentos XML: *Cascading Style Sheets* - CSS e *Extensible Style Language* - XSL.

Assim como os dados semi-estruturados, os dados XML também são autodescritíveis. A Fig. 9 apresenta um exemplo de documento XML que descreve informações sobre livros de uma livraria. Um documento XML consiste de vários elementos. Um elemento pode conter apenas texto, outros elementos (subelementos) ou então pode ter um conteúdo misto, ou seja, pode conter texto e outros elementos. Por exemplo, `livro` é um subelemento do elemento `livraria`, enquanto que `autor` e `titulo` são subelementos do elemento `livro`. Um elemento também pode ter atributos, os quais sempre são especificados no marcador que define o início do elemento. Por exemplo, `ano` é um atributo do elemento `livro` e define o ano de publicação de um determinado livro.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE livraria SYSTEM "livraria.dtd">
<livraria>
   <livro id="L01" ano="1997">
      <autor>
         <nome>Marie</nome >
         <sobrenome>Buretta</sobrenome >
      </autor>
      <titulo>Data Replication</titulo>
      <editora>John Wiley & Sons </editora>
   </livro>
   <livro id="L02" ano="1996" bib="L01">
      <autor>
         <nome>Serge </nome>
       <sobrenome>Abiteboul</sobrenome >
      </autor>
      <autor>
         <nome>Richard</nome>
       <sobrenome>Hull</sobrenome >
      </autor>
      <autor>
         <nome>Victor</nome>
       <sobrenome>Vianu</sobrenome >
      </autor>
      <titulo>Foundations of Data Bases</titulo>
      <editora>Addison Wesley</editora>
   </livro>
</livraria>
```

**Fig. 9- Exemplo de documento XML (livraria.xml).**

### 4.1.1 Categorias de Documentos XML

Os documentos XML são classificados como bem-formados quando estão de acordo com a sintaxe definida na especificação de XML [Bray 1998]. Os documentos XML também podem ser classificados como documentos válidos. Isto ocorre quando o documento XML é bem-formado e está de acordo com a gramática que define a sua estrutura. Esta gramática é definida em um esquema, o qual descreve: a estrutura de um documento XML, os elementos que podem participar do documento e os atributos que podem estar associados a estes elementos. Atualmente, existem várias propostas de linguagens de descrição de esquemas para documentos XML, entre elas destacam-se:

DTD [Bray 1998] e XML Schema [Biron 2000], [Thompson 2000], apresentadas na seção 4.3.1.

A verificação sintática de um documento XML, a fim de detectar se este é bem-formado ou não, é feita por um *parser*, que analisa o documento e envia mensagens de erro quando encontra erros de sintaxe. Os *parsers* podem ser classificados em duas categorias: i) *parser* de validação: além de detectar se um documento XML é bem-formado, também é capaz de verificar se o documento está de acordo com o esquema XML ao qual o documento está associado e ii) *parser* de não validação: não é capaz de verificar se um documento bem-formado está de acordo com o esquema ao qual o documento está associado.

### 4.1.2 APIs para XML

DOM (*Document Object Model*) e SAX (*The Simple API for XML*) são APIs (*Application Programming Interface*) para XML que oferecem meios para acessar e manipular o conteúdo de um documento XML. As aplicações podem utilizar as operações disponíveis na API para acessar o conteúdo de um documento XML. Estas duas APIs oferecem diferentes visões de um mesmo documento: DOM oferece uma visão baseada em árvore e SAX oferece uma visão baseada em eventos. É importante obervar que DOM é a API recomendada oficialmente pelo W3C.

Tanto DOM quanto SAX são APIs independentes de linguagem e plataforma que permitem programas e *scripts* acessarem e atualizarem o conteúdo e a estrutura de um documento dinamicamente.

DOM provê um conjunto de interfaces e objetos para representar a estrutura e o conteúdo de documentos XML e HTML. Esta API representa um documento como uma coleção de objetos (*nodes*), criando uma árvore de nós baseada na estrutura e na informação do documento XML.

SAX também é uma interface que permite a interação com documentos XML e apresenta um documento XML como uma seqüência de eventos. Esta API não permite acessos randômicos na manipulação do documento, sendo, portanto, mais adequada quando o processamento do documento é seqüencial.

### 4.1.3 Bancos de Dados e XML

De acordo com a comunidade XML:DB[1], três tipos diferentes de bancos de dados para XML podem ser definidos:

- **Banco de Dados XML Nativo** (*Native XML Database* - NXD): é um banco de dados projetado especificamente para armazenar e manipular dados XML. O acesso aos dados é através de XML e padrões relacionados, como XSLT, DOM ou SAX. Esta categoria inclui todos os bancos de dados onde a representação dos dados mantém a estrutura do documentos XML, bem como os metadados associados. A unidade fundamental de armazenamento é um documento XML. Tamino, dbXML e X-Hive são classificados nesta categoria.
- **Banco de Dados compatível com XML** (*XML Enabled Database* - XEDB): é um banco de dados que tem como funcionalidade adicional um nível de mapeamento

[1] http://www.xmldb.org/

para XML, que pode ser fornecido pelo próprio fabricante ou por terceiros. Este nível de mapeamento gerencia o armazenamento e a recuperação de dados XML. Os dados que são mapeados para o banco de dados são mapeados para formatos específicos da aplicação e, neste caso, a estrutura original do documento XML e os seus metadados podem ser perdidos. Além disso, os dados recuperados como XML não têm garantia de terem sido gerados no formato original de XML. A manipulação de dados pode ocorrer através de tecnologia específica para XML (i.e. XSLT, DOM ou SAX) ou outras tecnologias de banco de dados. (i.e. SQL). As soluções para XML propostas pela Oracle e Microsoft fazem parte desta categoria.

- **Banco de Dados XML Híbrido** (*Hybrid XML Database* - HXD): é um banco de dados que pode ser tratado ou como banco de dados nativo ou como um banco de dados compatível com XML, dependendo dos requisitos da aplicação. Exemplos de bancos de dados desta categoria são Excelon e Ozone.

### 4.2 Construindo Documentos XML

Os elementos são os blocos principais da estrutura hierárquica de XML. Como mencionado anteriormente, o conteúdo de um elemento pode ser composto de: outros elementos, caracteres, ou outros elementos e caracteres. Os elementos são delimitados por marcadores tais como `<autor>` e `</autor>` apresentados no exemplo da Fig. 9. Cada marcador de início (`<livro>`) deve ter um marcador de final correspondente (`</livro>`). Um documento XML também pode ter elementos com conteúdo vazio. Neste caso, o elemento deve ser representado de acordo com um dos seguintes formatos: `<referencia></referencia>` ou `<referencia/>`. Elementos vazios são úteis quando se deseja descrever um elemento apenas através de atributos ou para referenciar outros elementos.

Um elemento pode possuir um ou mais atributos que definem as propriedades dos elementos e adicionam novas características a eles. Os atributos são especificados no marcador de início do elemento ou no marcador que define um elemento vazio. Por exemplo, no documento "livraria.xml", `ano` é um atributo do elemento `livro`. É importante lembrar que os valores de atributos devem ser delimitados por aspas ("ou‘ ).

De forma geral, os elementos definem a estrutura de entidades do mundo real enquanto que os atributos definem propriedades sobre estas entidades. Como esta distinção nem sempre é clara, uma das dúvidas mais freqüentes quando se constrói um documento XML consiste em determinar se uma informação deve ser modelada como um elemento ou como um atributo. Entretanto, existem, algumas diretrizes que podem ser úteis no momento desta escolha. Quando a informação possui alguma estrutura hierárquica então esta informação deve ser modelada como um elemento, pois não é possível estabelecer uma hierarquia entre atributos. A ordem em que os atributos aparecem na definição de um elemento é irrelevante (`ano` e `id` em `livro` na Fig. 9). Por outro lado, a ordem em que os elementos aparecem na definição de um elemento sempre deve ser obedecida (`nome` e `sobrenome` em `autor` na Fig. 9). Outra diferença importante é que um atributo só pode ser definido uma única vez para cada elemento, enquanto que um subelemento pode se repetir várias vezes na definição de um elemento. Por exemplo, um livro pode ter vários autores, mas só pode ter um valor para ano de publicação.

A primeira informação de um documento XML é chamada de prólogo (*prolog*), o qual inclui a declaração XML e a declaração de tipo de documento. A declaração XML determina a versão de XML utilizada para a definição do documento. A declaração de tipo de documento especifica a DTD a qual o documento está associado. Após o prólogo, inicia-se o elemento raiz o qual contém todos os outros elementos do documento. Uma das regras básicas para criar um documento XML bem-formado é que deve existir apenas um único elemento raiz em um documento XML. O elemento raiz do documento "livraria.xml" é o elemento `livraria`. Além de outros elementos, o elemento raiz também pode conter:

- **Seções CDATA**: devem ser usadas em documentos XML que contêm grande número de caracteres especiais (ex: "<" e "&"). As seções CDATA são úteis porque quando um caractere especial se encontra dentro dos limites de uma seção CDATA ele passa a ser interpretado como um simples caractere. Considere, por exemplo a seção CDATA definida a seguir. Neste caso, o caractere "<" pode ser usado livremente sem ocasionar erros de sintaxe, pois este caractere está inserido dentro de uma seção CDATA.

```
<Documento>
<![CDATA [
se a<b e b<c então a<c ]]>
</Documento>
```

- **Comentários**: a definição de comentários é bastante útil e faz parte da maioria das linguagens. Em XML, os comentários são definidos como mostrado a seguir:

```
<!-- Exemplo de comentário -->
```

- **Instruções de processamento**: são utilizadas para enviar comandos e informações à aplicação que está processando o documento XML. Um exemplo de instrução de processamento é a declaração de XML (`<?xml version="1.0" encoding="utf-8"?>`).

Todo o conteúdo que vier após o elemento raiz é denominado de epílogo, o qual pode conter apenas comentários e instruções de processamento.

Para facilitar o desenvolvimento de aplicações em XML, existem várias ferramentas disponíveis, incluindo: editores para criação e modificação de documentos XML, ferramentas para criação e modificação de esquemas e folhas de estilo, e ferramentas que oferecem suporte para o gerenciamento e armazenamento de documentos XML.

### 4.3 Esquemas XML

Um documento XML pode, opcionalmente, estar associado a um esquema. Um esquema define os elementos que podem aparecer em um documento, o conteúdo destes elementos e os atributos que podem estar associados a eles. Um esquema também define a estrutura de um documento, por exemplo: quais os subelementos de um determinado elemento e a seqüência na qual estes subelementos podem aparecer.

Um esquema é útil para validar o conteúdo de um documento, ou seja, para determinar se um documento é válido de acordo com a gramática definida pelo esquema. Além disso, a gramática definida pelo esquema poderá ser reutilizada para a definição de outros documentos.

A habilidade de testar a validade de documentos XML é muito importante para aplicações Web que trocam informações entre fontes diversas. Com a definição de esquemas XML, é possível verificar se um determinado documento está de acordo com a estrutura esperada, facilitando o processamento de dados pelas aplicações. Além disso, diferentes fontes de dados, relacionadas a um mesmo domínio (ex: comércio eletrônico), podem definir uma gramática comum para ser utilizada como base para criação de documentos. Assim, a integração de dados pode tirar vantagem da gramática comum que foi utilizada para criar os documentos.

A primeira linguagem proposta para definição de esquemas XML foi a DTD, que apesar de suas limitações, atualmente é a forma mais utilizada para especificação de esquemas de documentos XML. Para sobrepor as limitações das DTDs, recentemente, várias linguagens para definição de esquemas XML foram propostas [Lee 2000], dentre elas se destacam: XML Schema [Biron 2000], [Thompson 2000], XDR [Frankston 1998], SOX [Davidson 1999], Schematron [Jellife 2000] e DSD [Klarlund 2000]. Estas linguagens são mais ricas em semântica e oferecem alguns recursos adicionais para definição de esquemas, como por exemplo: maior variedade de tipos primitivos, definição de novos tipos e hierarquia. Estas linguagens também oferecem a vantagem de seguir a sintaxe de XML, ou seja são linguagens baseadas em XML. Desta forma, é possível tirar proveito de toda a tecnologia que já foi desenvolvida para XML. A seguir, para ilustrar as principais características dos esquemas XML serão apresentadas as linguagens DTD e XML Schema.

### 4.3.1 DTD - Document Type Definition

Um método para descrever esquemas para documentos XML corresponde à definição de uma DTD (*Document Type Definition*), que define a seqüência dos elementos no documento, o conteúdo dos elementos e os atributos associados a cada elemento.

A declaração de tipo de documento que aparece no prólogo de um documento XML especifica a DTD a qual o documento está associado. Por exemplo, a declaração de tipo de documento `<!DOCTYPE livraria SYSTEM "livraria.dtd">` (Fig.9) define que o documento "livraria.xml" está associado a uma DTD armazenada em um arquivo externo chamado "livraria.dtd". É importante notar que a declaração de tipo de documento sempre deve referenciar o arquivo raiz, ou seja, sempre após a expressão `!DOCTYPE` deve-se colocar o nome do elemento raiz do documento.

As DTDs podem ser internas ou externas a um documento XML. Uma DTD é considerada interna quando a sua definição é inserida dentro do próprio documento XML. Por outro lado, uma DTD é externa quando a definição da DTD fica armazenada em um arquivo externo que é referenciado na declaração de tipo de documento. Considere, por exemplo, a definição da Fig. 10 (a), onde as declarações de elemento fazem parte da cláusula `!DOCTYPE` do documento XML. A Fig. 10 (b) apresenta um exemplo onde é feita uma referência a uma DTD externa que está armazenada no arquivo "livraria.dtd".

Uma DTD é composta de vários tipos de declaração, entre elas destacam-se: declarações de elemento, declarações de atributo e declarações de entidade, as quais serão explicadas a seguir.

```
<-- DTD Interna -->
<!DOCTYPE livraria [
<!ELEMENT livro (titulo, autor)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (#PCDATA)>]>
```

(a)

```
<-- DTD Externa -->
<!DOCTYPE livraria SYSTEM
"livraria.dtd">
```

(b)

**Fig. 10 - Exemplos de declaração de tipo de documento.**

- **Declaração de tipo de elemento**: permite a um documento XML restringir os elementos que ocorrem no documento. Todos os elementos utilizados em um documento XML devem ter uma declaração de elemento correspondente. Esta declaração especifica o conteúdo de um elemento, o qual pode ser dos seguintes tipos:
  - *Seqüência de elementos*: especifica que um elemento consiste de outros elementos exatamente na ordem em que for especificada a seqüência.

    Exemplo: `<!ELEMENT artigo (autor, titulo)>`.
  - *Seleção a partir de uma lista de elementos*: especifica que apenas um dos elementos definidos na declaração pode fazer parte do conteúdo do elemento que está sendo declarado.

    Exemplo: `<!ELEMENT publicacao (livro|artigo)>`.
  - *A ocorrência de um elemento é opcional* (?): especifica que a participação de um elemento é opcional e caso exista deve ser única.

    Exemplo: `<!ELEMENT livro (titulo, editora?)>`.
  - *Um elemento ocorre zero ou mais vezes* (*): especifica que um elemento pode participar zero ou mais vezes do elemento que está sendo declarado.

    Exemplo: `<!ELEMENT livros (livro*)>`.
  - *Um elemento ocorre uma ou mais vezes* (+): especifica que um elemento deve participar pelo menos uma vez do elemento que está sendo declarado, podendo também se repetir.

    Exemplo: `<!ELEMENT livro (autor+, titulo, editora)>`.
  - *Um elemento contém qualquer outro elemento em qualquer ordem*: especifica que um elemento consiste de qualquer combinação de elementos em qualquer ordem. O elemento também pode conter caracteres ou o elemento pode conter outros elementos e caracteres em qualquer ordem.

    Exemplo: `<!ELEMENT livro ANY>`.
  - *Um elemento também pode conter uma string de caracteres e pode ter conteúdo misto (caracteres e elementos)*.

    Exemplos:

    `<!ELEMENT titulo (#PCDATA)>`.

```
<!ELEMENT livro (#PCDATA, autor*)>.
```

- **Declaração de atributo**: especifica o nome, o tipo e, opcionalmente, o valor padrão dos atributos associados a um elemento. Os atributos podem ser dos seguintes tipos:
  - *String*: o valor de um atributo do tipo *string* é uma cadeia de caracteres de qualquer tamanho.

    Exemplo: `<!ATTLIST livro ano CDATA>.`
  - *Enumerado*: cada um dos valores possíveis que o atributo pode assumir está explicitamente enumerado na declaração. O atributo pode assumir apenas um dos valores especificados na sua declaração.

    Exemplo: `<!ATTLIST livro categoria (Banco de Dados|Web)>.`
  - *ID*: os atributos do tipo ID identificam unicamente elementos individuais em um documento. Todos os valores usados para os atributos do tipo ID em um documento devem ser diferentes. Cada elemento pode ter um único atributo ID.

    Exemplo: `<!ATTLIST livro id ID>.`
  - *IDREF/ IDREFS*: o valor de um atributo do tipo IDREF deve ser o valor de um único atributo ID definido para algum elemento do documento. O atributo IDREFS é uma variação do tipo IDREF. O valor de um atributo IDREFS pode conter valores IDREF múltiplos separados por espaços em branco.

    Exemplo: `<!ATTLIST livro bib IDREFS>.`
  - *ENTITY/ ENTITIES*: o valor de um atributo ENTITY deve ser o nome de uma única entidade (o conceito de entidade será explicado logo a seguir). O valor de um atributo ENTITIES pode conter valores de entidades múltiplos separados por espaços em branco.

    Exemplo: `<!ATTLIST livro resumo ENTITY>.`

  O valor padrão de um atributo associado a um elemento pode ser:
  - *Obrigatório (#REQUIRED)*: o atributo deve ter um valor explicitamente especificado em cada ocorrência do elemento no documento.

    Exemplo: `<!ATTLIST livro preco CDATA #REQUIRED>.`
  - *Opcional (#IMPLIED)*: o valor do atributo não é requerido e nenhum valor padrão é fornecido.

    Exemplo: `<!ATTLIST livro ano CDATA #IMPLIED>.`
  - *Fixo (#FIXED)*: um valor é fornecido na declaração. Sendo assim, nenhum valor precisa ser fornecido no documento. Entretanto, caso um valor seja fornecido no documento então ele deve corresponder ao valor fornecido na declaração.

    Exemplo: `<!ATTLIST livro qtdMinima CDATA #FIXED "15">.`
- **Declarações de entidade**: uma entidade é uma unidade de armazenamento que pode conter um bloco de texto, uma declaração de tipo de documento, uma referência a um arquivo externo, entre outros. Basicamente, uma entidade é utilizada para fazer uma associação entre o seu nome e algum conteúdo. As entidades podem ser internas

ou externas. Uma entidade é considerada interna quando o seu conteúdo estiver presente diretamente na declaração de entidade. Uma entidade é considerada externa quando o conteúdo ao qual a entidade está associada está armazenado em um recurso externo. Por exemplo, a entidade `Direitos`, declarada a seguir, é considerada um entidade interna, pois o conteúdo ao qual esta entidade está associada é o texto "Este produto é fabricado pela Intel.", que faz parte da declaração da entidade.

```
<!ENTITY Direitos "Este produto é fabricado pela Intel.">
```

Por outro lado, caso a entidade `Direitos` fosse declarada como se segue, esta entidade seria considerada uma entidade externa, pois estaria associada com o arquivo "direitos.txt".

```
<!ENTITY Direitos SYSTEM "direitos.txt">
```

Quando o *parser* encontra uma referência a uma entidade em um documento XML, o *parser* substitui esta referência pelo texto ou pelo conteúdo do arquivo ao qual a entidade está associada. O trecho de documento XML apresentado a seguir possui uma referência a entidade `Direitos (&Direitos;)`.

Exemplo:

```
<manual>
   <p> Este é o manual da Intel.
       &Direitos;
   </p>
</manual>
```

A Fig. 11 apresenta a DTD que define a gramática para o documento XML "livraria.xml" (Fig. 9). Esta DTD especifica que um elemento `livraria` consiste em vários elementos `livro`. Um elemento `livro`, por sua vez, contém um ou mais elementos `autor`, seguidos de um elemento `titulo` e de um elemento `editora`. Além desses elementos, `livro` também possui um atributo opcional `ano`, um atributo `id` e um atributo `bib`, que armazena referências a outros `livros`. Um elemento `autor` consiste em um elemento `nome` e um elemento `sobrenome`, os quais contêm *strings*, assim como os elementos `titulo` e `editora` também contêm apenas *strings*.

```
<!ELEMENT livraria (livro*)>
<!ELEMENT livro (autor+, titulo, editora)>
<!ATTLIST livro ano CDATA #IMPLIED
                id ID
                bib IDREF>
<!ELEMENT autor (nome, sobrenome)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT sobrenome (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT editora (#PCDATA)>
```

**Fig. 11 - Exemplo de DTD (livraria.dtd).**

### 4.3.2 XML Schema

XML Schema é uma proposta da W3C para descrever esquemas de documentos XML. Esta linguagem tem a vantagem de utilizar a mesma sintaxe de XML, além de oferecer um conjunto maior de tipos de dados, bem como suportar a criação de novos tipos de

dados. Nesta seção, algumas características necessárias para definição de esquemas XML são apresentadas. Maiores detalhes sobre a linguagem XML Schema podem ser encontrados em [Biron 2000], [Thompson 2000].

#### 4.3.2.1 Namespaces

Uma das principais diferenças entre uma DTD e um esquema definido em XML Schema está relacionada ao uso de *namespaces* XML. *Namespaces* [Bray 1999] também é um padrão definido pelo W3C e pode ser visto como uma maneira simples e direta de distinguir nomes usados em documentos XML. Um *namespace* é identificado por um nome único (uma URL) e consiste em uma coleção de tipos de elementos e nomes de atributos. Qualquer tipo de elemento ou nome de atributo pertencente a um *namespace* pode ser identificado unicamente através de um nome composto por duas partes: i) o nome do *namespace* e ii) o nome do tipo de elemento ou atributo.

*Namespaces* são declarados em um documento XML através do atributo `xmlns`, o qual associa um prefixo a um *namespace*. O escopo desta declaração inclui o elemento que contém o atributo `xmlns` e todos os seus descendentes. Quando um *namespace* contém um prefixo, então os tipos de elementos e nomes de atributos pertencentes a ele devem ser referenciados com o prefixo correspondente. Quando uma declaração de *namespace* não contém um prefixo, então este *namespace* será considerado *default* e os elementos pertencentes a ele são referenciados sem um prefixo. É importante lembrar que a única função de um *namespace* é prover uma forma de distinção entre nomes. Além disso, as URLs usadas como nomes de *namespaces* não apontam para esquemas, nem para informações sobre *namespaces*, ou seja, estas URLs são apenas identificadores.

Todo esquema XML está associado ao *namespace* `http://www.w3.org/2000/10/ XMLSchema`. Isto quer dizer que todos os elementos utilizados para a definição de esquemas XML pertencem a este *namespace* (por exemplo: `complexType`, `sequence`). Os elementos que são declarados em um esquema XML também podem estar associados a um *namespace*, o qual é denominado de `targetNamespace`. Por exemplo, no esquema "livraria.xsd", apresentado na Fig. 12, foi definido que o `targetNamespace` corresponde ao *namespace* `http://www.livraria.org`. Os *namespaces* utilizados na definição de um esquema XML devem ser especificados no elemento raiz do esquema. Todo esquema XML tem como elemento raiz o elemento `<schema>`.

#### 4.3.2.2 Sintaxe Básica

Assim como uma DTD, um esquema XML é um conjunto de declarações de elementos e atributos. Entretanto, ao contrário da DTD, a linguagem XML Schema oferece meios para definir tipos de dados, que podem ser tipos simples como PCDATA ou podem ser tipos mais complexos e estruturados. XML Schema dispõe de dois tipos de elementos que podem ser usados para a definição de tipos: o `simpleType` e o `complexType`. O `complexType` deve ser usado quando se deseja definir um elemento que possui subelementos ou atributos. Por exemplo, no esquema "livraria.xsd" foi utilizado o `complexType` para definir um elemento `livro`, composto pelos elementos `autor`, `titulo` e `editora`.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2000/10/XMLSchema
            targetNamespace = http://www.livraria.org
            xmlns: "http://www.livraria.org">
<xsd:element name="livro">
 <xsd:complexType>
  <xsd:sequence>
    <xsd:element name="autor" type="xsd:string"/>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="editora" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="ano" type="xsd:string"/>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

**Fig. 12 - Exemplo de esquema definido na linguagem XML Schema (livraria.xsd).**

Por outro lado, deve-se utilizar o elemento `simpleType` para criar tipos de dados que correspondem a refinamentos de tipos de dados primitivos (por exemplo: *string*, *integer*). Um novo tipo de dados pode ser definido a partir de um tipo de dados existente pela atribuição de valores às "facetas" dos tipos de dados (*facets*). Como mostrado a seguir, o tipo de dados `meuInteiro` foi criado a partir do tipo base *integer* através da atribuição de limites (`minInclusive` e `maxInclusive`) para os valores que o tipo pode receber.

Exemplo:

```
<simpleType name="meuInteiro">
   <restriction base="integer">
     <minInclusive value = "1">
     <maxInclusive value = "10">
   </restriction>
</simpleType>
```

Os tipos de dados definidos em um esquema podem ser usados para a definição de elementos e atributos. Definir e usar tipos de dados é comparável a definir uma classe e usá-la para criar objetos. Os tipos complexos (`complexType`) podem ser usados apenas para definição de elementos, enquanto que os tipos simples (`simpleType`) podem ser usados para definição tanto de elementos como de atributos. Ao contrário de outras linguagens para definição de esquemas, XML Schema permite a definição de cardinalidade para um elemento (i.e. o número possível de ocorrências do elemento). Para isto podem ser utilizados o atributo `minOccurs` e o atributo `maxOccurs`, que determinam respectivamente o número mínimo e máximo de ocorrências de um elemento.

Basicamente, existem três formas diferentes de declarar elementos:

- **A declaração de um elemento tem como subelemento a definição de um tipo complexo.**

  Exemplo:
  ```
  <xsd:element name="livro">
  ```

```
 <xsd:complexType>
  <xsd:sequence>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="editora" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

- **A declaração de um elemento tem como subelemento a definição de um tipo simples.**

  Exemplo:

```
<xsd:element name="meuInteiro">
  <simpleType>
   <restriction base="integer">
      <minInclusive value = "1">
      <maxInclusive value = "10">
   </restriction>
  </simpleType>
</xsd:element>
```

- **A declaração de um elemento faz referência a um tipo complexo já definido.**

  Exemplo:

```
<xsd:element name="livro" type="Tlivro"/>
<xsd:complexType name="Tlivro">
  <xsd:sequence>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="editora" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Declarações de elementos e tipos são ditas globais quando são filhas imediatas do elemento `<schema>`. Por outro lado, declarações de elemento e definições de tipos são consideradas locais quando estão aninhadas dentro de outros elementos ou tipos. Esta diferença é importante porque apenas elementos e tipos globais podem ser reusados.

XML Schema também permite que tipos complexos sejam derivados a partir de outros tipos. Esta derivação pode ser de duas formas:

- **Por extensão**: adicionando novos elementos a um tipo (semelhante a herança). Por exemplo, o tipo complexo `autorEstendido` é criado a partir de um elemento `autor` previamente definido através da adição do elemento `endereco`.

```
<complexType name="autorEstendido" base="autor">
   <element name="endereco" type="string"
    minOccurs='0' maxOccurs='*' />
</complexType >
```

- **Por restrição**: aplicando restrições aos elementos de um tipo. Por exemplo, o tipo `publicacaoAutorUnico` é criado a partir de um tipo publicação restringindo-se o número de elementos `autor` para um único elemento.

```
<xsd:complexType name="publicacaoAutorUnico">
```

```
    <xsd:restriction base ="publicacao">
      <xsd:sequence>
        <xsd:element name="autor" type="xsd:string"
         maxOccurs="1"/>
        <xsd:element name="titulo" type="xsd:string"/>
        <xsd:element name="editora" type="xsd:string"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexType>
```

XML Schema também permite a definição de grupos que especificam restrições sobre um conjunto fixo de subelementos. Os grupos podem ser de três tipos:

- **sequence**: todos os elementos pertencentes a este grupo devem aparecer na ordem em que foram definidos e nenhum elemento pode ser omitido.
  Exemplo:
```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="autor" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```
- **choice**: apenas um dos elementos pertencentes ao grupo deve aparecer em uma instância de documento XML.

  Exemplo:
```
<xsd:complexType>
  <xsd:choice>
    <xsd:element name="livro" type="xsd:string"/>
    <xsd:element name="artigo" type="xsd:string"/>
  </xsd:choice>
</xsd:complexType>
```
- **all**: os elementos podem aparecer em qualquer ordem e podem ser repetidos ou omitidos

  Exemplo:
```
<xsd:complexType>
  <xsd:all>
    <xsd:element name="titulo" type="xsd:string"/>
    <xsd:element name="autor" type="xsd:string"/>
  </xsd:all>
</xsd:complexType>
```

Com relação aos atributos, é importante notar que as declarações de atributo sempre aparecem após as declarações de elemento. Os atributos sempre dizem respeito ao elemento dentro do qual estão sendo definidos. Por exemplo, o esquema "livraria.xsd" tem uma declaração de atributo para o atributo `ano` do elemento `livro`. (`<xsd:attribute name="ano" use="required" type="xsd:string"/>`).

Assim como nas DTDs, também é possível declarar o valor padrão de um atributo. Isto é feito através da cláusula `use`, que pode assumir um dos seguintes valores: *required* (obrigatório), *optional* (opcional) e *fixed* (fixo). Neste último caso, deve-se dizer o valor padrão do atributo utilizando a cláusula `value`. Por exemplo, no

esquema “livraria.xsd” a declaração do atributo `ano` especifica que este atributo é obrigatório, ou seja, deve fazer parte de todos os elementos `livro`.

### 4.4 Modelos de Dados para XML

Um documento XML pode ser visto como uma “linearização” de uma estrutura em árvore, onde os nós da árvore podem representar elementos, atributos, instruções de processamento, entre outros. A Fig. 13 apresenta a representação em árvore para o documento “livraria.xml”, de acordo com o modelo de dados XML Query Data Model [Fernandez 2001]. Este modelo de dados é resultado de um refinamento do modelo de dados descrito na especificação XPath [Clark 1999a], no qual um documento é modelado como uma árvore de nós. No modelo de dados XML Query Data Model, os elementos são representados como nós, os subelementos são representados como arcos para os nós e os atributos são representados como campos que podem ser acessados a partir de seus elementos.

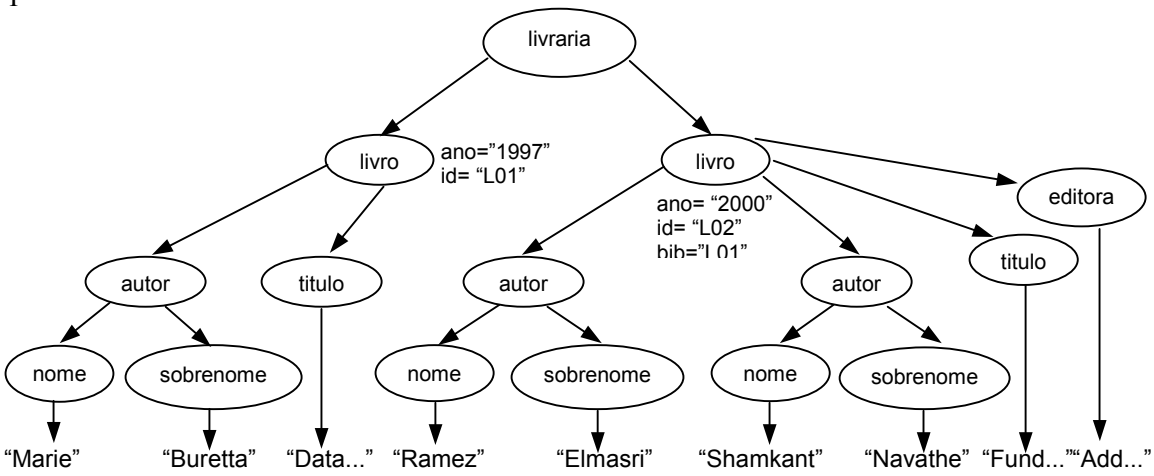


**Fig. 13 - Modelo de dados para o documento livraria.xml.**

Até o momento, quatro especificações de modelos de dados para XML foram proposta pelo W3C: o modelo de dados Infoset [Cowan 2001], o modelo de dados XPath [Clark 1999a], o modelo DOM [Apparo 1998], [Le Hors 2000] e o XML Query Data Model [Fernandez 2001]. Estes quatro modelos de dados descrevem um documento XML como uma estrutura em árvore, mas existem diferenças na estrutura das árvores e nas informações que estas disponibilizam.

### 4.5 Linguagens de Consulta para XML

Dados XML são diferentes dos dados de bancos de dados relacionais ou orientados a objetos e, por isso, as linguagens de consulta convencionais como SQL e OQL não são adequadas para especificar consultas em documentos XML. A principal distinção entre os dados convencionais e os dados XML está no fato de que os dados XML não seguem uma estrutura rígida, como os esquemas relacionais. Em modelos relacionais ou orientados a objetos, existe a noção de um esquema fixo, pré-definido e independente dos dados, a partir do qual as instâncias do banco de dados são definidas. Por outro lado, em XML o esquema existe juntamente com os dados, ou seja, XML é autodescritível e pode, naturalmente, modelar irregularidades que não podem ser modeladas quando modelos de dados relacionais ou orientados a objetos são usados. Por

exemplo, com o uso de XML é possível representar: i) itens de dados que podem ter elementos ausentes ou podem ter múltiplas ocorrências de um mesmo elemento, ii) elementos que podem ter valores atômicos em alguns itens de dados e valores estruturados em outros itens, e iii) coleções de elementos com estruturas heterogêneas.

Muitas linguagens para extração e reestruturação de dados XML têm sido propostas. Algumas dessas linguagens são baseadas nos conceitos de linguagens de consultas tradicionais, como SQL e OQL. Dentre as linguagens de consulta para XML destacam-se:

- **LOREL** foi projetada inicialmente como linguagem de consulta para dados semi-estruturados e, recentemente, foi estendida para manipulação de dados XML. Esta linguagem foi desenvolvida e implementada na Universidade de Stanford. LOREL é baseada nos conceitos de OQL, oferece um poderoso mecanismo para coerção de tipos e permite a definição de expressões de caminho complexas, que são extremamente úteis quando a estrutura do documento ainda não é conhecida [Abiteboul 1997b].
- **XML-QL** foi projetada por AT&T Labs como parte do projeto Strudel. Esta linguagem estende SQL com uma cláusula explícita CONSTRUCT que permite a construção de documentos como resultado de consultas e usa o conceito de *pattern* para encontrar os dados em um documento XML. XML-QL permite a definição de consultas e transformações nos dados XML, a fim de integrar dados XML de diferentes fontes de dados [Deutsch 1999].
- **XML-GL** foi projetada pela Politecnico di Milano, sendo uma linguagem de consultas gráfica, que tem como base uma representação em grafos de documentos XML e DTDs. Todos os elementos da linguagem são apresentados visualmente. Esta linguagem é adequada quando é necessária uma interface amigável para definição de consultas, semelhante ao QBE [Ceri 1999].
- **XSL** (*Extensible Stylesheet Language*) foi desenvolvida pelo *W3C XSL Working Group*. Um documento XSL consiste de uma coleção de regras *template*, onde cada *template* tem dois componentes: um *pattern*, que é comparado aos nós da árvore correspondente ao documento XML e um *template* que é instanciado para compor uma parte da árvore resultado. XSL utiliza a linguagem XPath para selecionar os elementos a serem processados e para geração de textos [Clark 1999b], [Schach 1998].
- **XQL** é uma notação para seleção de elementos e texto de documentos XML. XQL pode ser considerada uma extensão natural da sintaxe padrão de XSL e foi projetada com o objetivo de ser sintaticamente muito mais simples e compacta, entretanto com um poder de expressão reduzido. Esta linguagem foi projetada por J. Robie (Texcel Inc.), J. Lapp (webMethods, Inc.) e D. Schach (Microsoft Corporation) [Robie 1998a], [Robie 1998b].

### 4.5.1 XQuery

Para ilustrar a definição de consultas sobre dados XML, foi escolhida a linguagem de consultas XQuery [Chamberlin 2001]. Esta linguagem também é uma proposta da W3C, que agrega características de várias outras linguagens de consultas.

Assim como as linguagens de consulta para dados semi-estruturados, as linguagens de consulta para XML também usam o conceito de expressão de caminho para navegar em árvores. As expressões de caminho XQuery usam a sintaxe abreviada de XPath. Em XQuery, o resultado de uma expressão de caminho é uma lista ordenada de nós (cada nó inclui seus descendentes, logo o resultado pode ser visto como uma floresta ordenada). Os nós de nível mais alto da expressão de caminho são ordenados de acordo com sua posição na hierarquia original seguindo a ordem *top-down* e *left-right*.

Como visto anteriormente, uma expressão de caminho consiste de uma série de passos. Um passo representa um movimento ao longo de um documento em uma determinada direção. Cada passo pode aplicar um ou mais predicados para eliminar nós que não satisfazem uma determinada condição. O resultado de um passo é uma lista de nós que pode ser vista como um ponto de início para o próximo passo.

Uma expressão de caminho pode começar com uma expressão que identifica um nó específico, tal como a função `document` (*string*), que retorna o nó raiz de um dado documento. Uma consulta também pode conter uma expressão de caminho começando com "/" ou "//", que representa um nó raiz implícito, determinado pelo ambiente no qual a consulta está sendo executada. Uma discussão completa sobre a sintaxe abreviada de XPath pode ser encontrada em [Clark 1999a]. Além das expressões de caminho, existem outras expressões de XQuery que podem ser destacadas.

- **Expressões para construção de elementos**

Uma expressão para construção de elementos consiste de um marcador de início e um marcador de final que delimitam o elemento a ser criado. Estes marcadores agrupam uma série de outras expressões que provêem o conteúdo do elemento.

Exemplo: Gere um elemento `<livros>` contendo todos os títulos dos livros cujo autor tem como último nome "Abiteboul".

```
<livros>
   FOR $l IN ("livraria.xml")//livro
   WHERE $l/autor/sobrenome= "Abiteboul"
   RETURN $l/titulo
</livros>
```

- **Expressões FLWR**

Uma expressão FLWR (pronunciada "*flower*") é construída a partir de cláusulas FOR, LET, WHERE e RETURN. Assim como em uma consulta SQL, estas cláusulas devem aparecer em uma ordem específica. Uma expressão FLWR associa valores a uma ou mais variáveis e utiliza esses valores para construir um resultado. A primeira parte de uma expressão FLWR consiste de cláusulas FOR e/ou cláusulas LET que associam valores a uma ou mais variáveis. Os valores a serem associados às variáveis são representados por expressões (por exemplo: expressões de caminho).

Exemplo: Liste os títulos dos livros publicados pela editora "Addison Wesley". No ano de 1996.

```
FOR $l IN ("livraria.xml")//livro
WHERE $l/editora = " Addison Wesley"
AND $l/ano = "1996"
RETURN $l/titulo
```

- **Expressões condicionais**

Expressões condicionais são úteis quando a estrutura da informação a ser retornada depende de alguma condição. Assim como todas as outras expressões XQuery, as expressões condicionais podem ser aninhadas e podem ser usadas em qualquer lugar onde um valor é esperado.

Exemplo: Faça uma lista de todas as publicações, ordenadas por título. Para os livros, inclua a editora e para os artigos inclua a conferência onde eles foram publicados.

```
FOR  $p IN //publicacoes
RETURN
  <publicacoes>
    $p/titulo,
    IF $p/@tipo = "livro"
    THEN $p/editora
    ELSE $p/conferencia
  </publicacoes> SORTBY (titulo)
```

- **Expressões com quantificadores**

Em algumas situações pode ser necessário testar a existência de algum elemento que satisfaz uma determinada condição, ou para determinar se todos os elementos em alguma coleção satisfazem uma condição. Para isto XQuery provê quantificadores existenciais e universais, os quais são exemplificados a seguir.

Exemplo: Encontre todos os títulos de livro no qual a palavra "web" é mencionada em todos os parágrafos.

```
FOR $l IN //livro
WHERE EVERY $p IN $l//paragrafo SATISFIES
            contains ($p, "web")
RETURN $l/titulo
```

## Referências


Abiteboul, S. (1997a) "Querying semi-structured data", In Proc. of the 6th International Conference on Database Theory, p. 1-18, 1997.

Abiteboul, S., J.Mchugh, D., Widom, J. and Wiener, J. (1997b) "The lorel query language for semistructured data", International Journal on Digital Libraries, vol. 1, no. 1, p.68-88.

Abiteboul, S., McHugh, J., Rys, M., Vassalos V. and Weiner, J. (1998) "Incremental maintenance for materialized views over semistructured data," In Proc. of the 24th International Conference on Very Large Data Bases, p.38-49.

Abiteboul, S., Buneman, P. and Suciu, D., Data on the Web, 1st. Ed. Morgan Kaufmann Publishers, 2000.

Ambite, J., Ashish, N., Barish, G., Knoblock, A. C., Minton, S., Modi, P., Muslea, I., Philpot, A. and Tejada, S. (1998) "Ariadne: a system for constructing mediators for internet sources," In Proc. of ACM SIGMOD Conf. on Management of Data, Seattle, WA.

Apparo, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Le Hors, A., Nicol, G., Robie, J., Sutor, R., Wilson, C. and Wood, L. (1998) "Document Object Model

(DOM) Level 1 Specification (Second Edition) Version 1.0", World Wide Web Consortium, http://www.w3.org/TR/1998/ REC-DOM-Level-1-19981001/.

Arens, V., Chee, C. Y., Hsu, C-N. and Knoblock, C. A. (1993) "Retrieving and integrating data from multiple information sources," International Journal on Intelligent and Cooperative Information Systems, vol. 2, no. 2, p.127-158.

Arocena, G. O. and Mendelzon, A. O. (1998) "Weboql: restructuring documents, databases, and webs", In Proc. of Int. Conf. on Data Engineering (ICDE), Orlando, Florida, p. 24-33.

Baru, C., Gupta, A., Ludascher, B., Marciano, R., Papakostatinou, Y., Velikhov, P. and Chu, V. (1999) "Xml-based information mediation with mix", In Proc. of ACM SIGMOD Conf. On Management of Data.

Batini, C., Lenzerini, M. and Navathe, S. B. (1986) "A comparative analysis of methodologies for database schema integration", ACM Computing Surveys, vol. 18, no 4.

Bergamaschi, S., Castano, S., De Capitani Di Vimercati, S., Montanari, S. and Vincini, M. (1998) "A semantic approach to information integration: the momis project", In Proc. of Sesto Convegno della Associazione Italiana per l'Intelligenza Artificiale.

Biron, P. V. and Malhotra, A. (2000) "XML Schema Part 2: Datatypes", World Wide Web Consortium, http://www.w3.org/TR/xmlschema-2.

Bray, T., Paoli, J. and Sperberg-McQueen, C. M. (1998) "Extensible Markup Language (XML) 1.0", World Wide Web Consortium, http://www.w3.org/TR/REC-xml.

Bray, T., Hollander, D. and Layman, A. (1999) "Namespaces in XML", World Wide Web Consortium, http://www.w3.org/TR/REC-xml-names/.

Buneman, P., Davidson, S. B, Hillebrand, G. G. and Suciu, D. (1996) "A query language and optimization techniques for unstructured data", In Proc. of ACM SIGMOD Conf. on Management of Data, Montreal, Canada, p. 505-516.

Buneman, P. (1997) "Semi-structured data", In Proc. of ACM Symp. on Principles of Database Systems.

Castano, S. and De Antonellis, V. (1999) "Building views over semistructured data sources", In Proc. of 18th International Conference on Conceptual Modeling, Paris, France, p.146-160,.

Ceri, S. and Widom, J. (1991) "Deriving production rules for incremental view maintenance", In Proc. of Intl. Conf. on Very Large Data Bases, p. 577-589.

Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi S. and Tanca, L. (1999) "Xml-gl: a graphical language for querying and restructuring www data", In Proc. of 8th Int. World Wide Web Conference, WWW8, Toronto, Canada.

Chamberlin, D., Florescu, D., Robie, J., Siméon, J. and Stefanescu, M. (2001) "XQuery: A Query Language for XML", World Wide Web Consortium, http://www.w3.org/ TR/xquery/.

Chawathe, S., Garcia Molina, H. and Hammer, J. (1994) "The tsimmis project: integration of heterogeneous information sources", In Proc. of 10th Meeting of the Information Processing Society of Japan (IPSJ).

Clark, J. (1999a) “Xml Path Language (XPATH)”, World Wide Web Consortium, http://www.w3.org/TR/xpath.

Clark, J. (1999b) “XSL Transformations (XSLT specification)”, World Wide Web Consortium, http://www.w3.org/TR/WD-xslt.

Cowan J. and Tobin R. (2001) “XML Information Set”, World Wide Web Consortium, http://www.w3.org/TR/xml-infoset/

Davidson, A., Fuchs, M., Hedin, M. et al. (199) “Schema for Object-Oriented XML 2.0”, World Wide Web Consortium, http://www.w3.org/TR/NOTE-SOX.

Deutsch, A., Fernandez, M., Florescu, D., Levy, A. and Suciu, D. (1999) “A query language for xml”, In International World Wide Web Conference.

Fernandez, M. F., Florescu, D., Levy, A.Y. and Suciu, D. (1997) “A query language for a web-site management system,” SIGMOD Record, vol. 26, no. 3, p. 4-11,.

Fernandez, M. and Robie, J. (2001) “Query Datamodel”, World Wide Web Consortium, http://www.w3.org/TR/query-datamodel/.

Florescu, D., Levy A. and Mendelzon, A. (1998) “Database techniques for the world wide web: a survey”, ACM SIGMOD Record, vol.27, no.3, p.59-74.

Frankston C. and Thompson, H. S. (1998) “XML-Data Reduced”, Internet Document, http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm.

Genesereth, M., Keller, A. and Dushka, O. (1997) “Infomaster: an information integration system”, In Proc. of ACM SIGMOD.

Goldman, R. and Widom, J. (1997) “DataGuides: enabling query formulation and optimization in semistructured databases”, In Proc. of 23rd International Conference on Very Large Data Bases (VLDB), p. 436-445, 1997.

Goldman, R., Mchugh, J. and Widom, J. (1999) “From semistructured data to xml: migrating the lore data model and query language”, In Proc. of the 2nd International Workshop on the Web and Databases (WebDB '99), Philadelphia, Pennsylvania.

Gupta, A. and Mumick, I. S. (1995a) “Maintenance of materialized views: problems, techniques, and applications”, IEEE Data Eng. Bulletin, Special Issue on Materialized Views and data Warehousing.

Gupta, A. and Blakeley, J.A. (1995b) “Using partial information to update materialized views”, Information Systems, vol. 20, no. 8, p. 641-662.

Gupta, A. and Mumick, I. S. (1996) “What is the data warehousing problem? (Are materialized views the answer ?)”, In Proc. of the 22rd VLDB Conference.

Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M. M. and Vassalos, V. (1998) “Template-based wrappers in the tsimmis system”, In Proc. of ACM SIGMOD Conf. on Management of Data, Tucson, Arizona.

Himmeröder, R., Lausen, G., Ludäscher, B. and Schlepphorst, C. (1997) “On a declarative semantics for web queries”, In Proc. of the Int. Conf. on Deductive and Object-Oriented Databases (DOOD), Singapore, p. 386-398.

Hull, R. (1997) “Managing semantic heterogeneity in databases: a theoretical perspective”, In Proc. of ACM Symp. on Principles of Database Systems, p. 51-61.

Huyn, N. (1997) “Multiple view-self maintenance in data warehousing environments”, In Proc. of the 23rd Very Large Data Bases Conference, Athens, Greece.

Jellife, R. (2000) “Schematron”, Internet Document, http://www.ascc.net/ xml/resource/ schematron/.

Kim, W., Modern Database Systems, Addison-Wesley Pub. Co., 1995.

Klarlund, N., Moller, A. and Schwatzbach, M. I. (2000) “DSD: a schema language for xml”, In Proc. of 3rd ACM Workshop on Formal Methods in Software Practice.

Konopnicki, D. and Shmueli, O. (1995) “A query system for the world wide web”, In Proc. of Int. Conf. on Very large Databases (VLDB), Zurich, Switezerland, p. 54-65.

Kushmerick, N., Doorenbos, R. and Weld, D. (1997) “Wrapper induction for information extraction”, In Proc. of the 15th International Joint Conference on Artificial Intelligence.

Lakshmanan, L. V. S., Sadri, F. and Subramanian, I. N. (1996) “A declarative language for querying and restructuring the web”, In Proc. of 6th. International Workshop on research Issues in Data Engineering, RIDE'96, New Orleans.

Le Hors, A., Le Hégaret, P., Wood, L., Nicol, G., Robie, J., Champion, M. and Byrne, S. (2000) “Document Object Model (DOM) Level 2 Core Specification Version 1.0”, World Wide Web Consortium, http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/.

Lee D. and Chu, W.W. (2000) “Comparative analysis of six xml schema languages”, SIGMOD Record, vol. 29, no. 3, p. 76-87.

Levy, A. Y. (1999) “Combining artificial intelligence and databases for data integration”, Artificial Intelligence Today, p. 249-268.

McGrath, S., Xml by Example: Building E-Commerce Applications, Prentice Hall Computer Books, 1998.

Mendelzon, A. O., Mihaila, G. A. and Milo, T. (1997) “Querying the world wide web”, Int. J. on Digital Libraries, vol. 1, no. 1, p. 54-67.

Mirbel, I. (1995) “A fuzzy thesaurus for semantic integration of design schemes”, In J. Sharpe, editor, AI System Support for Conceptual Design (LIWED), p. 319-335.

Navathe, S. B., Elmasri, R. and Larson, J. (1986) “Integrating user views in database design,” IEEE Computer, vol. 19, no. 1, p. 50-62.

Nica, A. and Rundensteiner, E. A. (1999) “View maintenance after view synchronization”, In Proc. of International Database Engineering and Application Symposium (IDEAS'99).

Papakonstantinou, Y., Garcia-Molina, H. and Widom, J. (1995) “Object exchange across heterogeneous information sources”, In Proc. of the Eleventh International Conference on Data Engineering (IEEE Computer Society), Taipei, Taiwan, p. 6-10.

Robie, J. (1998a) “The design of XQL”, World Wide Web Consortium, http://www.w3.org/ Style/XSL/Group/1998/09/XQL-design.html.

Robie, J., Lapp, J. and Schach, D. (1998b) “Xml query language (xql)”, In Proc. of the Query Languages workshop, Cambridge, Mass.

S. Russell and P. Norvig, Artificial Intelligence – A modern Approach, Prentice Hall Series in Artificial Intelligence, New Jersey, 1995.

Savasere,A., Sheth, A., Gala, S., Navathe S. and Marcus, H. (1991) "On applying classification to schema integration", In *Proc. of the First International Workshop on Interoperability in Multidatabase Systems*, Kyoto.

Schach, D., Lapp, J. and Robie, J. (1998) "Querying and transforming xml", In Proc. of the Query Languages workshop, Cambridge, Mass.

Sheth, A. P. and Larson, J. (1990) "A federated database systems for managing distributed, heterogeneous, and autonomous databases", Computing Surveys, vol. 22, no. 3, p.183-236.

Spaccapietra, S. and Parent, C. (1994) "View integration: a step forward in solving structural conflicts", IEEE Trans. on Software Engineering, vol. 6, no. 2.

Subrahmanian, V. S., Adali, S., Brink, A., Emery, R., Lu, J., Rajput, A., Rogers, T.J., Ross, R. and Ward, C. (1995) "Hermes: a heterogeneous reasoning and mediator system", Technical Report, University of Maryland.

Thompson, H. S., Beech, D., Maloney, M. and Mendelsohn, N. (2000) "XML Schema Part 1: Structures", World Wide Web Consortium, http://www.w3.org/TR/xmlschema-1.

Tompa, F.W. and Blakeley, J. A. (1988) "Maintaining materialized views without accessing base data", Informations Systems, vol. 13, no. 4, p. 393-406.

Vidal, V. M. P. and Lóscio, B. F. (1999) "Solving the problem of semantic heterogeneity in defining mediator update translators", In Proc. of 18th International Conference on Conceptual Modeling, Paris, France, p.293-308.

Vidal, V. M. P., Lóscio, B. F. and Salgado, A. C. (2001) "Using correspondence assertions for specifying the semantics of xml-based mediators", In Proc. of WIIW 2001 - International Workshop on Information Integration on the Web - Technologies and Applications, Rio de Janeiro, Brasil.

Zhou, G., Hull, R. and King, R. (1996) "Generating data integration mediators that use materialization", Journal of Intelligent Information Systems, vol. 6, no. 2/3, p.199-221.

Zhuge, Y. and Garcia-Molina, H. (1998) "Graph structured views and their incremental maintenance", In Proc. of International Conference on Data Engineering, Orlando, Florida.

Widerhold, G. (1992) "Mediators in the architecture of future information systems", IEEE Computer, p.38-49.

Widom, J. (1995) "Research problems in data warehouse", In Proc. of the 4th Int'l Conference on Information and knowledge Management (CIKM).