UNIVERSIDADE FEDERAL DE CAMPINA GRANDE - UFCG
CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO - DSC
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

INTEGRAÇÃO DE BANCOS DE DADOS FEDERADOS NA WEB USANDO AGENTES MÓVEIS

Philip Stephen Medcraft

(Mestrando)

Ulrich Schiel

(Orientador)

CAMPINA GRANDE, FEVEREIRO DE 2003

Philip Stephen Medcraft

Integração de Bancos de Dados Federados na Web usando Agentes Móveis

Dissertação submetida ao Curso de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande, como requisito parcial para a obtenção de grau de Mestre em Informática.

Área de concentração: **Ciência da Computação**
Linha: **Sistemas de Informação e Banco de Dados**

Ulrich Schiel
Orientador

Campina Grande, Fevereiro de 2003

“A vida é aquilo que acontece enquanto
fazemos planos para o futuro”.

John Lennon

Agradecimentos

Primeiramente a **Deus** por sua misericórdia, amor e fidelidade.

À **minha filha** que mesmo muito nova foi a minha maior incentivadora.

Aos **meus pais e irmãs** pelo imenso apoio em todas as horas.

À **minha esposa** pelo carinho e paciência.

Ao **meu orientador Ulrich Schiel** pela amizade, apoio e conhecimento transmitido.

Ao professor **Cláudio Baptista** pela amizade, motivação e conhecimento transmitido.

A todos os **meus colegas** do LSI que sempre me apoiaram.

Aos **meus amigos**, Claudivan, Petrônio, Plácido, Éder, Josué, Alex, Rebouças, Jarbas, Guga, Vilar e Wesley, pelo companheirismo e força nos momentos mais difíceis.

Às funcionárias da COPIN, **Aninha** e **Vera**, pelo apoio e incentivo.

Aos **professores** da COPIN, pelos conhecimentos transmitidos e apoio na execução deste trabalho.

Aos **meus irmãos** da ACEV pelas orações.

À **CAPES** pelo suporte financeiro durante todo o trabalho de Mestrado.

Resumo

O problema clássico da integração de dados tem sido tratado há muito tempo. A Web Semântica está buscando definir uma infra-estrutura que permita a comunicação entre computadores na web. É uma visão que trata o problema da heterogeneidade semântica através do uso de ontologias para o compartilhamento de informações. Apesar de terem sido construídas para a web, as idéias da Web Semântica podem ser utilizadas em um ambiente de banco de dados federado, no qual as fontes de dados são previamente conhecidas e existe um esquema global, descrito em uma ontologia, que possibilita a integração dos dados. Para que o poder da Web Semântica seja perceptível, é necessário que sejam criados agentes de software que utilizem a infra-estrutura proporcionada pela aplicação desta visão, de forma a otimizar o processo de integração. Com base em ontologias, agentes de software serão capazes de encontrar os dados necessários nos locais adequados e processar a integração o mais cedo possível. Nesta dissertação apresentamos uma solução, denominada DIA, para a integração semântica de um banco de dados federado usando agentes móveis e ontologias.

Abstract

The classic problem of information integration has been addressed for a long time. The Semantic Web project is aiming to define an infrastructure that enables machine communication on the web. This is a vision that tackles the problem of semantic heterogeneity by using ontologies for information sharing. Although designed to be used on the entire web, the ideas behind the Semantic Web can be used in a federated database environment, in which the data sources are previously known and there is a global schema, described in an ontology, which enables data integration. In order that the power of the Semantic Web be perceived, software agents that utilize the infrastructure provided by the application of this vision need to be created, so as to improve the integration process. Based on ontologies, software agents will be capable of finding the correct data on the adequate sites of the federation, and achieve the integration process as soon as possible. In this dissertation we present a solution, known as DIA, for semantic integration of a federated database using mobile agents and ontologies.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O problema clássico da integração de dados tem sido tratado há muito tempo. Um dos principais objetivos de um sistema de integração de dados é preparar um ambiente de consulta, através do qual usuários possam realizar consultas sem conhecimento de como e onde os dados estão armazenados. Atualmente, existem duas abordagens de integração:

- Materializada: Os dados são previamente extraídos das fontes de dados e integrados em um único esquema, gerando um grande banco de dados, no qual usuários realizam suas consultas;
- Virtual: Uma vez que um usuário executa uma consulta, os dados são extraídos e integrados em tempo de execução a partir das fontes de dados.

A decisão de qual abordagem seguir deve ser baseada nas vantagens e desvantagens de cada uma. Com a abordagem materializada, não ocorre tradução de esquemas em tempo de execução, uma vez que, os dados são previamente extraídos e integrados em um único banco de dados. Dessa forma, essa abordagem tem como principal vantagem o desempenho. No entanto, manter o grande banco de dados atualizado representa uma tarefa complicada, pois, o conteúdo deste nem sempre reflete o estado atual das suas fontes.

No que diz respeito ao desempenho, a abordagem virtual perde para a abordagem materializada, uma vez que, ocorre a tradução de esquemas em tempo de execução. No entanto, a grande vantagem desta abordagem é que os dados extraídos e integrados a partir das fontes estão sempre atualizados. Além disso, somente os dados realmente necessários são extraídos, enquanto que, na forma materializada todos os dados são copiados antecipadamente.

Foram propostas três arquiteturas de banco de dados para solucionar o problema da integração de dados. São eles: Banco de Dados Federado, que segue a abordagem virtual; Data Warehouse, que segue a abordagem materializada; e Mediador, que segue a abordagem virtual. No caso dos Bancos de Dados Federados ainda é feita uma distinção se o Banco de Dados Federado possui um esquema global ou não. No primeiro caso, chamado de Banco de Dados Federado Fortemente Acoplado, o esquema global pode facilitar bastante o

processamento de consultas, mas, por outro lado, trata-se de uma federação fechada, ou seja, os membros são previamente fixados e novos membros devem passar por um processo de ingresso na federação.

(Garcia-Molina *et al*, 2001), apresentam algumas questões importantes referentes à integração de dados:

- As fontes podem possuir diferentes tipos de dados;
- As fontes podem utilizar nomes ou formatos diferentes para o mesmo objeto;
- Termos podem possuir semânticas diferentes, dependendo da forma com que são estruturados nas fontes;
- Alguns dados podem não estar presentes em todas as fontes.

Atualmente, a web interliga uma quantidade gigantesca de informações dispersas em inúmeras fontes de dados. Com o uso cada vez mais freqüente de bancos de dados na web, surgiu a necessidade de integrá-los. Para tanto, pesquisas têm sido realizadas no sentido de incorporar mecanismos dos bancos de dados tradicionais, de forma a possibilitar a integração de dados de fontes distribuídas e heterogêneas na web.

A visão da Web Semântica está buscando definir uma infra-estrutura capaz de possibilitar a comunicação entre computadores na web. Esta visão trata o problema da heterogeneidade semântica através do uso de ontologias na definição de esquemas globais para o compartilhamento de dados. Apesar de terem sido construídas para a web, as idéias da Web Semântica podem ser utilizadas em um ambiente de banco de dados federado, no qual as fontes de dados são previamente conhecidas e existe um esquema global, descrito em uma ontologia, que possibilita a integração dos dados.

Com base na ontologia, agentes de software podem realizar o processo de integração dos dados. No nosso contexto, agentes móveis representam uma ótima opção para promover a extração e integração de dados a partir das fontes distribuídas. Enviar agentes móveis para onde os dados estão armazenados pode diminuir o tempo de busca e recuperação dos dados (Goldschmidt *et al*, 2002).

Portanto, esta dissertação tem como foco principal apresentar uma nova solução para promover a integração semântica de dados em um banco de dados federado, usando agentes móveis e ontologias. Como classe de aplicação de banco de dados federado, idealizamos um ambiente contendo um conjunto de bancos de dados do modelo relacional que mantêm informações semelhantes, porém, apresentando heterogeneidades sintáticas e semânticas entre

seus esquemas. É considerada a possibilidade de replicação de dados a nível de registros, de tabelas, ou até de um banco de dados inteiro. A autonomia de modificação dos esquemas locais dos bancos de dados é preservada, considerando as conseqüentes manutenções do esquema global de forma a garantir a consistência da federação. A nossa solução promove a integração de dados em um ambiente de banco de dados federado com estas características, promovendo uma interface de consulta (apenas de leitura) para usuários localizados em qualquer ponto da web.

1.1 Motivação

Como citamos anteriormente, as arquiteturas para integração de bancos de dados que seguem a abordagem virtual de integração, possuem a baixa performance como maior desvantagem. Além da necessidade de traduzir esquemas em tempo de execução, existem outras razões para esta baixa performance:

- Acesso desnecessário a todas as fontes de dados: em cada consulta, todas as fontes de dados são acessadas;
- Fluxo de dados excessivo: a integração dos dados só ocorre após a extração a partir de todas as fontes, gerando um fluxo excessivo de dados.

1.2 Contribuição

Com este trabalho, foi desenvolvido um sistema de integração baseado na visão da Web Semântica e no paradigma de agentes móveis, solucionando alguns dos problemas enfrentados por sistemas que seguem a abordagem virtual de integração de dados. O uso de agentes móveis tem se mostrado adequado para a redução do fluxo de informações e na escolha das fontes de dados a serem consultadas.

1.3 Objetivos

Os principais objetivos deste trabalho são:

- Especificar e projetar uma arquitetura de integração de dados federados, baseada na visão da Web Semântica e no paradigma de agentes móveis;

- Desenvolver um mecanismo para garantir que o agente móvel visite apenas as fontes necessárias, ou seja, que possuem dados para a consulta;
- Realizar a integração dos dados diretamente nas fontes, de forma a garantir que, após visitadas todas as fontes, os dados já estejam devidamente integrados;
- Capacitar o agente móvel para que este, dependendo do resultado parcialmente obtido, verifique se a consulta foi satisfeita, e com isso, decida se deve ou não retornar à origem.

1.4 Estrutura da Dissertação

Esta dissertação é constituída de cinco capítulos, incluindo esta Introdução. Os capítulos estão organizados da seguintes forma:

- O capítulo 2 apresenta alguns problemas que envolvem um sistema de integração de dados e as arquiteturas de integração de dados existentes. Além disso, apresentamos a WEB Semântica e uma visão geral de agentes de software, concentrando-nos em agentes móveis.
- O capítulo 3 apresenta uma descrição completa do sistema de integração de dados que estamos propondo. Descrevemos a arquitetura e os padrões de mobilidade dos agentes móveis, bem como a integração destes com a proposta da WEB Semântica. Apresentamos como ocorre a integração dos dados entre as fontes heterogêneas, ressaltando o mapeamento da consulta do usuário para os esquemas locais a partir de um esquema global único.
- O capítulo 4 apresenta uma visão prática de implementação por meio de um estudo de caso.
- O capítulo 5 relata os resultados obtidos e estabelece as direções de pesquisa para este trabalho.

Capítulo 2

Integração de dados

A integração de dados distribuídos e heterogêneos tornou-se um requisito essencial para a sociedade da informação desde os primeiros anos de desenvolvimento da ciência da computação (Belian & Salgado, 2002). A necessidade de integração de dados deu origem a diferentes arquiteturas de bancos de dados, algumas delas obtendo grande sucesso. Com a explosão da web, novas propostas de integração estão sendo desenvolvidas, sendo um dos temas de banco de dados mais pesquisados atualmente.

Neste capítulo, apresentamos uma noção geral dos conceitos que envolvem a nossa pesquisa. Na seção 2.1, apresentamos alguns problemas relacionados à integração de dados entre bancos de dados distintos. Na seção 2.2 apresentamos as arquiteturas propostas para solucionar o problema da integração de dados. Na seção 2.3, apresentamos a Web Semântica, ressaltando como esta visão pode proporcionar novos meios para a integração de dados. Na seção 2.4 introduzimos o conceito de agentes de software. Na seção 2.5 apresentamos uma visão geral de agentes móveis.

2.1 Heterogeneidades entre bancos de dados

O projeto de um banco de dados é realizado através da representação de conceitos do mundo exterior em um esquema interno. O projetista modela o esquema do banco de dados segundo a sua visão do mundo, criando um conjunto de estruturas que representam conceitos e relacionamentos entre estes conceitos. Assim, cada elemento de um esquema de banco de dados está semanticamente relacionado a algum conceito do mundo exterior (García-Solaco *et al*, 1996).

A probabilidade de existir heterogeneidades entre bancos de dados distintos é muito grande, uma vez que, os mundos modelados pelos projetistas não são os mesmos. Caso os mundos modelados sejam coincidentes, ainda assim, deverão existir heterogeneidades, pois, os projetistas terão visões diferentes do mundo, portanto, terão projetados esquemas diferentes. Nesta seção, descrevemos as heterogeneidades mais comuns encontradas entre bancos de dados distintos.

2.1.1 Diferenças entre atributos

É bastante comum encontrar diferenças de nomes de atributos entre bancos de dados distintos. Por exemplo, um determinado banco de dados BD1 possui um atributo denominado *Identidade*, enquanto isso, um outro banco de dados BD2 possui um atributo correspondente denominado *NumRGIdentidade*. Ambos expressam o mesmo conceito, porém apresentam sintaxes diferentes.

Outra diferença bastante comum é a falta de atributos. Por exemplo, um banco de dados BD1 possui uma entidade *Comprador* com um atributo denominado *DataUltimaCompra*, enquanto isso, um outro banco de dados BD2 não possui um atributo correspondente.

Podemos encontrar diferenças de restrições dos atributos. Por exemplo, o atributo *DataUltimaCompra* da entidade *Comprador* do BD1 permite valores nulos, enquanto que, um atributo correspondente em um banco de dados BD3 não permite.

Outro exemplo de diferença de restrição dos atributos refere-se ao uso de valores *default*. Por exemplo, um valor *default* definido para um determinado atributo em um banco de dados BD1 pode ser diferente do valor *default* definido para o atributo correspondente em um outro banco de dados BD2. Além disso, um outro banco de dados BD3 pode ainda não definir qualquer valor *default* para o atributo.

Além disso, atributos correspondentes em diferentes bancos de dados podem ser de tipos diferentes, unidades diferentes, como também, podem possuir diferentes níveis de precisão.

2.1.2 Diferenças entre classes de entidades

É bastante comum encontrar diferenças de nomes de entidades entre bancos de dados distintos. Um banco de dados BDLoja1 possui uma entidade denominada *Comprador*, enquanto isso, um outro banco de dados BDLoja2 possui uma entidade denominada *Cliente*. Ambos expressam a mesma semântica, porém com sintaxes diferentes. A Figura 1 ilustra o esquema do banco de dados BDLoja1, e a Figura 2 ilustra o esquema do banco de dados BDLoja2.

No exemplo, *Comprador* e *Cliente* são sinônimos, ou seja, representam o mesmo conceito. Também podemos encontrar homônimos entre dois bancos de dados, ou seja, conceitos diferentes representados pelo mesmo nome.

Cada projetista de banco de dados pode adotar um esquema diferente na definição das chaves de cada entidade. Por exemplo, uma entidade *Comprador* de um banco de dados BD1 pode adotar o nome de cada cliente como chave da entidade, enquanto que, um outro banco de dados BD2 pode adotar algum tipo de código. Este código pode estar baseado em algum código existente no mundo exterior, como também, pode ser uma criação do projetista do banco de dados.

O número de classes de entidades pode variar bastante entre os esquemas dos bancos de dados. Por exemplo, a entidade *Comprador* do banco de dados BDLoja1 contém os atributos *Endereço*, *Bairro*, *Cidade*, *CEP* e *Estado*. Enquanto isso, o esquema do outro banco de dados BDLoja2 apresenta estes atributos em uma outra entidade, denominada *Contato*, que mantém um relacionamento de 1-1 com a entidade *Cliente*.

Podem também existir diferenças de cardinalidade entre as entidades dos bancos de dados. Por exemplo, dadas as entidades *Comprador* e *Representante* do BDLoja1 com relacionamento de 1-n, encontramos duas entidades equivalentes *Cliente* e *Corretor* do BDLoja2 com relacionamento de n-m.

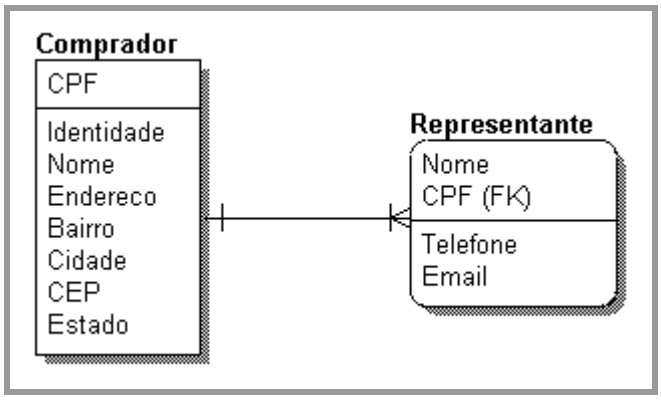

Figura 1: O esquema do banco de dados BDLoja1

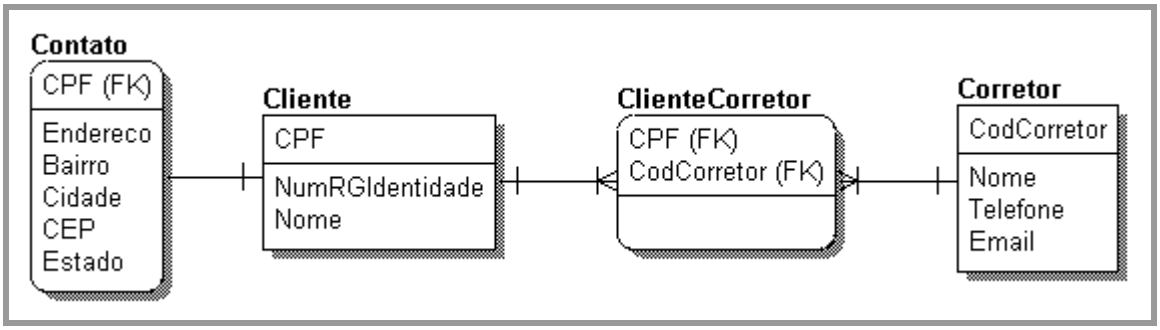

Figura 2: O esquema do banco de dados BDLoja2

2.2 Arquiteturas de integração de dados

Nesta seção, apresentamos três arquiteturas de bancos de dados para solucionar o problema da integração de dados.

2.2.1 Banco de Dados Federado

A arquitetura de *banco de dados federado* segue a abordagem virtual de integração. Em (Hurson & Bright, 1996), um banco de dados federado é descrito como um sistema distribuído que proporciona uma interface (esquema) global para SGBDs pré-existentes e heterogêneos. Neste tipo de sistema, usuários podem ter acesso a múltiplos bancos de dados remotos com uma única consulta. A localização dos bancos de dados é transparente para o usuário. O software do banco de dados federado realiza as devidas transformações entre o esquema da consulta global para o esquema de cada banco de dados local. Uma característica especial desta arquitetura é que os bancos de dados locais permanecem autônomos, ou seja, mantêm controle total sobre os dados e processamento locais. A arquitetura de uma federação de bancos de dados está ilustrada na Figura 3.

Esta arquitetura adota o seguinte princípio: ao invés de proporcionar transformações entre cada combinação de modelo de dados e linguagem dos bancos de dados, define-se um modelo de dados global e uma linguagem de consulta global. Dessa forma, cada modelo/linguagem local deverá ser convertido para apenas um modelo/linguagem global.

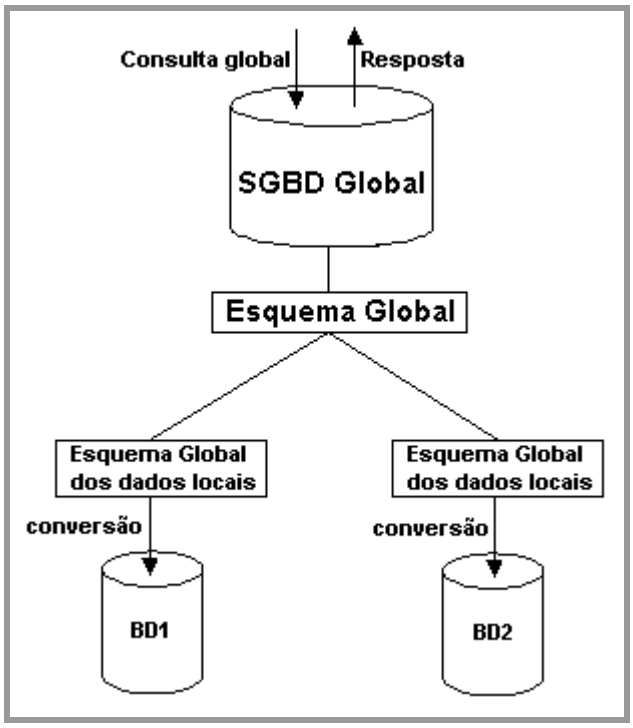

Figura 3: Arquitetura de uma federação de bancos de dados

Um sistema de banco de dados federado mantém um único esquema de dados global a partir dos esquemas dos bancos de dados locais, com o intuito de neutralizar os conflitos existentes entre estes esquemas. O esquema global é manipulado diretamente pelo usuário, através de uma única linguagem de definição e manipulação de dados, sem que necessite conhecer aspectos específicos dos esquemas dos bancos de dados locais. Em geral, um sistema de banco de dados federado permite apenas a realização de consultas, porém, algumas federações permitem a atualização dos seus componentes.

A seguir, destacamos algumas das principais funcionalidades de sistemas de bancos de dados federados:

- Integração de esquemas: permite uma visão integrada dos dados distribuídos;
- Gerência de consultas globais: análise, otimização e execução de consultas;
- Gerência de transações globais: trata das propriedades de atomicidade, confiabilidade, isolamento e durabilidade das transações no ambiente distribuído.

Existe também uma forma chamada de bancos de dados federados fracamente acoplados. Nesta forma, não há esquema global e a funcionalidade é semelhante aos sistemas com mediadores, a ser vista na seção 2.2.3 a seguir. Um sistema de banco de dados federado fracamente acoplado proporciona um nível mais alto de autonomia para os esquemas dos bancos de dados locais. Além disso, a ausência de uma centralização em um esquema único

(global) torna este tipo de arquitetura mais escalável, sendo adequado para federações de grande porte. No entanto, torna-se mais difícil manter a consistência da federação, como também, a localização dos bancos de dados deixa de ser transparente.

2.2.2 Data Warehouse

A arquitetura *data warehouse* segue a abordagem materializada de integração de dados. Nesta arquitetura, dados de várias fontes são extraídos e integrados em um único banco de dados, denominado *warehouse*, podendo ser consultados da mesma forma que em um banco de dados comum. Em geral, um data warehouse é analítico e não transacional, portanto, guarda a história dos dados, enquanto isso não é feito para bancos de dados federados, que são transacionais, sendo usados para OLTP[1]. A Figura 4 ilustra a arquitetura data warehouse.

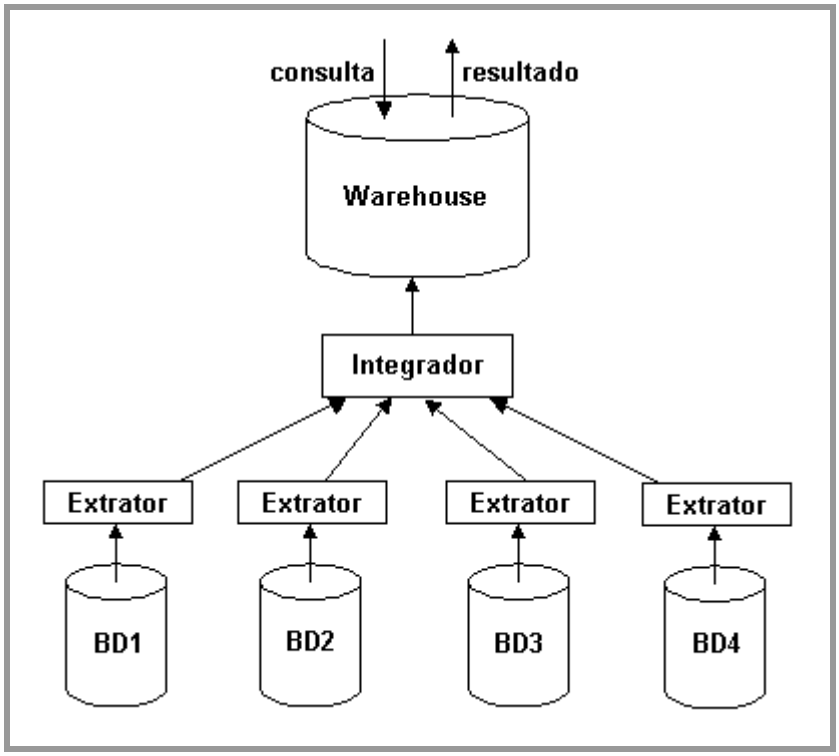

Figura 4: A arquitetura data warehouse

Além da performance, uma das grandes vantagens da arquitetura data warehouse é a disponibilidade das informações, uma vez que, consultas a ele podem ser imediatamente processadas. No entanto, isso implica que o warehouse nem sempre retorna o estado mais

[1] OLTP: On-Line Transaction Processing

recente dos dados. Dessa forma, torna-se necessário um mecanismo adequado de manutenção, promovendo a atualização dos dados contidos no warehouse.

Existem pelo menos três mecanismos de manutenção de um warehouse:

- Uma vez decorrido um certo espaço de tempo, o warehouse é completamente reconstruído, recebendo os dados atualizados das fontes. Esta reconstrução deve ocorrer em um período de tempo em que o warehouse não é consultado, uma vez que, é necessário interrompê-lo. Uma opção para evitar a interrupção do warehouse é gerar uma cópia e processar consultas na cópia durante a reconstrução;
- O warehouse é periodicamente atualizado com base nas modificações ocorridas nas fontes desde a última reconstrução. Dessa forma, a reconstrução envolve uma menor quantidade de dados, tornando-se um processo mais rápido. No entanto, engloba um processo bastante complexo, chamado de *atualização incremental*;
- O warehouse é atualizado imediatamente a cada mudança ocorrida em uma ou mais fontes. Esse mecanismo é pouco utilizado, uma vez que, exige muito processamento e comunicação entre o warehouse e suas fontes. Pode ser prático para um warehouse pequeno, com fontes que mudam pouco.

2.2.3 Mediadores

A arquitetura de *mediador* segue a abordagem virtual de integração de dados. Nesta arquitetura, um mediador envia a consulta do usuário para as fontes de dados. Em cada fonte, a consulta é traduzida, através de um *wrapper*, para a linguagem de consulta do SGBD local. Os resultados de cada consulta local são integrados e a resposta é entregue ao usuário. A Figura 5 ilustra a arquitetura de mediador.

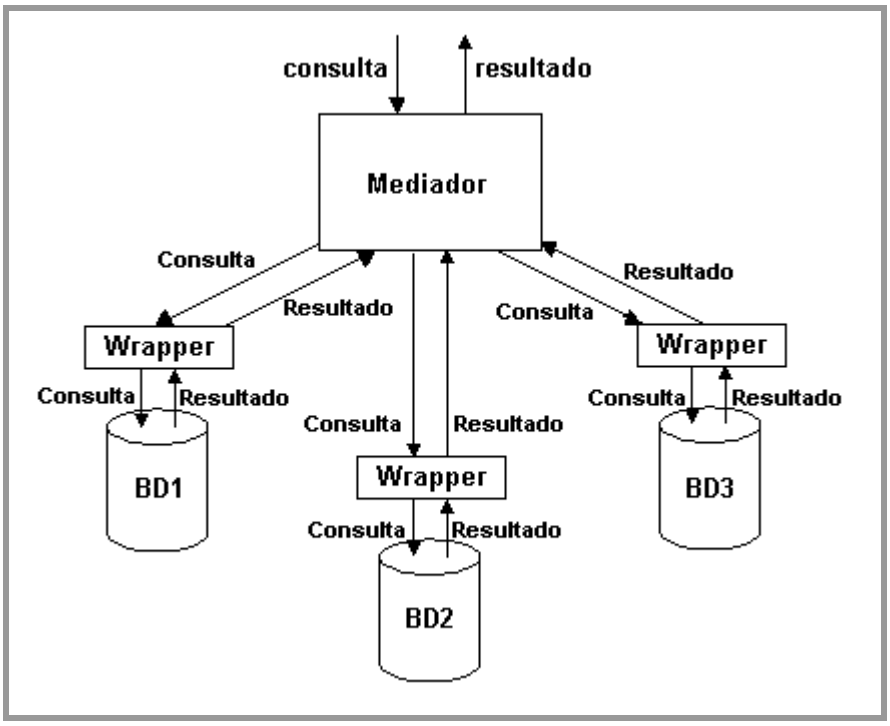

Figura 5: A arquitetura de mediador

Ao contrário do data warehouse, que segue a abordagem materializada, o mediador não armazena dados, portanto, todo o processo de extração e integração de dados é realizado sob-demanda. Esse processo é bastante adequado para fontes cujos dados mudam rapidamente, pois, os dados retornados sempre estão atualizados. No entanto, a tradução e integração de consultas sob-demanda tornam o baixo desempenho a maior desvantagem desta arquitetura.

2.3 Web Semântica

A World Wide Web atual é fundamentalmente uma mídia de publicação. Trata-se de um grande repositório de recursos (imagens, textos, etc.) para o consumo humano. A quantidade de recursos dispostos na web é enorme, enquanto que, descrições (metadados) do que cada recurso representa são escassas.

A adição de semântica modificará radicalmente a natureza da web – de um lugar onde as informações são meramente dispostas, para um lugar onde as informações são interpretadas, compartilhadas e processadas. A idéia é gerar uma web que não apenas liga um recurso ao outro, mas que também descreva o significado das informações contidas nos recursos, através de documentos que descrevem relacionamentos explícitos entre eles.

Segundo (Berners-Lee *et al*, 2001), a web semântica visa proporcionar estrutura ao conteúdo da web, criando um ambiente onde agentes de software poderão realizar tarefas sofisticadas aos usuários. Não se trata de uma web separada, mas uma extensão da web atual, na qual a informação possui um significado bem definido, possibilitando a cooperação entre computadores e pessoas.

Trata-se de um grande empreendimento. O primeiro passo é a criação de padrões que permitam a adição de metadados. O próximo passo é o desenvolvimento de métodos que permitam o relacionamento e a troca de metadados entre sistemas diferentes. A arquitetura de camadas para a web semântica, proposta por (Berners-Lee, 2000), está ilustrada na Figura 6.

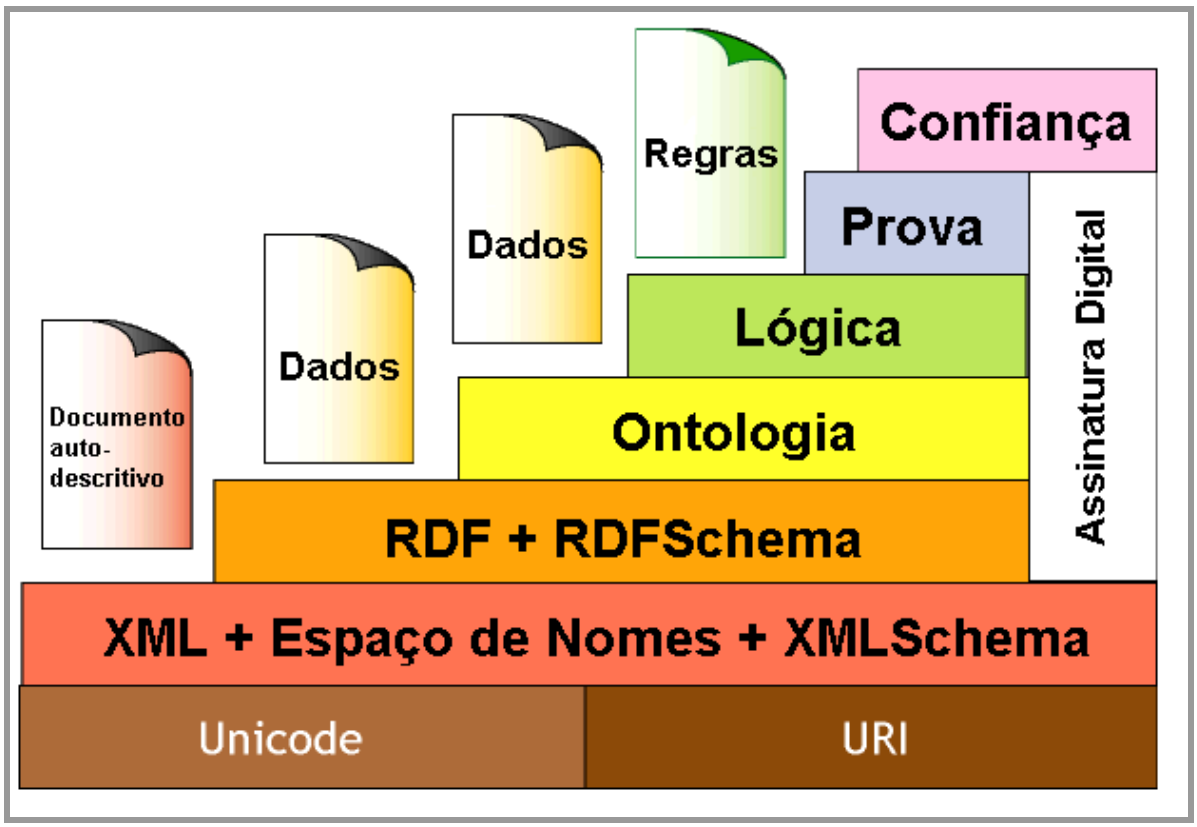

Figura 6: Arquitetura da web semântica

Melnik e Decker (Melnik & Decker, 2000) descrevem três níveis de interoperabilidade: *sintática*, *estrutural* e *semântica*. Na arquitetura descrita na Figura 6, o uso de XML proporciona a sintaxe necessária para garantir a interoperabilidade sintática; o RDF e RDFS (RDF Schema) proporcionam a interoperabilidade estrutural; e uma camada de Ontologia garante a interoperabilidade semântica.

Com base na semântica proporcionada pelas camadas inferiores, a camada de lógica permite o uso de regras para processamento dedutivo; a camada de prova executa as regras

geradas; e a camada de confiança avalia as regras, indicando às aplicações o nível de confiança destas. Atualmente, estão sendo desenvolvidas linguagens que atendam a estas camadas, como a linguagem RuleML (Boley, 2003).

Agentes de software não estão presentes na arquitetura da Figura 6. Na visão da web semântica, agentes de software são aplicações capazes de operar sobre o modelo, executando operações realmente úteis aos usuários. Estes agentes deverão compreender o significado e a relação entre os objetos, com base em ontologias, e de raciocinar sobre eles utilizando regras de inferência definidas na camada de lógica.

A autenticidade e confiabilidade das fontes adquirem um novo significado quando consideramos que agentes raciocinando sobre os dados podem chegar a conclusões que afetam a ação humana. As *assinaturas digitais* serão a forma de cada agente verificar a autenticidade das fontes (Afonso, 2001).

Por questões de escopo e devido ao fato das camadas de lógica, prova e confiança ainda estarem em fase de definições, decidimos concentrar nosso trabalho no nível da camada de ontologia. As seções seguintes descrevem sucintamente as camadas de XML, RDF(S) e Ontologia.

2.3.1 XML

A XML (eXtensible Markup Language) é uma linguagem de descrição e de troca de documentos estruturados. Apesar de ser uma tecnologia relativamente nova, a XML tem promovido um forte impacto na maneira como tratamos dados, tornando-se um padrão para o compartilhamento de dados. A linguagem XML influencia a maneira como vemos, processamos, transportamos, e gerenciamos dados, proporcionando várias utilidades que não eram possíveis no passado.

A linguagem XML é sintaticamente similar à linguagem HTML, sendo que uma das premissas básicas da XML é que todo e qualquer documento XML deverá ser bem formado, ou seja, deverá estar em conformidade com a sintaxe XML. Um interpretador XML não admite erros sintáticos, garantindo assim, uma manipulação bem mais confiável dos documentos gerados.

Um documento XML é organizado como uma hierarquia de elementos. Um elemento consiste em uma tag de abertura e uma tag de fechamento – por exemplo, <pessoa> e </pessoa>. Elementos podem conter outros elementos ou texto. Se um elemento não contiver

qualquer conteúdo, este pode ser abreviado como </pessoa>. Os elementos de um documento XML devem estar corretamente aninhados: as tags de abertura e fechamento de um elemento filho, devem estar entre as tags de abertura e fechamento do seu elemento parente. Todo documento XML deve possuir exatamente um elemento raiz, o qual contém todos os demais elementos do documento.

Como acontece com a linguagem HTML, elementos XML podem possuir atributos. Esses atributos são declarados como um par "nome = valor". Na Figura 7, o documento XML possui um elemento denominado <dica>, que possui vários atributos. O atributo "titulo" é o nome da dica, o atributo "autor" fornece uma abreviação do nome do autor da dica, e os atributos htmlURL e textoURL fazem referência a diferentes formatos de arquivos em que a dica está disponível.

```
<?xml version="1.0" encoding="UTF-8"?>
<dicas>
<autor id="phil" nomeCompleto="Philip Medcraft"/>
<dica  titulo="Como construir um documento XML"
       autor="phil"
       htmlURL="/~philip/dicas/2002/tt0509.html#dica1"
       textoURL="/~philip/dicas/txtarquivo/Junho01_Philip.txt"/>
</dicas>
```

Figura 7: Exemplo de documento XML

Como podemos ver no exemplo da Figura 7, a linguagem XML permite que o criador do documento especifique seus próprios elementos (tags). Assim, podemos afirmar que a linguagem XML é na verdade uma metalinguagem: um mecanismo para representar outras linguagens de forma padronizada. Isso torna a XML uma linguagem universalmente aplicável, através da qual, podemos definir linguagens de marcação customizadas para ilimitados tipos de documentos.

A linguagem XML trouxe consigo dois conceitos que, em primeiro instante, são bastante confundidos pelos seus novos usuários. Trata-se dos conceitos de *boa formação* e *validez* de documentos XML. Um documento XML é dito estar *bem formado*, quando este está em conformidade com a sintaxe da linguagem.

Para explicarmos claramente o conceito de *validez* de documentos XML, precisamos primeiramente descrever um DTD (Document Type Definition) ou um XML Schema. Basicamente, ambos são mecanismos com os quais o autor pode especificar a estrutura de

seus documentos XML. DTDs e XML Schemas não especificam o significado do conteúdo descrito pelo documento XML, no entanto, especificam os nomes dos elementos e atributos (o vocabulário), e como estes são usados nos documentos. Com isso, podemos validar documentos XML segundo a estrutura pré-definida por um DTD ou por um XML Schema. Dessa forma, podemos afirmar que um documento XML é dito ser válido quando este segue a estrutura pré-definida por um DTD ou por um XML Schema. A validação, segundo um destes mecanismos, permite a automação do tratamento dos dados e assegura uma possibilidade de controle de integridade.

O uso de DTDs é mais simples do que o uso de XML Schemas. Apesar disso, existe muita insatisfação em usar DTDs na definição de esquemas. Como primeira razão, podemos citar o fato do usuário ter que escrever seu documento XML usando uma sintaxe e o DTD usando outra. Uma segunda razão, é a capacidade limitada de tipos de dados que DTDs apresentam. Além disso, com DTDs não podemos usar técnicas da orientação a objetos, como herança. Assim, o XML Schema surge como um sucessor do DTD, buscando suprir essas e outras deficiências (Klein, 2001).

O XML Schema possui sintaxe XML e proporciona uma gramática mais rica para a descrição da estrutura de documentos XML. Além disso, um dos principais benefícios do uso de XML Schemas é a possibilidade de usar um mecanismo denominado *Espaço de Nomes*. Como já citamos, a linguagem XML permite que autores de documentos criem seus próprios elementos (tags). Com isso, pode acontecer que autores diferentes definam um mesmo elemento, no entanto, com semânticas diferentes. Dessa forma, é preciso haver um mecanismo que garanta a unicidade e devida separação de tais elementos. Assim, criou-se o Espaço de Nomes, que permite o uso de um esquema de atribuição de nomes ao longo de um conjunto de documentos. Cada Espaço de Nomes é identificado unicamente através de um URI (Universal Resource Identifier).

O exemplo a seguir apresenta o uso de Espaço de Nomes na solução de conflitos de nomes em documentos XML. Veja os documentos XML da Figura 8:

```
<?xml version="1.0"?>
<Cliente>
   <Nome>Philip</Nome>
   <Idade>24</Idade>
   <Endereco>
      <Rua>Rio Branco, 629</Rua>
      <Cidade>Patos</Cidade>
      <Estado>Paraíba</Estado>
   </Endereco>
</Cliente>
```

```
<?xml version="1.0"?>
<Servidor>
   <Nome>Canjica</Nome>
   <Endereco>150.165.75.164</Endereco>
</Servidor>
```

Figura 8: Documentos XML

Cada documento da Figura 8 utiliza um vocabulário XML diferente e cada vocabulário define um elemento denominado “Endereco”. Cada elemento “Endereco” possui um significado diferente, sendo interpretado por uma aplicação de maneiras diferentes. Isto não representa um problema desde que esses elementos existam em documentos separados. No entanto, caso eles sejam integrados em um mesmo documento, i.e., um documento descrevendo uma lista de departamentos (possuem endereço e servidores), uma determinada aplicação não conseguirá distinguir os elementos “Endereco” contidos no documento.

A solução para o problema descrito anteriormente é associar cada vocabulário a um Espaço de Nomes diferente, o que permite que utilizemos os elementos “Endereco” de ambos os vocabulários. O Espaço de Nomes associado a cada vocabulário é que distingue os elementos.

2.3.2 RDF e RDFS (RDF Schema)

O RDF (Resource Description Framework) é uma plataforma para descrição de metadados (Lassila & Swick, 1999). Trata-se de um modelo para a representação de dados sobre *recursos* na web. Um recurso, segundo o RDF, é qualquer coisa identificada por um URI. Além de recursos, o modelo RDF contém *propriedades* e *sentenças*. Uma propriedade é uma característica, atributo, ou relação que descreve um recurso. Uma sentença consiste de um recurso, uma propriedade para esse recurso, e o valor desta propriedade. Este valor da propriedade pode ser um outro recurso ou texto. Em resumo, uma descrição RDF é uma lista de triplas: um objeto (um recurso), um atributo (uma propriedade), e um valor (um recurso ou texto). Veja a Tabela 1.

Objeto	Atributo	Valor
http://www.livros.org/ISBN0012515866	type	http://descricao.org/schema/#livro

Tabela 1: Uma tripla RDF

Veja que a descrição RDF é uma tripla, onde o objeto é o recurso identificado pelo URI “http://www.livros.org/ISBN0012515866”, o atributo é a propriedade “type”, e o valor para o atributo é um recurso identificado pelo URI “http://descricao.org/schema/#livro”.

Um modelo RDF também pode ser representado através de um grafo de arestas rotuladas. Os nós são os recursos, as arestas são as propriedades, e o elemento terminal, representado por um retângulo, representa o valor da respectiva propriedade. O exemplo anterior pode ser representado através de um grafo, conforme ilustrado na Figura 9.

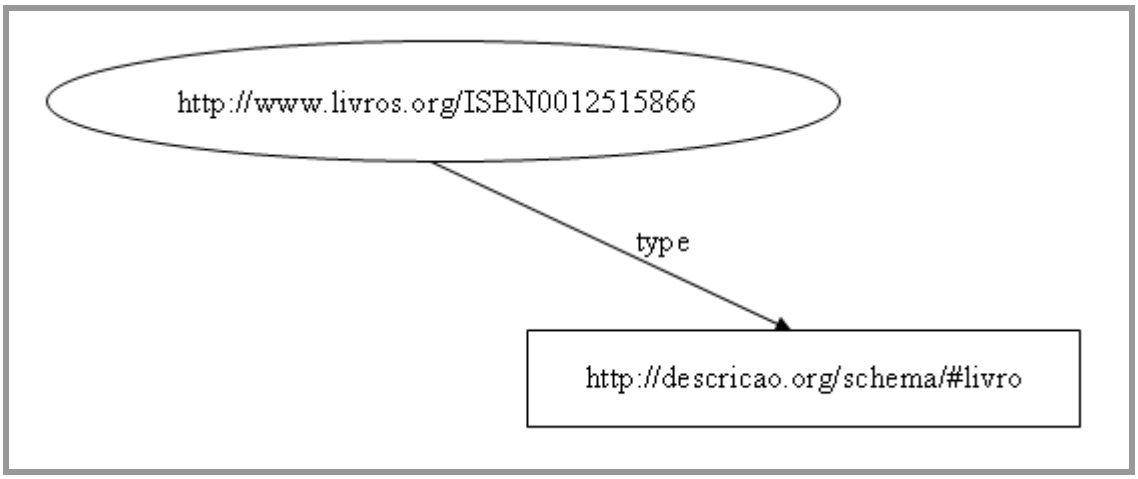

Figura 9: Representação do modelo RDF através de um grafo

O RDF proporciona apenas um modelo para a representação de metadados. A tripla é uma possível representação, assim como a representação através de um grafo, sendo possíveis outras representações sintáticas, dentre elas, a XML. Como a linguagem XML tem se tornado um padrão de interoperabilidade sintática, a especificação do modelo de dados inclui uma codificação XML para o RDF.

A Figura 10 apresenta a descrição RDF apresentada anteriormente, codificada na linguagem XML.

```
<rdf:Description about="http://www.livros.org/ISBN0012515866">
        <rdf:type rdf:resource="http://descricao.org/schema/#livro">
</rdf:Description>
```

Figura 10: Representação do modelo RDF em XML

O RDF Schema permite que projetistas definam um vocabulário particular para dados RDF (Brickley & Guha, 2000, Brickley & Guha, 2002). Em outras palavras, o mecanismo RDF Schema proporciona um sistema básico de tipos para modelos RDF. Esse sistema de tipos utiliza alguns termos pré-definidos, como *Class*, *subPropertyOf* e *subClassOf*, para esquemas específicos de aplicações (Decker *et al*, 2000).

As primitivas básicas de modelagem no RDF Schema são:

- Declarações do tipo *Class* e *subClassOf*, que juntos permitem a definição de hierarquias de classes;
- Declarações do tipo *Property* e *subPropertyOf*, usados para definir hierarquias de propriedades;
- Declarações do tipo *domain* e *range*, para restringir as combinações possíveis entre propriedades e classes;
- Declarações do tipo *type*, para declarar um recurso como uma instância de uma determinada classe.

Com as primitivas citadas anteriormente, construímos um exemplo de esquema RDFS para um determinado domínio. Veja o esquema da Figura 11.

```
<rdf:Description ID="Pessoa">
        <rdf:type resource="http://www.w3.org/TR/PR-rdf-schema#Class"/>
        <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/PR-rdf-schema#Resource"/>
</rdf:Description>
<rdf:Description ID="Homem">
        <rdf:type resource="http://www.w3.org/TR/PR-rdf-schema#Class"/>
        <rdfs:subClassOf rdf:resource="#Pessoa"/>
</rdf:Description>
<rdf:Description ID="Mulher">
        <rdf:type resource="http://www.w3.org/TR/PR-rdf-schema#Class"/>
        <rdfs:subClassOf rdf:resource="#Pessoa"/>
</rdf:Description>
<rdf:Description ID="nome">
        <rdf:type resource="http://www.w3.org/TR/PR-rdf-schema#Property"/>
        <rdfs:domain rdf:resource="#Pessoa"/>
        <rdfs:range rdf:resource="http://www.w3.org/TR/PR-rdf-schema#String"/>
</rdf:Description>
<rdf:Description ID="casadoCom">
        <rdf:type resource="http://www.w3.org/TR/PR-rdf-schema#Property"/>
        <rdfs:domain rdf:resource="#Pessoa"/>
        <rdfs:range rdf:resource="#Pessoa"/>
</rdf:Description>
```

Figura 11: Exemplo de um documento RDFS

A classe mais geral no RDF Schema é o *rdfs:Resource*. Ela possui duas subclasses, chamadas *rdfs:Class* e *rdf:Property*. Quando especificamos um esquema específico para um

domínio, como no exemplo acima, as classes e propriedades definidas no esquema serão instâncias desses dois recursos. O recurso *rdfs:Class* denota o conjunto de todas as classes em um sentido similar à orientação a objetos. Isso significa que *Pessoa* é uma instância da classe *rdfs:Class*. O mesmo vale para propriedades, por exemplo, cada propriedade definida em um esquema é uma instância de *rdf:Property* (Staab *et al*, 2000).

O RDF Schema define uma propriedade especial *rdfs:subClassOf* que define o relacionamento de subclasse entre classes. Uma vez que a propriedade *rdfs:subClassOf* é transitiva, definições são herdadas das classes mais genéricas para as classes mais específicas, e recursos que são instâncias de uma classe são automaticamente instâncias de todas as superclasses dessa classe. No RDF Schema, é proibido que qualquer classe seja uma *rdfs:subClassOf* dela mesma ou de uma de suas subclasses.

De maneira similar à propriedade *rdfs:subClassOf*, a qual define hierarquia de classes, um outro tipo especial de relação, denominada *rdfs:subPropertyOf*, define uma hierarquia de propriedades. Por exemplo, podemos definir que a propriedade *paiDe* é uma *rdfs:subPropertyOf* da propriedade *parenteDe*.

O RDF Schema também permite restringir o domínio e o alcance (*domain* e *range*) de propriedades. Por exemplo, podemos usar essas restrições para definir que pessoas e apenas pessoas podem possuir a propriedade *casadoCom*, e o valor dessa propriedade será sempre outra pessoa (Staab *et al*, 2000).

2.3.3 Os papéis de XML e RDF(S) no contexto da Web Semântica

XML e RDF são formalismos diferentes com suas próprias funções, e seus papéis na realização da Web Semântica são diferentes. A função da XML é proporcionar uma sintaxe de fácil utilização para dados na web. Com XML, podemos codificar todos os tipos de dados que serão trocados entre computadores, usando XML Schemas ou DTDs para prescrever a estrutura dos dados. Isso torna a XML uma linguagem fundamental para a Web Semântica, no sentido que muitas tecnologias provavelmente a utilizarão como sintaxe base. No entanto, como a XML não proporciona, de antemão, qualquer interpretação dos dados, ela não contribui muito para o aspecto semântico da Web Semântica.

Enquanto que um documento XML é fundamentalmente hierárquico, consistindo de elementos aninhados, o RDF estabelece uma rede de objetos conectados, sem que existam elementos hierarquicamente superiores ou inferiores. O papel principal do RDF é

proporcionar um modelo básico de objeto-atributo-valor aos metadados. Dessa forma, o RDF pode ser considerado como uma infra-estrutura que possibilita a codificação, troca e reuso de metadados estruturados.

O RDF Schema permite a definição de um vocabulário (ou esquema) particular usado pelos atributos RDF. Em outras palavras, o mecanismo de RDF Schema proporciona um sistema de tipos básicos para uso em modelos RDF. Tal sistema de tipos utiliza uma terminologia pré-definida, como por exemplo os termos *Class* e *subClassOf*.

A necessidade de funcionalidades e características que RDF e RDF Schema não possuem, como, por exemplo, tipos de dados, expressões consistentes para enumeração, mecanismos que permitam realizar inferências, etc., tornam esses mecanismos insuficientes para uma linguagem de definição de ontologias. Na verdade, a peça ausente é a associação efetiva entre dados e seu significado. Esta associação é que permitirá o compartilhamento e a transferência de informação através da diversidade de sistemas que compõem a web.

2.3.4 Ontologia

Assim como DTDs ou XML Schemas, uma ontologia define um vocabulário comum para o compartilhamento de informações em um domínio, através de definições interpretáveis por computador. No entanto, uma ontologia contém estruturas que definem a semântica do vocabulário de termos. Esta semântica pode ser utilizada para inferir informações baseadas no contexto de um domínio de conhecimento específico e integrar bases de dados diferentes. Por exemplo, duas bases de dados podem armazenar os mesmos conceitos utilizando terminologias diferentes. Para que a informação existente possa ser processada e relacionada, é necessária uma definição da relação entre os conceitos contidos nos diferentes esquemas. Ontologias proporcionam a estrutura necessária para a definição desses conceitos e os relacionamentos entre eles (Noy & McGuinness, 2001).

Para a web semântica, uma ontologia é um documento que define formalmente a relação entre conceitos. Uma ontologia possui uma taxonomia, usada na definição de classes e subclasses de objetos, além das relações entre eles; e um conjunto de regras de inferência, através das quais é possível que um agente de software extraia conclusões úteis.

Em (Gruber, 1993), uma ontologia é definida como "*uma especificação de uma conceitualização*". A conceitualização é a organização do conhecimento sobre o mundo em

forma de entidades, enquanto que, a especificação é a representação dessa conceitualização de uma forma concreta.

A grande vantagem do uso de ontologias é a capacidade de estabelecer uma hierarquia de classes de objetos e seus relacionamentos. Computadores não são capazes de realmente entender uma informação, no entanto, são muito eficientes na manipulação de termos. A adição de regras de inferência torna as ontologias ainda mais poderosas para serem usadas por computadores na manipulação de termos.

No caso de bancos de dados heterogêneos, o uso de ontologias pode tornar o conhecimento distribuído processável por computador, através da descrição de entidades e relações entre elas, além de regras de integridade para o domínio. Neste contexto, uma ontologia pode ser usada para a definição de um esquema global, para servir como base fundamental no processo de integração dos dados.

Em (Guarino, 1998), encontramos quatro classificações de ontologia:

- Ontologias de alto nível: descrevem conceitos bastante genéricos, independentes de um problema ou domínio particular;
- Ontologias de domínio: definem um vocabulário relacionado a um domínio genérico;
- Ontologias de tarefa: descrevem uma tarefa genérica ou atividade;
- Ontologias de aplicação: descrevem conceitos que dependem tanto de um domínio específico como de uma tarefa específica.

Além das classificações citadas acima, (Guarino, 1998) ainda faz a distinção entre ontologias refinadas e não-refinadas. Uma ontologia refinada tem um grande número de axiomas[2], e pede uma linguagem de alta expressividade. Uma ontologia não-refinada tem um número mínimo de axiomas.

2.3.5 Linguagens para especificação de ontologias

Nos últimos anos, foram desenvolvidas várias linguagens para especificação de ontologias. Com o desenvolvimento de tecnologias como XML, RDF e RDF Schema, surgiram linguagens apropriadas para especificação de ontologias na web. Nesta seção, introduzimos algumas destas linguagens. Realizar uma investigação detalhada das linguagens de especificação de ontologias existentes não é o foco do nosso trabalho, portanto, deixamos

[2] Um axioma é uma sentença em lógica de primeira ordem.

várias linguagens de fora da lista a seguir, apresentando apenas as linguagens que analisamos mais profundamente na nossa pesquisa.

- KIF (Knowledge Interchange Format). O KIF foi proposto como uma linguagem normalizada para a descrição de entidades em bases de dados, agentes inteligentes, etc. Esta linguagem também foi definida para servir nos processos de tradução entre linguagens. A descrição da linguagem inclui a especificação da sintaxe e da semântica. Além da descrição de entidades, fatos e conhecimento, o KIF permite ainda a descrição de procedimentos, isto é, a descrição de seqüências de ações para que os agentes as executem. Excluindo as especificidades próprias da linguagem, o KIF baseia-se fundamentalmente na lógica de primeira ordem (Connolly, 2000).
- OIL (Ontology Inference Layer). Foi criada a partir do projeto *OntoKnowledge* de um programa da União Européia denominado *Information Society Technologies*. Trata-se de uma proposta de uma camada de representação e inferência para ontologias. Permite a modelagem de primitivas geralmente usadas nas abordagens baseadas em *frames* (conceitos, relações, etc.), como também, a inclusão de uma semântica formal e de raciocínio encontrados nas abordagens de descrição lógica (description logic) (Fensel *et al*, 2000).
- DAML+OIL (DARPA Agent Markup Language + Ontology Inference Layer). Foi criada a partir da junção e extensão das linguagens DAML e OIL. Trata-se de uma linguagem de marcação semântica de recursos na web, construída a partir de linguagens W3C (World Wide Web Consortium) como RDF e RDFS (RDF Schema), estendendo estas linguagens com primitivas de modelagem mais ricas, como descrito na Tabela 2. Segundo (Gómez-Pérez & Corcho, 2002), a linguagem DAML+OIL suporta todas as características inerentes a taxonomia (subclasse de, decomposições exaustivas, decomposições de disjunção e não subclasse de), todas as características de conceitos (partições, documentação, atributos de instância, atributos de classe, escopo local e global, restrições de tipos e restrições de cardinalidade) exceto valores *default*, uma característica de axioma (lógica de primeira ordem) e todas as características de instância (instâncias de conceito, fatos e *claims*) (Horrocks *et al*, 2001a, Horrocks *et al*, 2001b, Horrocks *et al*, 2001c).
- OWL Web Ontology Language. Trata-se de uma linguagem de marcação semântica sendo desenvolvida pelo W3C, a partir de uma revisão da linguagem DAML+OIL.

Dentre as mudanças já realizadas, citamos as seguintes: inclusão de cinco novas primitivas para controle de versão; várias propriedades e classes foram renomeadas; a linguagem OWL permite subclasses cíclicas. Em geral, a linguagem OWL está sendo desenvolvida para tornar-se a linguagem padrão W3C para definição de ontologias, tomando como base uma linguagem bem amadurecida, a linguagem DAML+OIL (Hendler *et al*, 2002).

Na Tabela 2 a seguir apresentamos uma série de características fundamentais que uma linguagem de especificação de ontologias deve possuir. Para cada característica, assinalamos se as diferentes linguagens apresentadas anteriormente oferecem ou não suporte. Uma vez que as linguagens DAML+OIL e OWL estendem o RDF(S), incluímos também esta linguagem. Dessa forma, fica mais claro as distinções existentes entre as linguagens.

	KIF	**RDF(S)**	**OIL**	**DAML+OIL**	**OWL**
Elementos principais					
Classes	Sim	Sim	Sim	Sim	Sim
Relações	Sim	Sim	Sim	Sim	Sim
Funções	Sim	Não	Sim	Sim	Sim
Instâncias	Sim	Sim	Não	Sim	Sim
Axiomas	Sim	Não	Sim	Sim	Sim
Possibilidades de axiomas					
Negação	Sim	Não	Sim	Sim	Sim
Conjunção	Sim	Não	Sim	Sim	Sim
Disjunção	Sim	Não	Sim	Sim	Sim
Taxonomia					
Subclasse de	Sim	Sim	Sim	Sim	Sim
Herança múltipla	Sim	Sim	Não	Sim	Sim
Propriedades					
Propriedades multivaloradas	Sim	Sim	Não	Sim	Sim
Hierarquia de propriedades	Sim	Sim	Não	Sim	Sim
Restrições de propriedades					
Restrições de tipo	Sim	Sim	Sim	Sim	Sim
Cardinalidade	Sim	Não	Sim	Sim	Sim
Outras características					
Meta-informações sobre a ontologia	Sim	Não	Sim	Sim	Sim
Referência a outra ontologia	Não	Não	Sim	Sim	Sim
Tipos de dados primitivos	Sim	Não	Não	Não	Sim

Tabela 2: Tabela de comparação entre as linguagens

Como já citamos nesta seção, a linguagem OWL está em fase de desenvolvimento, portanto, decidimos não utilizá-la no nosso trabalho. Excluindo a linguagem OWL, podemos perceber na Tabela 2 que as linguagens KIF e DAML+OIL são bem mais completas que as

demais. Para o desenvolvimento do estudo de caso, descrito no capítulo 4, decidimos utilizar a linguagem DAML+OIL para a definição do esquema global de uma federação de bancos de dados. A existência de um pacote denominado JXML, descrito na seção 4.1, que gera uma representação gráfica em árvore a partir de uma ontologia descrita em DAML+OIL, foi determinante para esta escolha. Apresentamos uma descrição detalhada da linguagem DAML+OIL no Apêndice A desta dissertação.

2.4 Agentes de software

Para que o poder da web semântica seja perceptível, é necessário que sejam criados agentes de software que utilizem a infra-estrutura proporcionada pela aplicação desta visão, de forma a facilitar as atividades humanas. Com base em ontologias, agentes de software serão capazes de compreender o significado e a relação entre objetos, além de raciocinar sobre eles utilizando regras de inferência.

Devido à existência de inúmeras definições de agentes de software, torna-se necessário definir claramente o significado destes no contexto do nosso trabalho. Assim, definimos um agente de software como uma entidade de software capaz de navegar em uma rede, agindo de forma autônoma no lugar de um usuário.

Como acontece com qualquer campo de estudo que provoca algum interesse comercial, tem havido uma grande mudança de foco por parte da comunidade de Inteligência Artificial, no sentido de aplicar as técnicas básicas de inteligência artificial em sistemas distribuídos, intranets, Internet, e na web. Com a explosão do número de transações comerciais realizadas na web, tem crescido o interesse pela existência de agentes capazes de realizar ações em benefício do usuário.

A seguir, apresentamos algumas das características mais relevantes que um agente pode apresentar. A escolha de quais características devem estar presentes em um agente depende da funcionalidade que o projetista pretende dar ao seu agente. Os principais atributos que um agente pode apresentar são:

- Mobilidade: é a capacidade de um agente transportar-se de uma máquina para outra. Dois tipos de mobilidade podem ser identificados:
    - Mobilidade com script: os agentes são montados em uma máquina e enviados a outra, para execução em um ambiente apropriadamente seguro. Para esse tipo

de mobilidade, os dados atuais[3] não necessitam ser acoplados ao agente, visto que o agente viaja antes de iniciar sua execução;

- Mobilidade com estado: os agentes são transportados de máquina em máquina durante a execução. Para tanto, é necessário conduzir os valores atuais dos seus dados;

- Aprendizagem: é a habilidade apresentada pelo agente de acumular conhecimento baseado em experiências anteriores, e, com isso, modificar seu comportamento em resposta a novas situações;
- Pró-atividade ou iniciativa: representa um comportamento independente. Os agentes não somente reagem ao seu ambiente, mas também devem exibir um comportamento orientado à satisfação de seus objetivos. O agente que implementa este atributo possui maior flexibilidade, pois, é capaz de resolver problemas causados por situações inesperadas;
- Cooperação: pode ser entendida como a capacidade que os agentes possuem de trabalharem em conjunto de forma a concluírem tarefas de interesse comum. Para permitir esta cooperação, o agente deve ser dotado de uma certa habilidade social, capacitando-o a interagir com outros agentes e possivelmente humanos, através de alguma linguagem de comunicação;
- Persistência: é a capacidade apresentada por um agente de manter um estado interno conciso através do tempo, sem alterá-lo ao acaso;
- Confiabilidade: os agentes devem demonstrar benevolência e veracidade, ou seja, os usuários precisam ter certeza de que os agentes serão fidedignos nas suas ações;
- Abstração: é a habilidade do agente de detectar a relevância da informação ou ação para uma situação específica.

Os conceitos de agentes e objetos costumam ser confundidos devido a algumas similaridades existentes entre eles. Um objeto encapsula algum estado e possui controle sobre ele, de forma que o acesso a este estado só é conseguido através de métodos do próprio objeto. Agentes encapsulam estado de uma forma semelhante. Por outro lado, agentes incorporam uma noção mais forte de autonomia que os objetos, uma vez que, agentes também encapsulam comportamento, o que faz com que eles tenham controle sobre a execução dos seus métodos.

[3] Os valores atuais das variáveis do agente.

2.5 Agentes móveis

Atualmente, o principal paradigma utilizado em sistemas de objetos distribuídos baseia-se na passagem de mensagens entre objetos estacionários. Este paradigma, denominado *chamada de procedimentos remotos* (RPC), permite que um computador "chame" procedimentos contidos em outro computador na rede. Paradigmas de comunicação como o RPC, envolvem uma contínua troca de mensagens entre um cliente e um servidor distantes geograficamente, contribuindo para o aumento do congestionamento das redes de computadores.

Com o paradigma de agentes móveis, quando um usuário deseja que um determinado servidor realize tarefas em seu favor, ao invés de enviar uma seqüência de comandos na rede, ele envia um agente para intermediar a realização das tarefas localmente, junto ao servidor. Com isso, o número de mensagens enviadas na rede diminui consideravelmente. Devido a esta e outras vantagens, o paradigma de agentes móveis surge como uma nova alternativa para o desenvolvimento de sistemas distribuídos.

2.5.1 Benefícios

Segundo (Lange & Oshima, 1998), a implantação de agentes móveis no desenvolvimento de sistemas distribuídos apresenta uma série de benefícios como:

- Reduzem o tráfego da rede: Sistemas distribuídos geralmente envolvem protocolos de comunicação que demandam um grande volume de comunicação para realizar uma tarefa, principalmente quando a infra-estrutura de rede possui restrições de segurança. Com o uso de agentes móveis, toda a comunicação é empacotada e enviada para o computador remoto, onde a comunicação ocorre localmente. Agentes móveis também reduzem o tráfego de dados na rede, pois, permitem mover o processamento para o local onde os dados estão armazenados, ao invés de transferir os dados para depois processá-los. O princípio é simples: mover o processamento para onde estão os dados ao invés de mover os dados para o local de processamento;
- Executam de forma assíncrona e autônoma: Tarefas podem ser embutidas em agentes móveis, os quais são despachados na rede. Após serem despachados, esses agentes são independentes de quem os criou, podendo operar de forma assíncrona e autônoma.

Este benefício é bastante importante no caso de dispositivos móveis, como um laptop, onde não há conexão contínua com a rede;

- Adaptam dinamicamente: Assim como agentes fixos, agentes móveis possuem a habilidade de perceber mudanças no ambiente de execução e reagir autonomamente;
- Independentes de plataforma: Redes de computadores geralmente são heterogêneas, tanto na perspectiva de hardware como na de software. Como agentes móveis são independentes de computador e também de rede, sendo dependentes apenas do seu ambiente de execução, eles não dificultam a integração de sistemas. Agentes fixos também são independentes de plataforma, no entanto, esta característica é ainda mais importante para o caso de agentes móveis, uma vez que, estes migram entre diversos locais, os quais podem ser heterogêneos.

2.5.2 Desafios

O paradigma de agentes móveis tem sido alvo de pesquisas na área de Inteligência Artificial há vários anos. No entanto, o desenvolvimento de aplicações reais, baseadas em agentes móveis, é um fato recente. Dessa forma, devido à falta de experiência e maturidade da comunidade de agentes, existem alguns desafios que precisam ser levados em consideração, antes de optarmos pelo desenvolvimento de aplicações baseadas em agentes móveis. A seguir, citamos alguns desses desafios:

- Ausência de metodologias maduras: Para a engenharia de software, a concepção e desenvolvimento de aplicações baseadas em agentes móveis é relativamente recente. Dessa forma, não existem soluções maduras de metodologias de desenvolvimento de software para este tipo de aplicações. Além disso, métodos e técnicas da orientação a objetos não são suficientes para cobrir todos os aspectos de um sistema baseado em agentes móveis (Guedes & Machado, 2001);
- Segurança: As tecnologias de segurança ainda não são maduras o suficiente para oferecer um tratamento adequado às aplicações baseadas em agentes móveis. Trata-se de uma área bastante pesquisada, uma vez que, muitos problemas continuam sem solução (Douglis *et al*, 1999).

2.5.3 Plataformas

Para desenvolvermos aplicações baseadas em agentes móveis, é necessária a preparação de um ambiente de execução, de forma que cada local suporte a execução, migração, localização e comunicação de agentes. Este ambiente de execução é o que denominamos de plataforma de agentes móveis. A seguir, descrevemos brevemente algumas plataformas disponíveis atualmente.

- Grasshopper: Desenvolvida pela IKV++, trata-se de uma plataforma de agentes móveis que possibilita a integração do tradicional paradigma cliente/servidor com a tecnologia de agentes móveis. Foi a primeira plataforma a implementar o padrão MASIF (Mobile Agent System Interoperability Facility), criado pela OMG (Object Management Group), no intuito de promover interoperabilidade entre plataformas de agentes móveis de diferentes fabricantes (IKV++ GmbH, 1999);
- Aglets: Desenvolvido pelo laboratório de pesquisas da IBM no Japão, trata-se de um dos sistemas de agentes móveis mais conhecidos. Foi um dos primeiros sistemas de agentes móveis baseados em Java. O modelo de agente do Aglets é baseado no *applet*, que originou o termo Aglet, que é uma fusão das palavras *agent* e *applet*. Possui uma interface estável e uma grande base de usuários. Apesar de sua grande popularidade, o Aglets nunca foi comercializado, estando disponível gratuitamente (Lange & Oshima, 1999);
- JADE (Java Agent DEvelopment Framework): Esta plataforma pode ser vista como um middleware de agentes que implementa uma plataforma de agentes e um framework de desenvolvimento. Lida com aspectos que não fazem parte do agente em si, tais como: transporte de mensagens, codificação e ciclo de vida do agente. Inclui todos os agentes obrigatórios propostos pelo padrão FIPA[4]. Além disso, está integrada com o JESS (Java Expert System Shell), que é um motor de inferência que suporta o desenvolvimento, em Java, de sistemas baseados em regras (Bellifemine & Trucco, 2003, Friedman-Hill, 2003);
- Voyager: Desenvolvido pela ObjectSpace, consiste em uma plataforma com suporte a diferentes mecanismos de comunicação, como *chamada de procedimentos remotos* (RPC), ORB (Object Request Broker) e DCOM (Distributed Component Object

[4] A FIPA (Foundation for Intelligent Physical Agents) especificou um modelo de integração de agentes e aplicações não baseadas em agentes.

Model). O suporte a estes diferentes mecanismos de comunicação, somado à capacidade dos objetos moverem-se através da rede como agentes, fizeram do Voyager um sistema bastante usado (Lange & Oshima, 1999).

Tomando como base a avaliação descrita em (Altmann *et al*, 2001), apresentamos na Tabela 3 uma comparação entre as plataformas citadas anteriormente. Para cada quesito de comparação, atribuímos uma nota de 1 a 4. A seguir apresentamos uma breve descrição do que cada quesito representa:

- Criptografia de dados: Transporte de dados criptografados e identificação de qualquer mudança ocorrida nos dados durante uma migração;
- Certificados: Suporte a certificados de segurança;
- Autorização: Autorização de um agente, tomando como base o seu nível de acesso;
- Mecanismos de acesso: Mecanismo que permite o acesso aos agentes e aos recursos de um sistema de agentes;
- Interface gráfica de administração: Ferramenta gráfica para administração;
- Monitoração: Ferramenta para observar a execução e migração de agentes.

	Grasshopper	**Aglets**	**JADE**	**Voyager**
Criptografia de Dados	3	2	1	4
Certificados	3	1	1	1
Autorização	3	4	1	1
Mecanismos de acesso	3	4	1	1
Interface gráfica de administração	4	4	3	1
Monitoração	4	2	2	1

Tabela 3: Tabela de comparação entre as plataformas

Dentre as plataformas apresentadas nesta seção, escolhemos a plataforma Grasshopper (IKV++ GmbH, 1999) para o desenvolvimento do estudo de caso descrito no capítulo 4. Em média, esta plataforma obteve o melhor êxito na comparação descrita na Tabela 3, como também, na avaliação descrita em (Altmann *et al*, 2001). Além disso, as características citadas a seguir também influenciaram nossa escolha:

- Implementa o padrão MASIF (Mobile Agent System Interoperability Facility);
- Plataforma 100% Java;
- Possui excelente documentação;
- Uso gratuito e irrestrito.

2.5.4 Agentes em Java

O código Java é simples, seguro, compacto, e por ser orientado a objetos permite reuso e formas coerentes de explorar recursos como interfaces, encapsulamento e polimorfismo.

Lange e Oshima (Lange & Oshima, 1998) identificam algumas características da linguagem Java, essenciais para o desenvolvimento de aplicações baseadas em agentes móveis:

- Independência de plataforma: O código Java é compilado em um formato independente de arquitetura, denominado *byte code*, que permite a execução de aplicações Java em redes heterogêneas, sendo portanto, independente de plataforma. Isto permite a geração de agentes móveis capazes de executar em qualquer tipo de plataforma e rede;
- Carga dinâmica de classes: A máquina virtual Java carrega e define classes em tempo de execução, fornecendo um espaço de endereçamento privado para cada agente, podendo executar de forma segura e independente de outros agentes;
- Programação multithread: O modelo de programação multithread permite a implementação de agentes como entidades autônomas. As primitivas de sincronização entre threads permitem a interação entre agentes;
- Serialização de objetos: A linguagem Java permite a serialização e deserialização de objetos. Este mecanismo permite que objetos sejam empacotados com informação suficiente de forma a permitir posterior reconstrução. Esta é uma característica chave para a implementação de agentes móveis.

2.5.5 Grasshopper

Como citamos na seção 2.5.3, decidimos utilizar a plataforma Grasshopper para dar suporte aos agentes do sistema. A seguir, definimos alguns conceitos básicos de agentes móveis, segundo a plataforma de agentes escolhida.

Criação de um agente

Todo agente é criado dentro de um lugar mantido por uma agência, a qual cria um registro do agente em seu banco de dados interno. Ao ser criado, o novo agente recebe um

identificador e um conjunto de informações sobre si mesmo. Estas informações podem ser acessadas por ele mesmo, e algumas delas podem ser acessadas por outros agentes.

Execução remota

A execução remota é a forma mais tradicional de mobilidade de código. Significa que um programa é enviado a uma localidade remota *antes* de ser executado. O programa permanece nesta localidade durante todo o seu período de execução.

Migração

Um agente móvel é capaz de migrar, ou seja, mudar de localidade *durante* a sua execução, realizando partes de sua tarefa em diferentes locais de uma rede. Existem dois tipos de migração:

- Migração forte: O agente migra juntamente com o seu *estado de execução*. O estado de execução de um agente consiste em uma pilha de execução, que identifica o ponto de execução alcançado pelo agente. Após uma migração forte, o agente continua a sua execução a partir do ponto em que foi interrompido antes de realizar a migração;
- Migração fraca: O agente migra juntamente com o seu *estado de dados*. O estado de dados de um agente consiste nos valores de suas variáveis, as quais são serializadas antes da migração. Os valores das variáveis são capturados, transferidos pela rede, e providenciados para o agente na sua nova localização.

A linguagem Java, devido a restrições da JVM (Java Virtual Machine), não possibilita a captura do estado de execução de um processo ou de uma thread. Dessa forma, a plataforma Grasshopper, que é baseada nesta linguagem, suporta apenas a migração fraca.

Agentes estacionários

Agentes estacionários são agentes que não possuem a habilidade de migrar entre diferentes localidades de uma rede, estando sempre associados a um único local da rede.

Agência

Uma agência é o ambiente de execução de agentes móveis e estacionários. Para que um determinado computador seja capaz de receber e executar agentes, deverá existir pelo menos uma agência executando localmente.

Lugar

Um lugar oferece um conjunto de serviços aos agentes que nele residem, representando um agrupamento lógico das funcionalidades de uma agência. Uma agência pode conter muitos lugares, cada um devendo ser nomeado de acordo com o seu propósito. Toda agência Grasshopper possui um lugar padrão, denominado *InformationDesk.*

Região

O conceito de região facilita a gerência de agências distribuídas, pertencentes a um mesmo domínio. Por exemplo, uma região pode conter todas as agências pertencentes a uma empresa ou organização específica, auxiliando o gerenciamento destas.

Comunicação entre agentes

Na plataforma Grasshopper, a comunicação entre agentes é realizada de forma transparente à localização, através de um padrão de comunicação denominado *Proxy*. Com este padrão, a comunicação entre um agente-cliente e um agente-servidor é intermediada por uma entidade. Esta entidade é responsável por localizar e estabelecer uma conexão com o servidor. A vantagem do uso deste padrão é o fato do agente-cliente não precisar se preocupar com a localização do agente-servidor.

2.5.6 Padrões de agentes móveis

Padrões de projeto têm provado sua utilidade no desenvolvimento orientado a objetos, auxiliando projetistas e programadores a alcançarem uma melhor qualidade nas suas aplicações (Lange & Oshima, 1998). Com o desenvolvimento de aplicações baseadas em

agentes móveis, foram desenvolvidos padrões de projeto com a finalidade de promover o reuso de soluções validadas em desenvolvimentos anteriores.

Existem vários padrões de projeto para agentes móveis. Estes padrões são geralmente divididos em três classes: padrões de mobilidade, padrões de tarefa e padrões de interação (Lange & Oshima, 1998). A seguir, descrevemos estas três classes de padrões.

Padrões de mobilidade

A essência de um agente móvel é a habilidade de mover-se entre locais diferentes de uma rede. Os padrões de mobilidade permitem encapsular o controle da mobilidade, de forma a permitir o reuso e a simplificação do projeto de agentes (Lange & Oshima, 1998).

O padrão *Itinerário* é um exemplo de um padrão de mobilidade. Em resumo, este padrão trata do roteamento de agentes entre múltiplos destinos. Um itinerário contém uma lista de destinos, define um esquema de roteamento, trata casos especiais, i.e., um destino não existe ou não está acessível, e sempre conhece o próximo destino a ser visitado.

A idéia central do padrão itinerário é delegar a responsabilidade de migração do agente a um objeto associado, denominado Itinerario. A classe Itinerario proporciona uma interface para manter ou modificar o itinerário de um agente e despachá-lo para novos destinos. Um agente instancia a classe Itinerario, passando-lhe uma lista de destinos a serem visitados seqüencialmente. Antes de cada migração, este agente recebe do objeto Itinerario o endereço do próximo destino. Para isso, é necessário que o objeto Itinerario seja transferido juntamente com o agente.

Segundo (Lange & Oshima, 1998), o padrão Itinerário deve ser usado para alcançar os seguintes objetivos:

- Dividir o plano de migração de um agente do seu comportamento, de forma a promover a modularidade das duas partes;
- Proporcionar uma interface uniforme para os agentes móveis;
- Definir planos de migração que podem ser reusados e compartilhados entre agentes.

Padrões de tarefa

Padrões de tarefa tratam da divisão de tarefas e como estas são delegadas a um ou mais agentes. Em geral, uma tarefa pode ser dividida entre diferentes agentes trabalhando em paralelo, que cooperam entre si de forma a concluí-la.

Um exemplo de padrão de tarefa é o padrão *Mestre-escravo*. Este padrão permite que um agente-mestre delegue uma tarefa a um agente-escravo. Em geral, o agente-escravo migra para um destino, realiza sua tarefa, e retorna o resultado ao seu agente-mestre.

O padrão Mestre-escravo utiliza uma classe abstrata para definir as partes invariantes na delegação de tarefas entre agentes mestre e escravo. São elas: despachar um agente-escravo para um local remoto, iniciar a execução da tarefa, e tratar exceções ocorridas na execução da tarefa.

Segundo (Lange & Oshima, 1998), devemos aplicar o padrão Mestre-escravo nos seguintes casos:

- Quando um agente precisa realizar uma tarefa em paralelo com outras tarefas de sua responsabilidade;
- Quando um agente estacionário deseja realizar uma tarefa em um destino remoto.

Padrões de interação

Padrões de interação tratam da comunicação entre agentes, promovendo uma melhor cooperação entre eles.

Um exemplo de padrão de interação é o padrão *Reunião*. Este padrão proporciona um mecanismo para que dois ou mais agentes iniciem uma interação local, abstraindo a sincronização de tempo e lugar necessária para a realização da interação. Por exemplo, agentes podem migrar para um determinado destino, chamado *lugar de encontro*, onde são notificados da chegada de seus parceiros, podendo daí então, iniciar as interações.

Capítulo 3

DIA: Um sistema de integração de bancos de dados federados na web usando agentes móveis

O problema clássico da integração de dados tem sido tratado há muito tempo, e ainda continua sendo um campo de estudo bastante pesquisado, devido a sua complexidade, como também, ao surgimento de novos paradigmas provindos da web. Dentre estes novos paradigmas, surgiu uma nova visão denominada Web Semântica, que, em geral, procura resolver o problema da heterogeneidade semântica na web através do uso de ontologias.

Como citamos na seção 2.4, o poder da Web Semântica só será perceptível uma vez que forem criados agentes de software que utilizem a infra-estrutura proporcionada pela aplicação desta visão. Dessa forma, agentes de software exercem um papel importante na visão da Web Semântica. Como vimos no capítulo anterior, uma arquitetura de agentes, particularmente agentes móveis, apresenta diversas vantagens em relação a outras arquiteturas, como a cliente-servidor, para o desenvolvimento de sistemas distribuídos. Considerando estas vantagens e o fato do uso de agentes de software fazer parte da proposta da Web Semântica (Berners-Lee *et al*, 2001), apresentamos uma proposta de solução, denominada DIA[5], para a integração de bancos de dados federados na web, utilizando agentes móveis e ontologias.

Na seção 3.1 apresentamos os requisitos levantados para o desenvolvimento do sistema. Na seção 3.2 apresentamos o modelo de use-case do sistema. Na seção 3.3 apresentamos a arquitetura do sistema. Nas seções 3.4 e 3.5 detalhamos alguns aspectos importantes para a migração do agente móvel. Na seção 3.6 apresentamos como ocorre o processo de integração dos resultados obtidos dos bancos de dados locais. Na seção 3.7 apresentamos uma solução para o mapeamento das consultas do esquema global para os esquemas locais.

[5] DIA: Data Integration using Agents

3.1 Requisitos do sistema

Nesta seção, detalhamos os requisitos funcionais, não-funcionais e de interface do sistema de integração de dados.

3.1.1 Requisitos funcionais

- ✓ **Deverá existir uma ontologia que descreve todos os domínios do sistema.**

O objetivo da nossa arquitetura é proporcionar uma interface uniforme para SGBDs heterogêneos e distribuídos. Para isso, os bancos de dados locais têm que estar federados por um esquema global, definido como uma ontologia, que proporcione uma visão integrada dos esquemas destes SGBDs e permita o acesso aos dados de cada membro da federação.

- ✓ **Os termos da consulta construída deverão estar baseados na ontologia.**

O usuário realizará suas consultas através de uma interface web. Em seguida, o sistema deverá construir a consulta do usuário, em linguagem SQL, obedecendo à sintaxe dos termos definidos na ontologia. Isto visa facilitar o processo de adaptação da consulta aos diferentes bancos de dados locais.

- ✓ **O sistema não deverá sobrecarregar a carga de comunicação e transporte de dados.**

O sistema deverá evitar a excessiva troca de mensagens e dados, de forma a não sobrecarregar a rede de comunicação. O conhecimento necessário para o processamento de consultas deverá ser levado para perto de onde os dados estão armazenados. Para tanto, a consulta do usuário deverá ser entregue a um agente móvel, para que este visite os bancos de dados distribuídos, e realize o processo de integração diretamente nas fontes.

- ✓ **O sistema não deverá consultar bancos de dados desnecessariamente.**

O sistema segue a abordagem virtual de integração de dados. Quando comparada com a abordagem materializada, a abordagem virtual tem o desempenho como principal desvantagem. Com base nisso, visando um bom desempenho, o sistema só deverá consultar bancos de dados onde há dados que interessam à consulta. Para isso, deverá ser criado um roteiro específico para cada consulta, incluindo somente os locais que podem conter informação necessária para processar a consulta.

- ✓ **Em cada SGBD local, os elementos do esquema local deverão estar relacionados aos elementos da ontologia (esquema global).**

Para atender a este requisito, o sistema deverá ser uma federação de bancos de dados em que, para cada novo integrante, deverá ser criado este relacionamento. Para realizar a adaptação local da consulta do usuário, o sistema local precisa de uma prévia correspondência entre os elementos da ontologia, na qual a consulta é baseada, com os elementos do esquema local. Estas correspondências deverão estar definidas em uma tabela local.

- ✓ **O sistema local deverá retornar o resultado em XML.**

De forma a facilitar o processo de integração dos resultados, realizado pelo agente móvel, o sistema local deverá converter o resultado da consulta local para o formato XML.

- ✓ **O sistema deverá integrar os resultados parciais localmente.**

Após o sistema local ter entregue o resultado local ao agente móvel, este deverá integrar o novo resultado com os resultados previamente capturados, antes de prosseguir para o próximo destino.

- ✓ **O sistema deverá retornar à origem quando a consulta estiver satisfeita.**

Após a integração dos resultados de um local, o agente móvel deverá analisar o resultado acumulado, verificando se a consulta já está satisfeita. Caso a consulta já tenha sido

satisfeita, o agente móvel deverá retornar à origem. Caso contrário, deverá migrar para o próximo destino.

3.1.2 Requisitos não-funcionais

- ✓ **A visão da Web Semântica deverá ser a base do sistema.**

A Web Semântica visa proporcionar uma infra-estrutura que possibilite a comunicação entre máquinas na web. Não se trata de uma nova web, mas uma extensão da web atual. Nesta nova visão, a informação manipulada e compartilhada pelas máquinas tem semântica bem definida, de forma a permitir a automação de vários processos sem a necessidade de uma constante intervenção humana.

Acreditamos que os novos sistemas web deverão incorporar a visão da Web Semântica, utilizando tecnologias capazes de proporcionar a funcionalidade necessária para que a informação correta seja manipulada por máquinas. Nosso sistema deverá ser uma aplicação real desta visão, considerando a estrutura de uma aplicação distribuída definida por uma ontologia, garantindo a manutenção de um consenso semântico entre os bancos de dados envolvidos.

- ✓ **O sistema deverá ser de fácil manutenção.**

Nosso sistema trata da integração de dados entre bancos de dados autônomos e dinâmicos, os quais, ao longo do tempo, poderão sofrer alterações. Para que o sistema de integração entregue resultados coerentes ao usuário final, toda alteração nos bancos de dados locais deverá ser incorporada ao sistema. Para isso, cada banco de dados local deverá possuir um DBA[6] local, que informa o DBA Global sobre as alterações ocorridas. O DBA Global realiza as devidas manutenções na ontologia e nas correspondências entre os elementos da ontologia e os elementos do banco de dados local.

Dessa forma, o sistema deverá proporcionar uma fácil manutenção da ontologia, quando da inclusão de novos conceitos, etc., o que pode ser conseguido através do uso de editores de ontologias disponíveis gratuitamente na web, como o OntoEdit. Além disso, deverá existir uma fácil manutenção das correspondências entre os elementos de cada

[6] DBA: DataBase Administrator

esquema local e os elementos da ontologia, o que deverá ser conseguido através da atualização de uma tabela de correspondência global-local em cada banco de dados local.

3.1.3 Requisitos de interface

- ✓ **A ontologia deverá ser disponibilizada graficamente em um navegador web.**

A ontologia deverá estar disponibilizada, em modo gráfico, em um navegador web (browser) para que o usuário possa ter acesso ao sistema de qualquer local na web em que ele se encontre e conseqüentemente possa formular sua consulta.

- ✓ **A interface deverá ser de fácil utilização.**

Na preparação da consulta, não deverá ser exigido conhecimento de linguagens de acesso a bancos de dados, como SQL por exemplo. Assim, o formulário de geração da consulta deverá ser de fácil manuseio, de forma a simplificar a interação com o usuário.

3.2 Modelo de use-case

Um use-case é um documento narrativo que descreve a seqüência típica de eventos de um ator (um agente externo) usando um sistema para completar um processo (Jacobson *et al*, 1992). Com use-cases, buscamos entender melhor os requisitos funcionais do sistema, além de como deverá ocorrer a interação do usuário com o sistema.

Apresentamos os use-cases de forma expandida que, segundo (Larman, 2000), apresentam mais detalhes do que um use-case de alto nível, de modo que podemos obter um maior entendimento dos processos e requisitos do sistema. A diferença entre os dois tipos está na seção *seqüência típica dos eventos* que descreve a interação entre o ator e o sistema, e na seção *seqüências alternativas* que descreve situações alternativas ou exceções.

A Figura 12 apresenta o diagrama de use-case do nosso sistema. A seguir estão as descrições dos use-cases *Emitir Consulta*, *Processar Consulta*, *Apresentar Resultado* e *Incluir novo Banco de Dados na Federação*. Observe que já introduzimos o conceito de agentes.

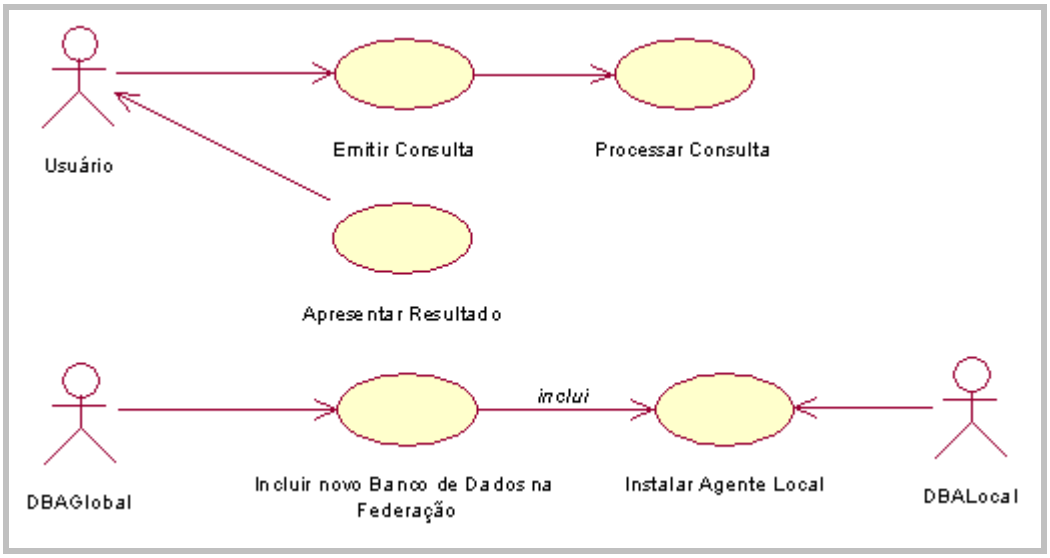

Figura 12: Diagrama de use-case

Use Case:	**Emitir Consulta**
Atores:	Um usuário do sistema
Agentes:	Agente-mestre
Descrição:	Um usuário aciona o sistema de consulta, prepara e emite uma consulta.

Seqüência típica de eventos:

Ação do ator	Resposta do sistema
1. O usuário aciona o sistema de consulta.	2. O sistema disponibiliza a ontologia em um modo gráfico.
3. O usuário seleciona um elemento da ontologia.	4. O sistema apresenta os campos de condições em um formulário, de acordo com o elemento da ontologia escolhido.
5. O usuário preenche o formulário e submete os dados ao sistema.	6. O sistema transforma os dados submetidos em uma instrução SQL.
	7. O sistema notifica o agente-mestre.

Use Case:	**Processar Consulta**
Atores:	Agente-mestre (representando o sistema)
Agentes:	Agente móvel (agente-escravo), AgenteBDLocal
Descrição:	O agente-mestre dispara um agente móvel, que realiza a integração de dados entre os bancos de dados.

Seqüência típica de eventos:

Ação do ator	Resposta do sistema
1. O agente-mestre recebe a consulta do sistema.	2. O agente-mestre prepara uma lista de destinos, incluindo somente os locais que podem conter a informação necessária para processar a consulta. Preparada a lista, o agente-mestre gera um agente móvel, passando-lhe a lista de destinos e a consulta emitida pelo usuário.
	3. O agente móvel migra para um SGBD remoto.
	4. O agente móvel notifica o agente que executa localmente, denominado AgenteBDLocal, e passa-lhe a consulta do usuário.
	5. O agente AgenteBDLocal adapta a consulta conforme o esquema local do banco de dados, extrai os dados, e retorna-os ao agente móvel.
	6. O agente móvel realiza uma integração dos dados coletados.
	7. Repetem-se os passos 3, 4, 5 e 6 até que o agente móvel tenha migrado para todos os

	destinos contidos na sua lista.
	8. O agente móvel retorna à origem.
	9. O agente móvel entrega os dados capturados ao agente-mestre.

Seqüências alternativas:

Evento 3: O agente móvel não conseguindo migrar para um determinado SGBD remoto, deverá gerar uma mensagem relatando o problema, guardar a mensagem, e migrar para o próximo destino.

Evento 6: Com a integração dos dados coletados, o agente móvel analisa o resultado e, percebendo que a consulta já foi atendida, deverá interromper o roteiro traçado e retornar ao agente-mestre.

Evento 8: O agente móvel não conseguindo migrar de volta à origem, deverá gerar uma mensagem de erro, guardar a mensagem, aguardar um período de tempo, e tentar migrar novamente.

Use Case:	**Apresentar Resultado**
Atores:	Um usuário do sistema
Agentes:	Agente-mestre
Descrição:	O resultado final obtido pelo agente móvel (incluindo mensagens de erro) é apresentado ao usuário pelo agente-mestre.

Use Case:	**Incluir novo Banco de Dados na Federação**
Atores:	DBAGlobal, DBALocal
Agentes:	AgenteBDLocal
Descrição:	Um novo banco de dados é inserido na federação.

Seqüência típica de eventos:

Ação do ator	Resposta do sistema
1. [DBAGlobal] Com base no esquema local do novo banco de dados, o DBAGlobal identifica objetos equivalentes a classes ou propriedades descritas na ontologia.	
2. [DBAGlobal] O DBAGlobal relaciona os elementos do esquema local com o esquema global. Com isso, ele prepara a tabela de correspondência global-local.	3. Cria a tabela de correspondência global-local no novo banco de dados.
4. [DBALocal] Prepara a instalação da agência local para abrigar o agente AgenteBDLocal.	5. Instala a agência.
6. [DBALocal] Prepara a instalação do agente AgenteBDLocal.	7. Instala o agente AgenteBDLocal que permanece executando localmente, aguardando as consultas trazidas pelo agente móvel.

Seqüências alternativas:

Evento 2: Se existir um objeto no novo banco de dados que não pode ser relacionado a qualquer classe ou propriedade da ontologia, a ontologia deve ser alterada de forma a expressar a semântica deste objeto.

3.3 Arquitetura proposta

Tradicionalmente, a recuperação da informação através da Internet é realizada utilizando-se a arquitetura cliente-servidor por meio de uma comunicação direta entre o cliente que necessita da informação e a fonte da informação. A comunicação indireta com um número maior de fontes, através do uso de agentes móveis, pode reduzir a sobrecarga de comunicação necessária para transmitir as requisições do cliente e as respostas do servidor. Além disso, o suporte a operações sem conexão contínua com a Internet e o menor volume de

dados transportados, representam algumas das várias vantagens do uso de agentes móveis neste tipo de aplicação.

A Figura 6 da seção 2.3 apresentou as camadas da Web Semântica, com linguagens de mais alto nível utilizando a sintaxe e as semânticas de linguagens de mais baixo nível. Nossa arquitetura concentra-se no nível da camada de ontologia. Linguagens de mais alto nível[7] podem proporcionar funcionalidades ainda mais interessantes, porém, devido a restrições de tempo, deixamos estas camadas para serem tratadas em trabalhos futuros.

O objetivo da nossa arquitetura é proporcionar uma interface uniforme para SGBDs heterogêneos e distribuídos, de forma que usuários possam gerar consultas que serão transportadas entre os diferentes bancos de dados por um agente móvel, produzindo um resultado final consolidado ao usuário. Para isso, é necessário identificar quais objetos nestes bancos de dados representam conceitos equivalentes, ou seja, que estejam semanticamente relacionados. Heterogeneidades entre os bancos de dados surgem como conflitos semânticos e sintáticos. Estes conflitos são detectados e solucionados no momento em que um novo banco de dados é inserido na federação.

Uma vez identificados os conceitos equivalentes no novo banco de dados, estes são definidos em uma ontologia (o esquema global). Esta ontologia descreve os conceitos como classes e propriedades, definindo também os relacionamentos entre eles. A interface do sistema para o usuário deverá ser construída baseada nesta ontologia. Dessa forma, as consultas do usuário conterão termos do esquema global do banco de dados federado. A adaptação da consulta do usuário ocorre verificando a relação de cada termo com o esquema local do banco de dados. Aplicando este mecanismo, estaremos resolvendo os conflitos semânticos e sintáticos existentes em cada banco de dados local.

Como descrevemos anteriormente, o uso de uma ontologia proporciona a semântica necessária para a solução de conflitos de heterogeneidade dos bancos de dados. Mostraremos agora como o uso de agentes, aliada a esta semântica, proporciona uma solução adequada para o processo de integração.

Como vimos no capítulo anterior, existem vários padrões de projeto para aplicações baseadas em agentes móveis. Esses padrões são normalmente divididos em três classes: padrões de *mobilidade, tarefa,* e *interação* (Lange & Oshima, 1998). Na nossa arquitetura, estamos propondo o uso de um padrão de mobilidade (Itinerário) e um padrão de tarefa (Mestre-escravo).

[7] Camadas de Lógica, Prova e Confiança.

O padrão de mobilidade Itinerário basicamente mantém uma lista de destinos e sempre sabe qual o próximo destino a ser visitado pelo agente. Este padrão também define casos especiais, como, por exemplo, que decisão o agente deve tomar caso o próximo destino não esteja disponível. Na nossa arquitetura, estamos propondo uma versão estendida deste padrão, descrita na seção 3.5.

Padrões de tarefa tratam da definição de tarefas e como estas tarefas são delegadas a um ou mais agentes. Na nossa arquitetura, propomos o uso do padrão de tarefa Mestre-escravo, o qual permite que um agente-mestre delegue tarefas a um agente-escravo.

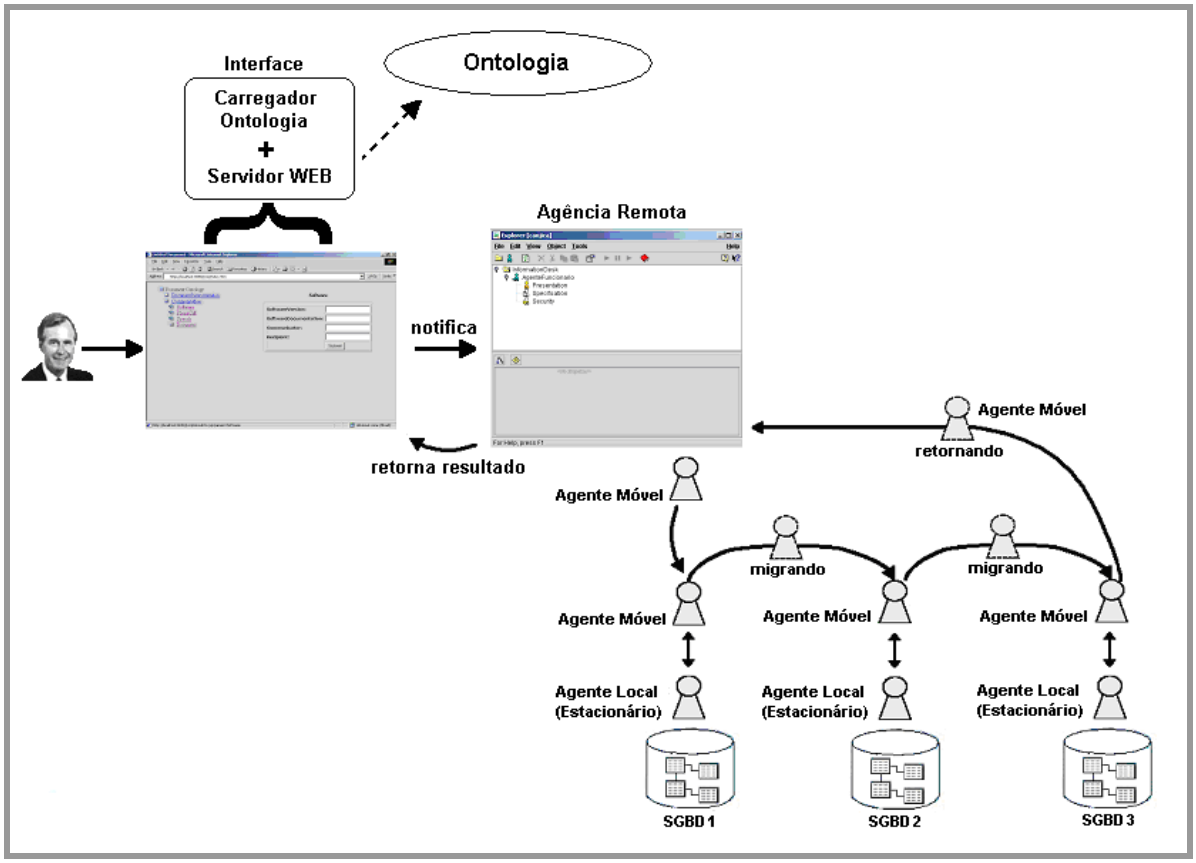

Figura 13: Arquitetura do DIA

Como podemos ver na Figura 13, o usuário aciona o sistema submetendo uma consulta. O sistema então notifica um agente estacionário, passando-lhe a consulta do usuário. Este agente cria um agente móvel (agente-escravo) delegando-lhe a tarefa de localização e integração dos dados. O agente estacionário (agente-mestre) entrega a consulta do usuário e uma lista de destinos (veja a seção 3.4) ao agente-escravo, o qual inicia sua migração.

Ao chegar em um destino, o agente móvel notifica um agente estacionário, que executa localmente, entregando-lhe a consulta do usuário. Este agente estacionário deverá

adaptar a consulta, capturar os dados e entregá-los ao agente móvel. A adaptação da consulta é realizada através de um processo que mapeia os termos da consulta (baseados no esquema global) para o esquema do banco de dados local. Adaptada a consulta, os dados são extraídos e apresentados ao agente móvel em XML.

O agente irá migrar com o resultado em XML para os próximos destinos, integrando-o aos resultados obtidos em cada novo local visitado (veja a seção 3.6). Com isso, reduz-se o tamanho do XML transportado, além de garantir que, ao final da migração, o resultado final já estará devidamente integrado.

As migrações continuam até que o agente móvel decide retornar à sua origem. Segundo o padrão básico de mobilidade Itinerário, o agente só retorna à origem após passar por todos os locais do itinerário. Na versão estendida deste padrão (veja a seção 3.5), o agente móvel pode retornar à origem antes disso. Baseando-se no resultado obtido, o agente móvel analisa se a consulta já foi satisfeita. Caso a consulta já tenha sido satisfeita, o agente retorna à sua origem, mesmo não tendo migrado para todos os destinos.

Uma vez retornado à origem, o agente móvel entrega o resultado final (já integrado) ao seu agente-mestre e remove-se automaticamente. O agente-mestre então apresenta o resultado da consulta ao usuário.

O comportamento geral do sistema, descrito nesta seção, pode ser mais bem compreendido através do diagrama de atividades da Figura 14.

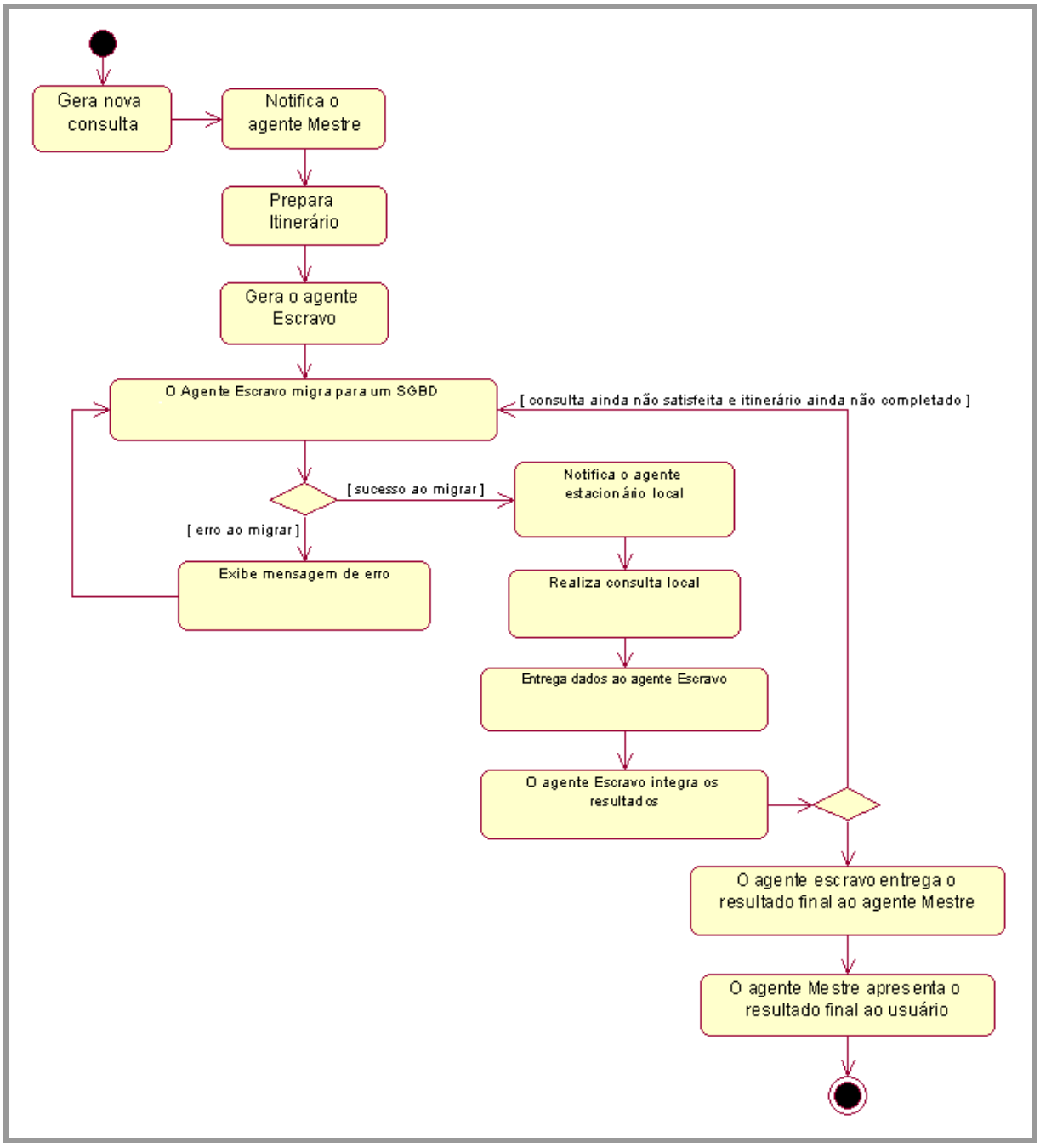

Figura 14: Diagrama de atividades

Na Figura 15, apresentamos o modelo conceitual do DIA, ilustrando os conceitos significativos que envolvem o sistema.

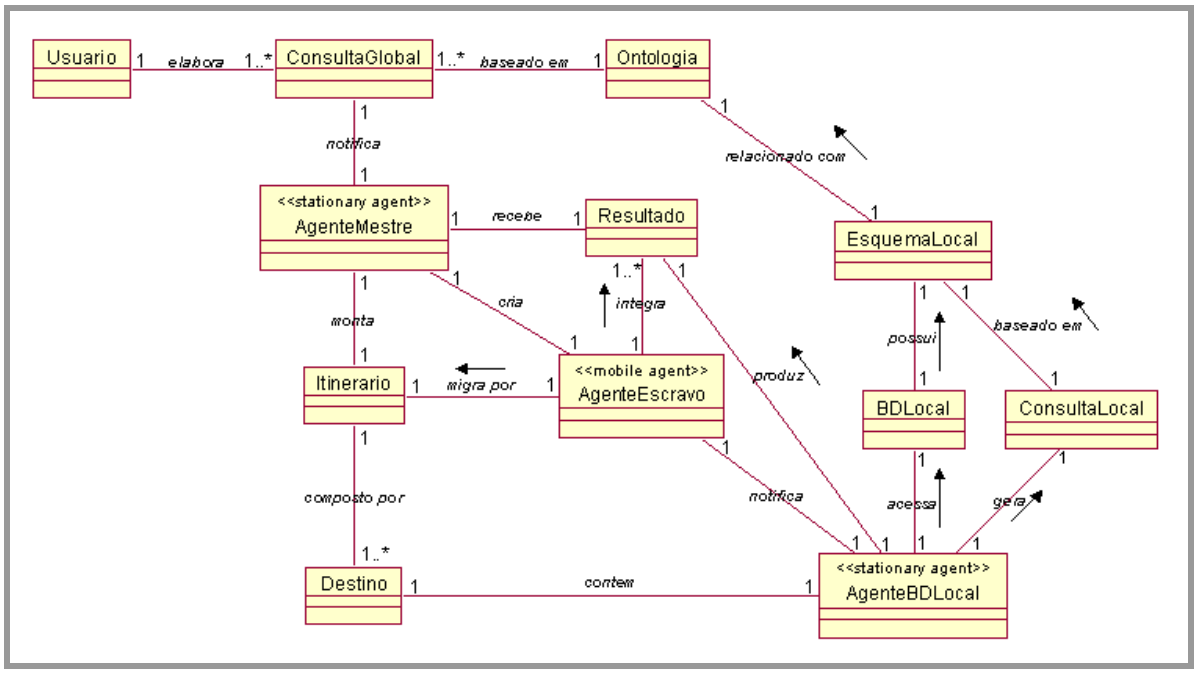

Figura 15: Modelo conceitual do DIA

A fim de representar agentes móveis no nosso modelo, utilizamos a extensão proposta por (Klein *et al*, 2001): o estereótipo <<mobile agent>> especifica uma classe de objetos agentes móveis.

A fim de representar agentes estacionários no nosso modelo, utilizamos a extensão proposta por (Guedes & Machado, 2001): o estereótipo <<stationary agent>> especifica uma classe de objetos agentes estacionários.

3.4 Preparando o itinerário do agente móvel

Como já citamos na seção anterior, um itinerário é basicamente uma lista de destinos pelos quais o agente móvel deverá passar. Segundo o modelo conceitual ilustrado na Figura 15, o agente estacionário *AgenteMestre* monta um itinerário que é composto por um ou mais destinos. O itinerário é entregue ao agente móvel *AgenteEscravo* no momento da sua criação, baseado no qual o agente realizará sua migração. Nesta seção, detalhamos como ocorre a preparação do itinerário para o agente móvel.

O esquema global, definido como uma ontologia, apresenta conceitos comuns presentes nos bancos de dados da federação. Dada uma consulta no esquema global, devido a heterogeneidade dos bancos de dados, é provável que alguns bancos de dados não possuam

dados para atender à consulta. Dessa forma, o agente móvel não deverá visitar estes bancos de dados.

Para garantir a geração de um itinerário otimizado, o agente-mestre (que cria o agente móvel) deverá conhecer todos os esquemas dos bancos de dados da federação. Dessa forma, de acordo com a comparação da consulta do usuário com cada esquema local, o agente-mestre é capaz de descobrir em quais bancos de dados há algo a ser processado. Assim, serão incluídos no itinerário do agente móvel apenas os endereços dos bancos de dados que possuam dados que atendem à consulta, garantindo que o agente não visitará locais desnecessariamente.

A seguir, apresentamos um exemplo que detalha o processo de preparação do itinerário. Considere a seguinte consulta em linguagem SQL:

```
Select Cod, SUM(Creditos) From Cliente
Where Cod = '02757'
```

A consulta acima representa a consulta do usuário, que é formulada de acordo com o esquema global. No processo de integração, que é realizado pelo agente móvel, em cada banco de dados, a consulta é mapeada segundo o esquema local. O processo de mapeamento está descrito na seção 3.7.

No entanto, os esquemas de alguns bancos de dados da federação podem não conter elementos correspondentes aos elementos ontológicos da consulta. Por exemplo, um banco de dados cujo esquema não contém um atributo correspondente ao elemento "Creditos", não possui dados para atender à consulta acima. Dessa forma, o agente-mestre, que conhece os esquemas dos bancos de dados locais, não incluirá o endereço deste banco de dados no itinerário do agente móvel.

3.5 Aprimorando o padrão de mobilidade Itinerário

Na seção anterior, apresentamos o critério usado na preparação do itinerário para o agente móvel. Nesta seção, mostraremos como o agente móvel poderá, durante a sua viagem, perceber que a consulta já foi atendida e retornar antecipadamente para o agente-mestre.

Como citamos anteriormente, o nosso sistema segue a abordagem virtual para a integração de dados. A grande desvantagem da nossa abordagem em relação à abordagem

materializada é o desempenho, pois, envolve a tradução de esquemas em tempo de execução. Buscando um melhor desempenho, propomos uma extensão ao padrão de mobilidade Itinerário. Dessa forma, distinguimos dois tipos de itinerários:

- Estático: o agente móvel só retorna à origem após ter migrado para o último destino;
- Dinâmico: o agente móvel decide continuar seguindo o itinerário ou retornar à origem. A decisão do agente depende de uma pré-classificação do itinerário, conforme descrito na Tabela 4, e dos dados que o agente acabou de obter no local.

Por exemplo, consultas como “Quando é o aniversário de Philip?” ou “Philip tem um número de créditos superior a 700?” sugerem um itinerário dinâmico, enquanto que, a consulta “Qual o número total de créditos de Philip” sugere um itinerário estático.

3.5.1. Taxonomia de consultas

Ao analisar os tipos de consultas que podem ser feitas a um banco de dados, descobrimos que elas podem ser divididas em duas categorias:

- Consultas para as quais o agente sempre tem que percorrer todos os bancos de dados locais do itinerário para serem respondidas (itinerário estático);
- Consultas em que a resposta já pode estar completa antes do agente ter percorrido todos os bancos de dados do itinerário, portanto, o agente pode retornar antecipadamente (itinerário dinâmico).

A fim de utilizar esta divisão para melhorar o desempenho do sistema, criamos uma taxonomia de consultas. A classificação do itinerário permite que o agente decida quando retornar à origem, mesmo que ainda não tenha sido alcançado o final do itinerário. A Tabela 4 a seguir apresenta a taxonomia de consultas.

Caso	Condição	Agregação	Satisfeita	Itinerário
1	Sim	Não	Todos	Estático
2	Sim	Não	Primeiro	Dinâmico
3	Sim	Sim	Todos	Estático
4	Sim	Sim	Primeiro	Dinâmico
5	Não	Não	--	Estático
6	Não	Sim	--	Estático

Tabela 4: Taxonomia de consultas

Quando o usuário submete os dados da consulta, o agente-mestre estrutura estes dados em uma consulta na linguagem SQL. Com isso, o agente-mestre analisa a estrutura da consulta e classifica-a segundo a taxonomia descrita na Tabela 4.

Analisando os tipos de consultas, percebemos que a existência ou não de condições e de operações de agregação, são determinantes para a classificação do itinerário em *Estático* ou *Dinâmico*. Por exemplo, o caso 5 da Tabela 4 diz que toda consulta que não possui condição e nem operação de agregação, sempre requer um itinerário estático.

No entanto, percebemos que alguns tipos de consultas possuem as mesmas características de condição e agregação[8], porém, tanto podem sugerir um itinerário estático como um dinâmico. Nestes casos, o agente-mestre analisa o atributo *Satisfeita*. Esta característica é fortemente dependente da semântica da consulta, devendo ser determinada pelo usuário.

A seguir, explicamos cada caso da Tabela 4 com exemplos:

- Caso 1: A consulta “Quero os clientes cujos salários são superiores a 1000” requer um itinerário estático, pois, todos os bancos de dados precisam ser consultados para ser satisfeita.
- Caso 2: A consulta “Existe algum cliente com salário superior a 1000?” requer um itinerário dinâmico, pois, uma vez encontrado um cliente com salário superior a 1000, a consulta estará satisfeita e o agente poderá retornar para o agente-mestre.

[8] Casos 1, 2 e casos 3, 4 da Tabela 4.

- Caso 3: A consulta “Quero a soma dos salários dos clientes cujos limites de crédito são superiores a 1000 e foram cadastrados após a data 01/01/2002” requer um itinerário estático, pois, só estará satisfeita após o agente ter passado por todos os bancos de dados.
- Caso 4: A consulta “Quero os nomes e a soma dos débitos dos clientes que foram cadastrados após a data 01/01/2002, sendo que a soma desses débitos não pode ser superior a 10.000 reais” requer um itinerário dinâmico, pois, uma vez que a soma dos débitos resultar em um valor maior ou igual a 10.000 reais, a consulta estará satisfeita e o agente poderá retornar para o agente-mestre.
- Caso 5: A consulta “Quero os nomes dos clientes” requer um itinerário estático.
- Caso 6: A consulta “Conte quantos clientes eu tenho” requer um itinerário estático.

3.6 Integração local dos dados

Após o processo de adaptação local da consulta e havendo sido extraídos os dados do banco de dados local, o agente estacionário local entrega os dados ao agente móvel em XML. A Figura 16 ilustra como uma tupla com três atributos é entregue ao agente móvel.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">027571664-33</coluna>
     <coluna nome="NOME" operacao="nenhuma">Philip</coluna>
     <coluna nome="CREDITOS" operacao="SUM">200</coluna>
  </linha>
</ResultSet>
```

Figura 16: Exemplo do resultado na sintaxe XML

O elemento “coluna” possui dois atributos: “nome” e “operação”. O primeiro apresenta o nome do atributo, e o segundo apresenta a operação a ser realizada sobre o atributo durante a integração. Para tornar o processo de integração possível, os nomes dos atributos deverão ser coerentes com os nomes dos termos da consulta do usuário.

A integração dos resultados pode ser um processo de concatenação ou de junção. A escolha pela concatenação ou pela junção dos resultados é baseada na consulta do usuário. Para isso, a interface do sistema deve proporcionar um componente de seleção, através do qual o usuário seleciona o processo de integração que deverá ser aplicado. Com base nisso, o agente móvel decide qual dos processos deve ser aplicado na integração dos resultados.

O processo de concatenação apenas adiciona as novas linhas, obtidas localmente, ao resultado obtido previamente. A Figura 17 ilustra a concatenação de resultados.

```
<!-- Primeiro XML -->
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">027571664-33</coluna>
     <coluna nome="DATAADMISSAO" operacao="nenhuma">19/03/2001</coluna>
     <coluna nome="NOME" operacao="nenhuma">Philip</coluna>
  </linha>
</ResultSet>

<!-- Segundo XML -->
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">001122334-44</coluna>
     <coluna nome="DATAADMISSAO" operacao="nenhuma">02/08/1999</coluna>
     <coluna nome="NOME" operacao="nenhuma">Petronio</coluna>
  </linha>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">014457012-55</coluna>
     <coluna nome="DATAADMISSAO" operacao="nenhuma">15/11/2000</coluna>
     <coluna nome="NOME" operacao="nenhuma">Claudivan</coluna>
  </linha>
</ResultSet>

<!-- XML Resultante -->
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">027571664-33</coluna>
     <coluna nome="DATAADMISSAO" operacao="nenhuma">19/03/2001</coluna>
     <coluna nome="NOME" operacao="nenhuma">Philip</coluna>
  </linha>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">001122334-44</coluna>
     <coluna nome="DATAADMISSAO" operacao="nenhuma">02/08/1999</coluna>
     <coluna nome="NOME" operacao="nenhuma">Petronio</coluna>
  </linha>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">014457012-55</coluna>
     <coluna nome="DATAADMISSAO" operacao="nenhuma">15/11/2000</coluna>
     <coluna nome="NOME" operacao="nenhuma">Claudivan</coluna>
  </linha>
</ResultSet>
```

Figura 17: Concatenação de resultados

A junção de resultados é um processo mais elaborado de integração. Para isso, a adaptação local da consulta deve garantir que cada linha resultante terá um atributo que o identifica unicamente. Como pode ser visto na Figura 18, a junção de um XML atual com um novo XML ocorre através da comparação de seus elementos “linha”, realizando as seguintes etapas:

- Se o atributo-chave de uma linha no novo XML corresponde ao atributo-chave de uma linha no XML antigo, todos os demais atributos são integrados baseados nas suas respectivas operações.
- Se o atributo-chave de uma linha no novo XML não corresponde a nenhum atributo-chave no XML antigo, a linha corrente do novo XML é inteiramente adicionada ao XML antigo.

- Se o atributo-chave de uma linha no novo XML corresponde ao atributo-chave de uma linha no XML antigo, e a linha do novo XML possui um atributo que não está presente na linha correspondente do XML antigo, então o atributo é adicionado à linha corrente do XML antigo.
- A operação “nenhuma” é atribuída a todo atributo não-numérico. Se um conflito ocorrer, o sistema mantém o valor do atributo do XML antigo.

```
<!-- Primeiro XML -->
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">027571664-33</coluna>
     <coluna nome="NOME" operacao="nenhuma">Philip</coluna>
     <coluna nome="CREDITOS" operacao="SUM">200</coluna>
  </linha>
</ResultSet>


<!-- Segundo XML -->
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">027571664-33</coluna>
     <coluna nome="NOME" operacao="nenhuma">Philip</coluna>
     <coluna nome="CREDITOS" operacao="SUM">300</coluna>
     <coluna nome="DataNascimento" operacao="nenhuma">03/07/1978</coluna>
  </linha>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">014457012-55</coluna>
     <coluna nome="NOME" operacao="nenhuma">Ulrich</coluna>
     <coluna nome="CREDITOS" operacao="SUM">1000</coluna>
     <coluna nome="DataNascimento" operacao="nenhuma">12/10/1946</coluna>
  </linha>
</ResultSet>

<!-- XML Resultante -->
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">027571664-33</coluna>
     <coluna nome="NOME" operacao="nenhuma">Philip</coluna>
     <coluna nome="CREDITOS" operacao="SUM">500</coluna>
     <coluna nome="DataNascimento" operacao="nenhuma">03/07/1978</coluna>
  </linha>
  <linha>
     <coluna nome="CPF" operacao="nenhuma">014457012-55</coluna>
     <coluna nome="NOME" operacao="nenhuma">Ulrich</coluna>
     <coluna nome="CREDITOS" operacao="SUM">1000</coluna>
     <coluna nome="DataNascimento" operacao="nenhuma">12/10/1946</coluna>
  </linha>
</ResultSet>
```

Figura 18: Junção de resultados

3.7 O processo de mapeamento de consultas

Como vimos anteriormente, o processo de adaptação de uma consulta no esquema global a uma consulta no esquema local é automático. Para isso, é necessário um processo anterior que gere as correspondências sintáticas entre os dois esquemas. No nosso sistema, usamos a solução descrita em (Arruda, 2002), que se baseia na criação de uma tabela contendo um conjunto de regras de correspondências para o modelo relacional. Nesta seção, apresentaremos essas regras de correspondências, extraídas de (Arruda, 2002).

Apresentaremos as regras de correspondências para cada possibilidade básica de relacionamento no modelo relacional. Para todas as regras, utilizamos as seguintes definições:

- Entidade **Eq_C1**: entidade equivalente à classe C1;
- Entidade **Eq_p**: entidade equivalente à propriedade p;
- **FK**: chave estrangeira;
- **PK**: chave primária;
- **null**: Nulo.

Regra 1: Quando uma tabela "A", em um determinado esquema, corresponde a uma classe "C1" na ontologia e um atributo "a" desta tabela A, corresponde a uma propriedade "p" da mesma classe da ontologia, o preenchimento da tabela de correspondências deve seguir o modelo descrito na Tabela 5.

Classe	Propriedade	Tabela	Atributo	Junção
C1	p	Eq_C1	Eq_p	null

Tabela 5: Tabela de correspondência da regra 1

Exemplo 1:

Consideremos os seguintes elementos de uma ontologia:

Classe C1 = Cliente

Propriedade p = nome

Um exemplo de consulta segundo o esquema global:

- ✓ **Select * from Cliente where nome = 'Philip'**

Esquema relacional: Entidade Eq_C1 com atributo Eq_p

Tabela Eq_C1 = Comprador

Ou seja, a tabela Comprador corresponde à classe Cliente da ontologia. A mesma lógica aplica-se à propriedade: Eq_p = nome. Dessa forma, pelo nosso exemplo 1, apresentamos a consulta mapeada segundo a regra 1, e a tabela de correspondência preenchida (Tabela 6).

Consulta adaptada:

- ✓ **Select * from Comprador where nome = 'Philip'**

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	nome	Comprador	nome	null

Tabela 6: Tabela de correspondência do exemplo 1

Regra 2: Quando uma tabela "A", em um determinado esquema, corresponde a uma classe "C1" na ontologia e uma tabela "B" relacionada com a tabela "A" com multiplicidade (1,*) ou (1,1) corresponde a uma propriedade "p" da mesma ontologia, o preenchimento da tabela de correspondências deve seguir o modelo descrito na Tabela 7.

Classe	Propriedade	Tabela	Atributo	Junção
C1	p	Eq_C1 x, Eq_p y	y.atributo	y.FK = x.PK

Tabela 7: Tabela de correspondência da regra 2

Exemplo 2:

Consideremos os seguintes elementos de uma ontologia:

Classe C1 = Cliente

Propriedade p = estado

Um exemplo de consulta segundo o esquema global:

- ✓ **Select * from Cliente where estado = 'Paraíba'**

Esquema relacional: Entidade Eq_C1 com Entidade Eq_p

Tabela Eq_C1 = Comprador

Tabela Eq_p = Contato (1,1) ou (1,*)

Dessa forma, pelo nosso exemplo 2, apresentamos a consulta mapeada segundo a regra 2, e a tabela de correspondência preenchida (Tabela 8).

Consulta adaptada:

- ✓ **Select * from Comprador x, Contato c**
  where c.estado = "Paraíba" AND c.ident = x.ident

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	estado	Comprador x, Contato c	c.estado	c.ident = x.ident

Tabela 8: Tabela de correspondência do exemplo 2

Regra 3: Quando uma tabela "A", em um determinado esquema, corresponde a uma classe "C1" na ontologia e uma tabela "B" relacionada com a tabela "A" com multiplicidade (*,1) ou (1,1), corresponde a uma propriedade "p" da mesma ontologia, o preenchimento da tabela de correspondências deve seguir o modelo descrito na Tabela 9.

Classe	Propriedade	Tabela	Atributo	Junção
C1	p	Eq_C1 x, Eq_p y	y.atributo	x.FK = y.PK

Tabela 9: Tabela de correspondência da regra 3

Exemplo 3:

Consideremos os seguintes elementos de uma ontologia:

Classe C1 = Cliente

Propriedade p = corretor

Um exemplo de consulta segundo o esquema global:

- ✓ **Select * from Cliente where corretor = 'Ulrich'**

Esquema relacional: Entidade Eq_C1 com Entidade Eq_p

Tabela Eq_C1 = Comprador

Tabela Eq_p = Corretor (1,1) ou (*,1)

Dessa forma, pelo nosso exemplo 3, apresentamos a consulta mapeada segundo a regra 3, e a tabela de correspondência preenchida (Tabela 10).

Consulta adaptada:

- ✓ **Select * from Comprador x, Corretor c**
  where c.nome = "Ulrich" AND x.corretor = c.nome

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	corretor	Comprador x, Corretor c	c.nome	x.corretor = c.nome

Tabela 10: Tabela de correspondência do exemplo 3

Regra 4: Quando uma tabela "A", em um determinado esquema, corresponde a uma classe "C1" na ontologia e uma tabela "B", que pertence a uma propriedade "p" da mesma ontologia, está relacionada com a tabela "A" através de uma tabela de junção "AB" (devido a um relacionamento (*,*)), o preenchimento da tabela de correspondências deve seguir o modelo descrito na Tabela 11.

Classe	Propriedade	Tabela	Atributo	Junção
C1	P	Eq_C1 x, Eq_p y, tabelaJuncao xy	y.atributo	y.PK = xy.FK_y AND xy.FK_x = x.PK

Tabela 11: Tabela de correspondência da regra 4

Exemplo 4:

Consideremos os seguintes elementos de uma ontologia:

Classe C1 = Cliente

Propriedade p = salario

Um exemplo de consulta segundo o esquema global:

- ✓ **Select * from Cliente where salario = 220,00**

Esquema relacional: Entidade Eq_C1 com Entidade Eq_p

Tabela Eq_C1 = Comprador

Tabela Eq_p = Emprego

Tabela xy = compradorEmprego

Dessa forma, pelo nosso exemplo 4, apresentamos a consulta mapeada segundo a regra 4, e a tabela de correspondência preenchida (Tabela 12).

Consulta adaptada:

- ✓ **Select * from Comprador x, Emprego y, compradorEmprego xy**
  where y.salario = 220,00 AND y.cod = xy.codEmprego
  AND xy.codComprador = x.CPF

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	salario	Comprador x, Emprego y, compradorEmprego xy	y.salario	y.cod = xy.codEmprego AND xy.codComprador = x.CPF

Tabela 12: Tabela de correspondência do exemplo 4

Capítulo 4

Estudo de caso: Um sistema de consulta de informações de clientes de uma corporação comercial distribuída

No capítulo anterior apresentamos uma proposta de acesso a bancos de dados federados na web, denominada DIA. Neste capítulo, apresentamos uma visão prática de implementação por meio de um estudo de caso. A meta é ratificar as propostas apresentadas no capítulo anterior, dessa vez, com um exemplo completo.

Na seção 4.1 apresentamos a interface desenvolvida para o protótipo. Na seção 4.2 apresentamos a estrutura de classes do protótipo. Na seção 4.3 apresentamos os agentes implementados. Na seção 4.4 apresentamos a infra-estrutura preparada para a execução do protótipo. Na seção 4.5 apresentamos o esquema global. Na seção 4.6 apresentamos os esquemas dos bancos de dados, como também, exemplos práticos do processo de mapeamento de consultas. Na seção 4.7 apresentamos o processo de inclusão de um novo banco de dados na infra-estrutura de execução do protótipo. Na seção 4.8 finalizamos o capítulo com um exemplo de uma execução do protótipo.

4.1 A interface do protótipo

Dentre os requisitos de interface, descritos na seção 3.1.3, a apresentação da ontologia (o esquema global) em um modo gráfico representa o maior desafio na preparação da interface. Durante o projeto da interface para o protótipo, decidimos utilizar um pacote denominado JXML, desenvolvido por Carlos Alexandre de Lima (Arruda *et al*, 2002), o qual gera uma representação gráfica em árvore de uma ontologia escrita na linguagem DAML+OIL.

O pacote JXML é um conjunto de classes que se propõe ao tratamento e apresentação de ontologias no formato de uma árvore hierárquica. A estrutura geral de funcionamento do JXML está ilustrada na Figura 19.

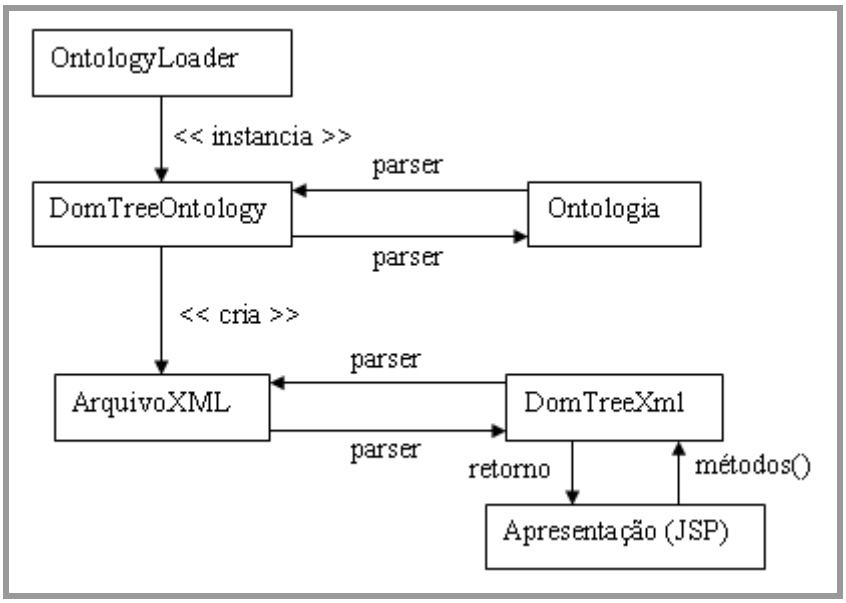

Figura 19: Funcionamento do pacote JXML

Como entrada para o JXML, deve ser apresentado um arquivo com uma ontologia definida na linguagem DAML+OIL. O pacote possui um conjunto de classes responsáveis pela análise (parse) da ontologia, retirando e colocando em estruturas internas destas classes os valores que serão interessantes para as operações com a ontologia. Após separar as classes, subclasses e propriedades da ontologia, realiza-se um mapeamento destes para elementos e atributos em um arquivo XML. Com isso, o arquivo XML é analisado (parse) para a formação da árvore hierárquica.

Com o arquétipo da árvore gerado, desenvolvemos arquivos JSP para invocar os métodos fornecidos pela classe de tratamento do arquivo XML (DomTreeXML), de modo a apresentar a árvore em um navegador. A Figura 20 apresenta a interface do nosso protótipo.

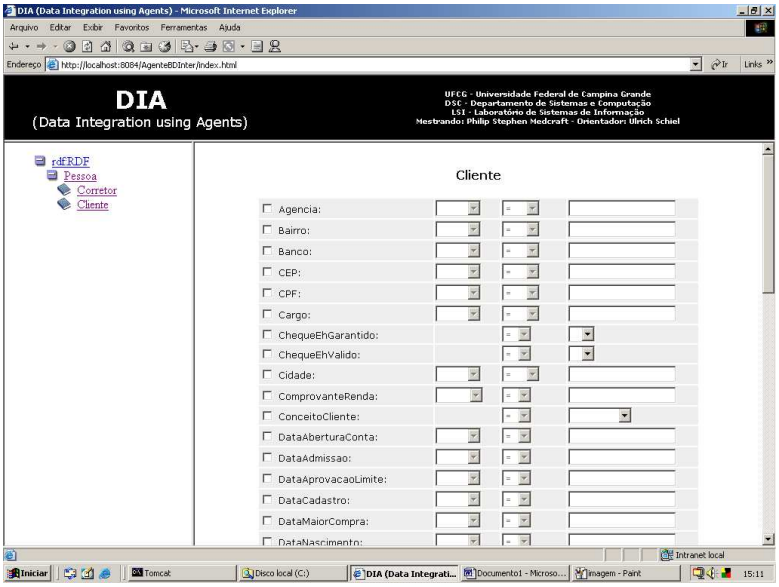

Figura 20: A interface do protótipo

Como destacamos na Figura 21, a árvore gerada a partir da ontologia fica disposta no lado esquerdo do navegador. Na seção 4.1.1 discutiremos limitações na estrutura da árvore gerada pelo pacote JXML.

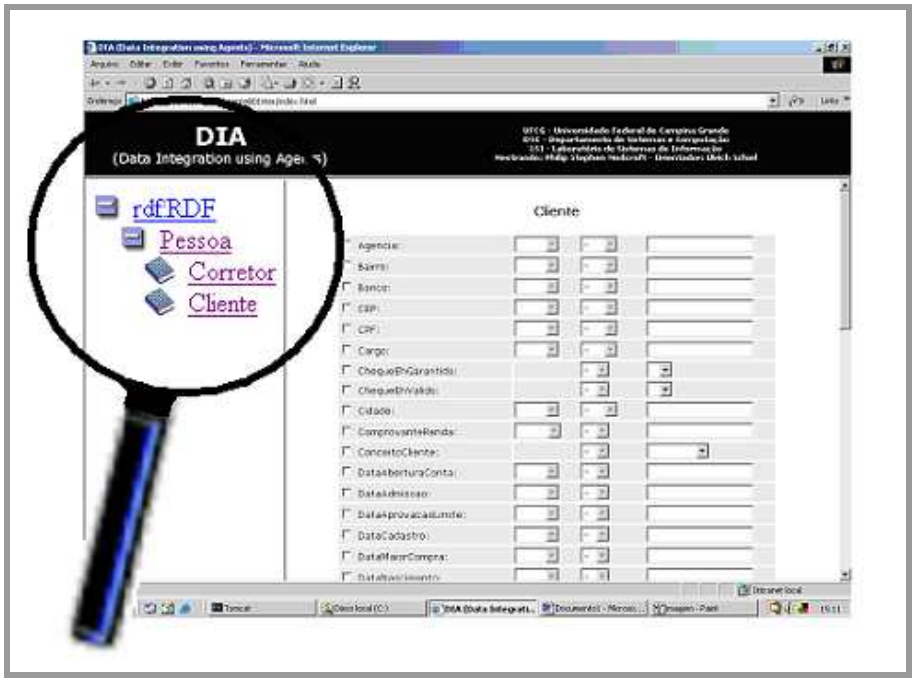

Figura 21: A árvore gerada a partir da ontologia

Uma vez que um dado elemento da árvore é selecionado, os atributos do elemento são dispostos no lado direito do navegador. Estes atributos são organizados em um formulário HTML contendo um conjunto de componentes para a criação da consulta. A Figura 22 ilustra a estrutura do formulário da classe Cliente.

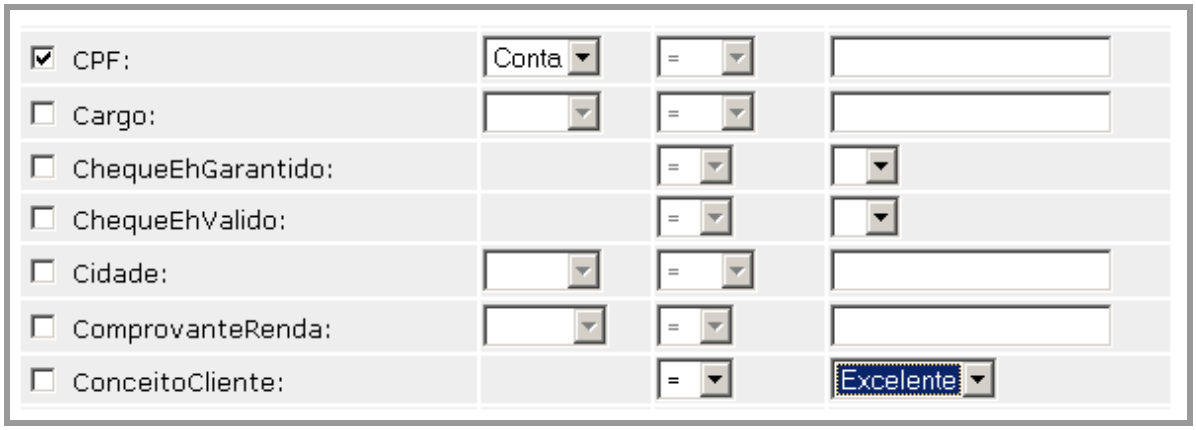

Figura 22: A estrutura do formulário da classe Cliente

Para cada atributo, o formulário apresenta quatro componentes. O primeiro componente é do tipo *CheckBox*, situado ao lado esquerdo do nome do atributo. Quando selecionado, este componente indica que o atributo deverá fazer parte da projeção da consulta. Por exemplo, no formulário da Figura 22, o usuário selecionou o atributo "CPF" para a projeção da nova consulta.

O segundo componente é do tipo *ComboBox*, e contém os tipos possíveis de operações de agregação SQL sobre o atributo. Este componente só estará acessível caso o atributo já tenha sido incluído na projeção da consulta. Os tipos de operações SQL desenvolvidos no protótipo foram: COUNT, SUM, AVG, MAX e MIN. No formulário da Figura 22, o usuário selecionou a operação "Conta" (COUNT) sobre o atributo "CPF" da projeção.

Caso o usuário deseje que um dado atributo seja uma das condições da consulta, este deverá selecionar, no terceiro componente, o operador para a condição, além de digitar ou selecionar, o valor da condição no quarto componente do atributo.

O terceiro componente é do tipo *ComboBox*, e possui os possíveis operadores para as condições da consulta. Os operadores são: >, <, =, <> e LIKE. Este componente só estará acessível caso o valor da condição já tenha sido determinado.

O quarto componente pode ser do tipo *Text* ou do tipo *ComboBox*. Caso o atributo seja do tipo *Alternative*, então o componente será do tipo *ComboBox*, e conterá as alternativas de valores para a condição. Caso contrário, o componente será do tipo *Text*, no qual o usuário deverá digitar o valor da condição.

No formulário da Figura 22, o usuário determinou uma condição: *ConceitoCliente* = *Excelente*, com o operador "=" selecionado no terceiro componente e o valor da condição selecionado no quarto componente.

Para finalizar o formulário, definimos três novos componentes. Como está ilustrado na Figura 23, o primeiro deles (Agrupar por) é do tipo *ComboBox*, que define o atributo de agrupamento (*group by*) da consulta. O segundo componente (Satisfeito por) é do tipo *ComboBox*, possuindo dois valores para seleção: *Todos* e *Primeiro*. Como citamos na seção 3.5.1, algumas consultas podem sugerir tanto um itinerário estático como um itinerário dinâmico. Portanto, selecionando o valor *Todos*, o usuário estará declarando que deseja todas as tuplas que satisfazem as condições da consulta[9], enquanto que, selecionando o valor *Primeiro*, o usuário se dará por satisfeito se uma tupla que satisfaz as condições for encontrada[10]. O terceiro componente (Fundir resultado) é do tipo *CheckBox*, o qual estando selecionado, determina a junção dos dados no processo de integração, caso contrário, determina a concatenação dos dados no processo de integração. Para que as funções de agregação definidas anteriormente sejam aplicadas, é preciso que seja realizada a junção dos dados no processo de integração. Neste caso, o terceiro componente (Fundir resultado) deverá estar selecionado.

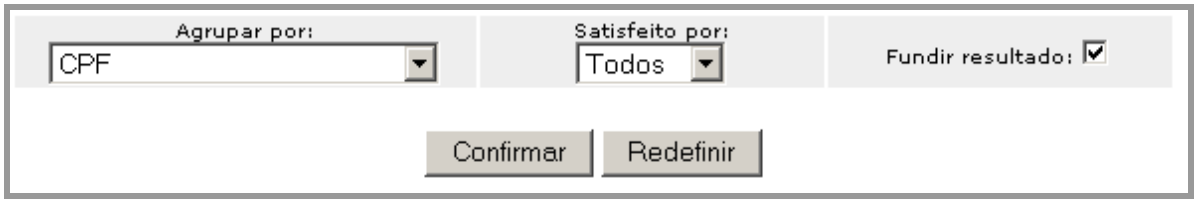

Figura 23: A estrutura final do formulário

Conforme descrito nesta seção, o protótipo que desenvolvemos possibilita ao usuário realizar consultas visuais a partir de elementos de uma ontologia DAML+OIL. A linguagem DAML+OIL é um formalismo orientado a objetos e SQL é relacional, portanto, existe uma dificuldade na tradução da consulta em DAML+OIL para a linguagem SQL devido à incompatibilidade existente entre os modelos destas linguagens. No nosso protótipo, a montagem da consulta SQL ocorre de forma bastante simples. Na verdade, o nosso protótipo não realiza uma tradução entre uma instância DAML+OIL e a sua versão em SQL. Quando o usuário submete uma consulta, os dados incluídos no formulário são tratados e guardados em vetores (vetor de atributos, vetor de condições, etc.). Com isso, o protótipo monta a consulta em linguagem SQL, segundo definido na interface pelo usuário.

[9] Requer itinerário estático.
[10] Requer itinerário dinâmico.

4.1.1 Limitações da interface do protótipo

Como citamos anteriormente, decidimos utilizar o pacote JXML para gerar uma representação gráfica em árvore da ontologia. Na árvore da Figura 21, temos os elementos *Pessoa*, *Corretor* e *Cliente*, todos extraídos da ontologia. Como já citamos anteriormente, uma vez selecionado um desses elementos, os atributos deste são apresentados no lado direito do navegador, estruturados em um formulário HTML para a criação da consulta pelo usuário. Dessa forma, a consulta do usuário sempre será criada com atributos de um único elemento, o que representa uma restrição na criação de consultas.

Os elementos da árvore da Figura 21 estão definidos como classes na ontologia. Na ontologia do protótipo, definimos a superclasse *Pessoa* e as suas subclasses *Corretor* e *Cliente*. Além da relação de classe e sub-classe, uma ontologia pode definir outros relacionamentos. Por exemplo, a relação *Cliente tem-emprego Empresa* pode ser perfeitamente modelada em uma ontologia. O pacote JXML não gera árvores que modelem este tipo de relacionamento, pois, está limitado a relações classe e sub-classe, o que representa uma limitação em um domínio de problema como o nosso. Mas, se todos os bancos de dados envolvidos são relacionais e cada relação é considerada uma classe na ontologia, não haverá prejuízo no potencial de formular consultas.

O desenvolvimento de uma interface mais completa para o protótipo deverá ser realizado em trabalhos futuros, com a extensão do pacote JXML ou com o desenvolvimento de um novo pacote para a geração da representação gráfica da ontologia, de forma a permitir a criação de consultas mais elaboradas.

4.2 Estrutura de classes do protótipo

Nesta seção, apresentamos a estrutura de classes do protótipo. O diagrama de classes e agentes pode ser visto na Figura 24. Com base neste diagrama, descrevemos a estrutura do nosso protótipo.

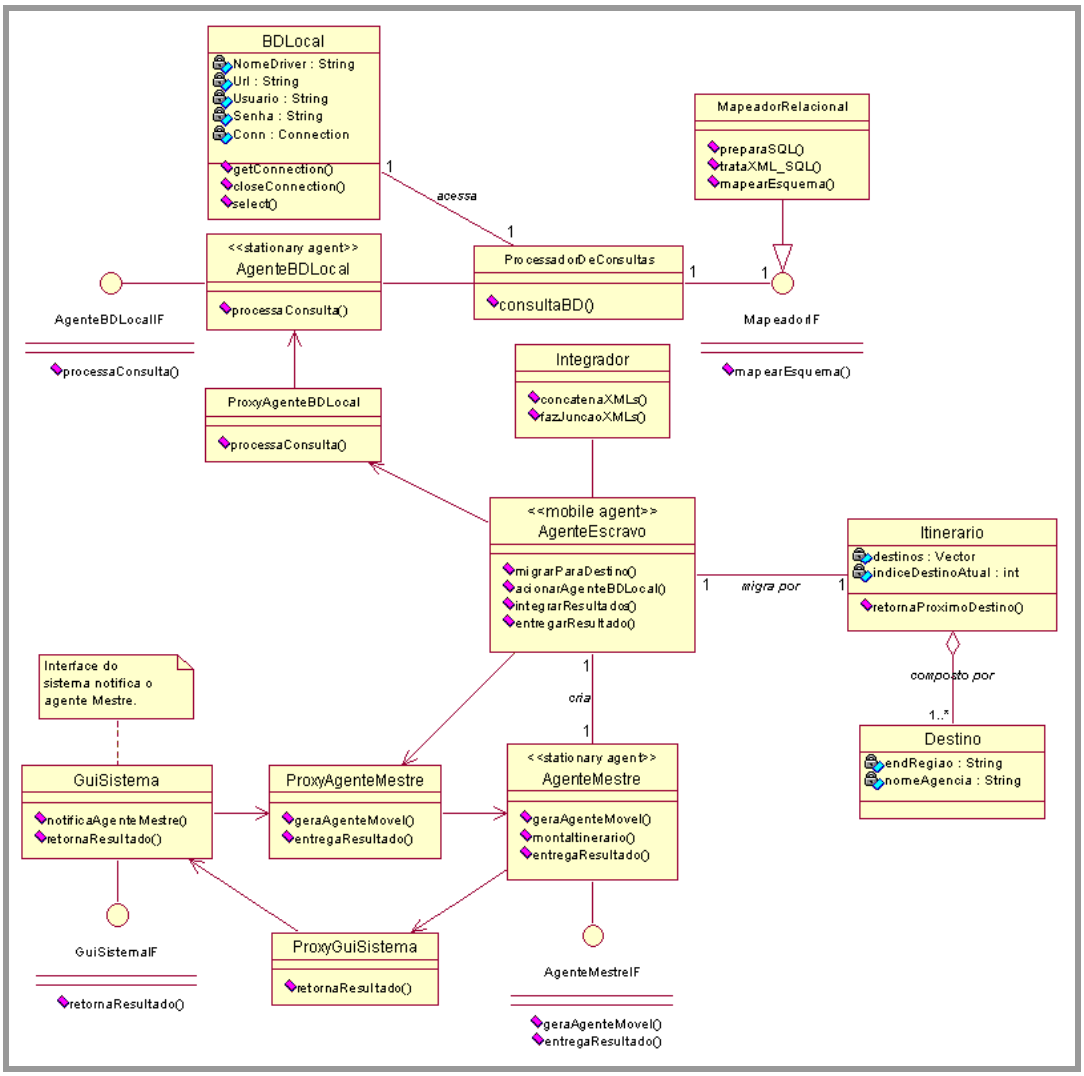

Figura 24: Diagrama de classes do protótipo

Como descrevemos na seção 2.5.5, a plataforma Grasshopper determina o uso do padrão Proxy na comunicação entre agentes. Este padrão consiste em uma entidade intermediária entre um cliente e um servidor, a qual é responsável por localizar o servidor e estabelecer uma conexão com o mesmo. Assim, para estabelecer uma conexão com um servidor, um cliente cria um objeto *proxy* que corresponde ao servidor.

No nosso protótipo, limitamos a implementação para bancos de dados do modelo relacional. A interface ***MapeadorIF*** permite que futuramente sejam incluídas novas classes, que tratem dos modelos objeto-relacional e orientado a objeto.

Quando o usuário submete a consulta, é instanciada a classe ***GuiSistema***, que é responsável por iniciar o processo de integração dos bancos de dados da federação. Esta classe, através do método *notificaAgenteMestre()*, cria um objeto *proxy*, denominado

ProxyAgenteMestre, correspondente à classe ***AgenteMestre***. Com isso, é invocado o método *geraAgenteMovel()* da classe ***AgenteMestre***, sendo passada a nova consulta como argumento.

O método *geraAgenteMovel()* cria um agente móvel denominado ***AgenteEscravo***, passando-lhe a consulta do usuário, um itinerário para o processo de migração, e um identificador de tipo do itinerário. O identificador de tipo do itinerário indica ao agente móvel a classificação do itinerário segundo o critério apresentado na Tabela 4. O itinerário do agente é preparado pelo ***AgenteMestre***, através do método *montaItinerario()*, seguindo o critério descrito na seção 3.4. Uma vez criado, o ***AgenteEscravo*** instancia a classe ***Itinerario***, passando-lhe sua lista de destinos. A classe ***Itinerario*** irá acompanhar o ***AgenteEscravo*** na sua migração.

Através do método *retornaProximoDestino()* da classe ***Itinerario***, o ***AgenteEscravo*** pega o endereço do próximo destino que precisa visitar. O ***AgenteEscravo*** migra para um destino através do método *migrarParaDestino()*. Ao alcançar um destino, o ***AgenteEscravo*** aciona um agente estacionário, que executa localmente, denominado ***AgenteBDLocal***. O acionamento ocorre através do método *acionarAgenteBDLocal()*, que cria um objeto *proxy*, denominado ***ProxyAgenteBDLocal***, correspondente à classe ***AgenteBDLocal***. Com isso, o método *processaConsulta()* da classe ***AgenteBDLocal*** é invocado, sendo passada a consulta como argumento.

O método *processaConsulta()* instancia um ***ProcessadorDeConsultas*** passando a consulta do usuário para ser adaptada ao esquema do banco de dados local. Ao instanciar um ***ProcessadorDeConsultas***, instancia-se também um ***BDLocal***, obtendo-se uma conexão com o banco de dados local, necessária ao processo de adaptação da consulta, e posterior extração dos dados.

Com isso, o método *consultaBD()* instancia um ***MapeadorRelacional***. Esta classe tem como primeira tarefa adaptar a consulta de forma que seja válida localmente. A segunda tarefa da classe é estruturar a consulta sintaticamente. Por fim, a classe deverá receber os dados extraídos e estruturá-los no formato XML.

A adaptação local da consulta é realizada através do método *mapearEsquema()*. O mecanismo de mapeamento é detalhado na seção 3.7. Após ter sido mapeada, a consulta é estruturada sintaticamente pelo método *preparaSQL()*. O método *trataXML_SQL()* modela os dados extraídos do banco de dados na sintaxe XML.

O resultado obtido localmente é entregue ao agente móvel (***AgenteEscravo***). Neste instante, o agente móvel, caso já tenha visitado pelo menos um destino, conterá o resultado local e o resultado obtido anteriormente. A integração destes resultados, ambos no formato

XML, é realizada pela classe ***Integrador***, que é instanciada pelo método *integrarResultados()* da classe ***AgenteEscravo***. A classe ***Integrador*** possui dois métodos: *concatenaXMLs()* e *fazJuncaoXMLs()*. Na integração, apenas um destes métodos é invocado. A escolha de qual método invocar é realizada pelo agente móvel, através do método *integrarResultados()*. O critério de escolha está baseado na consulta do usuário.

Após a integração dos resultados, o agente-escravo analisa o resultado obtido. Baseado neste resultado e na classificação do itinerário (estático ou dinâmico), o agente decide se deve ou não retornar à origem com o resultado atual. Esta análise do resultado está embutida no método *integrarResultados()* da classe ***AgenteEscravo***.

Uma vez concluído o itinerário do ***AgenteEscravo***, este então retorna à sua origem, ou seja, retorna para o local onde está o ***AgenteMestre***. Chegando na origem, o agente-escravo entrega o resultado final, previamente integrado, ao agente-mestre. A entrega do resultado ocorre através do método *entregarResultado()* do ***AgenteEscravo***. Este método cria um objeto *proxy*, denominado ***ProxyAgenteMestre***, que corresponde à classe ***AgenteMestre***. Com isso, o método *entregaResultado()* da classe ***AgenteMestre*** é invocado. Por sua vez, este método cria um objeto *proxy*, denominado ***ProxyGuiSistema***, que corresponde à classe ***GuiSistema***. Após isso, é invocado o método *retornaResultado()* da classe ***GuiSistema***, que finaliza o processo de integração, apresentando o resultado final ao usuário.

4.3 Os agentes do protótipo

Com a implementação do protótipo foram gerados três pacotes. O primeiro pacote contém a classe *AgenteMestre* e a interface *AgenteMestreIF*. O esforço de implementação deste pacote resultou em um total de 170 linhas de código.

O segundo pacote contém a classe *AgenteEscravo*, que representa o agente móvel do protótipo, responsável por migrar pelos bancos de dados. O pacote também contém outras classes e interfaces importantes para a execução do *AgenteEscravo*. O esforço de implementação deste pacote resultou em um total de 644 linhas de código.

O último pacote contém a classe *AgenteBDLocal*, que representa o agente estacionário local do nosso protótipo, responsável pelo tratamento local da consulta em cada banco de dados. Além desta classe, este pacote também contém todas as classes e interfaces necessárias para a realização das tarefas do agente. O esforço de implementação deste pacote resultou em um total de 667 linhas de código.

4.4 A infra-estrutura de execução dos agentes

Para podermos executar o protótipo, montamos uma infra-estrutura capaz de proporcionar um ambiente propício à realização de testes, contendo os seguintes componentes:

- Um servidor web para disponibilizar a interface;
- Uma agência Grasshopper executando o agente-mestre do sistema, denominado *AgenteMestre*;
- Três computadores remotos, cada um executando um SGBDR de algum fabricante, junto com uma agência Grasshopper executando um agente estacionário local, denominado *AgenteBDLocal*.

O primeiro computador preparado pertence ao Laboratório de Sistemas de Informação e Bancos de Dados (LSI), do Departamento de Sistemas e Computação (DSC), da Universidade Federal de Campina Grande (UFCG). Este computador possui um SGBD Oracle 9i do fabricante Oracle, com esquema local conforme ilustrado na Figura 26.

O segundo computador preparado pertence ao Poligene (Centro Softex Genesis em Campina Grande). É preciso salientar que a rede de computadores do Poligene encontra-se sob o domínio da UFCG. Este computador possui um SGBD Interbase do fabricante Borland, com esquema local conforme ilustrado na Figura 27.

O terceiro computador preparado pertence ao laboratório do PCT (Programa de Capacitação Tecnológica). A rede de computadores do PCT também encontra-se sob o domínio da UFCG. Este computador possui um SGBD SQL Server do fabricante Microsoft, com esquema local conforme ilustrado na Figura 28.

Cada um desses computadores executa uma agência da plataforma Grasshopper. Cada agência abriga um agente denominado *AgenteBDLocal*, responsável pelo tratamento local da consulta do usuário.

4.5 O esquema global

O protótipo que desenvolvemos promove o acesso a uma federação composta por três bancos de dados, citados na seção anterior. O esquema global identifica os objetos destes

bancos de dados que representam conceitos equivalentes, proporcionando uma visão integrada dos seus esquemas. O esquema global da federação está ilustrado na Figura 25.

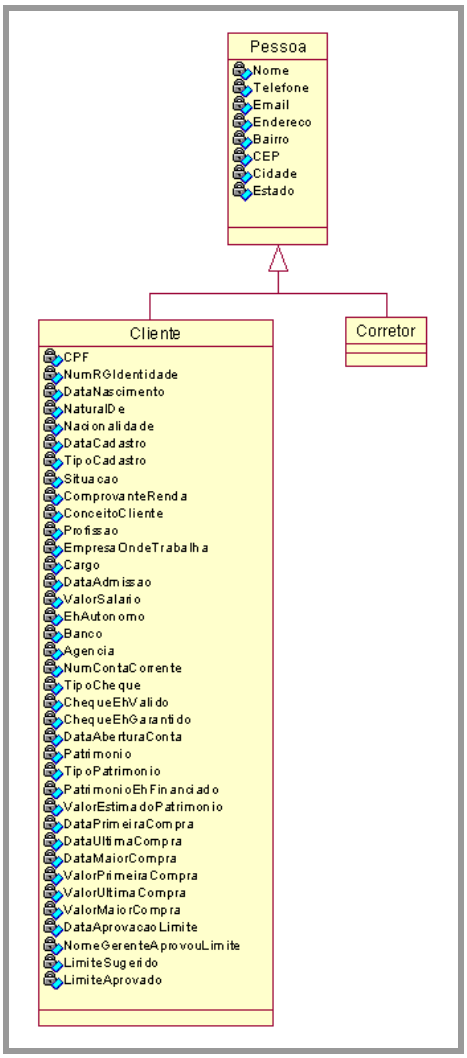

Figura 25: O esquema global

Na nossa arquitetura, o esquema global de uma federação de bancos de dados é definido como uma ontologia. Para o protótipo, utilizamos a linguagem DAML+OIL para a criação da ontologia, que está apresentada no Apêndice B desta dissertação.

4.6 Os esquemas dos bancos de dados

Como citamos na seção 4.4, preparamos três bancos de dados para a execução do protótipo. Nesta seção, apresentamos o esquema local de cada um deles, a fim de demonstrar as heterogeneidades existentes.

A Figura 26 ilustra o esquema do banco de dados Oracle.

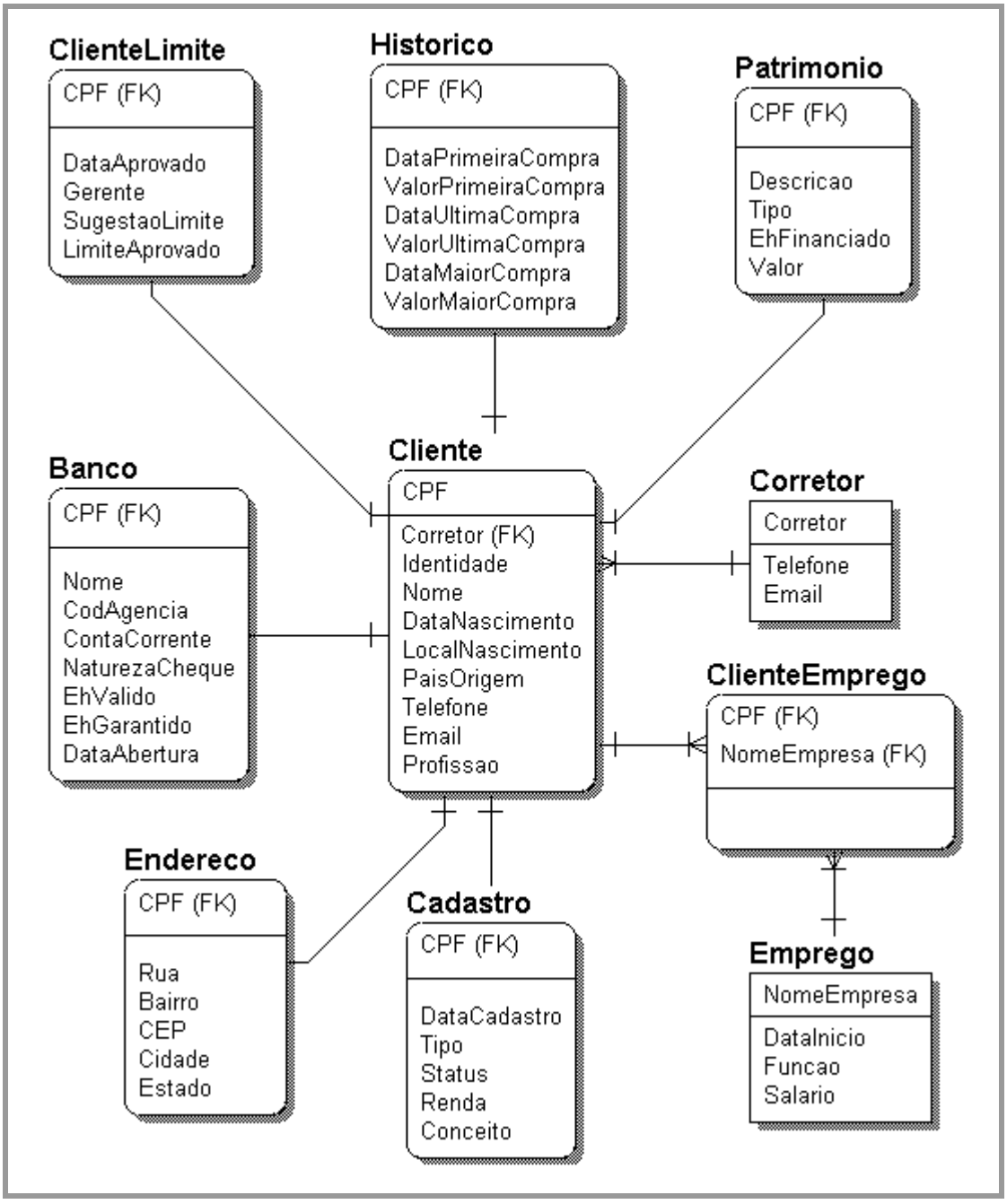

Figura 26: Esquema relacional do SGBD Oracle

Esquema relacional do SGBD Oracle
- Classes ontológicas não mapeadas: Pessoa - Atributos ontológicos não mapeados: EhAutonomo

A Figura 27 ilustra o esquema do banco de dados Interbase.

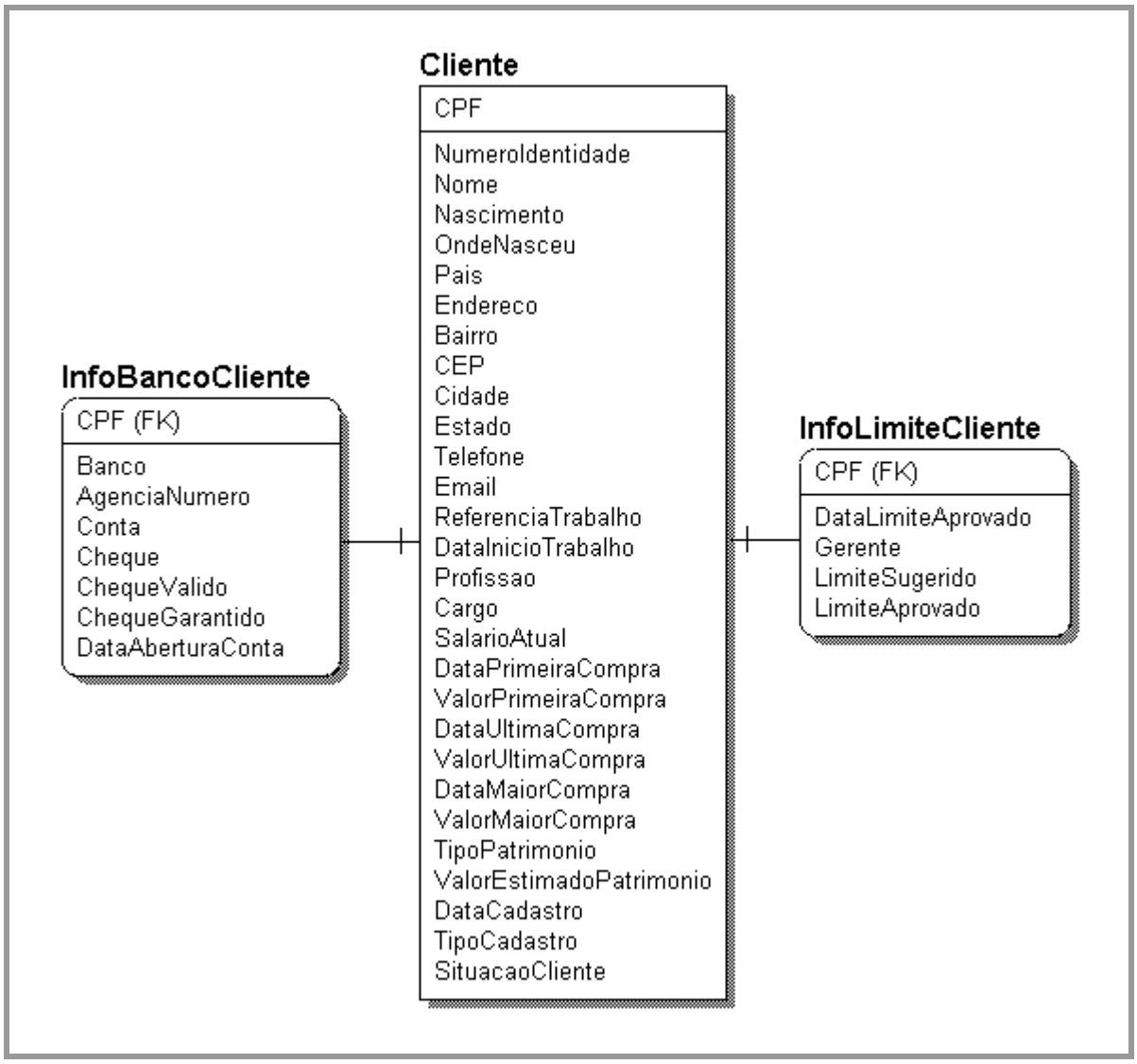

Figura 27: Esquema relacional do SGBD Interbase

Esquema relacional do SGBD Interbase

- Classes ontológicas não mapeadas: Pessoa e Corretor
- Atributos ontológicos não mapeados:
  - ComprovanteRenda
  - ConceitoCliente
  - EhAutonomo
  - Patrimonio
  - PatrimonioEhFinanciado

A Figura 28 ilustra o esquema do banco de dados SQL Server.

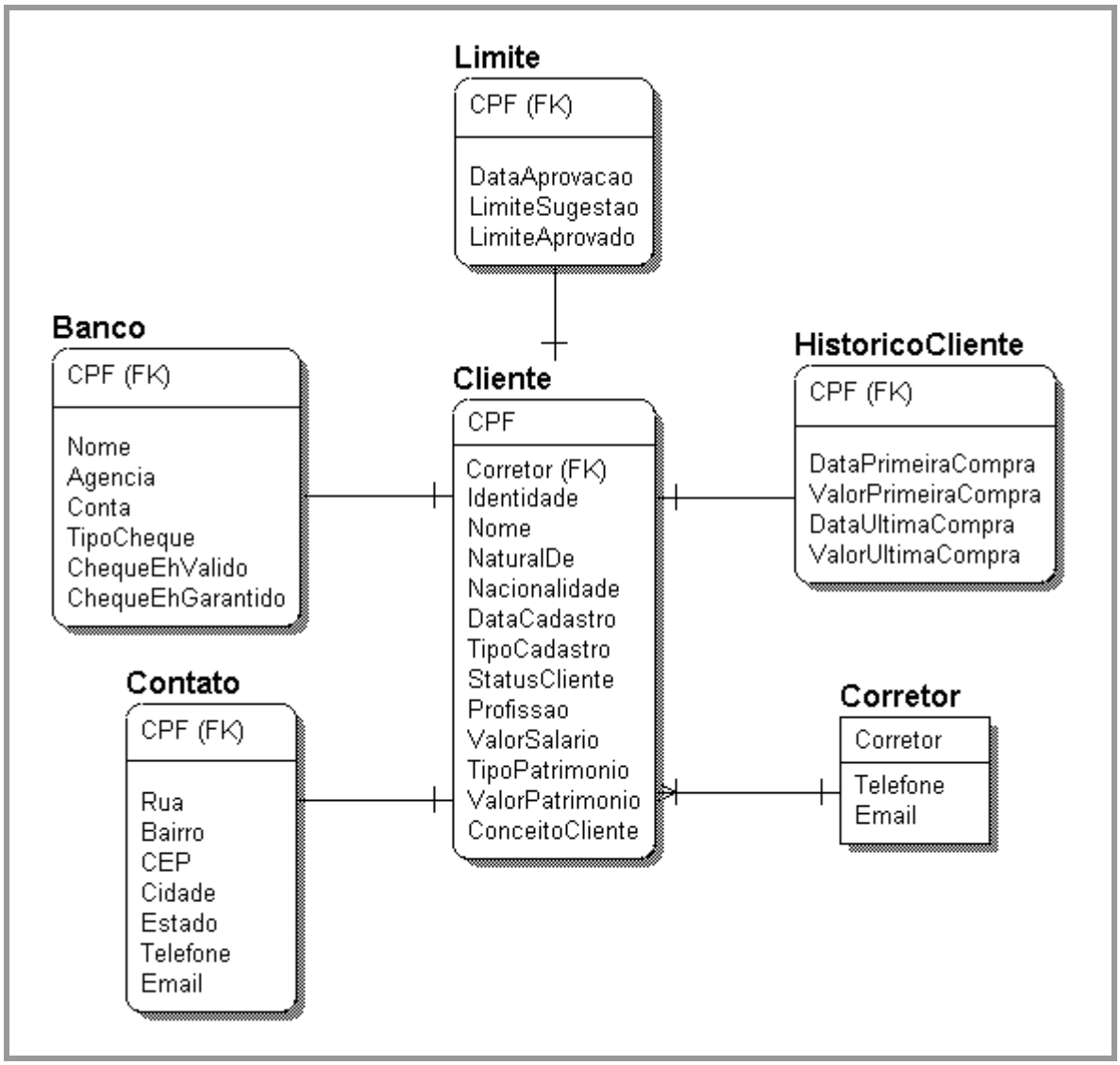

Figura 28: Esquema relacional do SGBD SQL Server

Esquema relacional do SGBD SQL Server

- Classes ontológicas não mapeadas: Pessoa
- Atributos ontológicos não mapeados:
  - ComprovanteRenda
  - DataNascimento
  - EmpresaOndeTrabalha
  - EhAutonomo
  - DataAdmissao
  - Cargo
  - DataAberturaConta
  - Patrimonio
  - PatrimonioEhFinanciado

DataMaiorCompra ValorMaiorCompra NomeGerenteAprovouLimite

Abaixo, listamos dois exemplos de consultas no esquema global:

Consulta 1	Select * from Cliente where LimiteSugerido > 2000
Consulta 2	Select * from Cliente where ValorSalario > 220,00

A seguir apresentamos tuplas da tabela de correspondência do SGBD Oracle para a adaptação de cada consulta:

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	LimiteSugerido	Cliente x, ClienteLimite c	c.SugestaoLimite	c.CPF = x.CPF

Tabela 13: Tupla do SGBD Oracle para adaptação da Consulta 1

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	ValorSalario	Cliente x, Emprego e, ClienteEmprego ce	e.Salario	x.CPF = ce.CPF AND e.NomeEmpresa = ce.NomeEmpresa

Tabela 14: Tupla do SGBD Oracle para adaptação da Consulta 2

A seguir listamos os dois exemplos de consultas, agora mapeadas para o esquema relacional da Figura 26:

Consulta 1 (Oracle)	Select * from Cliente x, ClienteLimite c where c.SugestaoLimite > 2000 AND c.CPF = x.CPF
Consulta 2 (Oracle)	Select * from Cliente x, Emprego e, ClienteEmprego ce where e.Salario > 220,00 AND x.CPF = ce.CPF AND e.NomeEmpresa = ce.NomeEmpresa

A seguir apresentamos tuplas da tabela de correspondência do SGBD Interbase para a adaptação de cada consulta:

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	LimiteSugerido	Cliente x, InfoLimiteCliente c	c. LimiteSugerido	c.CPF = x.CPF

Tabela 15: Tupla do SGBD Interbase para adaptação da Consulta 1

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	ValorSalario	Cliente	SalarioAtual	null

Tabela 16: Tupla do SGBD Interbase para adaptação da Consulta 2

A seguir listamos os dois exemplos de consultas, agora mapeadas para o esquema relacional da Figura 27:

Consulta 1 (Interbase)	Select * from Cliente x, InfoLimiteCliente c where c.LimiteSugerido > 2000 AND c.CPF = x.CPF
Consulta 2 (Interbase)	Select * from Cliente where SalarioAtual > 220,00

A seguir apresentamos tuplas da tabela de correspondência do SGBD SQL Server para a adaptação de cada consulta:

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	LimiteSugerido	Cliente x, Limite c	c.LimiteSugestao	c.CPF = x.CPF

Tabela 17: Tupla do SGBD SQL Server para adaptação da Consulta 1

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	ValorSalario	Cliente	ValorSalario	null

Tabela 18: Tupla do SGBD SQL Server para adaptação da Consulta 2

A seguir listamos os dois exemplos de consultas, agora mapeadas para o esquema relacional da Figura 28:

Consulta 1 (SQL Server)	Select * from Cliente x, Limite c where c.LimiteSugestao > 2000 AND c.CPF = x.CPF
Consulta 2 (SQL Server)	Select * from Cliente where ValorSalario > 220,00

4.7 Incluindo um novo banco de dados

A inclusão de um novo banco de dados na infra-estrutura de execução do protótipo, pressupõe a existência de um esquema do novo banco de dados. Dessa forma, o DBA local entrega este novo esquema para o DBA global, responsável pelo ingresso de novos bancos de dados na federação. O primeiro passo de inclusão é a identificação de objetos equivalentes a classes ou propriedades descritas na ontologia. Se existir um objeto no novo banco de dados que não pode ser relacionado a qualquer classe ou propriedade da ontologia, a ontologia deve ser alterada, pelo DBA global, de forma a expressar a semântica deste objeto.

Com isso, a tabela de correspondência local pode ser definida. Através desta tabela, o DBA global relaciona os objetos do novo banco de dados com as classes e propriedades definidas na ontologia. A estrutura da tabela de correspondência está descrita na seção 3.7.

O próximo passo é a instalação da agência *Grasshopper*, juntamente com a inclusão do agente estacionário *AgenteBDLocal*. Estes procedimentos são realizados pelo DBA local. Como citamos anteriormente, o agente *AgenteBDLocal* permanece executando localmente, sendo responsável pelo tratamento da consulta trazida pelo agente móvel.

Na inclusão de um novo banco de dados, nenhuma classe do protótipo precisa ser re-compilada. Cada agente *AgenteBDLocal*, executando junto aos diferentes bancos de dados, possui o mesmo código, independente do tipo de sistema operacional e SGBD utilizados. Para garantir esta portabilidade do código, os parâmetros necessários para a conexão com o banco de dados devem ser definidos em um arquivo externo. Este arquivo é lido pelo agente *AgenteBDLocal* no momento da abertura da conexão.

4.8 Executando o protótipo

Para finalizar este capítulo, apresentamos um exemplo completo de execução do protótipo, desde a criação da consulta até a apresentação do resultado final. Como exemplo, consideremos que o usuário, através da interface do protótipo, tenha escolhido a classe Cliente da ontologia, e tenha preenchido o formulário conforme ilustrado na Figura 29.

Figura 29: Exemplo de consulta do usuário

Para acionar o sistema de integração, o usuário submete os dados contidos no formulário da figura acima. As características da consulta do usuário estão descritas abaixo.

Consulta do usuário:

- Classe escolhida da ontologia: Cliente
- Atributos selecionados para projeção: DataCadastro e LimiteAprovado
- Operação de agregação: SUM (LimiteAprovado)
- Condições da consulta: Nome = ‘Andre’ e ConceitoCliente = ‘Regular’
- Atributo de agrupamento: DataCadastro
- Satisfeita por: Todos
- Fundir resultados: Sim

Consulta criada:

✓ Select DataCadastro, SUM(LimiteAprovado) from Cliente
where Nome = ‘Andre’ AND ConceitoCliente = ‘Regular’
GROUP BY DataCadastro

Uma vez submetida a consulta, o sistema notifica o *AgenteMestre*. Este agente conhece todos os esquemas locais dos bancos de dados distribuídos, e, portanto, é capaz de saber quais bancos de dados podem atender à consulta.

Neste exemplo, o banco de dados Interbase não pode atender à consulta, pois o seu esquema local, ilustrado na Figura 27, não possui qualquer correspondência ao atributo

ConceitoCliente da classe *Cliente* da ontologia. Assim, o itinerário para o processo de migração conterá os endereços das agências executando nos computadores que possuem os SGBDs Oracle e SQL Server, cujos esquemas podem atender à consulta do usuário.

Além de preparar o itinerário para o processo de migração, o *AgenteMestre* ainda classifica-o em *Estático* ou *Dinâmico*. Neste exemplo, a consulta possui uma operação de agregação e duas condições. Segundo a taxonomia de consultas apresentada na seção 3.5.1, uma consulta com essas características tanto sugere um itinerário estático como um dinâmico, e, portanto, a classificação do itinerário será determinada pelo parâmetro *Satisfeita por todos*, da consulta, o qual determina a classificação do itinerário como sendo *Estático*.

Com isso, o agente *AgenteMestre* cria o agente *AgenteEscravo*, passando-lhe a consulta do usuário, o seu itinerário, e um identificador de tipo do itinerário. Uma vez criado, o agente móvel migra para o primeiro destino (SGBD Oracle). Ao chegar no destino, este agente notifica um agente executando localmente, denominado *AgenteBDLocal*, entregando-lhe a consulta.

Analisando a tabela de correspondência local, o agente *AgenteBDLocal* mapeia a consulta do esquema global para o esquema relacional da Figura 26. Uma vez mapeada, a consulta local fica da seguinte forma:

- ✓ Select c.DataCadastro, SUM(l.LimiteAprovado)
  from Cliente x, Cadastro c, ClienteLimite l
  where x.Nome = ‘Andre’ AND c.Conceito = ‘Regular’
  AND c.CPF = x.CPF AND l.CPF = x.CPF
  GROUP BY c.DataCadastro

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	Nome	Cliente	Nome	null
Cliente	DataCadastro	Cliente x, Cadastro c	c.DataCadastro	c.CPF = x.CPF
Cliente	ConceitoCliente	Cliente x, Cadastro c	c.Conceito	c.CPF = x.CPF
Cliente	LimiteAprovado	Cliente x, ClienteLimite l	l.LimiteAprovado	l.CPF = x.CPF

Tabela 19: Tuplas para a adaptação da consulta no SGBD Oracle

Concluída a adaptação da consulta, o agente *AgenteBDLocal* executa a consulta no banco de dados local, e apresenta o resultado ao agente *AgenteEscravo* conforme ilustrado na Figura 30. Após isso, o agente *AgenteBDLocal* permanece ativo.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="DataCadastro" operacao="nenhuma">01/01/2002</coluna>
     <coluna nome="LimiteAprovado" operacao="SUM">800</coluna>
  </linha>
</ResultSet>
```

Figura 30: Resultado da consulta no SGBD Oracle

Como está ilustrado na Figura 30, a execução da consulta no SGBD Oracle resultou em uma tupla. Como este é o primeiro resultado obtido pelo agente *AgenteEscravo*, este não realiza qualquer integração de resultados. Com isso, o agente migra para o próximo destino contido no seu itinerário.

O próximo destino do agente móvel é o SGBD SQL Server. Ao chegar no destino, este agente realiza o mesmo conjunto de passos da migração anterior. O agente *AgenteBDLocal* então mapeia a consulta segundo a tabela de correspondência local. Uma vez mapeada, a consulta fica da seguinte forma:

- ✓ Select x.DataCadastro, SUM(l.LimiteAprovado)
  from Cliente x, Limite l
  where x.Nome = ‘Andre’ AND x.ConceitoCliente = ‘Regular’ AND l.CPF=x.CPF
  GROUP BY x.DataCadastro

Classe	Propriedade	Tabela	Atributo	Junção
Cliente	Nome	Cliente	Nome	null
Cliente	DataCadastro	Cliente	DataCadastro	null
Cliente	ConceitoCliente	Cliente	ConceitoCliente	null
Cliente	LimiteAprovado	Cliente x, Limite l	l.LimiteAprovado	l.CPF = x.CPF

Tabela 20: Tuplas para a adaptação da consulta no SGBD SQL Server

Concluído o processo de adaptação da consulta, o agente *AgenteBDLocal* executa a consulta no banco de dados local, e apresenta o resultado obtido ao agente *AgenteEscravo* segundo ilustrado na Figura 31.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="DataCadastro" operacao="nenhuma">01/01/2002</coluna>
     <coluna nome="LimiteAprovado" operacao="SUM">1000</coluna>
  </linha>
</ResultSet>
```

Figura 31: Resultado da consulta no SGBD SQL Server

Como está ilustrado na Figura 31, a execução da consulta no SGBD SQL Server resultou em uma tupla. Ao receber este resultado, o agente *AgenteEscravo* realiza a integração deste com o resultado obtido anteriormente. A forma com que os resultados são integrados é determinada pela consulta do usuário. Neste exemplo, o usuário optou pela junção dos resultados. O processo de junção, detalhada na seção 3.6, ocorre da seguinte forma:

- Cada linha do novo XML é comparada a todas as linhas do XML antigo;
    - Caso a linha corrente do novo XML corresponda a uma linha do XML antigo, estas são integradas conforme a operação de cada um dos seus atributos;
    - Caso a linha corrente do novo XML não corresponda a nenhuma linha do XML antigo, a linha corrente do novo XML é adicionada ao XML antigo.

No exemplo, o sistema integra o resultado obtido do SGBD SQL Server (novo XML) com o resultado obtido do SGBD Oracle (XML antigo). Neste exemplo, a linha do novo XML corresponde à linha do XML antigo, uma vez que, o valor do atributo-chave (DataCadastro) de cada XML é correspondente. Dessa forma, o XML resultante contém apenas uma linha, conforme ilustrado na Figura 32, resultante do processo de integração. No XML resultante, o atributo “LimiteAprovado” contém a soma dos valores do atributo em cada XML, conforme a operação “SUM” definida para este atributo.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<ResultSet>
  <linha>
     <coluna nome="DataCadastro" operacao="nenhuma">01/01/2002</coluna>
     <coluna nome="LimiteAprovado" operacao="SUM">1800</coluna>
  </linha>
</ResultSet>
```

Figura 32: Resultado da junção dos resultados

Como o agente *AgenteEscravo* alcançou o último destino do seu itinerário, este então retorna à origem levando consigo o resultado da Figura 32. Ao chegar na origem, o agente entrega o resultado ao agente *AgenteMestre*, e após isso, remove-se automaticamente.

Uma vez recebido o resultado do processo de integração, o agente *AgenteMestre* analisa o documento XML, e então apresenta o resultado ao usuário de uma forma adequada. Com isso, este agente permanece ativo, aguardando novas consultas.

Capítulo 5

Conclusão

Ao contrário das abordagens tradicionais de consultas na web, onde o universo de recursos pesquisados é altamente dinâmico e desconhecido para a máquina cliente, existe um grande número de aplicações complexas que possuem um ambiente bastante diferente, como corporações multinacionais, governos, e cadeias de lojas. Estes ambientes possuem um conjunto de bancos de dados bem conhecidos que mantêm informações relacionadas. Em um ambiente deste tipo, conhecido como Banco de Dados Federado, o conhecimento das informações armazenadas e a relativa estabilidade dos membros da federação podem ser usados para melhorar o processamento de consultas globais.

Nesta dissertação, apresentamos uma solução de integração, denominada DIA, para um ambiente de banco de dados federado, através do uso de uma ontologia para informar ao usuário quais informações estão armazenadas na federação, e para ajudar o sistema a encontrar os resultados de uma consulta específica. O processamento da consulta é realizado por um agente móvel, o qual após receber a tarefa de integração através de um agente estacionário mestre, percorre a federação de forma a requisitar os dados desejados a partir de agentes estacionários locais. Os resultados são integrados localmente, em cada banco de dados, e após cada processamento local, o agente móvel decide se pode retornar à origem ou se deve continuar percorrendo os membros da federação. Esta proposta de integração é aparentemente melhor do que outras propostas, como aquelas baseadas em mediadores, uma vez que diminui o fluxo de dados e a quantidade de comunicação na rede.

5.1 Trabalhos relacionados

Atualmente, existem muitos trabalhos que tratam da integração de dados usando ontologias. Apesar disso, nas nossas pesquisas, não encontramos trabalhos aplicando ontologias e agentes móveis como solução para o problema da integração de dados. Dessa forma, podemos considerar nosso trabalho como pioneiro neste aspecto.

A seguir, apresentamos alguns trabalhos de pesquisa que se relacionam com o nosso trabalho:

- Observer (Ontology Based System Enhanced with Relationships for Vocabulary hEteRogeneity): Este trabalho apresenta uma abordagem de processamento de consultas para Sistemas de Informação Globais (GIS), utilizando várias ontologias pré-existentes em repositórios de dados. Uma ou mais ontologias descrevem cada repositório através do uso de Description Logics (DLs), os quais são traduzidos para as linguagens locais de consulta dos repositórios (Mena *et al*, 1996);
- MEDIWeb (A Mediator-Based Environment for Data Integration on the Web): Este trabalho apresenta uma arquitetura de um sistema de consultas na web, no qual usuários, a partir do uso de uma ontologia, podem especificar e submeter consultas para um conjunto de fontes de dados. Estas fontes de dados tanto podem ser bancos de dados como arquivos XML (Arruda *et al*, 2002);
- Integration of Databases using the Mobility of the Code: Este trabalho aplica a mobilidade de código para extrair dados a partir de bancos de dados heterogêneos (Claro & Sobral, 2000).

Dentre os trabalhos citados, apenas o último aplica o paradigma de agentes móveis, porém, não há o uso de ontologias como base para o processo de integração. Para cada banco de dados a ser visitado, existe um agente estacionário responsável por realizar o mapeamento local da consulta. Este mapeamento está implementado na própria classe do agente, portanto, torna-se necessária a implementação de uma classe diferente para cada banco de dados componente. Além disso, ao ocorrer qualquer alteração no esquema local, a classe do agente precisa ser atualizada e re-compilada, acarretando em um grande trabalho de manutenção. Para cada consulta realizada, o agente móvel visita todos os bancos de dados componentes e a integração dos dados ocorre apenas no final do processo de extração, sobrecarregando o agente móvel, que leva consigo uma carga considerável de dados. Um aspecto bastante interessante deste trabalho é que, além de bancos de dados relacionais, também são tratados bancos de dados objeto-relacionais e orientados a objeto.

No nosso trabalho, o uso de uma ontologia na definição do esquema global da federação, somado ao mecanismo de mapeamento descrito na seção 3.7, tornou possível a geração de uma única classe de agente, totalmente portável, capaz de atender a qualquer banco de dados relacional. Como está descrito na seção 4.7, o processo de inclusão de um novo banco de dados na federação é bastante simples, sem necessidade de implementação de

uma classe específica para a nova fonte. Além disso, ao ocorrer uma mudança no esquema local, não é preciso re-compilar qualquer classe, bastando apenas atualizar a tabela de correspondência local (descrita na seção 3.7), de acordo com a descrição do esquema global definido na ontologia. No nosso sistema, o agente móvel visita apenas as fontes que possuem dados que satisfazem a consulta do usuário, evitando assim, que este visite locais desnecessariamente. Após o processo de integração dos dados, que ocorre diretamente nas fontes, o agente móvel é capaz de decidir se pode retornar à origem ou se deve migrar para o próximo banco de dados. Dessa forma, garantimos um melhor desempenho, pois, evitamos que o agente móvel carregue uma quantidade excessiva de dados.

5.2 Trabalhos futuros

A seguir, citamos alguns trabalhos que podem ser desenvolvidos a partir do DIA:

- Desenvolver um modelo de interface que solucione as restrições descritas na seção 4.1.1, de forma a permitir a criação de consultas mais elaboradas. Isso deve ser feito a partir da extensão do pacote JXML ou com o desenvolvimento de um novo pacote para a geração da representação gráfica da ontologia;
- Estender a estrutura da ontologia, permitindo a modelagem de relacionamentos e definição de regras associadas aos conceitos;
- Desenvolver uma GUI para dar suporte à criação e atualização da tabela de correspondência global-local;
- Incorporar novos bancos de dados na federação;
- Introduzir as demais camadas da arquitetura da Web Semântica, incluindo assim, regras que possibilitem aos agentes obterem resultados ainda melhores;
- Realizar uma análise do desempenho do sistema proposto. Para isso, sugerimos uma análise comparativa entre o DIA e o sistema MEDIWeb (nas bases relacionais);
- Estender o mecanismo de mapeamento, descrito na seção 3.7, de forma a suportar bancos de dados objeto-relacionais e orientados a objeto.

Referências bibliográficas

(Afonso, 2001) Afonso, M. M. R., "**Semantic Web**", World Wide Web. Disponível em: http://www.fe.up.pt/~mgi00014/ari/SW.doc

(Altmann *et al*, 2001) Altmann, J., Gruber, F., Klug, L., Stockner, W., Weippl, E., "**Using Mobile Agents in Real World: A Survey and Evaluation of Agent Platforms**". In: 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS at the 5th International Conference on Autonomous Agents, Canada, 2001.

(Arruda *et al*, 2002) Arruda, L. S., Baptista, C. S., Lima, C. A., "**MEDIWeb (A Mediator-Based Environment for Data Integration on the Web**". In: ICEIS'2002, 4th International Conference on Enterprise Information Systems, ICEIS Press, 2002.

(Arruda, 2002) Arruda, L. S., "**MEDIWeb: Um Integrador Semântico de Dados na Web baseado em Mediador**", Dissertação (Mestrado), Universidade Federal de Campina Grande, UFCG - COPIN, Campina Grande, 2002.

(Belian & Salgado, 2002) Belian, R., Salgado, A. C., "**Integração de Informações em Saúde na Web: Uma Visão Tecnológica**". In: VIII Congresso Brasileiro de Informática em Saúde, Natal, Brasil, 2002.

(Bellifemine & Trucco, 2003) Bellifemine, F., Trucco, T., "**JADE**", World Wide Web, 2003. Disponível em: http://sharon.cselt.it/projects/jade

(Berners-Lee *et al*, 2001) Berners-Lee, T., Hendler, J., Lassila, O., "**The Semantic Web**", Scientific American, 2001. Disponível em: http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html

(Berners-Lee, 2000) Berners-Lee, T., "**Semantic Web on XML**". In: XML 2000 Conference, 2000. Disponível em: http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

(Boley, 2003) Boley, H., "**RuleML - The Rule Markup Initiative**", World Wide Web, 2003. Disponível em: http://www.dfki.uni-kl.de/ruleml

(Brickley & Guha, 2000) Brickley, D., Guha, R. V., "**Resource Description Framework (RDF) Schema Specification 1.0**", W3C Candidate Recommendation, 2000. Disponível em: http://www.w3.org/TR/2000/CR-rdf-schema-20000327

(Brickley & Guha, 2002) Brickley, D., Guha, R. V., "**RDF Vocabulary Description Language 1.0: RDF Schema**", W3C Working Draft, 2002. Disponível em: http://www.w3.org/TR/rdf-schema

(Claro & Sobral, 2000) Claro, D. B., Sobral, J. B., "**Integration of Databases using the Mobility of the Code**". In: Workshop on Agent-Based Simulation, Passau, Alemanha, 2000, p. 227-232.

(Connolly, 2000) Connolly, D., "**Knowledge Interchange Format (KIF) as an RDF Schema**", World Wide Web, 2000.
Disponível em: http://www.w3.org/2000/07/hs78/KIF

(Decker *et al*, 2000) Decker, S., Melnik, S., Harmelen, F. V., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., Horrocks, I., "**The Semantic Web: The Roles of XML and RDF**", IEEE Internet Computing, 2000, vol. 4, nº. 5, p. 63-74.

(Douglis *et al*, 1999) Douglis, F., Milojicic, D. S., Wheeler, R., "**Mobility: Processes, Computers, and Agents**", Addison-Wesley, EUA, 1999.

(Fensel *et al*, 2000) Fensel, D., Horrocks, I., Harmelen, F., Decker, S., Erdmann, M., Klein, M., "**OIL in a Nutshell**", Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management, 2000.

(Friedman-Hill, 2003) Friedman-Hill, E., "**Jess, the Expert System Shell for the Java Platform**", World Wide Web, 2003. Disponível em: http://herzberg.ca.sandia.gov/jess

(Garcia-Molina *et al*, 2001) Garcia-Molina, H., Ullman, J. D., Widom, J., "**Database Systems: The Complete Book**", Prentice-Hall, EUA, 2001.

(García-Solaco *et al*, 1996) García-Solaco, M., Saltor, F., Castellanos, M., "**Semantic Heterogeneity in Multidatabase Systems**". In: Bukhres, O. A., Elmagarmid, A. K., Object-Oriented Multidatabase Systems: A Solution for Advanced Applications, Prentice-Hall, EUA, 1996, p. 129-202.

(Goldschmidt *et al*, 2002) Goldschmidt, B., Laszlo, Z., Doller, M., Kosch, H., "**Mobile Agents in a Distributed Heterogeneous Database System**", 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP 2002), Ilhas Canárias, Espanha, 2002.

(Gómez-Pérez & Corcho, 2002) Gómez-Pérez, A., Corcho, O., "**Ontology Specification Languages for the Semantic Web**", IEEE Intelligent Systems, 2002, vol. 17, nº. 1, p. 54-60.

(Gruber, 1993) Gruber, T., "**What is an Ontology?**", World Wide Web.
Disponível em: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

(Guarino, 1998) Guarino, N., "**Formal Ontology and Information Systems**", Proceedings of FOIS'98, Trento, Itália, 1998.

(Guedes & Machado, 2001) Guedes, F. P., Machado, P. D. L., "**Um Modelo para o Desenvolvimento de Aplicações Baseadas em Agentes Móveis**". In: Anais do VI Worshop de Teses em Engenharia de Software (WTES), Rio de Janeiro, 2001, p. 31-34.

(Hendler *et al,* 2002) Hendler, J., Dean, M., Horrocks, I., Connolly, D., Harmelen, F., McGuinness, D., Patel-Schneider, P. F., Stein, L. A., "**Web Ontology Language (OWL) Reference Version 1.0**", W3C Working Draft, 2002.
Disponível em: http://www.w3.org/TR/owl-ref/#ref-guide

(Horrocks *et al,* 2001a) Horrocks, I., Connolly, D., Harmelen, F., McGuinness, D., Patel-Schneider, P. F., Stein, L. A., "**DAML+OIL Reference Description**", W3C Note, 2001. Disponível em: http://www.w3.org/TR/daml+oil-reference

(Horrocks *et al,* 2001b) Horrocks, I., Harmelen, F., Patel-Schneider, P. F., "**A Model-Theoretic Semantics for DAML+OIL**", W3C Note, 2001.
Disponível em: http://www.w3.org/TR/daml+oil-model

(Horrocks *et al,* 2001c) Horrocks, I., Connolly, D., Harmelen, F., McGuinness, D., Patel-Schneider, P. F., Stein, L. A., "**Annotated DAML+OIL Ontology Markup**", W3C Note, 2001. Disponível em: http://www.w3.org/TR/daml+oil-walkthru

(Hurson & Bright, 1996) Hurson, A. R., Bright, M. W., "**Object-Oriented Multidatabase Systems**". In: Bukhres, O. A., Elmagarmid, A. K., Object-Oriented Multidatabase Systems: A Solution for Advanced Applications, Prentice-Hall, EUA, 1996, p. 1-36.

(IKV++ GmbH, 1999) IKV++ GmbH, "**Grasshopper – A Platform for Mobile Software Agents**", 1999. Disponível em:
http://www.grasshopper.de/download/doc/GrasshopperIntroduction.pdf

(Jacobson *et al,* 1992) Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G., "**Object-Oriented Software Engineering: A Use-Case Approach**", Addison-Wesley, USA, 1992.

(Klein *et al,* 2001) Klein, C., Rausch, A., Sihling, M., Wen, Z., "**Extension of the Unified Modeling Language for Mobile Agents**". In: Siau, K., Halpin, T., Unified Modeling Language: Systems Analysis, Design and Development Issues, Idea Publishing Group, 2001, p. 116-128.

(Klein, 2001) Klein, M., "**XML, RDF, and Relatives**", IEEE Intelligent Systems, 2001, vol. 16, nº. 2, p. 26-28.

(Lange & Oshima, 1998) Lange, D. B., Oshima, M., "**Programming and Deploying Java Mobile Agents with Aglets**", Addison-Wesley, EUA, 1998.

(Lange & Oshima, 1999) Lange, D. B., Oshima, M., "**Mobile Agents with Java: The Aglet API**". In: Douglis, F., Milojicic D. S., Wheeler R., Mobility: Processes, Computers, and Agents, Addison-Wesley, EUA, 1999, p. 495-512.

(Larman, 2000) Larman, C., "**Utilizando UML e Padrões – Uma Introdução à Análise e ao Projeto Orientados a Objetos**", Bookman, Brasil, 2000.

(Lassila & Swick, 1999) Lassila, O., Swick, R. R., "**Resource Description Framework (RDF) Model and Syntax Specification**", W3C Recommendation, 1999.
Disponível em: http://www.w3.org/TR/REC-rdf-syntax

(Melnik & Decker, 2000) Melnik, S., Decker, S., "**A Layered Approach to Information Modeling and Interoperability on the Web**". In: ECDL 2000 Workshop on the Semantic Web, 2000.

(Mena *et al,* 1996) Mena, E., Kashyap, V., Sheth, A., Illarramendi, A., "**OBSERVER: An approach for Query Processing in Global Information Systems based on Interoperation across Preexisting Ontologies**". In: proceedings of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96), 1996.

(Noy & McGuinness, 2001) Noy, N. F., McGuinness, D., "**Ontology Development 101: A Guide to Creating Your First Ontology**", Knowledge Systems Laboratory, Stanford University, 2001.

(Staab *et al,* 2000) Staab, S., Erdmann, M., Maedche, A., Decker, S., "**An Extensible Approach for Modeling Ontologies in RDF(S)**". In: ECDL 2000 Workshop on the Semantic Web, 2000.

Apêndice A

A linguagem DAML+OIL

A DAML+OIL é uma linguagem semântica de marcação para recursos na web. A linguagem é construída sobre padrões W3C como RDF e RDF Schema, estendendo-os com primitivas de modelagem mais ricas. A DAML+OIL foi construída a partir da linguagem DAML, em uma tentativa de combinar os componentes dessa linguagem com os componentes da linguagem OIL (Fensel *et al*, 2000).

Uma base de conhecimento DAML+OIL é uma coleção de triplas RDF. A DAML+OIL prescreve um significado específico para tais triplas. Assim como com qualquer conjunto de triplas RDF, triplas DAML+OIL podem ser representadas em várias formas sintáticas diferentes. A especificação original da DAML+OIL, no entanto, é um tipo específico de marcação RDF, que por sua vez possui sintaxe XML.

(Horrocks *et al*, 2001b) especificam exatamente quais triplas possuem um significado designado, e o que cada significado representa. A DAML+OIL apenas proporciona uma interpretação semântica para aquelas partes de um grafo RDF que instanciam o esquema definido em (Horrocks *et al*, 2001a).

A linguagem DAML+OIL permite a declaração direta de classes e propriedades, conforme ilustrado na Figura 33:

```
<daml:Class rdf:ID="Produto">
        <rdfs:label>Produto</rdfs:label>
        <rdfs:comment>Um item que é vendido nas lojas da empresa.</rdfs:comment>
</daml:Class>

<daml:DatatypeProperty rdf:ID="numeroProduto">
        <rdfs:label>Número do Produto</rdfs:label>
        <rdfs:domain rdf:resource="#Produto"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</daml:DatatypeProperty>
```

Figura 33: Declarando classes e propriedades em DAML+OIL

Na Figura 34, ilustramos a criação de uma instância da classe descrita na Figura 33, através da definição de algum recurso como sendo do seu tipo, e fornecendo valores para as devidas propriedades.

```
<Produto rdf:ID="GarrafaTermica">
        <rdfs:label>Garrafa térmica</rdfs:label>
        <numeroProduto>38267</numeroProduto>
</Produto>
```

Figura 34: Definindo uma instância da classe Produto

No exemplo, definimos que a propriedade *numeroProduto* seria do tipo definido em "http://www.w3.org/2000/01/rdf-schema#Literal". Vejam que essa declaração é bastante restrita, uma vez que o tipo *Literal* pode ser qualquer string, incluindo aqueles que não são números. A DAML+OIL permite que valores sejam restringidos a tipos de dados do XML Schema ou a tipos de dados definidos pelo usuário.

```
<daml:DatatypeProperty rdf:ID="numeroProduto">
        <rdfs:label>Número do Produto</rdfs:label>
        <rdfs:domain rdf:resource="#Produto">
        <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</daml:DatatypeProperty>
```

Figura 35: Aplicando tipos de dados do XML Schema

Usamos o prefixo "daml" para representar o espaço de nomes da DAML+OIL (http://www.w3.org/2001/10/daml+oil#). A DAML+OIL estende o RDF Schema, adicionando primitivas, como a *DatatypeProperty*. Há algumas mudanças nas semânticas de *rdfs:domain* e *rdfs:range* em sistemas DAML+OIL. A mudança principal é o fato de uma propriedade poder ter múltiplos domínios (*domains*) e múltiplos alcances (*ranges*). Em DAML+OIL, qualquer propriedade que não é uma *daml:DatatypeProperty* é considerada uma *daml:ObjectProperty*, cujo alcance (range) deve ser uma classe definida em DAML+OIL ou em RDF Schema.

Em DAML+OIL, uma propriedade também poder ser definida como sendo idêntica a uma outra propriedade. Podemos fazer isso de duas maneiras: com a propriedade *daml:equivalentTo* ou com a propriedade *daml:samePropertyAs*. Suponha que a nossa empresa-exemplo tenha publicado seu sistema baseado em RDF, e alguma organização de comércio surja com um vocabulário padrão para a apresentação de produtos na web. Se este

vocabulário utiliza uma propriedade chamada *produtoID* para indicar o número do produto, a nossa empresa não precisará alterar todo o seu código para utilizar a nova propriedade. Com a DAML+OIL, a nossa empresa-exemplo simplesmente estenderá a definição de *numeroProduto* conforme ilustrado na Figura 36.

```
<rdf:Description about="#numeroProduto">
        <daml:samePropertyAs rdf:resource="http://www.padraoLojas.org/vocab#produtoID"/>
</rdf:Description>
```

Figura 36: Definindo uma propriedade como idêntica a outra propriedade

Também podemos especificar que uma dada propriedade é única, o que significa que haverá apenas um valor da propriedade para cada instância. Como exemplo, podemos definir que cada produto terá um único número. Para isso, devemos modificar a definição da propriedade da forma ilustrada na Figura 37.

```
<daml:DatatypeProperty rdf:ID="numeroProduto">
        <rdfs:label>Número do Produto</rdfs:label>
        <rdfs:domain rdf:resource="#Produto">
        <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
        <rdf:type rdf:resource="http://www.w3.org/2001/10/daml+oil#UniqueProperty"/>
</daml:DatatypeProperty>
```

Figura 37: Definindo a unicidade de propriedades

Estrutura da linguagem

Uma ontologia descrita em DAML+OIL consiste em vários componentes, dos quais alguns são opcionais e alguns podem ser repetidos. Para um melhor entendimento, apresentamos as construções da DAML+OIL em um formato estruturado, e não como triplas RDF. Em resumo, uma ontologia DAML+OIL consiste de zero ou mais cabeçalhos, seguido de zero ou mais classes, propriedades e instâncias.

Cabeçalho

Um elemento *daml:Ontology* contém zero ou mais informações de versão e elementos *import*. Veja a Figura 38.

```
<daml:Ontology rdf:about="">
        <daml:versionInfo>$Id: Exemplo, versão 1.0 2002</daml:versionInfo>
        <rdfs:comment>Um exemplo de ontologia </rdfs:comment>
        <daml:imports rdf:resource="http://www.w3.org/2001/10/daml+oil"/>
</daml:Ontology>
```

Figura 38: Cabeçalho de uma ontologia DAML+OIL

O elemento *daml:versionInfo* geralmente contém texto com informação sobre a versão da ontologia. Esse elemento não contribui para o significado lógico da ontologia.

Cada declaração *daml:imports* referencia outra ontologia DAML+OIL contendo definições que se aplicam à nova ontologia. Cada referência consiste de um URI especificando de onde a ontologia deverá ser importada. Declarações de importação são transitivas, ou seja, se uma ontologia A importa B, e B importa C, então A importa B e C.

Valores Object e Datatype

A linguagem DAML+OIL divide o universo em duas partes disjuntas. Uma parte consiste nos valores que pertencem aos tipos definidos no XML Schema, chamada de *datatype domain*. A outra parte consiste em objetos que são considerados membros de classes descritas em DAML+OIL ou RDF, chamada de *object domain*.

A DAML+OIL está mais voltada para a criação de classes que fazem parte do *object domain*. Tais classes são chamadas *classes-objeto* e são elementos de *daml:Class*, uma subclasse de *rdfs:Class*. A DAML+OIL também permite o uso de tipos definidos pelo XML Schema para descrever parte do *datatype domain*. Esses tipos de dados são usados na DAML+OIL através da inclusão de seus URIs na ontologia.

Classes

Um elemento *daml:Class* contém a definição de uma classe objeto. Um elemento deste tipo referencia o nome da classe e contém:

- zero ou mais elementos *rdfs:subClassOf* (cada um contém uma expressão de classe). Cada elemento *daml:subClassOf* afirma que a classe sendo definida é uma subclasse da classe contida na expressão de classe;
- zero ou mais elementos *daml:disjointWith* (cada um contém uma expressão de classe). Cada elemento *daml:disjointWith* afirma que a classe sendo definida não possui instâncias em comum com a expressão de classe no elemento;
- zero ou mais elementos *daml:disjointUnionOf* (cada um contém uma expressão de classe). Cada elemento *daml:disjointUnionOf* afirma que a classe sendo definida possui as mesmas instâncias que a união disjunta da expressão de classe no elemento;
- zero ou mais elementos *daml:sameClassAs* (cada um contém uma expressão de classe). Cada elemento *daml:sameClassAs* afirma que a classe sendo definida é equivalente à expressão de classe no elemento (terão as mesmas instâncias);
- zero ou mais elementos *daml:equivalentTo* (cada um contendo uma expressão de classe). Quando aplicado a uma classe, esse elemento possui a mesma semântica que *daml:sameClassAs*;
- zero ou mais combinações *booleanas* de expressões de classe. A classe sendo definida deverá ser equivalente à classe definida pela combinação das expressões booleanas;
- zero ou mais elementos de enumeração. Cada elemento de enumeração afirma que a classe sendo definida contém exatamente as instâncias enumeradas no elemento.

Expressões de classe

Uma expressão de classe pode ser:

- o nome de uma classe;
- uma enumeração, definida entre as tags <daml:Class>. . .</daml:Class>;
- uma restrição de propriedade;
- uma combinação *booleana* de expressões de classe, definida entre tags <daml:Class>. . .</daml:Class>.

Cada expressão de classe pode fazer referência a uma classe nomeada ou definir implicitamente uma classe anônima. Dois nomes de classes já são pré-definidos, são eles: *daml:Thing* e *daml:Nothing*. Todo objeto é um membro de *daml:Thing*, e nenhum objeto é membro de *daml:Nothing*. Conseqüentemente, toda classe é uma subclasse de *daml:Thing* e *daml:Nothing* é uma subclasse de toda classe.

Enumerações

Uma enumeração é um elemento *daml:oneOf* contendo uma lista dos objetos que são suas instâncias. Isso permite definir uma classe pela exaustiva enumeração de seus elementos. A classe definida pelo elemento *daml:oneOf* contém exatamente os elementos enumerados. Veja a Figura 39.

```
<daml:oneOf parseType="daml:collection">
        <daml:Thing rdf:about="#Eurasia"/>
        <daml:Thing rdf:about="#Africa"/>
        <daml:Thing rdf:about="#South_America"/>
</daml:oneOf>
```

Figura 39: Definindo uma enumeração

Restringindo propriedades

Uma restrição de propriedade é um tipo especial de expressão de classe. Ela implicitamente define uma classe anônima, em outras palavras, a classe de todos os objetos que satisfazem à restrição. Existem dois tipos de restrição: *ObjectRestriction* e *DatatypeRestriction*. O primeiro tipo aplica-se em propriedades que relacionam objetos a outros objetos. O segundo tipo aplica-se em propriedades que relacionam objetos a tipos de valores. Ambos os tipos de restrição são criados usando a mesma sintaxe. A única diferença está na referência a um tipo de valor ou a uma classe.

Um elemento *daml:Restriction* contém um elemento *daml:onProperty*, o qual se refere a uma propriedade, e um ou mais dos seguintes elementos:

- elementos indicando o tipo da restrição:
    - um elemento *daml:toClass* (que contém uma expressão de classe). Um elemento *daml:toClass* define a classe de todos os objetos para a qual todos os valores da propriedade pertencem à expressão de classe. Em outras palavras, o elemento define a classe de objetos x onde se o par (x,y) é uma instância da propriedade, então y é uma instância da expressão de classe ou tipo de dado;
    - um elemento *daml:hasValue* (que contém uma referência a um objeto ou a um valor). Um elemento *daml:hasValue* define a classe de todos os objetos para a qual a propriedade possui pelo menos um valor igual ao objeto nomeado ou ao valor;
    - um elemento *daml:hasClass* (que contém uma expressão de classe ou uma referência a um valor). Um elemento *daml:hasClass* define a classe de todos os objetos para a qual pelo menos um valor da propriedade é um membro da expressão de classe ou ao valor;
- elementos contendo um inteiro não negativo (para o qual nos referimos como sendo N), indicando uma restrição de cardinalidade:
    - um elemento *daml:cardinality*. Esse elemento define a classe de todos os objetos que possuem exatamente N valores distintos para a propriedade;
    - um elemento *daml:maxCardinality*. Esse elemento define a classe de todos os objetos que possuem no máximo N valores distintos para a propriedade;
    - um elemento *daml:minCardinality*. Esse elemento define a classe de todos os objetos que possuem pelo menos N valores distintos para a propriedade.
- elementos contendo um inteiro não negativo (para o qual nos referimos como sendo N), indicando uma restrição de cardinalidade e um elemento *daml:hasClassQ*, contendo uma expressão de classe ou uma referência a um valor:
    - um elemento *daml:cardinalityQ*. Esse elemento define a classe de todos os objetos que possuem exatamente N valores distintos para a propriedade, e que são instâncias da expressão de classe ou tipo de dado. Em outras palavras, x é uma instância da classe definida (x satisfaz a restrição), se e somente se, existem exatamente N valores y distintos tal que (x,y) é uma instância da propriedade, e y é uma instância da expressão de classe ou tipo de dado;

- um elemento *daml:maxCardinalityQ*. Esse elemento define a classe de todos os objetos que possuem no máximo N valores distintos para a propriedade, que são instâncias da expressão de classe ou tipo de dado;
- um elemento *daml:minCardinality*. Esse elemento define a classe de todos os objetos que possuem pelo menos N valores distintos para a propriedade, que são instâncias da expressão de classe ou tipo de dado.

Perceba que uma restrição do tipo *daml:toClass* é análoga ao quantificador universal (para-todos) da lógica de predicados – para cada instância da classe ou tipo de dado sendo definido, todo valor da propriedade deve satisfazer a restrição. A restrição do tipo *daml:hasClass* é análoga ao quantificador existencial da lógica de predicados – para cada instância da classe ou tipo de dado sendo definido, existe pelo menos um valor para o predicado que satisfaz a restrição.

Combinação booleana de expressões de classe

Uma combinação booleana de expressões de classe pode ser construída a partir de:

- um elemento *daml:intersectionOf*, contendo uma lista de expressões de classe. Isso define a classe que consiste em exatamente todos os objetos comuns a todas as expressões de classe na lista;
- um elemento *daml:unionOf*, contendo uma lista de expressões de classe. Isso define a classe que consiste em exatamente todos os objetos que pertencem a pelo menos uma das expressões de classe na lista;
- um elemento *daml:complementOf*, contendo uma única expressão de classe. Isso define a classe que consiste em todos os objetos que não pertencem à expressão de classe.

Propriedades

Um elemento *rdf:Property* faz referência ao nome de uma propriedade. Propriedades que são usadas como restrição de propriedades podem ser de dois tipos: propriedades que relacionam objetos a outros objetos e são instâncias de *daml:ObjectProperty*; ou propriedades que relacionam objetos a tipos de dados, e são instâncias de *daml:DatatypeProperty*.

Um elemento *rdf:Property* contém:

- zero ou mais elementos *rdfs:subPropertyOf*, cada um contendo o nome de uma propriedade. Cada elemento *rdfs:subPropertyOf* afirma que a propriedade é uma sub-propriedade da propriedade nomeada no elemento;
- zero ou mais elementos *rdfs:domain*, cada um contendo uma expressão de classe. Cada elemento *rdfs:domain* afirma que a propriedade se aplica apenas a instâncias da expressão de classe definida no elemento;
- zero ou mais elementos *rdfs:range*, cada um contendo uma expressão de classe. Cada elemento *daml:range* afirma que a propriedade assume apenas valores que são instâncias da expressão de classe definida no elemento;
- zero ou mais elementos *daml:samePropertyAs*, cada um contendo o nome de uma propriedade. Cada elemento *daml:samePropertyAs* afirma que a propriedade é equivalente à propriedade nomeada no elemento;
- zero ou mais elementos *daml:equivalentTo*, cada um contendo o nome de uma propriedade. Quando aplicado a uma propriedade, o elemento *daml:equivalentTo* possui a mesma semântica do elemento *daml:samePropertyAs*;
- zero ou mais elementos *daml:inverseOf*, cada um contendo o nome de uma propriedade. Cada elemento *daml:inverseOf* afirma que a propriedade é a relação inversa da propriedade nomeada no elemento. Mais formalmente: se o par (x,y) é uma instância da propriedade, então o par (y,x) é uma instância da propriedade nomeada no elemento;
- um elemento *daml:TransitiveProperty*, que é uma subclasse de *ObjectProperty*. Por exemplo: se o par (x,y) é uma instância da propriedade, e o par (y,z) é uma instância da propriedade, então o par (x,z) é também uma instância da propriedade;
- um elemento *daml:UniqueProperty*. Esse elemento afirma que a propriedade poderá possuir apenas um único valor para cada instância;
- um elemento *daml:*UnambigousProperty, que é uma subclasse de *ObjectProperty*. Esse elemento afirma que uma instância y só pode ser o valor da propriedade para uma instância única x. Por exemplo: não podem existir duas instâncias distintas y1 e y2 tais que os pares (x,y1) e (x,y2) são ambos instâncias da propriedade.

Coleções

A linguagem DAML+OIL precisa representar coleções desordenadas de itens em um número de construções, como *intersectionOf*, *unionOf*, *oneOf* e *disjointUnionOf*. A DAML+OIL explora o atributo *rdf:parseType* para estender a sintaxe do RDF com uma notação conveniente para tais coleções. Toda vez que um elemento possui o atributo *rdf:parseType* com valor igual a *daml:collection*, os elementos listados devem ser interpretados como elementos em uma lista. Veja a Figura 40.

```
<daml:oneOf rdf:parseType="daml:collection">
        <daml:Thing rdf:about="#vermelho"/>
        <daml:Thing rdf:about="#branco"/>
</daml:oneOf>
```

Figura 40: Definindo uma coleção

Os parsers RDF atuais não suportam o *daml:collection parseType*. De forma a processar documentos DAML+OIL, tais parsers deverão ser estendidos, ou um pré-processamento separado é necessário para traduzir a forma inicial para uma nova, antes que o documento DAML+OIL seja entregue como entrada para o RDF parser.

Apêndice B

O esquema global da federação

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
         xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
         xmlns="http://www.dsc.ufpb.br/~philip/onto/GlobalOntology#">

<daml:Ontology rdf:about="">
        <daml:versionInfo>Ontologia Versão 1.0</daml:versionInfo>
        <daml:comment>Esquema global de um BD federado</daml:comment>
        <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil#"/>
</daml:Ontology>

<daml:Class rdf:ID="Pessoa">
        <rdfs:label>Pessoa</rdfs:label>
</daml:Class>

<daml:Class rdf:ID="Cliente">
        <rdfs:label>Cliente</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Pessoa"/>
</daml:Class>

<daml:Class rdf:ID="Corretor">
        <rdfs:label>Corretor</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Pessoa"/>
</daml:Class>

<rdf:Property rdf:ID="Nome">
        <rdfs:label>Nome</rdfs:label>
        <daml:domain rdf:resource="#Pessoa"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="Telefone">
        <rdfs:label>Telefone</rdfs:label>
        <daml:domain rdf:resource="#Pessoa"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="Email">
        <rdfs:label>Email</rdfs:label>
        <daml:domain rdf:resource="#Pessoa"/>
```

```
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="Endereco">
        <rdfs:label>Endereco</rdfs:label>
        <daml:domain rdf:resource="#Pessoa"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="Bairro">
        <rdfs:label>Bairro</rdfs:label>
        <daml:domain rdf:resource="#Pessoa"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="CEP">
        <rdfs:label>CEP</rdfs:label>
        <daml:domain rdf:resource="#Pessoa"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="Cidade">
        <rdfs:label>Cidade</rdfs:label>
        <daml:domain rdf:resource="#Pessoa"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="Estado">
        <rdfs:label>Estado</rdfs:label>
        <daml:domain rdf:resource="#Pessoa"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="DataCadastro">
        <rdfs:label>DataCadastro</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/>
</rdf:Property>

<rdf:Property rdf:ID="TipoCadastro">
        <rdfs:label>TipoCadastro</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range>
                <rdf:Alt>
                        <rdf:li>Normal</rdf:li>
                        <rdf:li>Recadastramento</rdf:li>
                </rdf:Alt>
        </daml:range>
</rdf:Property>
```

```
<rdf:Property rdf:ID="Situacao">
        <rdfs:label>Situacao</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range>
                <rdf:Alt>
                        <rdf:li>Abertura</rdf:li>
                        <rdf:li>BCC</rdf:li>
                        <rdf:li>Negativo</rdf:li>
                        <rdf:li>Indeterminado</rdf:li>
                </rdf:Alt>
        </daml:range>
</rdf:Property>

<rdf:Property rdf:ID="ComprovanteRenda">
        <rdfs:label>ComprovanteRenda</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="ConceitoCliente">
        <rdfs:label>ConceitoCliente</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="CPF">
        <rdfs:label>CPF</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:comment>Heterogeneous database primary key</daml:comment>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="NumRGIdentidade">
        <rdfs:label>NumRGIdentidade</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="DataNascimento">
        <rdfs:label>DataNascimento</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/>
</rdf:Property>

<rdf:Property rdf:ID="NaturalDe">
        <rdfs:label>NaturalDe</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>
```

```
<rdf:Property rdf:ID="Nacionalidade">
        <rdfs:label>Nacionalidade</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="EmpresaOndeTrabalha">
        <rdfs:label>EmpresaOndeTrabalha</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="EhAutonomo">
        <rdfs:label>EhAutonomo</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#boolean"/>
</rdf:Property>

<rdf:Property rdf:ID="DataAdmissao">
        <rdfs:label>DataAdmissao</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/>
</rdf:Property>

<rdf:Property rdf:ID="Profissao">
        <rdfs:label>Profissao</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="Cargo">
        <rdfs:label>Cargo</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="ValorSalario">
        <rdfs:label>ValorSalario</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</rdf:Property>

<rdf:Property rdf:ID="Banco">
        <rdfs:label>Banco</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="Agencia">
        <rdfs:label>Agencia</rdfs:label>
```

```
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="NumContaCorrente">
        <rdfs:label>NumContaCorrente</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="TipoCheque">
        <rdfs:label>TipoCheque</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range>
                <rdf:Alt>
                        <rdf:li>Especial</rdf:li>
                        <rdf:li>Normal</rdf:li>
                        <rdf:li>Cartao</rdf:li>
                </rdf:Alt>
        </daml:range>
</rdf:Property>

<rdf:Property rdf:ID="ChequeEhValido">
        <rdfs:label>ChequeEhValido</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#boolean"/>
</rdf:Property>

<rdf:Property rdf:ID="ChequeEhGarantido">
        <rdfs:label>ChequeEhGarantido</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#boolean"/>
</rdf:Property>

<rdf:Property rdf:ID="DataAberturaConta">
        <rdfs:label>DataAberturaConta</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="Patrimonio">
        <rdfs:label>Patrimonio</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="TipoPatrimonio">
        <rdfs:label>TipoPatrimonio</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
```

```
        <daml:range>
                <rdf:Alt>
                        <rdf:li>Casa</rdf:li>
                        <rdf:li>Apartamento</rdf:li>
                        <rdf:li>Terreno</rdf:li>
                        <rdf:li>Veiculo</rdf:li>
                        <rdf:li>Predio</rdf:li>
                </rdf:Alt>
        </daml:range>
</rdf:Property>

<rdf:Property rdf:ID="PatrimonioEhFinanciado">
        <rdfs:label>PatrimonioEhFinanciado</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#boolean"/>
</rdf:Property>

<rdf:Property rdf:ID="ValorEstimadoPatrimonio">
        <rdfs:label>ValorEstimadoPatrimonio</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</rdf:Property>

<rdf:Property rdf:ID="DataPrimeiraCompra">
        <rdfs:label>DataPrimeiraCompra</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/>
</rdf:Property>

<rdf:Property rdf:ID="DataMaiorCompra">
        <rdfs:label>DataMaiorCompra</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/>
</rdf:Property>

<rdf:Property rdf:ID="DataUltimaCompra">
        <rdfs:label>DataUltimaCompra</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/>
</rdf:Property>

<rdf:Property rdf:ID="ValorPrimeiraCompra">
        <rdfs:label>ValorPrimeiraCompra</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</rdf:Property>

<rdf:Property rdf:ID="ValorMaiorCompra">
        <rdfs:label>ValorMaiorCompra</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
```

```
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</rdf:Property>

<rdf:Property rdf:ID="ValorUltimaCompra">
        <rdfs:label>ValorUltimaCompra</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</rdf:Property>

<rdf:Property rdf:ID="DataAprovacaoLimite">
        <rdfs:label>DataAprovacaoLimite</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#date"/>
</rdf:Property>

<rdf:Property rdf:ID="NomeGerenteAprovouLimite">
        <rdfs:label>NomeGerenteAprovouLimite</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</rdf:Property>

<rdf:Property rdf:ID="LimiteSugerido">
        <rdfs:label>LimiteSugerido</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</rdf:Property>

<rdf:Property rdf:ID="LimiteAprovado">
        <rdfs:label>LimiteAprovado</rdfs:label>
        <daml:domain rdf:resource="#Cliente"/>
        <daml:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#double"/>
</rdf:Property>

</rdf:RDF>
```