INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Configuração e Interrogação com Sistemas Federados

Hugo Miguel Ferreira da Cruz
Licenciado

Dissertação de natureza científica realizada para obtenção do grau de Mestre em Engenharia Informática e de Computadores

Orientador: Doutor Paulo Trigo, ISEL - DEETC

Júri:
Presidente:
Mestre Vítor Almeida, ISEL - DEETC
Vogais:
Doutora Graça Gaspar, FCUL - DI
Doutor Paulo Trigo, ISEL - DEETC

Dezembro de 2010

Resumo

Apesar da existência de produtos comerciais e da investigação na área, a construção de sistemas de informação com diversos componentes distribuídos, heterogéneos e autónomos - conhecidos como sistemas de informação federados - é ainda um desafio. Estes sistemas de informação oferecem uma visão global unificada sobre os vários modelos de dados (parciais). No entanto, a modelação destes sistemas é um desafio, já que modelos de dados como o relacional não incluem informação sobre a distribuição e tratamento de heterogeneidade. É também necessário interagir com estes sistemas de informação, através de interrogações sobre os diversos componentes dos sistemas, sem ser necessário conhecer os detalhes dos mesmos. Este trabalho propõe uma abordagem a estes desafios, através da utilização de modelos para descrição semântica, e.g. linguagem OWL (*Ontology Web Language*), para construir uma descrição unificada dos seus diversos modelos parciais. O modelo criado para dar suporte a esta descrição é, em parte, baseado em ontologias existentes, que foram alteradas e extendidas para resolver diversos desafios de modelação. Sobre este modelo, é criado um componente de *software* que permite a execução de interrogações SQL (Structured Query Language) sobre o sistema federado, resolvendo os problemas de distribuição e heterogeneidade existentes.

Palavras-chave: Sistemas de Informação, Sistemas Federados, Integração, Ontology Web Language, Lógica de Descrição

CONTEÚDO

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

Actualmente está implantada a utilização generalizada de modelos de dados com suporte formal, e.g., o modelo relacional, ou com origem em sistemas específicos, e.g. folhas de cálculo. Numa mesma organização é comum surgirem sistemas de informação em diversas localizações, com diferentes estruturas (heterogéneas) e utilizados por diferentes aplicações. Mesmo quando todas as fontes de dados envolvidas seguem o mesmo modelo formal (e.g. o modelo relacional), existem problemas provocados pela utilização de produtos de diferentes fabricantes, como a falta de comunicação ou a utilização de formatos de dados incompatíveis. Um sistema de informação federado [1] pode ser construído sobre estes vários sistemas de informação, com o objectivo de oferecer uma visão global e uniforme sobre os mesmos. As fontes de dados de um sistema de informação federado caracterizam-se essencialmente por serem:

- Distribuídas - podem-se encontrar em localizações físicas distintas;
- Heterogéneas - podem ser baseadas em modelos distintos, assim como ser suportadas por diferentes tecnologias;
- Autónomas - cada uma das fontes de dados criada independentemente do sistema federado continua a funcionar independentemente deste.

Construir uma perspectiva global e uniforme de diversas fontes de dados parciais apresenta alguns desafios. Uma das grandes dificuldades é o facto de não existir um modelo formal que permita descrever não só a estrutura dos dados, mas também a sua distribuição e o contributo de cada fonte de dados para o sistema global. Também se deve evitar a construção de uma perspectiva global que constitua, ela própria, mais uma solução *ad-hoc*.

A análise das principais abordagens aos sistemas federados permite identificar problemas em aberto que motivam a colocação de duas hipóteses, em torno das quais esta dissertação se desenvolve:

1. Os formalismos que suportam descrição semântica têm poder expressivo para construir uma visão homogénea sobre modelos heterogéneos.
2. A adopção do *standard* de interrogação SQL (Structured Query Language) mantém-se válido num contexto onde a visão uniforme se descreve à base de um modelo construído sobre os princípios da hipótese 1.

A abordagem das duas hipóteses originou a construção de um modelo, de um protótipo e de uma validação experimental. O trabalho daqui resultante permite modelar e interrogar um sistema de informação constituído por fontes de dados distribuídas, heterogéneas e autónomas.

1.1 Resumo das Contribuições

Para além da validação das hipóteses colocadas, outras contribuições resultam desta dissertação.

Relacionadas directamente com a validação das hipóteses colocadas, foi materializado o modelo de suporte para a meta informação que permite descrever os diversos modelos de dados envolvidos no sistema, assim como as suas características de distribuição e heterogeneidade tecnológica e estrutural. Foi também implementado um executor de interrogações SQL sobre sistemas de informação federados descritos por este modelo e que permite interrogar fontes de dados mantidas em folhas de cálculo com formato compatível com o *Microsoft Excel 2007* (*xlsx*) e fontes de dados com conectividade a bases de dados (interface *jdbc*), para além de ser facilmente adicionado suporte a outros tipos de fontes de dados.

São também contribuições desta dissertação a criação de uma versão alterada da ontologia *relational.owl* [2] que introduz informação de acesso às fontes de dados e trata uma falha no suporte a chaves estrangeiras. Foi criado um conjunto de classes sobre o analisador de interrogações do *Eclipse Data Tools* [3] que simplifica a sua extensão para o modelo federado. O código do *plugin Datamaster* [4] para a ferramenta *Protege* [5] foi também extendido para dar suporte à versão alterada da ontologia *relational.owl* e para simplificar a implementação de suporte a novas fontes de dados para além do suporte já implementado para fontes de dados *xlsx* e fontes com interface *jdbc*.

1.2 Estrutura da Dissertação

Os restantes capítulos desta dissertação estão organizados seguinte forma:

No **capítulo 2** é apresentado o estado da arte na área dos sistemas de informação federados, dos formalismos de descrição de dados e da execução de interrogações em sistemas de informação federados. São analisados os problemas em aberto na construção de soluções para sistemas de informação federados e a forma como produtos comerciais e projectos de código de fonte aberta se propõem resolvê-los. São também abordadas as alternativas existentes para a descrição do modelo de dados destes sistemas, com destaque para os formalismos de descrição semântica e para a linguagem OWL (*Ontology Web Language*). Por fim, são apresentados os aspectos mais relevantes da execução de interrogações em sistemas de informação federados.

O **capítulo 3** descreve os vários aspectos do modelo proposto neste trabalho. Apresenta um modelo de dados baseado em formalismos que suportam descrição semântica. São apresentadas as soluções para descrever os modelos parciais (fontes de dados) e o modelo global (modelo público do sistema federado), assim como as alternativas de mapeamento entre os modelos, nomeadamente a utilização de partições horizontais e verticais, réplicas, colunas calculadas e constantes. Termina-se com um exemplo ilustrativo de apoio à concretização dos conceitos apresentados.

O **capítulo 4** apresenta o suporte desenvolvido neste trabalho que permite utilizar a linguagem padronizada SQL em interrogações sobre um modelo federado. É descrita uma solução de interrogação que adopta o *standard* de interrogação SQL e se executa sobre o sistema descrito pelo modelo apresentado no capítulo 3. São apresentados os aspectos essenciais da solução implementada, em cada uma das fases da execução de uma interrogação: análise, planeamento e execução. É apresentada uma abordagem de resolução de heterogeneidade de acesso às fontes de dados através de adaptadores.

O **capítulo 5** apresenta a implementação de uma aplicação gráfica que auxilia a concretização e teste de um sistema de informação federado baseado nas hipóteses apresentadas. Assume-se que os utilizadores alvo estão familiarizados com o modelo relacional. Esta aplicação permite criar o modelo descrito no capítulo 3 sem ser necessário ter conhecimento sobre modelação semântica e reduzindo potenciais erros introduzidos através da descrição manual dos modelos.

No **capítulo 6** é apresentado um exemplo completo que ajuda a validar as soluções apresentadas.

O relatório termina com as conclusões do trabalho realizado e com o alinhamento de trabalho futuro.

CAPÍTULO 2

ESTADO DA ARTE

Neste capítulo faz-se a análise do estado da arte na área dos sistemas de informação federados. Apresenta-se uma arquitectura de referência, a meta-informação que suporta estes sistemas e os problemas em aberto. Segue-se uma análise de produtos e projectos de código aberto. É dada ênfase às soluções apresentadas para os problemas típicos dos sistemas de informação federados. É analisada a forma como a abordagem proposta e desenvolvida nesta dissertação se distingue das soluções existentes.

São também analisadas alternativas para a descrição do modelo de dados de sistemas de sistemas federados, nomeadamente a utilização do modelo relacional, de um modelo de objectos ou de um modelo com suporte para descrição semântica.

São ainda apresentados os aspectos mais relevantes da execução de interrogações em sistemas de informação federados, e como este processo difere da execução de interrogações em sistemas monolíticos.

2.1 Sistemas de Informação Federados

A implementação de sistemas de informação federados levanta uma série de problemas que têm sido alvo de investigação desde o início dos anos 80 [6]. Desde então, acompanhando as evoluções nas várias áreas relacionadas (bases de dados [7], integração de sistemas [8] [9], comunicações), o foco da investigação tem variado muito ao longo dos anos. Nos anos mais recentes, a arquitectura proposta em [1] e apresentada na figura 2.1 tem sido sido adoptada como referência por vários autores e projectos de investigação [10] [11] [12].

A arquitectura apresenta os seguintes níveis de meta-informação, utilizados na construção de um sistema de informação federado:

1. Técnico - informação sobre os aspectos técnicos dos componentes, como a sua localização, forma de acesso, etc.
2. Lógico - as relações e dependências entre os diversos modelos de dados.
3. Meta modelos - a descrição dos diversos modelos de dados de forma a lidar com a sua heterogeneidade.
4. Semântico - descrição do significado dos conceitos existentes no sistema de informação.
5. Qualidade - informação sobre a qualidade das fontes de dados, como a frequência de actualização, a correcção dos dados e a disponibilidade.

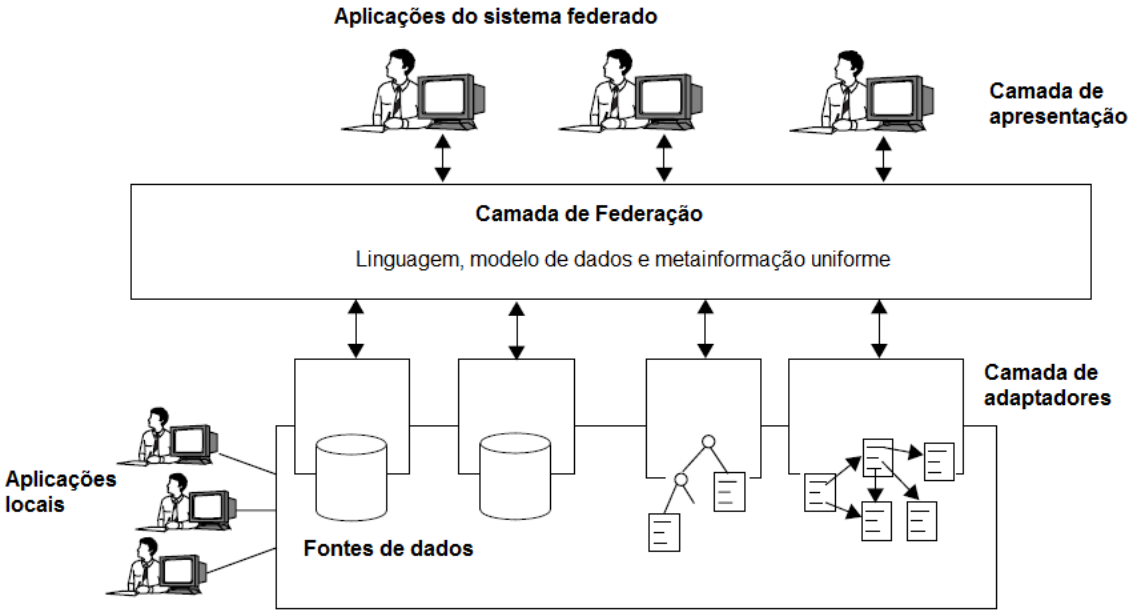


Figura 2.1: Arquitectura de referência (adaptado de [1])

6. Infraestrutura - informação que auxilia os utilizadores a encontrar informação relevante. Tipicamente, inclui anotações sobre os elementos de dados do sistema.
7. Utilizador - Suporte a opções personalizadas de apresentação dos dados.

Os níveis de meta informação técnico, lógico e de meta modelos são essenciais para o funcionamento básico de um sistema de informação federados. Estas três camadas são utilizadas pelas camadas de federação e de adaptadores e constituem a base de um modelo de dados para um sistema federado.

Existe também investigação que, actualmente, aborda os seguintes aspectos:

- Definição dos modelos de dados [13] [14] [15];
- Criação automática de modelos de dados globais a partir dos modelos de dados das fontes de dados participantes na federação [11];
- Processamento de interrogações [10] [9];
- Detecção e processamento de informação semântica como, por exemplo, informação repetida entre fontes de dados [8];
- Controlo de transacções [7].

Nesta dissertação parte-se da arquitectura proposta em [1] (figura 2.1) e abordam-se os problemas da definição dos modelos de dados e do processamento de interrogações. O trabalho aqui proposto incide sobre os níveis de meta informação técnico, lógico e de meta modelos.

2.2 Representação dos Modelos de Dados

Existem diversas abordagens alternativas para a descrição do modelo de dados de sistemas de sistemas federados. Os modelos mais discutidos e implementados são:

- Modelo relacional - utilizado por várias implementações [16] [17] [18], principalmente por sistemas de gestão de bases de dados que também suportam sistemas federados. A escolha deste modelo permite a reutilização das ferramentas já existentes para utilização com bases de dados, mas impõe restrições na definição de qual a participação das fontes de dados no sistema federado. Não é possível, por exemplo, definir réplicas de dados neste modelo.
- Modelo de objectos [7] [19] - os modelos de dados das fontes e da visão global dos dados são definidos através de classes. A heterogeneidade e o contributo das fontes para o sistema são resolvidos através das hierarquias das classes e de código executável. Este modelo tem como principal característica aglutinar a estrutura dos dados e o código. Daqui resulta a dificuldade de configurar o modelo por uma pessoa especialista na modelação dos dados mas que não conheça as particularidades da linguagem de programação em que as classes estão implementadas.
- Modelos de descrição semântica - é uma ideia frequentemente apresentada como preferencial na literatura [1] [13] [14] [15] mas continua a existir uma escassez de implementações. Estes modelos permitem soluções muito flexíveis para descrever de forma completa fontes de dados com características heterogéneas. É, no entanto, importante a existência de aplicações que permitam a utilização destas soluções por utilizadores familiarizados com o modelo relacional. Também a linguagem de interrogação do sistema é uma questão em aberto, sendo que a utilização de uma nova linguagem traz uma nova barreira de aprendizagem e discussões de qualidade e aplicabilidade sobre a mesma.

Nesta dissertação exploram-se os modelos de descrição semântica (cf.. terceira opção acima), concretizada sobre a linguagem OWL. Esta linguagem permite representar informação de forma a facilitar o seu processamento automático por aplicações e é um *standard W3C*. A linguagem OWL tem como suporte formal a lógica de descrição (DL *Description Logic*) e como suporte tecnológico o RDFS (*Resource Definition Framework Schema*). Esta linguagem está na base para os esforços actuais na área da *web* semântica, está vocacionada para lidar com a pouca estrutura existente na *web* e oferece uma visão unificada sobre dados e meta-dados.

A linguagem OWL tem três sub-linguagens com diferentes capacidades de descrição semântica: *OWL Full*, *OWL-DL* e *OWL Lite*, por ordem decrescente das capacidades descritivas dos operadores disponíveis. No entanto, os motores de inferência que suportam a variante *OWL Full* não garantem que seja possível chegar a resultados em tempo útil, pelo que a variante *OWL-DL* é a versão mais expressiva que dá garantias de inferência completa. Esta variante tem a sua base na lógica de descrição $\mathcal{SHOIN}^{\mathcal{D}}$:

- $\mathcal{S}$ - abreviatura da linguagem $\mathcal{ALC}$, com papéis (propriedades) transitivos:
  - $\mathcal{AL}$ - *Attributive Language* - negação de conceitos atómicos, intersecções, uniões, restrições, qualificação existencial limitada;
  - $\mathcal{C}$ - negação de conceitos;
- $\mathcal{H}$ - hierarquia de papéis (propriedades);
- $\mathcal{O}$ - nominais, restrições sobre valores;
- $\mathcal{I}$ - propriedades inversas;
- $\mathcal{N}$ - restrições de cardinalidade;
- $\mathcal{D}$ - utilização de tipos XML.

A representação do conhecimento (*Knowledge Representation* - KR) de um domínio de aplicação utilizando DL é realizado através da caracterização de dois aspectos essenciais:

- Um contexto da terminologia (*Terminological Box* - **TBox**) onde se define o conjunto de **conceitos** e das relações entre esses conceitos através de **papéis**, e
- Um contexto das asserções (*Assertion Box* - **ABox**) onde se utilizam as definições da TBox para caracterizar cada objecto (**indivíduo**) concreto que ocorre no domínio de aplicação.

A linguagem OWL utiliza terminologias diferentes do DL, nomeadamente, um conceito em DL é uma classe em OWL e um papel em DL é uma propriedade em OWL.

Nesta dissertação as listagens de taxonomia são apresentadas usando DL, e são utilizados apenas alguns dos símbolos disponíveis. Nomeadamente, são utilizados:

- $C \sqsubseteq D$ - subsunção (subclasse em OWL). O conceito C é subsumido em D.
- R^- - papel simétrico.
- R_+ - papel transitivo.
- $\bot$ - conceito vazio.
- $\top$ - conceito universal.
- $A \sqcup B$ - união.
- $\exists R.C$ - qualificador existencial. Esta asserção caracteriza indivíduos que têm relação R com o conceito C.
- $\forall R.C$ - qualificador universal. Esta asserção caracteriza indivíduos que têm todas as suas relações R com o conceito C.

Numa TBox descrevem-se axiomas de inclusão e de igualdade. Um axioma é uma asserção sobre o modo como os conceitos e os papéis se interligam uns com os outros. O conceito base das taxonomias OWL é "owl:Thing". Por exemplo, o axioma de inclusão "Pessoa $\sqsubseteq$ owl:Thing" define o conceito "Pessoa" na taxonomia. Um papel é definido da mesma forma, com a excepção de o papel se subsumido pelo conceito "owl:Property", e.g. "estuda $\sqsubseteq$ owl:Property".

Na definição de um papel é também essencial indicar o domínio e contradomínio desse papel. Para o efeito são usados dois axiomas:

- $\exists R \top \sqsubseteq C$ - limita o **domínio** do papel "R" a indivíduos do conceito "C".
- $\top \sqsubseteq \forall R.C$ - limita o **contradomínio** do papel "R" a indivíduos do conceito "C".

De forma a facilitar a leitura das listagens DL, estes axiomas são substituídos por duas construções (propostas em [20]):

- **Domain(R, C)** - limita o **domínio** do papel "R" a indivíduos do conceito "C".
- **Range(R, C)** - limita o **contradomínio** do papel "R" a indivíduos do conceito "C".

Na definição de uma ABox são realizadas asserções sobre os conceitos e papéis da TBox. Um indivíduo é definido realizando uma asserção do tipo "C(a)" em que "a" é um indivíduo do conceito "C". Asserções sobre os papéis são representadas por "R(a, b)" em que existe uma relação "R" entre "a" e "b".

Na taxonomia apresentada na listagem 1 são criados os conceitos "Professor" e "Aluno" e o papel "ensina" que relaciona os dois conceitos. De seguida são realizadas asserções para a definição dos indivíduos "António" (um "Professor") e "Pedro" (um "Aluno") e para a definição de uma relação ensina entre os indivíduos.

```
1  #Definição de conceitos na TBox
2  Professor ⊑ owl:Thing
3  Aluno ⊑ owl:Thing
4
5  #Definição de papéis na TBox
6  ensina ⊑ owl:Property
7  Domain(ensina, Professor)
8  Range(ensina, Aluno)
9
10 #Asserções sobre conceitos na ABox
11 Professor(António)
12 Aluno(Pedro)
13
```

```
14 #Asserções sobre papéis na ABox
15 ensina(António, Pedro)
```

Listagem 1: Exemplo de uma taxonomia

Nesta dissertação, as capacidades de inferência sobre taxonomias OWL são utilizadas para tirar partido dos papéis simétricos e transitivos. Por exemplo, a definição de uma propriedade que estabeleça uma relação de réplica entre dois indivíduos é simultaneamente simétrica e transitiva.

2.3 Interrogações em Sistemas de Informação Federados

O processamento de interrogações em sistemas de informação é realizado em três fases [21] (figura 2.2):

1. Análise - tradução de uma representação, tipicamente textual, para uma expressão que possa ser processada. Por exemplo, numa fonte de dados relacional, seria traduzido um texto SQL numa expressão de álgebra relacional.
2. Optimização - a consulta de estatísticas sobre os dados permite reescrever a interrogação e seleccionar o plano de execução que oferece melhor desempenho.
3. Avaliação - acede às fontes de dados segundo o plano de execução seleccionado e devolve os resultados.

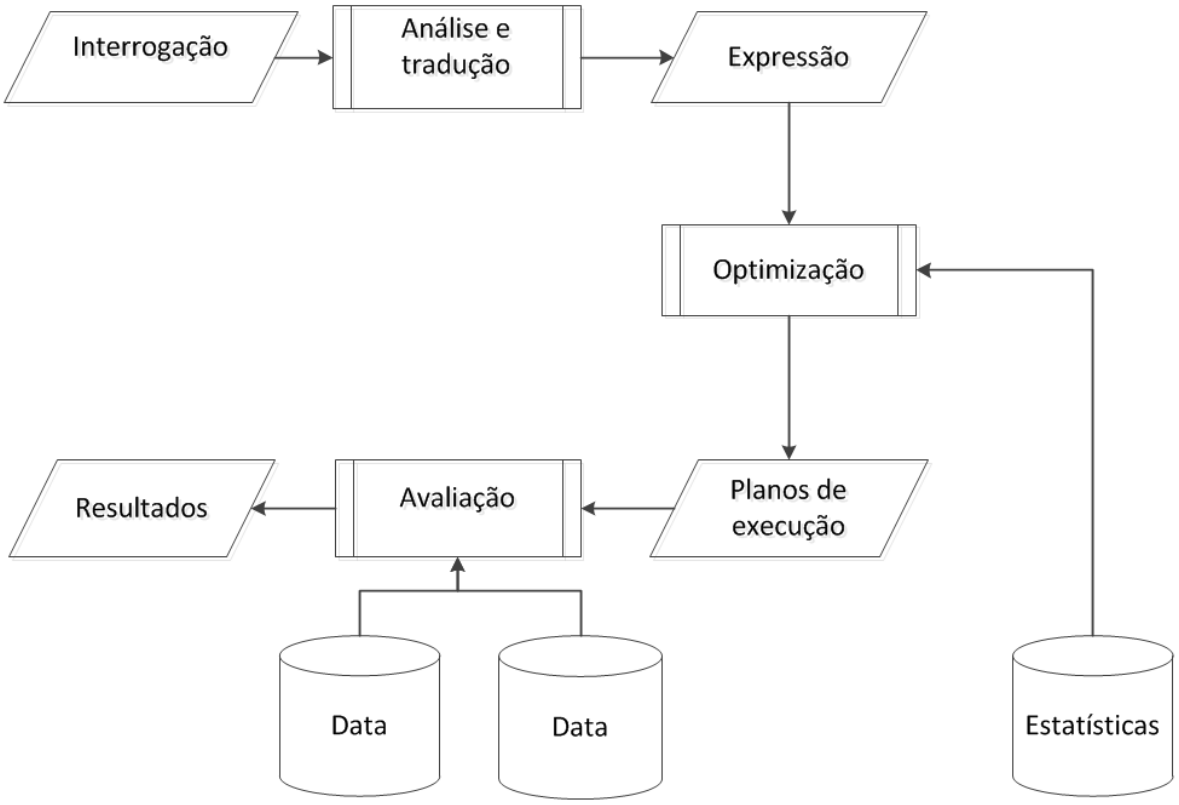


Figura 2.2: Fases da execução de uma interrogação

Num sistema de informação monolítico, e.g. numa base de dados suportada por um único SGBD, uma interrogação é realizada sobre um único modelo de dados e executado directamente. No entanto, num sistema de informação federado, uma interrogação é realizada sobre o modelo de dados da federação que

envolve várias fontes de dados distribuídas e heterogéneas, sendo necessário criar interrogações parciais sobre os modelos de dados das fontes e traduzir as interrogações parciais para poderem ser executadas sobre cada uma das fontes de dados [10].

Apesar das três fases de execução de uma interrogação (cf.. figura 2.2) se manterem válidas para os sistemas de informação federados, as suas características obrigam a que as operações a realizar em cada uma das fases sejam diferentes:

- Distribuição - não é suficiente criar um plano de execução para a interrogação. É necessário criar interrogações parciais para cada uma das fontes de dados e pelo menos um plano para unificar os resultados parciais e realizar pós-processamentos.
- Heterogeneidade - a linguagem de interrogação de cada uma das fontes de dados pode ser diferente. A linguagem de interrogação do sistema federado pode ainda ser distinta das linguagens das fontes de dados, pelo que existe a necessidade de traduzir a interrogação nas diferentes linguagens em causa.
- Autonomia - é difícil manter estatísticas sobre a globalidade dos dados, pelo que as optimizações baseadas nestas estatísticas, em geral, apenas se realizam ao nível individual de cada uma das fontes de dados.

Assim, o processamento de interrogações em sistemas de informação federados é detalhado nas seguintes fases:

1. Análise - esta fase mantém os mesmo princípios de um sistema de informação não federado.
2. Planeamento
   (a) Criar planos de execução para as interrogações parciais, que podem ser executadas sobre as fontes de dados.
   (b) Criar planos de execução para juntar os resultados das interrogações parciais.
   (c) Criar planos de execução para realizar pós-processamentos.
3. Avaliação
   (a) Executar os planos de execução parciais através dos adaptadores de cada fonte de dados.
   (b) Executar os planos de execução para juntar e pós-processar os resultados.

A optimização das interrogações é um factor problemático nos sistemas de informação federados. Devido às características destes sistemas, não é possível aplicar as estratégias de optimização utilizadas em sistemas de bases de dados [22]. As estratégias de optimização típicas em sistemas de informação federados estão divididas em três grupos [19]:

1. Estratégia de pós-processamento.
2. Execução paralela.
3. Estimação de custos globais.

A optimização das estratégias de pós-processamento pretende que o máximo de trabalho seja realizado pelas fontes de dados. Tira-se partido do paralelismo, da distribuição de carga e de possíveis optimizações locais. Um caso muito relevante deste processamento é a realização de filtragens dos dados. Se o processamento dos filtros for realizado nas fontes de dados pode-se tirar partido de eventuais índices existentes localmente, reduz-se o tráfego entre os componentes do sistema e reduz-se a quantidade de dados que tem de ser processada centralmente.

É também sugerido em [19] uma estratégia de passar às fontes de dados resultados parciais de outras interrogações parciais. Esta estratégia é, no entanto, mais complicada de planear e assume que as fontes de dados suportam a leitura desses resultados parciais. A comunicação entre as fontes de dados também pode não ser possível.

A execução paralela pretende optimizar principalmente as operações de junção dos dados através da execução paralela dessas operações. Esta execução pode ser realizada sobre a forma de múltiplos fios de execução numa camada de federação [23], mas é especialmente interessante num cenário de execução distribuída [24] . Quando existe um conjunto de camadas de federação distribuídas e colaborativas (vários mediadores), esta estratégia é tipicamente utilizada para realizar junções intermédias de dados.

A estimação de custos globais tem como objectivo a reutilização de técnicas de optimização normalmente aplicadas a bases de dados e a algoritmos de execução distribuída. A estimação destes custos pode permitir a escolha da melhor forma de juntar resultados parciais, a escolha dos melhores mediadores para realizar um processamento distribuído ou qual a réplica dos dados que apresenta melhor desempenho [23]. O *GaianDB* [24] tem como principal atractivo a apresentação de contribuições significativas na utilização destas técnicas de optimização.

Nesta dissertação é proposta e implementada uma solução de interrogação com as três fases da execução (análise, planeamento e avaliação) e que aborda os problemas relacionados com a distribuição, heterogeneidade e autonomia das fontes de dados. Apesar de não ser o foco da dissertação, realizam-se optimizações na estratégia de pós-processamento, nomeadamente na maximização do trabalho realizado pelas fontes de dados e minimização das operações realizadas na camada de federação.

2.4 Produtos e protótipos

Existem alguns produtos comerciais e projectos de código de fonte aberto (*open-source*) para a implementação de sistemas federados. No entanto, alguns utilizam o termo "sistema federado" com outros significados. Por exemplo, a definição utilizada pelo produto Microsoft *SqlServer* é a de um conjunto de servidores com partições horizontais de dados [25], com o objectivo de realizar balanceamento de carga. Outros produtos disponibilizam funcionalidades apenas focadas no modelo relacional e na *Web* (vista como uma colecção de páginas *html*), ignorando outros tipos de fontes de dados.

Segue-se a análise de produtos que se apresentam como referências na área e que apresentam as soluções mais abrangentes.

IBM DB2 [16]

Talvez a implementação mais completa dos conceitos seja a disponibilizada pela IBM através do seu sistema de gestão de bases de dados DB2. Uma instância do SGBD DB2 assume o papel de servidor central e assume as funções de camada federada para o sistema. Esta camada proporciona transparência de localização, esquema e linguagem e contempla um conjunto de fontes de dados relacionais (DB2, *SqlServer* e *Oracle*) e não relacionais (*Excel* e *WebServices*) através de um conjunto de adaptadores (*wrappers*) distribuídos com o produto. Estes adaptadores são componentes de *software* que são invocados pelo servidor para fazer a comunicação com cada uma das fonte de dados. É da responsabilidade destes a tradução da entrada (em linguagem SQL) para as operações a realizar efectivamente, seja a tradução para outra linguagem ou para um conjunto de operações a realizar (por exemplo, um conjunto de chamadas a funções).

As fontes de dados têm, no entanto, de ser representadas sobre a forma tabular no servidor federado. Note-se também que todas as fontes de dados não relacionais apenas suportam operações de leitura. A autonomia das fontes é mantida excepto quando o contexto de uma transacção requer que essa transacção corra na fonte de dados.

O facto de todo o controlo ser executado num servidor central torna necessária a existência de uma máquina com grande capacidade para não comprometer a disponibilidade do sistema federado. Não existem também muitas possibilidades de definição de alguns tipos de meta-dados, como por exemplo para definir relações e restrições entre participantes. Para além disso, as partições/replicações de dados só são suportados indirectamente através de vistas, que estão limitadas a operações de leitura.

O modelo de dados global é definido tal como o de uma base de dados comum no sistema DB2, através da *Data Definition Language* (DDL) do SGBD. É então utilizado o modelo relacional com as habituais extensões de tipos (que permitem campos com ficheiros binários, xml, etc). Também não existe processamento semântico da informação e o processamento transaccional está limitado pelas capacidades dos adaptadores.

A principal factor diferenciador das soluções apresentadas nesta dissertação está nas capacidades de modelação de dados. Nomeadamente, é possível criar um modelo de dados com informação de distribuição e relações entre as fontes de dados. Sem esta possibilidade, os problemas de distribuição dos dados têm de ser resolvidos caso a caso, através da criação de vistas e procedimentos armazenados de acesso aos dados.

Sybase Adaptive Server Enterprise [17]

Um sistema de informação federado baseado no *Sybase Adaptative Server* assenta numa base de dados local que contém representantes das tabelas externas (*proxy tables*). As *proxy tables* contêm a meta-informação importada das fontes originais dos dados e permitem aceder às fontes de dados através de interrogações em linguagem SQL. Quando uma tabela da vista global é constituída por mais de uma tabela local, a solução passa pela criação de vistas, com a limitação de essas vistas apenas permitirem operações de leitura.

O suporte a fontes de dados para além do modelo relacional é realizada através de ligações a outros produtos do mesmo fabricante. Todas estas fontes de dados são mapeadas para o modelo relacional para manter a consistência com os restantes esquemas parciais. Por exemplo, é possível criar uma *proxy table* baseada num sistemas de ficheiros, sendo criada uma tabela com colunas como o nome do ficheiro, a sua localização, meta-informação (datas de criação, acesso, tamanho, etc) e conteúdo (pesquisável como texto). A grande desvantagem desta solução é a dependência de outros produtos do fabricante para o acesso às fontes de dados.

A solução apresentada nesta dissertação tem uma arquitectura em que os adaptadores de acesso às fontes de dados podem ser implementados facilmente, apenas tendo de implementar uma interface pré-estabelecida.

UniSql

O UniSql é um SGBD orientado a objectos que permite a criação de um sistema de informação federado através de representantes dos objectos (objectos *proxy*). A linguagem de interrogação utilizada é a mesma que é utilizada para as interrogações realizadas sobre os objectos: SQL/X. A comunicação com outras fontes de dados é feita através de adaptadores já incluídos na distribuição. Por ser um SGBD orientado a objectos, permite modelações distintas das possíveis nas soluções baseadas no modelo relacional puro. É possível, por exemplo, a execução de algum processamento semântico dos dados.

A utilização de um modelo de objectos permite soluções flexíveis mas pouco reutilizáveis devido à mistura do código com a estrutura dos dados. A solução apresentada nesta dissertação permite a utilização de modelos de descrição semântica e a separação efectiva da estrutura e do comportamento.

Apache Derby [18]

O *Apache Derby* é um sistema de base de dados relacional de fonte aberta que contém um ponto de extensão (VTI - *Virtual Table Interface* [26]) que pode ser usado para a implementação de sistemas de informação federados. Este ponto de extensão permite a criação de funções que acedem a fontes externas de dados, escondendo os pormenores do modelo de dados, da distribuição e da heterogeneidade das fontes de dados acedidas.

A criação destas funções pode ser realizada de forma *had-hoc* ou de modo a permitir aceder a todas as fontes de dados do mesmo tipo, resolvendo a heterogeneidade tecnológica entre as fontes. Sobre estas VTI podem ser criadas vistas que criem a visão global sobre as VTI parciais para resolver a heterogeneidade da estrutura lógica dos dados. A solução apresentada nesta dissertação permite a criação de um modelo de dados que contenha toda a informação necessária para resolver tanto a heterogeneidade tecnológica como a heterogeneidade de modelos de dados.

GaianDB [24]

O *GaianDB* [24] é um projecto da IBM, que é disponibilizado como uma biblioteca *open-source* escrita em Java, e que é construído sobre o projecto *Apache Derby*. Uma base de dados *GaianDB* é constituída por uma rede de nós, todos eles a correr instâncias do *GaianDB*. Utilizando as soluções do *Apache Derby* para resolver as questões do acesso às fontes de dados, o contributo do *GaianDB* está na comunicação e planeamento de operações entre os nós de uma base de dados *GaianDB*.

Nesta dissertação não se abordam os problemas que são o foco da implementação do *GaianDB*, no entanto a sua análise fornece uma importante base para trabalho futuro.

MIX [27]

Projecto ligado à investigação, sucessor do projecto de referência *TSIMMIS* (*The Stanford-IBM Manager of Multiple Information Sources* [28]). Documentos XML são utilizados para definir o modelo de dados, para resolver as questões de heterogeneidade e até para devolver os resultados. A linguagem de interrogação utilizada foi criada como parte do projecto e tem como nome *XMAS*.

A utilização de XML, uma linguagem com suporte para dados semi-estruturados, como forma de retorno de dados, permite uma maior facilidade de integração de fontes semi-estruturadas ou mesmo não estruturadas. A utilização de uma outra linguagem de interrogação baseada no XML e já estabelecida, como a *XQuery*, poderia ser uma abordagem interessante. No entanto, ao contrário da solução apresentada nesta dissertação, foi criada uma linguagem de interrogação própria, dificultando a aceitação da solução por parte do utilizadores.

CAPÍTULO 3

DESCRIÇÃO DO MODELO FEDERADO

A modelação de um sistema de informação federado partilha algumas das características da modelação de um sistema de informação baseado no modelo relacional mas apresenta também outros desafios. Nomeadamente, é essencial descrever:

- o modelo de dados público (**modelo global**) do sistema federado;
- a estrutura das fontes de dados envolvidas (**modelos parciais**), considerando a sua heterogeneidade;
- a contribuição de cada fonte de dados para o sistema federado.

A descrição dos modelos de dados, ilustrada na figura 3.1, está dividida em quatro componentes (figura 3.1):

1. TBox_P - Componente fixa que define a forma como os modelos parciais são descritos, baseada na taxonomia *relational.owl* [2];
2. ABox_P - Descrição dos modelos parciais através de asserções sobre a TBox_P. É criada uma ABox_P para cada fonte de dados parcial;
3. TBox_G - Descrição da estrutura do modelo global;
4. ABox_G - Descrição das relações entre o modelo global e os modelos parciais através de asserções sobre a TBox_G e criando relações com as diversas ABox_P.

Todos os conceitos, relações e indivíduos das taxonomias apresentadas têm um prefixo que mapeia o seu espaço de nomes. De forma a facilitar a leitura das listagens, foi omitido o prefixo "fis" correspondente ao espaço de nomes de base da taxonomia apresentada. Os restantes prefixos utilizados são:

- rdf - espaço de nomes da *Resource Description Framework*;
- owl - espaço de nomes da *Ontology Web Language*;
- xsd - espaço de nomes dos elementos *XML Schema Definition*;
- dbs - espaço de nomes da taxonomia *relational.owl*.

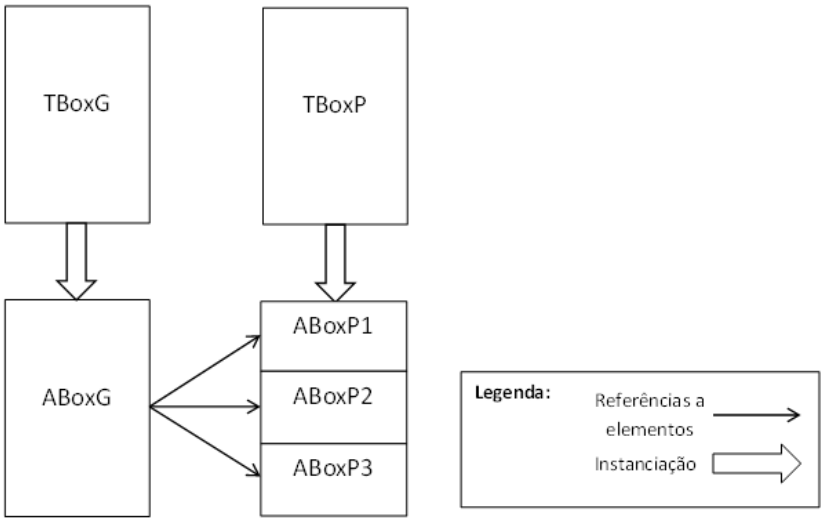


Figura 3.1: Componentes da descrição de um sistema

3.1 Descrição dos Modelos Parciais

Os modelos parciais são descritos sobre uma versão alterada da taxonomia *relational.owl* [2]. Esta taxonomia, na sua versão original permite descrever a estrutura de fontes de dados relacionais, pelo que as fontes de dados não relacionais são mapeadas para este modelo.

A versão original da taxonomia *relational.owl* inclui a representação de:

- base de dados (*dbs:Database*);
- tabelas (*dbs:Table*);
- colunas (*dbs:Column*);
- chaves primárias (*dbs:PrimaryKey*);
- chaves estrangeiras (*dbs:references*);
- relações entre cada um dos elementos (*dbs:hasTable*, *dbs:hasColumn*, *dbs:isIdentifiedBy*, *dbs:hasTable*).

A TBox desta taxonomia é apresentada na listagem 2.

```
1  dbs:Database ⊑ rdf:Bag
2  dbs:Table ⊑ rdf:Seq
3  dbs:Column ⊑ rdf:Resource
4  dbs:PrimaryKey ⊑ rdf:Bag
5
6  dbs:hasTable ⊑ owl:Property
7  Domain(dbs:hasTable, dbs:Database)
8  Range(dbs:hasTable. dbs:Table)
9
10 dbs:hasColumn ⊑ owl:Property
11 Domain(dbs:hasColumn, dbs:Table ⊔ dbs:PrimaryKey)
12 Range(dbs:hasColumn, dbs:Column)
13
14 dbs:isIdentifiedBy ⊑ owl:Property
15 Domain(dbs:isIdentifiedBy, dbs:Table)
16 Range(dbs:isIdentifiedBy, dbs:PrimaryKey)
17
18 dbs:references ⊑ owl:Property
19 Domain(dbs:references, dbs:Column)
20 Range(dbs:references, dbs:Column)
```

Listagem 2: TBox relational.owl original

Foram identificadas duas lacunas nesta taxonomia que motivaram uma proposta de alteração da mesma.

- A primeira lacuna tem a ver com a definição de chaves estrangeiras. Na versão original da *relational.owl*, as chaves estrangeiras são definidas apenas pelo estabelecimento de relações entre duas colunas. Isto permite interpretar cada relação estabelecida como uma chave estrangeira ou todo o conjunto de relações estabelecidas numa tabela como uma única chave estrangeira. No entanto, uma chave estrangeira correctamente definida é caracterizada por um conjunto de uma ou mais colunas relacionadas, que devem ser avaliadas em conjunto. Assim, definiu-se um novo conceito *dbs:ForeignKey* que representa uma chave estrangeira e contém pares de colunas (*dbs:ColumnsRelation*).
- A segunda lacuna identificada é a ausência de informação de acesso à fonte de dados. Enquanto a taxonomia original pode ser utilizada para fins que não necessitem desta informação, no caso desta dissertação essa é uma necessidade efectiva. Assim, para cada fonte de dados (*dbs:database*), é necessário saber a sua localização (um *Unique Resource Identifier*), qual o adaptador a usar para aceder a essa fonte e, possivelmente, as credenciais necessárias (*username* e *password* ou apenas um dos dois). Como o acesso às colunas pode não poder ser feito através do seu nome, é registada informação que permita o seu acesso. O formato desta informação não é estabelecido neste modelo, mas tem de ser acordado com o adaptador utilizado para acesso à fonte de dados.

A estrutura desta versão modificada da *relational.owl* corresponde à TBox_P e é apresentada na listagem 3. Asserções realizadas sobre esta TBox constituem uma ABox_P.

```
1  #conceitos relational.owl
2  dbs:Database ⊑ rdf:Bag
3  dbs:Table ⊑ rdf:Seq
4  dbs:Column ⊑ rdf:Resource
5  #novos conceitos
6  dbs:ForeignKey ⊑ owl:Thing
7  dbs:ColumnsRelation ⊑ owl:Thing
8
9  #papéis relational.owl
10 dbs:hasTable ⊑ owl:Property
11 Domain(dbs:hasTable, dbs:Database)
12 Range(dbs:hasTable, dbs:Table)
13
14 dbs:hasColumn ⊑ owl:Property
15 Domain(dbs:hasColumn, dbs:Table)
16 Range(dbs:hasColumn, dbs:Column)
17
18 #novos papéis para suporte a chaves estrangeiras
```

```
dbs:hasForeignKey ⊑ owl:Property
Domain(dbs:hasForeignKey, dbs:Table)
Range(dbs:hasForeignKey, dbs:ForeignKey)

dbs:toTable ⊑ owl:Property
Domain(dbs:toTable, dbs:ForeignKey)
Range(dbs:toTable, dbs:Table)

dbs:relatedColumns ⊑ owl:Property
Domain(dbs:relatedColumns, dbs:ForeignKey)
Range(dbs:relatedColumns, dbs:ColumnsRelation)

dbs:fromColumn ⊑ owl:Property
Domain(dbs:fromColumn, dbs:ColumnsRelation)
Range(dbs:fromColumn, dbs:Column)

dbs:toColumn ⊑ owl:Property
Domain(dbs:toColumn, dbs:ColumnsRelation)
Range(dbs:toColumn, dbs:Column)

#novos papéis para suporte a acesso aos dados
dbs:provider ⊑ owl:Property
Domain(dbs:provider, dbs:Database)
Range(dbs:provider, xsd:string)

dbs:uri ⊑ owl:Property
Domain(dbs:uri, dbs:Database)
Range(dbs:uri, xsd:string)

dbs:username ⊑ owl:Property
Domain(dbs:username, dbs:Database)
Range(dbs:username, xsd:string)

dbs:password ⊑ owl:Property
Domain(dbs:password, dbs:Database)
Range(dbs:password, xsd:string)

dbs:columnAccess ⊑ owl:Property
Domain(dbs:columnAccess, dbs:Column)
Range(dbs:columnAccess, xsd:string)
```

Listagem 3: TBox relational.owl modificada

3.2 Descrição do Modelo Global

A TBox_G descreve a estrutura do modelo de dados global. Cada elemento da TBox_G representa um constituente do modelo de dados global. Uma subclasse de "FederatedEntity" descreve uma tabela do modelo global. Uma propriedade que tenha essa subclasse como domínio descreve uma coluna da tabela. Tomando como referência o modelo de dados relacional, a TBox_G pode ser construída seguindo apenas duas regras:

1. Para cada tabela, é criada uma subclasse de "FederatedEntity".
2. Para cada coluna dessa tabela, é criada uma propriedade com a classe correspondente à tabela como domínio e com o conceito "dbs:Column" como contradomínio.

O exemplo da listagem 4 contém uma TBox que representa uma tabela "Product" com as colunas "prodId" e "prodDesc".

```
FederatedEntity ⊑ owl:Thing
Product ⊑ FederatedEntity

prodId ⊑ owl:Property
```

```
5 Domain(prodId, Product)
6 Range(prodId, dbs:Column)
7
8 prodDesc ⊑ owl:Property
9 Domain(prodDesc, Product)
10 Range(prodDesc, dbs:Column)
```

Listagem 4: Exemplo de uma TBox de um esquema global

3.3 Mapeamento entre os Modelos

O mapeamento entre o modelo global e os modelos parciais é realizado na ABox_G e permite modelar:

- Mapeamento directo;
- Partições horizontais;
- Partições verticais;
- Réplicas;
- Colunas constantes;
- Colunas calculadas.

3.3.1 Mapeamento Directo

O mapeamento directo entre o modelo global e um modelo parcial é realizado criando uma instância do conceito referente a uma tabela do modelo global e indicando quais as colunas dos modelos parciais que correspondem a cada coluna do modelo global, como apresentado na listagem 5. Todas as colunas dos modelos parciais têm que pertencer à mesma tabela na fonte de dados.

```
1 Product (productInstance1)
2 prodId (productInstance1, dbSource1.product.id)
3 prodDesc (productInstance1, dbSource1.product.description)
```

Listagem 5: Exemplo de ABox de mapeamento de modelos

3.3.2 Partições Horizontais

Um particionamento horizontal divide os tuplos de uma mesma tabela por várias fontes. Num sistema de informação federado, isto significa que várias fontes contêm dados sobre um mesmo conceito do modelo global. Cada partição horizontal representa um subconjunto da totalidade dos dados.

Neste modelo, cada instância de uma classe da TBox_G descreve uma partição de dados horizontal. Assim, os tuplos correspondentes à tabela do modelo global são obtidos através da união dos resultados de cada uma das suas instâncias. Caso apenas exista uma instância de uma classe, essa instância descreve a única partição horizontal dos dados, o que é equivalente a não existir particionamento. Na listagem 6 é apresentado um exemplo em que existem duas partições horizontais da tabela "Product".

```
1 Product (productInstance1)
2 prodId (productInstance1, dbSource1.product.id)
3 prodDesc (productInstance1, dbSource1.product.description)
4
5 Product (productInstance2)
6 prodId (productInstance2, dbSource2.produto.id)
7 prodDesc (productInstance2, dbSource2.produto.nome)
```

Listagem 6: Exemplo de ABox com duas partições horizontais

3.3.3 Partições Verticais

O particionamento vertical distribui as colunas de uma tabela por várias outras tabelas. Num sistema de informação federado, isto significa que várias fontes contêm diferentes colunas para um mesmo conceito do modelo de dados global. A junção das colunas de cada uma das partições verticais corresponde à estrutura do conceito do modelo de dados global.

A modelação de uma partição vertical envolve dois passos. O primeiro passo a realizar é o mapeamento directo das colunas. As colunas podem ter origem em tabelas diferentes da mesma fonte de dados ou mesmo em fontes de dados diferentes. Cada tabela do modelo parcial envolvida é interpretada como uma partição vertical dos dados.

O segundo passo é o estabelecimento da relação existente entre as partições verticais. Esta relação é descrita de forma semelhante a uma chave estrangeira:

1. A tabela esquerda da relação;
2. A tabela direita da relação;
3. Pares de colunas que irão constituir a condição de junção.

A TBox relativa a esta informação é apresentada na listagem 7.

```
1  FederatedRelation ⊑ rdf:Bag
2  ColumnRelation ⊑ rdf:Bag
3
4  #a relação entre as partições verticais
5  implicitJoin ⊑ owl:Property
6  Domain(implicitJoin, FederatedEntity)
7  Range(implicitJoin, FederatedRelation)
8
9  tableLeft ⊑ owl:Property
10 Domain(tableLeft, FederatedRelation)
11 Range(tableLeft, dbs:Table)
12
13 tableRight ⊑ owl:Property
14 Domain(tableRight, FederatedRelation)
15 Range(tableRight, dbs:Table)
16
17 #pares de colunas que forma a condição de junção
18 relatedColumns ⊑ owl:Property
19 Domain(relatedColumns, FederatedRelation)
20 Range(relatedColumns, ColumnRelation)
21
22 fromColumn ⊑ owl:Property
23 Domain(fromColumn, ColumnRelation)
24 Range(fromColumn, dbs:Column)
25
26 toColumn ⊑ owl:Property
27 Domain(toColumn, ColumnRelation)
28 Range(toColumn, dbs:Column)
```

Listagem 7: TBox para registar relações entre tabelas

3.3.4 Réplicas

Para além das partições, é possível definir réplicas. As réplicas são definidas ao nível das instâncias das tabelas do modelo global. Isto é, pode-se indicar que duas instâncias de uma tabela descrita no modelo global são réplicas uma da outra. Esta definição é suportada pela propriedade *replic*, cuja definição é apresentada na listagem 8. Esta propriedade é simétrica e transitiva.

```
1  replic ⊑ owl:Property
2  Domain(replic, FederatedEntity)
```

```
Range(replic , FederatedEntity)

#replic é simétrica e transitiva
replic⁻₊
```

Listagem 8: TBox para a definição de réplicas

3.3.5 Colunas Constantes

Quando as fontes de dados são heterogéneas, é comum existir informação em falta ou em formatos incompatíveis. Para ultrapassar estas barreiras é possível inserir valores constantes ou colunas calculadas através da execução de funções.

Para o caso da utilização de constantes, foi criada uma fonte de dados especial que apenas contém a informação dessas constantes. Esta fonte de dados não tem existência física e representa uma tabela virtual com apenas um tuplo, que contém o valor da constante. Por essa razão, esta fonte de dados tem de ser tratada de forma por um adaptador criado para o efeito. Por outro lado, esta solução permite a implementação de uma solução que segue os princípios já existentes no resto do modelo. A listagem 9 apresenta a descrição desta fonte de dados.

```
dbs:Database (srcConstants)
dbs:Table (srcConstants.constantsTable)
dbs:Column (srcConstants.constantsTable.int#3)
dbs:Column (srcConstants.constantsTable.string#NA)
```

Listagem 9: Exemplo de uma ABox para a fonte de dados de constantes

Ao inserir constantes e colunas de outras fontes numa mesma instância, formam-se partições verticais e é necessário estabelecer a relação entre as tabelas. Neste caso, como a tabela com as constantes pode ser vista como uma tabela apenas com uma linha, junta-se esse valor a todas as linhas da outra partição vertical. Isto é conseguido com o estabelecimento da relação entre as tabelas das partições sem indicar colunas para a condição de junção. Na prática, isto é equivalente a um produto cartesiano (*cross join*). A listagem 10 apresenta um exemplo em que é utilizada uma coluna constante.

```
Product (prodInstance1)
prodId (prodInstance1, dbSource1.product.id)
prodDesc (prodInstance1, srcConstants.constantsTable.string#Descricao)

FederatedRelation (relationWithConstants)
tableLeft (relationWithConstants, dbSource1.product)
tableRight (relationWithConstants, srcConstants.constantsTable)
relatedColumns (⊥)

implicitJoin(prodInstance1, relationWithConstants)
```

Listagem 10: Exemplo de uma ABox com utilização de constantes

3.3.6 Colunas Calculadas

Para a utilização de colunas calculadas, é necessário modelar a execução de funções que executem as transformações pretendidas. Modelar o comportamento destas funções na taxonomia daria origem a um modelo complexo e com capacidade para descrever apenas um conjunto limitado de funções.

Assim, adoptou-se a solução de incluir na taxonomia a indicação de qual a função a executar e quais as colunas que são utilizadas como parâmetros, tal como é ilustrado na listagem 11. O comportamento das funções é da responsabilidade das aplicações construídas sobre o modelo.

```
1  Function ⊑ ⊤
2  FunctionCall ⊑ ⊤
3
4  operation ⊑ owl:Property
5  Domain(operation, FunctionCall)
6  Range(operation, Function)
7
8  arguments ⊑ owl:Property
9  Domain(arguments, FunctionCall)
10 Range(arguments, rdf:List) #rdf:List mantém a lista ordenada de argumentos
```

Listagem 11: TBox para modelação de execução de funções

As funções disponíveis são as instâncias da classe *Function*. Instâncias de *FunctionCall* modelam a execução funções com uma lista concreta de argumentos. A propriedade *arguments* contém a lista de colunas, sobre a forma de uma lista *rdf*. Tal como no mapeamento directo de colunas, as colunas usadas podem ser provenientes de tabelas diferentes, desde que a relação entre as tabelas esteja definida.

Para que seja possível mapear uma coluna do modelo global através de uma função, é necessário que o contradomínio das propriedades correspondentes a essa coluna seja a união de *dbs:Column* (mapeamento directo de colunas) e *FunctionCall* (mapeamento através da execução de funções), como apresentado na listagem 12.

```
1 prodId ⊑ owl:Property
2 Domain(prodId, Product)
3 Range(prodId, dbs:Column ⊔ FunctionCall)
```

Listagem 12: Exemplo da TBox de uma coluna com suporte a funções

3.4 Exemplo Ilustrativo

O exemplo que se apresenta nesta secção pretende ilustrar os conceitos apresentados nas secções anteriores. Este exemplo inclui a definição de dois modelos parciais, um modelo global e a associação entre modelos recorrendo a mapeamento directo, partições horizontais, partições verticais, funções e constantes. Para o exemplo, considere-se o seguinte contexto:

> Um *website* realiza vendas de uma cadeia de livrarias e usa uma base de dados relacional para guardar a informação dos livros disponíveis. Pretende-se que este *website* passe também a vender livros cuja informação tem origem numa outra loja. A loja tem a informação dos livros disponíveis registada numa folha de cálculo com apenas três colunas <isbn, name, price>. Existem algumas regras para a informação acedida através desta fonte de dados. Primeiro, o preço dos livros deve incluir uma margem de lucro de 10% para o *website*. A indicação da categoria dos livros não existe na fonte de dados, pelo que deve ser utilizada a informação existente no *website*. Finalmente, os livros devem ter a indicação da loja de origem.
>
> A pesquisa de livros vai passar a ser realizada através de um sistema federado, que apresenta como resultado tuplos com o formato <isbn, name, price, category, store>. A figura 3.2 ilustra este problema.

3.4.1 Modelos parciais

A descrição dos modelos parciais é realizada em duas ABox_P.

A criação de uma ABox_P é realizada aplicando as seguintes regras:

1. É criada uma instância de *dbs:Database*;

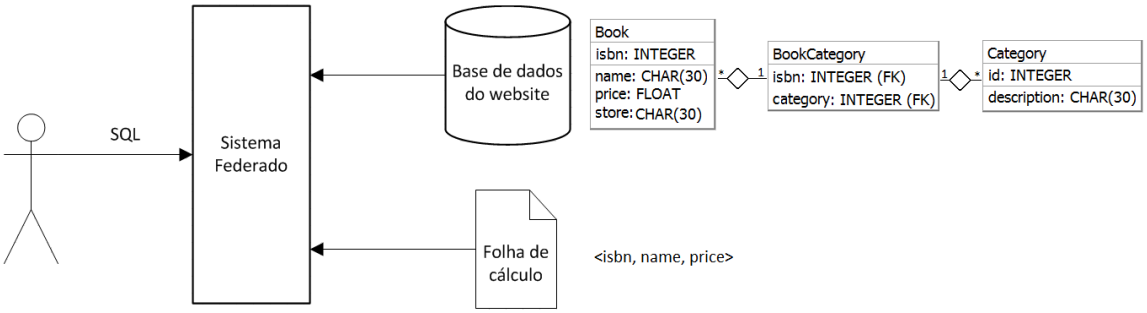


Figura 3.2: Ilustração do problema

2. Para cada entidade, é criada uma instância de *dbs:Table* e uma propriedade *dbs:hasTable* é estabelecida entre a instância de *dbs:Database* e a instância de *dbs:Table* criadas;
3. Para cada coluna, é criada uma instância de *dbs:Column* e uma propriedade *dbs:hasColumn* é estabelecida entre a instância de *dbs:Table* e a instância de *dbs:Column* criadas.

No caso do modelo relacional (*website*), cada "entidade"(cf. linha 2 da regra acima) corresponde a uma tabela. A listagem 13 apresenta o resultado da aplicação destas regras, apenas considerando a tabela *Book*. As regras 2 e 3 aplicam-se a cada uma das restantes tabelas (*BookCategory* e *Category*).

```
1  dbs:Database(wsDb)
2
3  dbs:Table(Book)
4  dbs:hasTable(wsDb, Book)
5
6  dbs:Column(Book.isbn)
7  dbs:hasTable(Book, Book.isbn)
8  dbs:Column(Book.name)
9  dbs:hasTable(Book, Book.name)
10 dbs:Column(Book.price)
11 dbs:hasTable(Book, Book.price)
12 dbs:Column(Book.store)
13 dbs:hasTable(Book, Book.store)
```

Listagem 13: ABox_P que descreve o modelo de dados relacional do *website*

No caso do modelo baseado em ficheiros compatíveis com o *Microsoft Excel 2007* (loja), cada folha (*sheet*) corresponde a uma tabela. A listagem 14 apresenta o resultado da aplicação destas regras.

```
1  dbs:Database(individualSpreadsheet)
2
3  dbs:Table(Spreadsheet1)
4  dbs:hasTable(individualSpreadsheet, Spreadsheet1)
5
6  dbs:Column(Spreadsheet1.isbn)
7  dbs:hasTable(Spreadsheet1, Spreadsheet1.isbn)
8  dbs:Column(Spreadsheet1.name)
9  dbs:hasTable(Spreadsheet1, Spreadsheet1.name)
10 dbs:Column(Spreadsheet1.price)
11 dbs:hasTable(Spreadsheet1, Spreadsheet1.price)
```

Listagem 14: ABox_P que descreve o modelo de dados da folha de cálculo

3.4.2 Modelo global

Para a descrição do modelo global, é criada uma TBox_G. A TBox_G é criada de acordo com as seguintes regras:

1. Para cada tabela, é criada uma subclasse de *FederatedEntity*.
2. Para cada coluna de uma tabela, é criada uma propriedade que tem a classe correspondente à tabela como domínio. O contradomínio destas propriedades é sempre a união entre *dbs:Column* e *FunctionCall*.

No cenário deste exemplo, existe apenas uma tabela com as colunas *isbn, name, price, category* e *store*. A TBox_G resultante é apresentada na listagem 15.

```
1  FederatedEntity ⊑ owl:Thing
2  BookForSale ⊑ FederatedEntity
3
4  isbn ⊑ owl:Property
5  Domain(isbn, BookForSale)
6  Range(isbn, dbs:Column ⊔ FunctionCall)
7
8  name ⊑ owl:Property
9  Domain(name, BookForSale)
10 Range(name, dbs:Column ⊔ FunctionCall)
11
12 price ⊑ owl:Property
13 Domain(price, BookForSale)
14 Range(price, dbs:Column ⊔ FunctionCall)
15
16 category ⊑ owl:Property
17 Domain(category, BookForSale)
18 Range(category, dbs:Column ⊔ FunctionCall)
19
20 store ⊑ owl:Property
21 Domain(store, BookForSale)
22 Range(store, dbs:Column ⊔ FunctionCall)
```

Listagem 15: TBox_G para o exemplo

3.4.3 Mapeamento entre os modelos

As duas fontes de dados contém informações sobre livros, constituindo duas partições horizontais dos dados. São assim criadas duas instâncias da classe *BookForSale*, modelando essas duas partições horizontais. A primeira dessas instâncias vai mapear os elementos da ABox_P da base de dados do *website*. Neste modelo parcial, a informação está dispersa em diversas tabelas, isto é, existem partições verticais dos dados. É então necessário não só realizar o mapeamento das colunas, mas também indicar as relações existentes entre as partições. A listagem 16 apresenta a ABox_G resultante deste mapeamento.

```
#Mapear directamente as colunas
BookForSale (bookInstance1)
isbn (bookInstance1, Book.isbn)
name (bookInstance1, Book.name)
price (bookInstance1, Book.price)
category (bookInstance1, Category.description)
store (bookInstance1, Book.store)

#Criar a relação entre Book e BookCategory
FederatedRelation (relBookBookCategory)
tableLeft (relBookBookCategory, Book)
tableRight (relBookBookCategory, BookCategory)

#Indicar a condição de junção
ColumnsRelation (colRelBookBookCategory)
fromColumn (colRelBookBookCategory, Book.isbn)
toColumn (colRelBookBookCategory, BookCategory.isbn)
relatedColumns (relBookBookCategory, colRelBookBookCategory)

#Adicionar a relação à instância
implicitJoin (bookInstance1, relBookBookCategory)
```

Listagem 16: ABox_G com o mapeamento para a base de dados do *website*

Para além do mapeamento das colunas, é definida a relação entre as tabelas que correspondem às partições verticais. Esta relação é estabelecida em três passos:

- criar a relação entre Book e BookCategory;
- indicar a condição de junção;
- adicionar a relação à instância.

O mapeamento da ABox_P da fonte de dados correspondente à folha de cálculo apresenta outros desafios. Primeiro, para a inserção da margem de lucro vai ser usada uma função que multiplica por 1.1 o preço original. A indicação da categoria dos livros está em falta, pelo que é estabelecida uma partição vertical para realizar o cruzamento com a informação existente no *website*. Finalmente, a indicação da loja é substituída por uma constante. A ABox_G resultante deste processo de mapeamento é apresentada na listagem 17.

```
#Cria instância e mapeia as colunas
BookForSale(bookInstance2)
isbn (bookInstance2, Spreadsheet1.isbn)
name (bookInstance2, Spreadsheet1.name)
category (bookInstance2, Category.description)

#Cria a função para o cálculo do preço
FunctionCall (addMarginProfit)
operation (addMarginProfit, Multiply)
arguments (addMarginProfit, Spreadsheet1.price)
arguments (addMarginProfit, Constants.#1.1)
price (bookInstance2, addMarginProfit)

#Cria a relação entre as tabelas para obter informação de categoria
FederatedRelation (relSSBookCategory)
tableLeft (relSSBookCategory, Book)
tableRight (relSSBookCategory, BookCategory)

#Cria a condição de junção desta relação
ColumnsRelation (colRelBookBookCategory)
fromColumn (colRelBookBookCategory, Book.isbn)
toColumn (colRelBookBookCategory, BookCategory.isbn)
relatedColumns (relSSBookCategory, colRelBookBookCategory)

#Adiciona a relação à instância reutilizando a relação relBookCategoryCategory
implicitJoin (bookInstance2, relSSBookCategory)
implicitJoin (bookInstance2, relBookCategoryCategory)

#Adiciona a relação com a tabela de constantes
FederatedRelation (relSSBookConstants)
tableLeft (relSSBookConstants, Book)
tableRight (relSSBookConstants, Constants)

#Mapeia a coluna store para uma constante
store (bookInstance2, Constants.#individualSeller)
```

Listagem 17: ABox do mapeamento da folha de cálculo

Neste mapeamento, para além do mapeamento directo das colunas, existe o estabelecimento de uma relação entre as tabelas correspondentes às partições verticais, a definição de uma coluna calculada e de uma coluna constante.

O estabelecimento da relação é realizado como no mapeamento da ABox_P da base de dados do *website*. A coluna constante é mapeada para uma coluna da tabela "Constants", que faz parte do modelo parcial de constantes. É também necessário estabelecer a relação com a tabela de constantes. A definição da coluna calculada implica a criação de uma instância de "FunctionCall", a indicação da operação a realizar ("operation") - neste caso a operação "Multiply" - e a lista de argumentos ("arguments") - neste caso, a coluna "Spreadsheet1.price" e a coluna constante Constants.#1.1.

CAPÍTULO 4

INTERROGAÇÃO DO MODELO FEDERADO

De forma a oferecer transparência na execução de interrogações sobre o sistema federado, foi construída uma interface aplicacional (API). Para o efeito, a interface para interrogações é semelhante a uma interface *jdbc* (*Java Database Connectivity*), utilizando a linguagem SQL e retornando os resultados sobre a forma de objectos *RowSet*.

Esta API foi desenvolvida em *Java* e a operação fundamental disponibilizada é:

```
1 RowSet execute(String queryString);
```

A execução de interrogações suporta as seguintes cláusulas SQL:

- **select**.
- **from**.
- **where**.
- **order by**.
- **group by**.
- **having**.

A listagem 18 apresenta o EBNF (*Extended Backus–Naur Form* [29]) simplificado de uma interrogação SQL. Para aumentar a legibilidade não é detalhado o EBNF correspondente às condições (elemento "cond_exp").

```
1  select ::=
2     <SELECT> ( <DISTINCT> )? select_list
3     <FROM> <IDENTIFIER> ( join_expression (ON cond_exp)? )*
4     ( <WHERE> cond_exp )?
5     ( <GROUP> <BY> columns_list )?
6     ( <HAVING> cond_exp )?
7     ( <ORDER> <BY> columns_list )?
8
9  select_list ::=
10    select_column ( <COMMA> select_column )*
11
12 join_expression ::=
13    <INNER>|<CROSS>|<LEFT OUTTER>|<RIGHT OUTTER>|<FULL OUTTER> <JOIN>
14
15 columns_list ::=
16    <IDENTIFIER> ( <COMMA> <IDENTIFIER> )* ( <ASC> | <DESC> )?
17
18 select_column ::=
19    <ASTERISK> | <IDENTIFIER> ( <DOT> <ASTERISK> | <IDENTIFIER> )?
20    ( <AS> <IDENTIFIER> )?
```

Listagem 18: EBNF simplificado de uma interrogação SQL

A arquitectura sobre a qual a execução de interrogações é implementada é a arquitectura apresentada no capítulo 2 e ilustrada na figura 4.1.

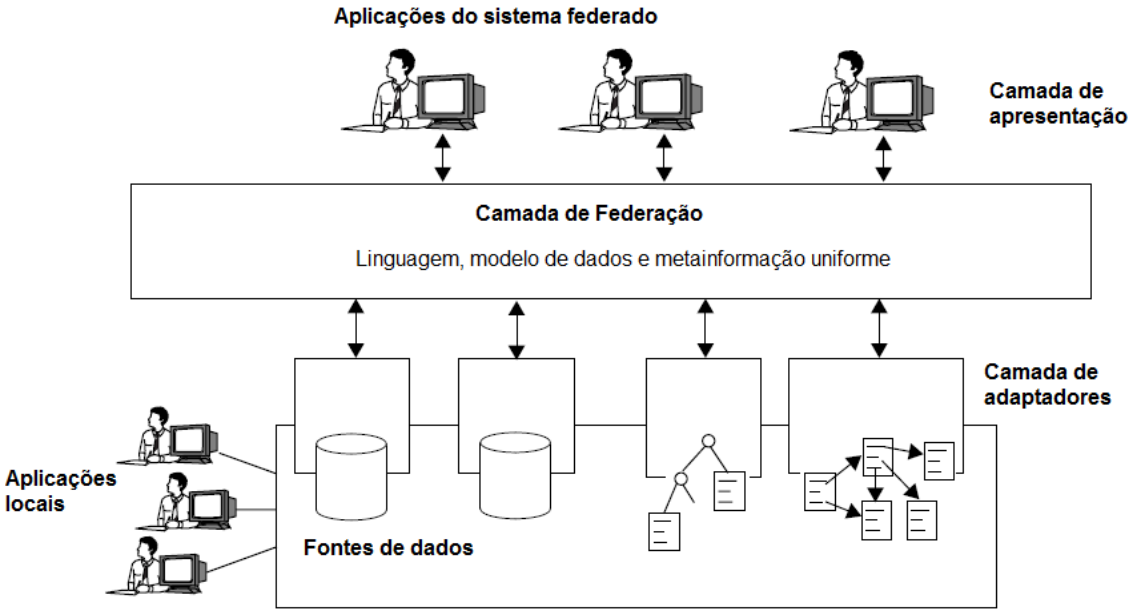


Figura 4.1: Arquitectura de referência (adaptado de [1])

Nesta arquitectura, o executor de interrogações faz parte da camada de federação, utilizando a camada de adaptadores para aceder às fontes de dados.

A execução de uma interrogação está dividida em três fases:

1. **Análise** - tendo como entrada uma *string*, são construídos objectos que representam a estrutura da interrogação.

2. **Planeamento -** é necessário planear a execução da interrogação tendo em conta a modelação do sistema federado. Nesta fase, a interrogação original é decomposta em várias interrogações parciais.
3. **Processamento -** seguindo o planeamento criado no ponto anterior, são executadas as interrogações parciais (através de adaptadores) e é realizado todo o pós-processamento necessário, nomeadamente a execução de operações sobre a globalidade dos dados e que não puderam ser executadas nas fontes de dados.

Este processo está resumido na figura 4.2.

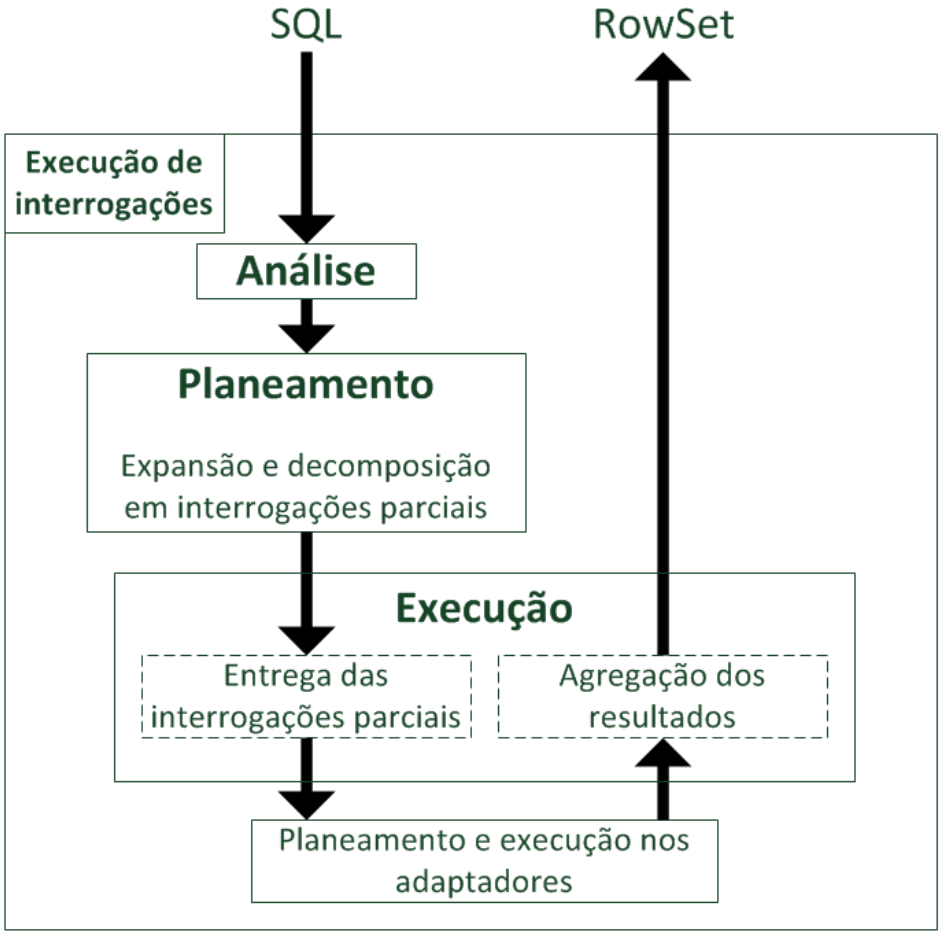


Figura 4.2: Resumo do processo de execução de uma interrogação

É essencial que a implementação não interfira com a autonomia das fontes de dados, o que tem algumas consequências no planeamento e execução das interrogações. Nomeadamente, não é possível manter estruturas auxiliares como estatísticas e índices actualizados, pelo que muitas das técnicas de optimização utilizadas em cenários de bases de dados não podem ser aproveitadas para o cenário dos sistemas de informação federados. As questões de optimização das interrogações não são o foco do trabalho, embora se tenha garantido que, sempre que possível, as operações sejam realizadas junto das fontes de dados, aproveitando as optimizações locais das fontes de dados, reduzindo o tráfego de dados e minimizando a quantidade de dados a processar na camada de federação.

No resto do capítulo apresentam-se os principais pontos da implementação das três fases da execução da interrogação. Na fase de análise da interrogação, discutem-se as hipóteses de implementação de raíz ou do reaproveitamento de soluções existentes, apresentando mais ao pormenor a solução *Eclipse Data Tools* [3] e o modelo de interrogação construído sobre essa solução. Na fase de planeamento da interrogação,

é descrito acesso ao modelo através do motor de inferência *Pellet* [30], a estrutura de um plano de interrogação e as regras aplicadas para a sua construção. Na fase do processamento da interrogação, são apresentadas as operações de detecção de réplicas, interrogação das fontes através de adaptadores, realização de pós processamento como junções, funções, uniões, nomes alternativos, filtros e agregações.

4.1 Análise da Interrogação

Uma interrogação SQL é passada à API sobre a forma de uma *String*, sendo necessário realizar a sua análise. É necessário que a análise das interrogações suporte as cláusulas essenciais do *SQL standard* [31] (*select, from, where, group by, order, having*). Foram consideradas duas alternativas para abordar este problema:

1. Implementar um novo *parser* SQL;
2. Utilizar um *parser* disponibilizado como *open-source* e alterar/acrescentar o necessário para satisfazer os requisitos.

Na implementação de um novo analisador (*parser*) SQL, considerou-se a utilização de uma ferramenta de construção de *parsers* como a *antlr* [32]. Existem, inclusive, alguns exemplos de implementação da linguagem SQL através desta ferramenta. No entanto, devido à complexidade da linguagem SQL, nenhuma das implementações disponíveis era completa, e a criação de uma nova implementação envolveria um esforço demasiado elevado e pouco alinhado com os objectivos traçados.

Optou-se então pela utilização de um analisador de fonte aberta e foram analisadas algumas alternativas:

- SQLJEP [33]: gerado pelo JavaCC [34] mas baseado numa versão incompleta do *standard* SQL, centrando-se mais nas particularidades dos dialectos *Oracle* e *MaxDB*.
- Apache Derby [35]: implementação bastante completa mas criada para funcionar em modo "ligado", pelo que a utilização num sistema federado (sobre um modelo de dados virtual) obrigaria a recorrer a opções rebuscadas de implementação que poderiam gerar comportamentos pouco previsíveis.
- *Eclipse Data Tools*[1] [3]: implementação bastante completa, apesar de gerar um modelo de objectos complexo, pouco documentado e por vezes pouco organizado.

Optou-se por utilizar o parser do pacote *Eclipse Data Tools*, por ter bom suporte ao *standard* da linguagem SQL e por ser possível utilizar directamente.

O código apresentado na listagem 19 mostra como utilizar o pacote *Eclipse Data Tools* para interpretar uma *string* e gerar uma árvore de objectos que representam a interrogação.

```
1  public void parseAndFill(String query) throws SQLParserException,
       SQLParserInternalException
2  {
3      //Get SQLQueryParserManager object
4      //Parameters: product, version
5      //Both parameters as null for sql standard
6      SQLQueryParserManager parserManager = SQLQueryParserManagerProvider.
7              getInstance().getParserManager(null, null);
8
9      //Parse
10     SQLQueryParseResult parseResult = parserManager.parseQuery(query);
11
12     //Get the sql statement
13     QueryStatement resultObject = parseResult.getQueryStatement();
14
15     //Process the sql objects
16     runThroughQueryTree(resultObject);
17 }
```

[1] Eclipse Public License 1.0 [36]

Listagem 19: Utilização do pacote Eclipse Data Tools

A interpretação da interrogação sobre a forma de *string* é realizada pelo *Eclipse Data Tools*, tendo sido acrescentada uma segunda fase da interpretação da interrogação de forma a:

1. Simplificar o modelo de objectos;
2. Complementar os objectos com informação do modelo de dados do sistema federado;
3. Realizar de imediato algumas validações à interrogação, como a utilização de colunas não existentes ou a não indicação das tabelas correctas na cláusula *where*.

As classes resultantes desta análise são mais simples que as classes do *Eclipse Data Tools*. O diagrama de classes resultante da segunda fase da análise é apresentado na figura 4.3. As propriedades e os métodos foram ocultados de forma a aumentar a legibilidade do diagrama.

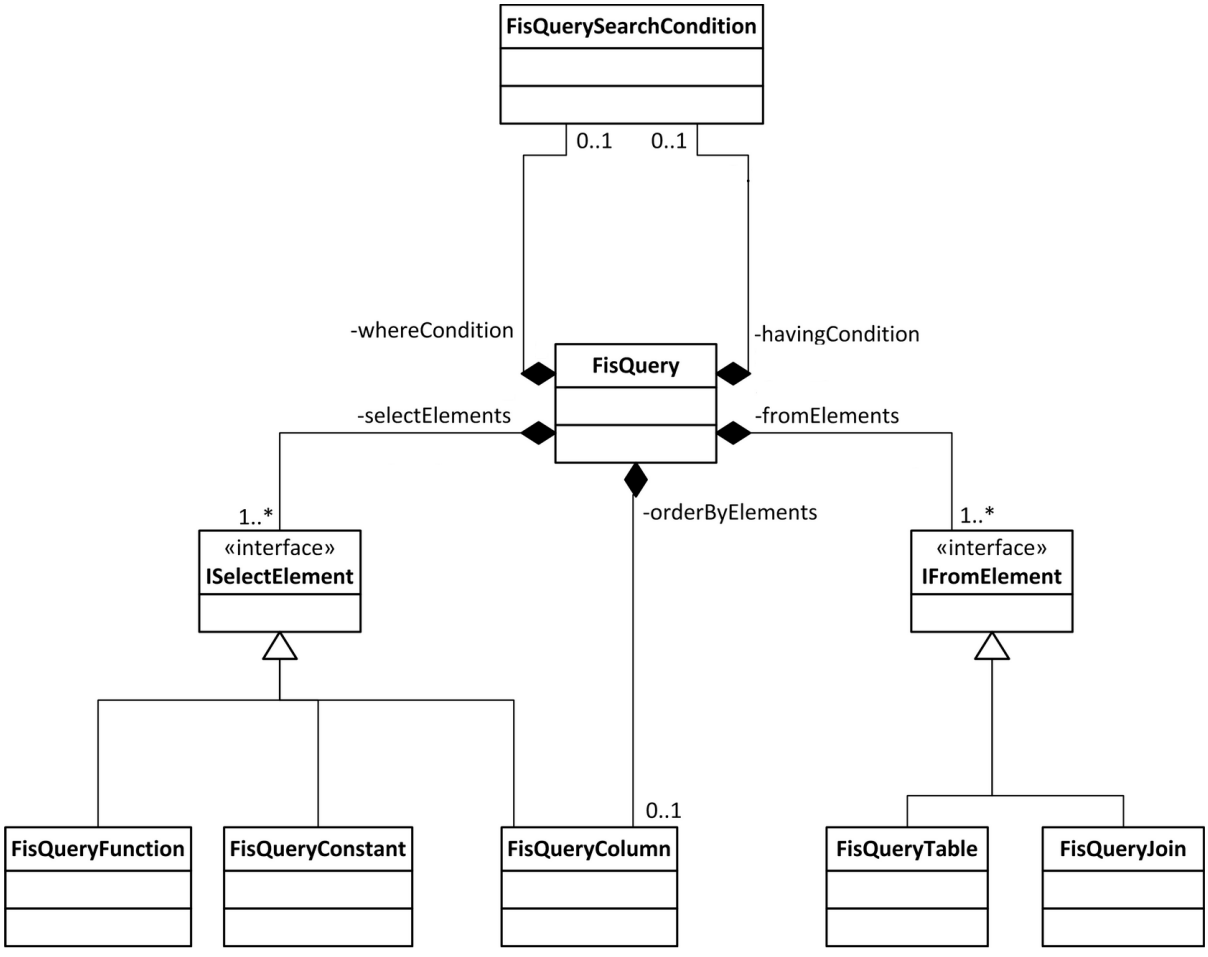


Figura 4.3: Diagrama de classes para as *queries* interpretadas

4.2 Planeamento da Interrogação

O planeamento das interrogações tem como objectivo definir os passos necessários para a execução de uma interrogação. Cada fase do plano é vista como uma interrogação parcial, com o seu próprio

planeamento.

O planeamento é realizado em quatro passos:

1. Expandir a interrogação com a informação do modelo.
2. Criar sub interrogações executáveis nas fontes de dados.
3. Planear o pós-processamento necessário para juntar os resultados de cada uma das sub interrogações locais.
4. Planear as operações que têm de ser realizadas como pós-processamento de forma centralizada, e.g., junções globais, alguns filtros e agregações.

4.2.1 Acesso aos Modelos

No processo de planeamento de uma interrogação, é necessário aceder ao modelo que descreve o sistema federado para alterar, complementar ou dividir a interrogação. O modelo está descrito em linguagem OWL e tem de ser acedido em código Java. No entanto esse acesso não é feito de forma directa, já que é necessário ter em conta a semântica que é proporcionada pela linguagem OWL, nomeadamente inferência baseada na lógica de descrição (Description Logic - DL [37]). Assim, é utilizado um motor de inferência, que conhece e processa a semântica da linguagem OWL. Foi adoptado o motor de inferência *Pellet*[2] [30] permite aceder aos dados através de diversas interfaces como a interface OWLAPI[39] e a interface *Jena* [40]. Ambas permitem aceder a taxonomias escritas em OWL através de modelos de objectos, mas a interface *Jena* permite também a utilização da linguagem de interrogação SPARQL-DL [41].

A linguagem SPARQL-DL tem como origem a linguagem SPARQL, utilizada para interrogar grafos RDF. A SPARQL-DL é em tudo semelhante à SPARQL, mas realiza as pesquisas considerando a inferência DL.

A listagem que se segue apresenta uma parcela de uma taxonomia descrita em DL. Neste exemplo, diz-se que "instancia1" é uma réplica de "instancia2". Note-se que "replic" é uma propriedade simétrica e transitiva (cf. secção 3.3.4).

```
1 replic(instancia1, instancia2)
```

A seguinte interrogação pode ser executada em SPARQL ou SPARQL-DL, sendo válida nas duas linguagens. O objectivo é obter as réplicas de "instancia2". Os elementos iniciados pelo caracter "?" são as variáveis da interrogação.

```
1 SELECT ?replic
2 WHERE { instancia2 replic ?replic }
```

O resultado da execução da interrogação em SPARQL é vazio, enquanto em SPARQL-DL o resultado é "fm:instancia1". A diferença dos resultados está no facto de as interrogações SPARQL apenas terem em conta o que existe directamente no modelo, enquanto as interrogações SPARQL-DL realizam inferências sobre o modelo, obtendo informações que não têm de ser directamente inseridas no modelo. Neste caso, "replic"é simétrica e portanto "replic(instancia2, instancia1)"é possível de inferir de "replic(instancia1, instancia2)".

O código para executar uma interrogação SPARQL-DL através da interface *Jena* é apresentado na listagem 20.

[2]GNU Affero General Public License, Version 3 [38]

```
public ResultSet getReplicsOfInstance(String instance) {
    //Create the query to be executed
    //The query is in a separate file
    final String queryFile = "file:" + getSparqlDirectory() +
        "FindReplicsOfInstance.sparql";
    Query q = QueryFactory.read( queryFile );

    //Create the execution environment over an existing model
    //The environment will use SparqlDL
    QueryExecution qe = SparqlDLExecutionFactory.create( q,getRdfsModel());

    //Set the variables of the query
    Resource parameterInstance = getRdfsModel().getResource(instance);

    QuerySolutionMap qs;
    qs = new QuerySolutionMap();
    qs.add("instance", parameterInstance);
    qe.setInitialBinding(qs);

    //Execute and get results
    ResultSet rs = qe.execSelect();
    return rs;
}
```

Listagem 20: Executar uma interrogação SPARQL-DL através da interface *Jena*

4.2.2 Regras Aplicadas no Planeamento

Partindo dos objectos gerado pela análise da interrogação interpretada, é aplicado um conjunto de regras dando origem a uma lista de planos com base num objecto *FisQueryMasterPlan*. Este objecto contém um ou mais objectos que implementam a interface *FisQueryPlan* (base para todos os planos de execução). Existem quatro classes que implementam esta interface:

- *FisQueryMasterPlan*: Plano de interrogação referente a uma interrogação completa ou a uma interrogação aninhada (*inner query*).
- *FisTablePlan*: Plano de interrogação referente a uma tabela do esquema global.
- *FisInstancePlan*: Plano de interrogação referente a uma instância de uma tabela do esquema global.
- *FisSourcePlan*: Plano de interrogação executável numa fonte de dados.

O diagrama de classes desta componente é apresentado na figura 4.4.

As regras aplicadas no planeamento são apresentadas de seguida. As consultas ao modelo são realizadas através de interrogações SPARQL-DL que constam do anexo A. As regras são acompanhadas de um exemplo da sua aplicação sobre a interrogação da listagem 21. A listagem 21 apresenta a interrogação utilizada como exemplo.

```
SELECT A.COL1, B.COL2
FROM A JOIN B
ON A.ID = B.ID
WHERE A.ID < 1000 AND A.COL3 < A.COL4
```

Listagem 21: Exemplo de uma interrogação para aplicação das regras de planeamento

Neste exemplo consideram.se os seguintes pressupostos sobre o modelo:

1. Existe uma instância da tabela global “A”;
2. Existe uma instância da tabela global “B”;

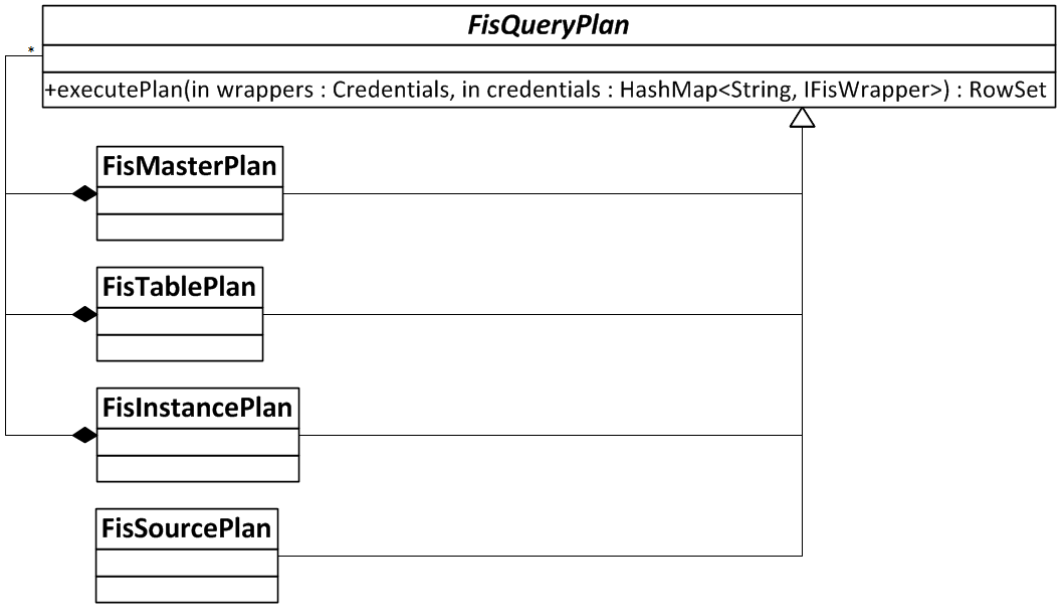


Figura 4.4: Diagrama de classes para os planos de execução de interrogações

3. Existe uma partição vertical entre as tabelas parciais "table1" e "table2". A condição de junção utilizada é "table1.id=table2.id";
4. As colunas da tabela global "A" estão definidas como:
   (a) A.ID = table1.id *(Mapeamento directo)*
   (b) A.COL1 = FUNCAO(table1.col1, table1.col2) *(Coluna calculada)*
   (c) A.COL3 = table1.col3 *(Mapeamento directo)*
   (d) A.COL4 = table2.col4 *(Mapeamento directo)*
5. As colunas da tabela global "B" estão definidas como:
   (a) B.ID = table3.id *(Mapeamento directo)*
   (b) B.COL2 = table3.col2 *(Mapeamento directo)*
6. As tabelas parciais "table1", "table2" e "table3" existem em fontes de dados distintas.

Regra 1. Por cada interrogação sobre o modelo federado, é criado um objecto do tipo *FisQueryMasterPlan*.

Regras aplicada no planeamento dos objectos *FisMasterPlan*

Regra 2. Por cada tabela global envolvida na interrogação, é criado um objecto do tipo *FisTablePlan*. Cada um desses objectos contém uma interrogação parcial contendo apenas as colunas existente na tabela respectiva.

No exemplo da listagem 22, são criados dois objectos: um para a interrogação sobre a tabela global "A" e outro para a interrogação sobre a tabela global "B".

```
1 SELECT A.COL1
2 FROM A
3 WHERE A.ID < 1000 AND A.COL3 < A.COL4
4
5 SELECT B.COL2
6 FROM B
```

Listagem 22: Aplicação da regra 2 do planeamento sobre a interrogação de exemplo

Regra 3. As colunas necessárias para a realização das operações globais de junção, filtragem e agrupamento são adicionadas à cláusula de selecção do objecto *FisTablePlan* correspondente à tabela que contém cada coluna.

No exemplo da listagem 23, são acrescentadas as colunas "A.ID" e B.ID às cláusulas de selecção, para ser possível juntar as tabelas globais "A" e "B".

```
1 SELECT A.COL1, A.ID
2 FROM A
3 WHERE A.ID < 1000 AND A.COL3 < A.COL4
4
5 SELECT B.COL2, B.ID
6 FROM B
```

Listagem 23: Aplicação da regra 3 do planeamento sobre a interrogação de exemplo

Regras aplicada no planeamento dos objectos *FisTablePlan*

Regra 4. Por cada instância existente para a tabela global (anexo A.1), é criado um objecto do tipo *FisInstancePlan*.

No planeamento de FisInstancePlan

Regra 5. Se existirem partições verticais (anexo A.2), a cláusula de junção das partições é adicionada à interrogação inicial (anexo A.3).

No exemplo da listagem 24, é introduzida o *join* entre as partições verticais da tabela global "A" (pressuposto 3).

```
1 #nota:para a tabela global "A" existe uma partição parcial entre table1 e table2
2
3 SELECT A.COL1, A.ID
4 FROM table1 JOIN table2
5 ON table1.id = table2.id
6 WHERE A.ID < 1000 AND A.COL3 < A.COL4
7
8 SELECT B.COL2, B.ID
9 FROM table3
```

Listagem 24: Aplicação da regra 5 do planeamento sobre a interrogação de exemplo

Regra 6. Cada coluna do modelo global é substituída pela coluna do modelo parcial correspondente (anexo A.2).

No exemplo da listagem 25, as colunas globais são substituídas pelos seus mapeamentos nos modelos parciais (pressupostos 4 e 5). Note-se o caso do mapeamento da coluna "A.COL1" realizado como uma coluna calculada (pressuposto 4.a).

```
1 #nota: A.COL1 está mapeada para uma função FUNCAO(table1.col1, table1.col2)
2
3 SELECT FUNCAO(table1.col1, table1.col2), table1.id
4 FROM table1 JOIN table2
5 ON table1.id = table2.id
6 WHERE table1.id < 1000 AND table1.col3 < table2.col4
7
8 SELECT table3.col2, table3.id
9 FROM table3
```

Listagem 25: Aplicação da regra 6 do planeamento sobre a interrogação de exemplo

Regra 7. Se uma coluna do modelo global está mapeada para uma função, as colunas utilizadas como argumento são adicionadas à cláusula de selecção da interrogação (anexo A.4) e a função fica registada para ser realizada na fase de pós-processamento.

No exemplo da listagem 26, a função "FUNCAO(table1.col1, table1.col2)" é substituída pelas colunas necessárias para a sua execução: "table1.col1" e "table1.col2".

```
1 SELECT table1.col1, table1.col2, table1.id
2 FROM table1 JOIN table2
3 ON table1.id = table2.id
4 WHERE table1.id < 1000 AND table1.col3 < table2.col4
5
6 SELECT table3.col2, table3.id
7 FROM table3
```

Listagem 26: Aplicação da regra 7 do planeamento sobre a interrogação de exemplo

Regra 8. Para cada fonte de dados da instância, é criado um objecto do tipo *FisSourcePlan* que contém a sub-interrogação apenas com as colunas e tabelas que lhe correspondem.

No exemplo da listagem 27, são criados objectos para conter as sub-interrogações individuais para cada fonte de dados, dividindo a interrogação sobre "table1" e "table2" (pressuposto 6).

```
1 SELECT table1.col1, table1.col2, table1.id
2 FROM table1
3 WHERE table1.id < 1000 AND table1.col3 < table2.col4
4
5 SELECT table2.id
6 FROM table2
7 WHERE table1.id < 1000 AND table1.col3 < table2.col4
8
9
10 SELECT table3.col2, table3.id
11 FROM table3
```

Listagem 27: Aplicação da regra 8 do planeamento sobre a interrogação de exemplo

Regras aplicada no planeamento dos objectos *FisSourcePlan*

Regra 9. A cláusula *where* é calculada de forma a conter a maior parcela possível da cláusula *where* global que pode ser executada nessa fonte de dados.

Esta regra tem como objectivo diminuir o número de tuplos que têm de ser processados na camada de federação. Assume-se que a realização destas operações junto das fontes de dados é mais eficiente pois implica menos tráfego de dados, maior distribuição do trabalho e é possível tirar partido de optimizações específicas das fontes (por exemplo, a utilização de índices).

Para ilustrar a forma de verificar se uma condição é válida no contexto de uma fonte de dados, considere-se como exemplo a condição A=B AND C=D OR E=F, em que cada letra pode ser uma coluna ou uma

constante. Esta condição pode ser reescrita para reflectir as precedências dos operadores envolvidos (por ordem decrescente de precedência: =, NOT, AND, OR) como [(A=B) AND (C=D)] OR (E=F). Esta condição pode também ser vista como uma árvore binária, como é apresentado na figura 4.5,

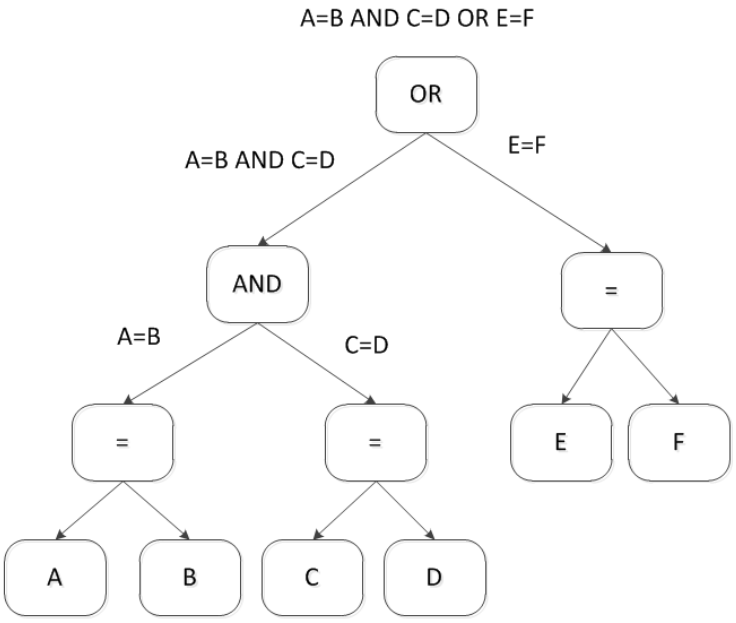


Figura 4.5: Árvore da condição A=B AND C=D OR E=F

O objectivo é processar esta árvore e determinar os elementos válidos na interrogação de cada fonte de dados.

Uma folha da árvore é válida se for uma coluna existente na fonte de dados ou se for uma constante. A validade de um nó depende da condição a validar nesse nó, e acordo com as seguintes regras:

- AND - válido se ambos os seus ramos forem válidos.
- OR - válido se algum dos seus ramos for válido. Se apenas um dos ramos for válido, o outro ramo é descartado.
- NOT - válido se o seu ramo descendente for válido.
- Operadores relacionais - válido se ambos os ramos forem válidos.

Um ramo da árvore é válido se todos os nós que dele derivam forem válidos.

A listagem 28 ilustra a aplicação desta regra. Neste caso, não é possível avaliar "table1.col3 < table2.col4" na mesma fonte de dados, pelo que a condição é substituída pelas colunas que permitem a condição posteriormente. Neste caso, a coluna "col3" é acrescentada à cláusula de selecção da interrogação sobre a tabela "table1" e a coluna "col4" é acrescentada à cláusula de selecção da interrogação sobre a tabela "table2".

```
1  #nota: table1 e table2 estão em fontes de dados diferentes
2  condição:
3      AND(
4          LESS_THAN( table1.id, 1000 ),
5          LESS_THAN( table1.col3, table2.col4 )
6      )
7
8  SELECT table1.col1, table1.col2, table1.id, table1.col3
9  FROM table1
10 WHERE table1.id < 1000
```

```
11
12 SELECT table2.id, table2.col4
13 FROM table2
14
15 SELECT table3.col2, table3.id
16 FROM table3
```

Listagem 28: Aplicação da regra 9 do planeamento sobre a interrogação de exemplo

4.3 Processamento da Interrogação

O processamento das interrogações utiliza os objectos resultantes do planeamento (cf. figura 4.4) como base para o seu funcionamento. Existem duas classes essenciais de operações que são realizadas pelo executor de interrogações:

1. Entrega das interrogações parciais aos adaptadores das fontes de dados;
2. Realização centralizada das operações que não podem ser realizadas nas fontes de dados.

A figura 4.6 apresenta o diagrama de sequência das operações realizadas pelo processador de interrogações. O diagrama serve de guião à apresentação das operações.

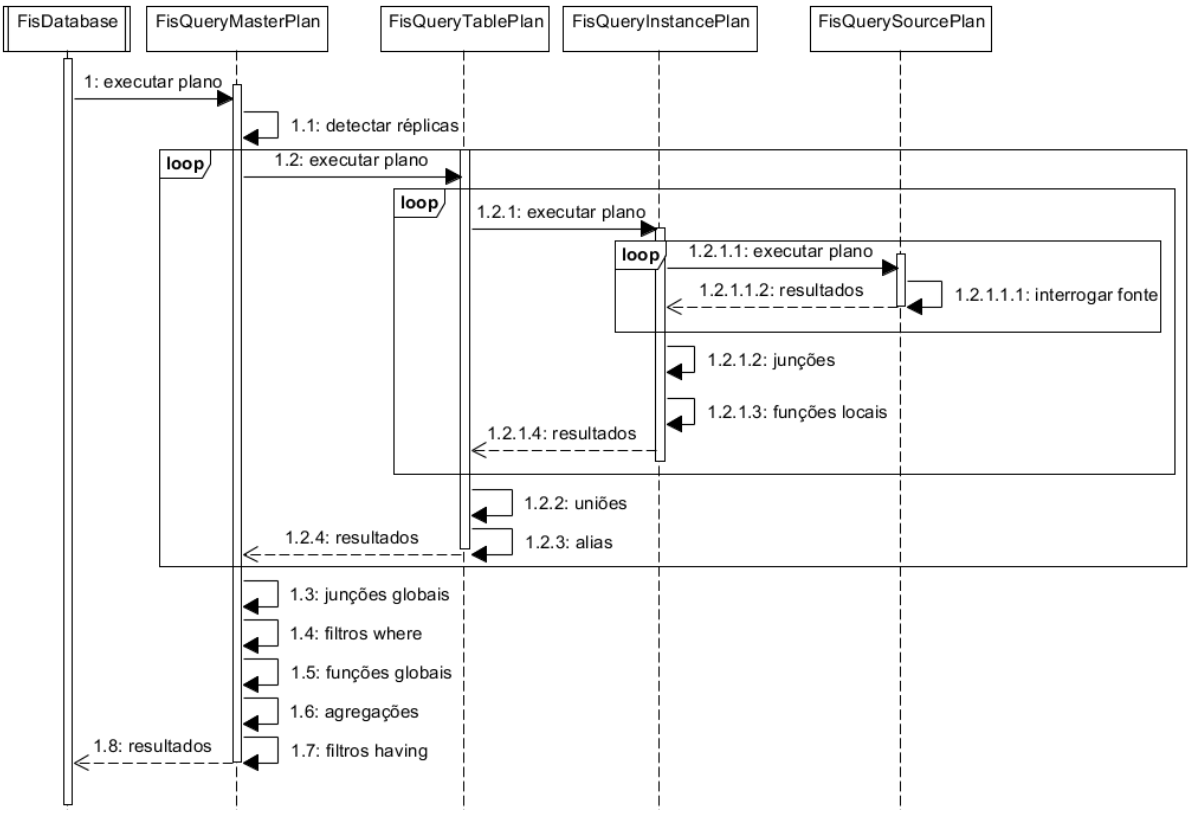


Figura 4.6: Diagrama de sequência do processador de interrogações

4.3.1 Detectar réplicas

As réplicas estão definidas ao nível das instâncias de uma tabela do modelo global (anexo A.5). Se o acesso a uma dessas instâncias falhar, é possível tentar aceder a uma das suas réplicas. Basta que uma

das réplicas seja acedida com sucesso para que a interrogação possa ser realizada com sucesso.

Por outro lado, se o acesso a uma instância for realizado com sucesso, todas os acessos às suas réplicas são considerados completos, de forma a não aceder desnecessariamente a informação duplicada.

4.3.2 Interrogar fonte

As interrogações parciais são realizadas sempre através dos adaptadores (wrappers). Estes componentes, que implementam a interface presente na listagem 29, são responsáveis por realizar o acesso aos dados resolvendo o problema da heterogeneidade física dos dados: os dados retornados através do *RowSet* estão sempre no formato relacional.

```
1 public interface IFisWrapper {
2    public RowSet executePlan(FisQuery query, Credentials credentials) throws
         FISWrapperException;
3 }
```

Listagem 29: Interface dos adaptadores de fontes de dados

Adaptadores implementados

Foram implementados dois adaptadores para servirem de suporte à validação aos restantes componentes. Os adaptadores criados permitem o acesso a fontes de dados relacionais através da interface JDBC e a ficheiros suportados pela aplicação *Microsoft Excel 2007*.

O acesso a fontes de dados *jdbc* é realizado de forma directa: os objectos do tipo *FisQuery* geram *strings SQL standard* que podem ser executadas através de uma ligação *jdbc* e o resultado pode ser retornado directamente.

O acesso a fontes de dados *Excel* tem de ser realizado através de uma API que permita o acesso a ficheiros deste tipo. Para o efeito foi utilizada a biblioteca *POI*[3] [43]. O acesso através desta biblioteca é feito ao nível de folhas (*sheets*), colunas e células, como é possível verificar na listagem 30.

```
1 Sheet sheet = null;
2 Table table = null;
3 Column column = null;
4
5 //for each sheet...
6 for(int i = 0; i < wb.getNumberOfSheets(); ++i)
7 {
8   sheet = wb.getSheetAt(i);
9
10  for (Cell cell : row) {
11    CellReference cellRef = new CellReference(row.getRowNum(), cell.getColumnIndex());
12
13    //do something with cell
14
15  }
16 }
```

Listagem 30: Aceder a células num ficheiro *xlsx*

Construção de novos adaptadores

Um novo adaptador tem de implementar a interface *IFisWrapper* (listagem 29) e tem como requisito as operações necessárias para a imposição das cláusulas *select* e *from*. As restantes cláusulas da interrogação

[3] Apache License, Version 2.0 [42]

não são obrigatórias mas devem ser implementadas de forma a diminuir o tráfego entre componentes e o trabalho realizado na camada de federação.

Um novo adaptador tem de ser configurado através do ficheiro *wrappers.properties*. O ficheiro de configuração para a utilização dos dois adaptadores implementados é apresentado na listagem 31.

```
1 wrappers=excel, mysql
2
3 excel_file=FisExtensionsExample.jar
4 excel_class=edu.ipl.isel.deetc.fis.extend.wrappers.ExcelPOIWrapper
5
6 mysql_file=FisApi.jar
7 mysql_class=edu.ipl.isel.deetc.fis.wrappers.MysqlJdbcWrapper
```

Listagem 31: Configuração dos adaptadores

4.3.3 Junções locais, inter esquema e globais

Sempre que possível, tenta-se que as junções sejam realizadas nas fontes de dados. No entanto, existem várias situações em que isso não é possível, pois os dados a juntar provêm de fontes de dados distintas. Nesses casos, é necessário realizar as junções em memória.

O pacote *javax.sql.rowset* contém a interface *JoinRowSet* para dar suporte a objectos *RowSet* que permitam realizar junções em modo desligado das fontes. No entanto, a única implementação disponibilizada neste pacote - *JoinRowSetImpl* - apenas suporta a variante *inner join*, pelo que não pode ser utilizada directamente. Foi por isso necessário realizar a implementação das junções de raíz.

Na escolha do algoritmo a implementar, foram consideradas as duas variantes tipicamente utilizadas pelos SGBDs para realizar este tipo de operações:

- *Nested-loop join*: dois ciclos aninhados percorrem todas as colunas das tabelas a juntar, sendo necessário percorrer a tabela interior para cada linha da tabela exterior.
- *Merge-scan join*: as tabelas a juntar estão ordenadas pela chave de junção, sendo cada tabela percorrida apenas uma vez.

A utilização do *merge-scan loop* tem o problema de ser necessária a ordenação dos dados. Não sendo possível a utilização de índices, as operações de ordenação em memória têm um custo muito elevado, sendo mais viável optar pela implementação de um algoritmo do tipo *nested-loop join*, apesar do seu custo também elevado.

O algoritmo implementado executa dois ciclos aninhados em que, para cada linha da tabela percorrida pelo ciclo interno, é verificado o predicado de junção. A tabela a percorrer pelo ciclo interno é a tabela de menor dimensão, de forma a optimizar a execução da junção.

A implementação deste algoritmo tem algumas diferenças consoante o tipo de junção a efectuar:

- *Cross join* - junção de todas as linhas das duas tabelas. Não é verificado nenhum predicado.
- *Inner join* - junção de todas as linhas das duas tabelas que cumpram o predicado.
- *Left outter join* - como o tipo *Inner join*, com a garantia que todas as linhas da tabela à esquerda estão presentes no resultado. Para além das linhas que cumpram o predicado, também são incluídas todas as linha da tabela à esquerda que não cumprem esse predicado.
- *Right outter join* - como o tipo *Inner join*, com a garantia que todas as linhas da tabela à direita estão presentes no resultado. Para além das linhas que cumpram o predicado, também são incluídas todas as linha da tabela à direita que não cumprem esse predicado.

- *Full outter join* - como o tipo *Inner join*, com a garantia que todas as linhas das duas tabelas estão presentes no resultado. Para além das linhas que cumpram o predicado, são sempre incluídas as restantes linhas de ambas as tabelas.

4.3.4 Funções

As funções a executar podem ser de uma de duas categorias:

- Funções de linha - aplicadas sobre cada linha da tabela;
- Funções de agregado - aplicadas sobre um conjunto de linhas agrupadas pela cláusula de agregação *group by*.

Uma função de linha implementa a interface apresentada na listagem 32 e realiza, de uma só vez, as suas operações sobre a sua lista de parâmetros.

```
1 public interface IFISRowFunction extends IFISFunction {
2     public Object execute(List parameters) throws FISExecutionException;
3 }
```

Listagem 32: Interface para funções de linha

Um função de agregação implementa a interface apresentada na listagem 33 e realiza, de forma faseada, as suas operações sobre os dados de cada linha. Para obter o valor final da sua execução, é necessário chamar o método *getFinalResult*.

```
1 public interface IFISAgregationFunction extends IFISFunction {
2     public void initialize();
3     public void accumulate(Object parameter) throws FISExecutionException;
4     public Object getFinalResult();
5
6 }
```

Listagem 33: Interface para funções de agregação

Novas funções podem ser adicionadas. Uma nova função tem de implementar a interface *IFISRowFunction* (listagem 32) ou a interface *IFISAgregationFunction* (listagem 33).

Uma nova função tem de ser configurado através do ficheiro *functions.properties*. O ficheiro de configuração para a utilização da função *add* é apresentado na listagem 34.

```
1 functions=add
2 add_file=FisExtensionsExample.jar
3 add_class=edu.ipl.isel.deetc.fis.extend.function.FisAdd
```

Listagem 34: Configuração da função *add*

4.3.5 Uniões

Os objectos *RowSet* não disponibilizam a operação de união e, devido à forma como os objectos do pacote *javax.sql.rowset* estão implementados, não é possível garantir de que forma os dados estão a ser guardados, tornando-se difícil realizar a operação de forma eficiente.

A solução passou por realizar a união de objectos *RowSet* copiando linhas entre objectos. Isto implica que todas as linhas sejam percorridas de forma a serem copiadas. O objecto percorrido é sempre o menor dos objectos a unir em cada momento.

4.3.6 Agregações

A primeira acção a realizar para a realização de uma agregação é a ordenação dos dados. Depois de os dados estarem ordenados pelas colunas de agregação, são removidas as colunas repetidas e, se necessário, executadas as funções de agregação.

A ordenação dos dados é, no entanto, uma operação problemática. Mais uma vez, a inexistência de objectos auxiliares como índices dos dados, limita-nos a escolha de algoritmos. Os objectos *RowSet* não implementam acesso directo, pelo que a escolha do algoritmo teve também esse factor em conta. O algoritmo utilizado foi o *mergesort* [44], que tem um custo *O(n log(n))*. No entanto, para grandes volumes de dados, qualquer algoritmo de ordenação em memória terá problemas de desempenho ao nível de tempo e do espaço ocupado.

4.3.7 Nome Alternativo (*Alias*)

Sendo utilizados nomes alternativos numa interrogação, há momentos em que é necessário alterar o nome das colunas, que passam a ser conhecidas pelo seu *alias*. A alteração do nome de uma coluna num objecto *rowset* implica a alteração da sua *metadata*, alteração essa que não pode ser realizada sobre um objecto preenchido. Por essa razão, tenta-se realizar esta alteração de nomes juntamente com outras operações que obriguem a utilizar novos objectos *rowset*, como por exemplo a realização da união entre objectos.

4.3.8 Filtros

O pacote *javax.sql.rowset* contém a interface *FilteredRowSet* para dar suporte a objectos *RowSet* que possam ser filtrados em modo desligado das fontes. A implementação de referência incluída no pacote é a classe *FilteredRowSetImpl* e deixa a decisão de quais as linhas a filtrar para um objecto que implemente a interface *Predicate*, apresentada na listagem 35.

```
1  public interface Predicate {
2        //utilizado para validar o predicado sobre um objecto preenchido
3      public boolean evaluate(RowSet rs);
4
5        //utilizados ao inserir novas linhas
6      public boolean evaluate(Object value, int column) throws SQLException;
7      public boolean evaluate(Object value, String columnName) throws SQLException;
8  }
```

Listagem 35: Interface javax.sql.rowset.Predicate

Na fase de análise da interrogação, são criados objectos do tipo *FisQuerySearchCondition* para conter as informações dos predicados. Para validar condições contidas nestes objectos, foi implementada a classe *FisQueryPredicate* (listagem 36).

```
1  public class FisQueryPredicate implements Predicate, Cloneable {
2      protected FisQuerySearchCondition _conditionToEvaluate;
3
4      public FisQueryPredicate(FisQuerySearchCondition conditionToEvaluate) { ... }
5
6      public boolean evaluate(RowSet rs) { ... }
7
8      @Override
9      public Object clone() throws CloneNotSupportedException { ... }
10 }
```

Listagem 36: Classe FisQueryPredicate

CAPÍTULO 5

APLICAÇÃO DE CRIAÇÃO DO MODELO

A aplicação de criação do modelo do sistema tem como objectivo permitir a criação do modelo de descrição do sistema federado (do capítulo 3) sem ser necessário ter conhecimentos sobre a linguagem OWL. Também se pretende simplificar certas partes da criação do modelo de forma a diminuir o trabalho do utilizador. A existência de uma aplicação deste género permite ainda reduzir os erros na construção do modelo, ao proporcionar auxílios visuais, como a apresentação apenas da informação válida num contexto e a validação de consistência no momento da inserção de novos elementos.

Tal como o modelo de dados, a aplicação está organizada em três componentes (figura 5.1):

1. Definição dos modelos de dados parciais (área 1);
2. Definição do modelo de dados global (área 2);
3. Mapeamento entre os modelos de dados (área 3).

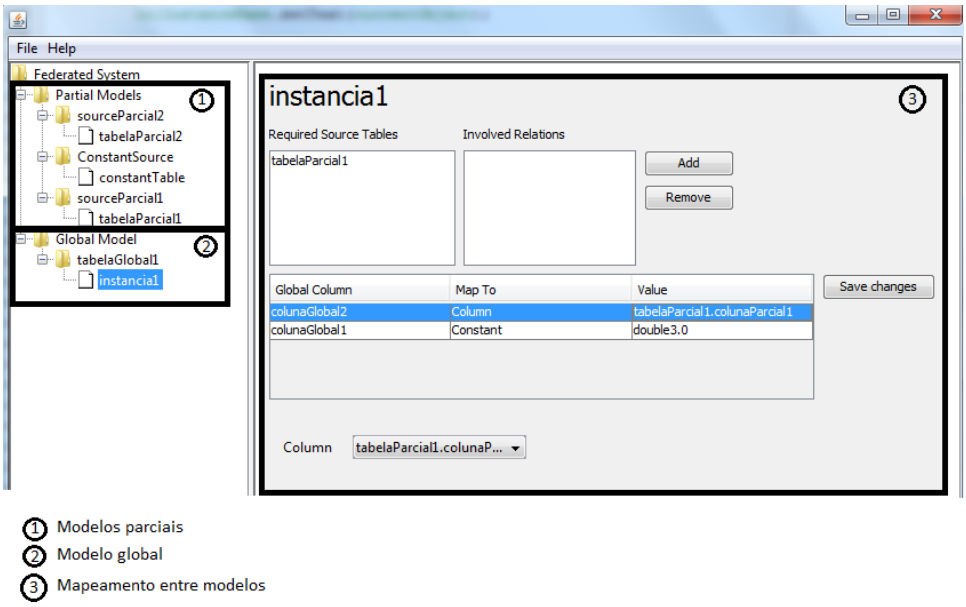


Figura 5.1: Organização geral da aplicação

No resto deste capítulo apresentam-se as diferentes estratégias utilizadas no acesso ao modelo do sistema, a forma como são criados os modelos parciais e o modelo global e ainda o suporte ao mapeamento entre os modelos. São apresentadas diversas alternativas de acesso ao modelo do sistema através de código gerado pela ferramenta *Protégé*, das interfaces *OWLAPI* e *Jena* e das linguagens *SPARQL* e *SPARQL-DL*. Na criação dos modelos de dados são apresentados os ecrãs que permitem a sua criação manual e um módulo de importação automática de modelos parciais. Por fim, é apresentado o suporte disponibilizado para o mapeamento entre os modelos parciais e o modelo global.

5.1 Acesso aos Modelos

A componente do modelo de dados que descreve os modelos parciais tem uma TBox_P fixa, i.e, não alterável pelo utilizador. A componente que descreve o modelo global tem uma TBox_G que pode ser alterada.

O acesso à descrição dos modelos parciais é realizado através de código gerado pela ferramenta *Protégé* e modificado consoante necessário. Este código é constituído por um conjunto de classes de acesso às classes OWL definidas no modelo (figura 5.2) e contém métodos de leitura e escrita nas diversas propriedades. A base de todos as classes geradas é a classe *AbstractCodeGeneratorIndividual* que forma uma camada de abstracção de acesso à interface *OWLAPI*. Para além disso, é também disponibilizada uma *factory* para a criação de novas instâncias. A listagem 37 mostra como se pode criar uma coluna e adicionar valores às suas propriedades através destas classes.

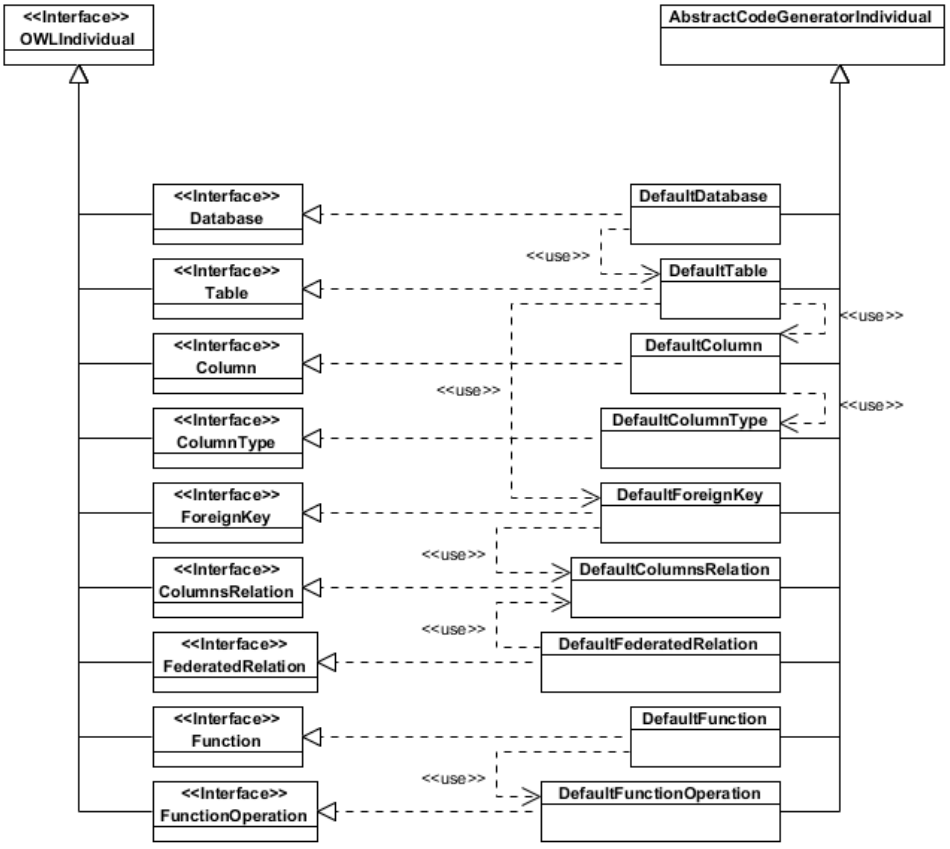


Figura 5.2: Classes utilizadas para o acesso aos elementos fixos da taxonomia

```
1  //Create a factory for a OWLModel object
2  FisFactory factory = new FisFactory(ontology.getOwlModel());
3
4  //Create the column
5  Column col = factory.createColumn(strColumnName);
6
7  //Set the column properties
8  col.setLength(lenght);
9  col.setScale(scale);
10 col.setColumnAccess(columnAccess);
```

Listagem 37: Criar uma coluna no modelo e dar valores às suas propriedades

O acesso à descrição do modelo global é realizado directamente através da interface *OWLAPI*, já que a geração de código não é viável para as classes que só são conhecidas em tempo de execução. A listagem 38 apresenta a criação de uma tabela através da *OWLAPI*.

```
1  public boolean addGlobalTable(String table)
2  {
3     //get active OWL model
4     OWLModel owlModel = Global.getCurrentProject().getOntology().getOwlModel();
5
6     OWLNamedClass newTable = null;
7     OWLNamedClass fedEntity = null;
8
9     try{
10       //get resource name for the class fm:FederatedEntities, superclass of global
             tables
11       String fedEntityString = owlModel.getResourceNameForURI(getNsFm() +
             "FederatedEntities");
12
13       //get class fm:FederatedEntities
14       fedEntity = owlModel.getOWLNamedClass(fedEntityString);
15
16       //create table
17       newTable = FISOntologyUtils.createOWLClassSafe(owlModel, getNsFm() + table);
18
19       //add superclass
20       newTable.addSuperclass(fedEntity);
21    }
22    catch(Exception ex){
23       ...
24    }
25    return true;
26 }
```

Listagem 38: Criar uma tabela no modelo global

O acesso pode ainda ser realizado através de pesquisas através da interface *Jena*, recorrendo às linguagens *SPARQL* e *SPARQL-DL*. As duas linguagens têm sintaxes iguais, mas a linguagem *SPARQL-DL* é executada sobre um motor de inferência, o que permite processar a semântica da *OWL*. No entanto, devido a limitações conceptuais, não é possível realizar interrogações sobre a TBox de uma taxonomia utilizando a variante *SPARQL-DL*. Como essas interrogações são necessárias para obter informações sobre as tabelas e colunas do modelo de dados global, as duas variantes são utilizadas.

Os recursos obtidos através destas interrogações podem depois ser processados através das duas técnicas apresentadas anteriormente: código gerado pelo *protégé* ou interface *OWLAPI*.

5.2 Modelos Parciais

Os modelos de dados parciais podem ser inseridos de forma manual ou através de um componente que importa automaticamente o modelo a partir de uma ligação à fonte de dados.

A inserção manual é realizada através de um conjunto de ecrãs que disponibilizam acesso a campos dos modelos parciais. Nomeadamente, são disponibilizados os ecrãs correspondentes a:

- *dbs:database* - uma fonte de dados parcial, permite inserir novas tabelas e definir os campos *dbs:uri*, *dbs:provider*, *dbs:username* e *dbs:password*.
- *dbs:table* - uma tabela de uma fonte de dados parcial, permite inserir novas colunas e aceder aos seus campos *dbs:columnAccess* e *dbs:columnType*.

A importação de fontes de dados foi implementada para fontes de dados relacionais (com interface *jdbc*) e ficheiros *xlsx* (criados com o programa *Microsoft Excel*). Tendo como entrada a *string* de conexão *jdbc* ou o *url* para o ficheiro *xlsx*, a aplicação obtém a estrutura dos dados e adiciona informação à taxonomia actualmente activa na aplicação. Desta forma facilita-se a configuração do sistema e reduzem-se eventuais erros que são mais fáceis de ocorrer nas configurações manuais.

Este módulo é baseado no código do *plugin Datamaster* [4] para a ferramenta *Protégé* que gera uma ABox para a ontologia *relational.owl*. Esta base de código foi alterada de forma a:

- reflectir as alterações realizadas à ontologia *relational.owl*;
- permitir a importação de fontes de dados sem interface *jdbc* ou *odbc* (as únicas suportadas pelo *Datamaster*).

A figura 5.3 apresenta o diagrama de classes do módulo de importação automática.

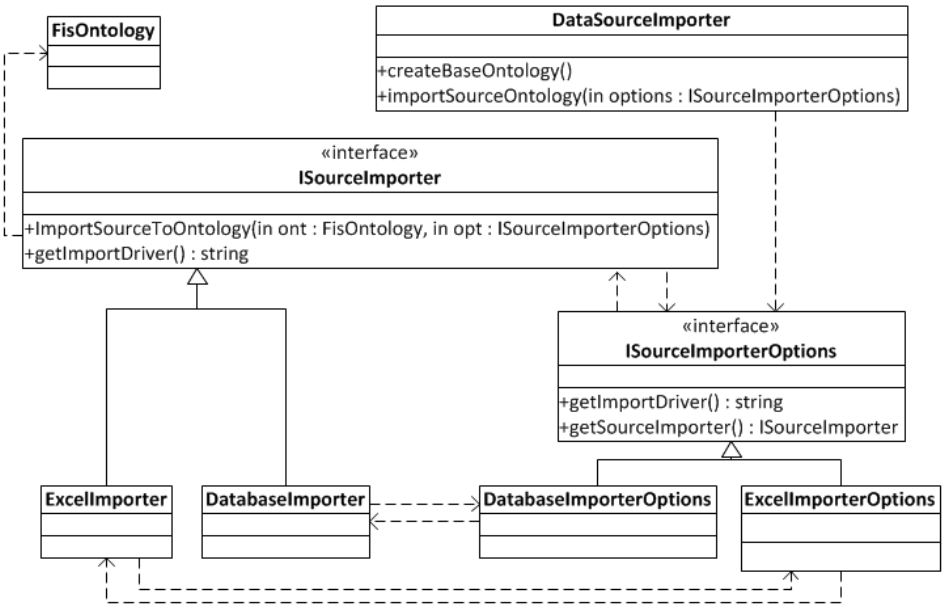


Figura 5.3: Diagrama de classes para a importação de fontes de dados

A classe "DataSourceImporter" é o ponto de entrada para a criação de uma ontologia vazia (método *createOntology*) e para a importação (método *importSourceOntology*). Este segundo método recebe um objecto "ISourceImporterOptions" com toda a informação necessária para aceder aos dados e com a indicação de qual a instância de "ISourceImporter" que vai realizar a importação dos dados.

Foi implementada a importação automática de modelos parciais a partir de fontes de dados que possam ser acedidas através da interface *jdbc* ("DatabaseImporter") e fontes de dados suportadas no formato *xlsx* "ExcelImporter".

Este módulo de importação pode ser utilizado como exemplificado na listagem 39.

```
FISOntology ontology;
//initialize ontology as needed

DataSourceImporter sourceImporter = new DataSourceImporter(ontology);

//set options for importer
ISourceImporterOptions res = new DatabaseImporterOptions(driver, uri, schema, user,
    pass, importDriverName);

//import
sourceImporter.ImportSourceOntology(options);
```

Listagem 39: Utilizar importador de fontes de dados

5.2.1 Importação automática de modelos parciais

A importação do modelo de dados é específica para cada tipo de fonte de dados suportado.

Para as fontes de dados *jdbc*, já existem implementações da interface *DatabaseMetaData* [45] que disponibilizam os serviços necessários para aceder à meta informação. Este código segue o modelo do código do *plugin* para o *Protégé DataMaster*[1] [4]. A listagem 40 mostra como é possível obter o nome das tabelas de uma base de dados de nome "dbteste" através desta interface *DatabaseMetaData*.

```
Connection conn = null;
DatabaseMetaData dbMetaData = null;

try {
  //get jdbc connection to source
  conn = getConnection(conOptions);

  //get metadata
  dbMetaData = conn.getMetaData();

  ResultSet tablesSet = dbMetaData.getTables(``dbteste'', null, null, null);

  while(tablesSet.next())
  {
     String tableName = tablesSet.getString("TABLE_NAME");
  }
}
catch(SQLException e) {
    e.printStackTrace();
}
```

Listagem 40: Código Java para obter o nome das tabelas

Para obter a mesma informação das fontes de dados *xlsx* a estratégia é muito diferente, pois não existe uma classe com estes serviços e a estrutura dos dados não pode ser obtida de forma tão directa. A listagem 41 apresenta um exemplo de importação das folhas de um ficheiro *xlsx*. A importação automática destas fontes de dados assume que a primeira linha com células preenchidas é a linha dos cabeçalhos e faz uma recolha simples de colunas, sem distinguir chaves primárias ou estrangeiras.

[1] Mozilla Public License [46]

```
1  private void importOntology(FISOntology ontology, Workbook wb, File file)
2  {
3    //create factory for ontology objects
4    FisFactory fact = new FisFactory(ontology.getOwlModel());
5
6    //create database(source)
7    Database db = fact.createDatabase( FISOntologyUtils.getOWLNameForTable(
8        FISOntologyUtils.getABOXNamespace(), file.getName()) );
9    db.setUri(file.getAbsolutePath());
10   db.setProvider(``excel'');
11
12   Sheet sheet = null;
13   Table table = null;
14
15   //for each sheet, create a table
16   for(int i = 0; i < wb.getNumberOfSheets(); ++i)
17   {
18     sheet = wb.getSheetAt(i);
19
20     //sheet -> table
21     table = fact.createTable(FISOntologyUtils.getABOXNamespace() +
           sheet.getSheetName());
22     db.addHasTable(table);
23
24     //import columns
25     ...
26    }
27 }
```

Listagem 41: Código Java para a importação de folhas de um ficheiro *xlsx*

5.3 Modelo Global

A definição do modelo de dados global tem de ser realizada de forma manual através da interface gráfica da aplicação. Para o efeito, são disponibilizados ecrãs para acesso à componente TBox_G (taxonomia que descreve o modelo de dados global). Nomeadamente, são disponibilizados os ecrãs correspondentes a:

- tabelas - permite criar subclasses de *fm:FederatedEntities* correspondentes a tabelas no modelo de dados global.
- colunas - permite criar propriedades correspondentes a colunas no modelo de dados global.

5.4 Mapeamento entre modelos

O mapeamento entre modelos é realizado através de instâncias das classes correspondentes às tabelas do modelo global. Cada instância corresponde a uma partição horizontal dessa tabela. Ao seleccionar uma instância, é apresentado o ecrã da figura 5.4 onde podem ser introduzidas as informações do mapeamento.

O primeiro passo do mapeamento é indicar a tabela dos modelos parciais que irá ser utilizada para o mapeamento. Se for utilizada mais de uma tabela, é necessário indicar as colunas que vão definir a relação entre as tabelas (figura 5.5). A indicação da primeira tabela não tem qualquer influência no modelo, já que essa informação pode ser inferida do mapeamento das colunas. No entanto, essa indicação é utilizada para determinar as colunas que são apresentadas na interface gráfica. Para cada tabela, para além da primeira, é inserido no modelo uma instância de *FederatedRelation*, com uma instância de *ColumnsRelation* para cada par de colunas indicado para a relação entre as tabelas.

O mapeamento é realizado definindo as ligações individuais das colunas do modelo global. Assim, cada coluna do modelo global pode ser mapeada para:

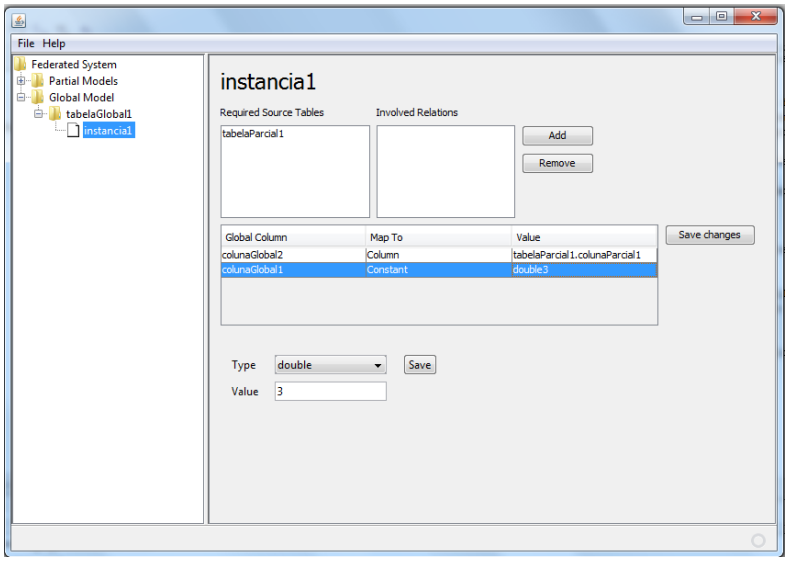


Figura 5.4: Ecrã para mapeamento de uma instância

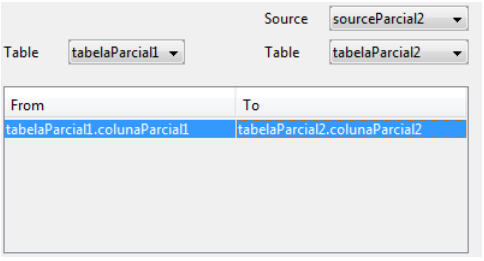
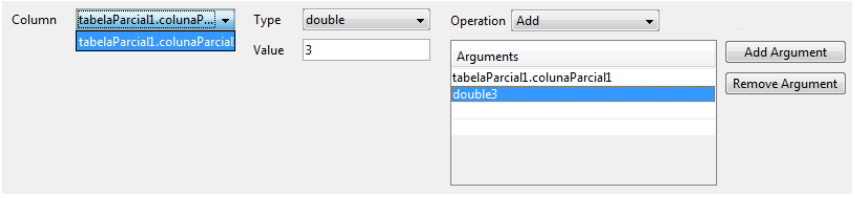


Figura 5.5: Ecrã para definição de relação entre tabelas

1. Uma coluna de um modelo parcial (figura 5.6a);
2. Uma constante (figura 5.6b);
3. Uma função (figura 5.6c).



(a) Coluna (b) Constante (c) Função

Figura 5.6: Ecrãs alternativos para mapeamentos de colunas

Esta aplicação foi utilizada e testada para definição dos exemplos apresentados nos capítulos 3 e 4, assim como no cenário de avaliação apresentado no capítulo 6.

CAPÍTULO 6

VALIDAÇÃO EXPERIMENTAL

Neste capítulo apresenta-se uma experiência em contexto real, usada para validar a aplicabilidade concreta das propostas e implementações que sustentam esta dissertação.

O cenário apresentado baseia-se num caso analisado em contexto empresarial, em que existe a necessidade de construir perspectivas unificadas sobre dados que descrevem contratos e dados que caracterizam variáveis que afectam diariamente o valor representado por cada contrato. Ou seja, um contrato é assinado numa data e é válido para determinado período. Diariamente, os itens desse contrato são afectados por vários parâmetros (de mercado) que é preciso analisar (também diariamente) para se perceber o efectivo valor de mercado de cada contrato.

No resto do capítulo apresenta-se a descrição do problema, o modelo construído para a descrição do sistema e os pormenores da execução de uma interrogação sobre o sistema descrito. No mapeamento do modelo de dados e na execução de interrogações são apresentados casos de partições horizontais, partições verticais, mapeamento directo e mapeamento através de constantes e de colunas calculadas.

6.1 Descrição do problema

O sistema de informação *SIT* contém os dados relativos a contratos, ou negócios (*deals*), a tratar. No essencial, cada contrato é estabelecido com uma contraparte (empresa outorgante), refere pelo menos um produto, a ser transaccionado (compra ou venda) em determinada quantidade e preço unitário mantendo registo da data contratual e do período (data início e fim) de aplicação desse contrato.

Os contratos são analisados numa base mensal, i. é, usando como unidade temporal o mês. Esta normalização é, em geral, efectuada no *SIT* e tornada disponível no seu modelo de dados. No entanto, existem também sequências de subcontratos externas ao *SIT* e que aqui serão posteriormente descritos como *contratos-fora-SIT*. Um dos exemplos de *contratos-fora-SIT* é o de contratos de carvão, cujos dados estão numa ficheiro *excel*.

A criação de uma perspectiva global, designada por base_mensal, recorre às seguintes fontes de dados:

- *contratos-no-SIT* obtidos via interrogações SQL ao sistema *SIT*,
- *contratos-fora-SIT* que actualmente se materializa em ficheiros *excel* com dados que não estão presentes no *SIT*.

O *contratos-no-SIT* será uma fonte de dados parcial do sistema federado e será descrita numa ABox_P. O ficheiro excel com os contratos de carvão (*contratos-fora-SIT*) será uma outra fonte de dados parcial descrita numa outra ABox_P.

O modelo de dados global irá descrever, numa TBox_G, as tabelas *BASE_MENSAL_DEAL* e *BASE_MENSAL_SERIES* (ver figura 6.1).

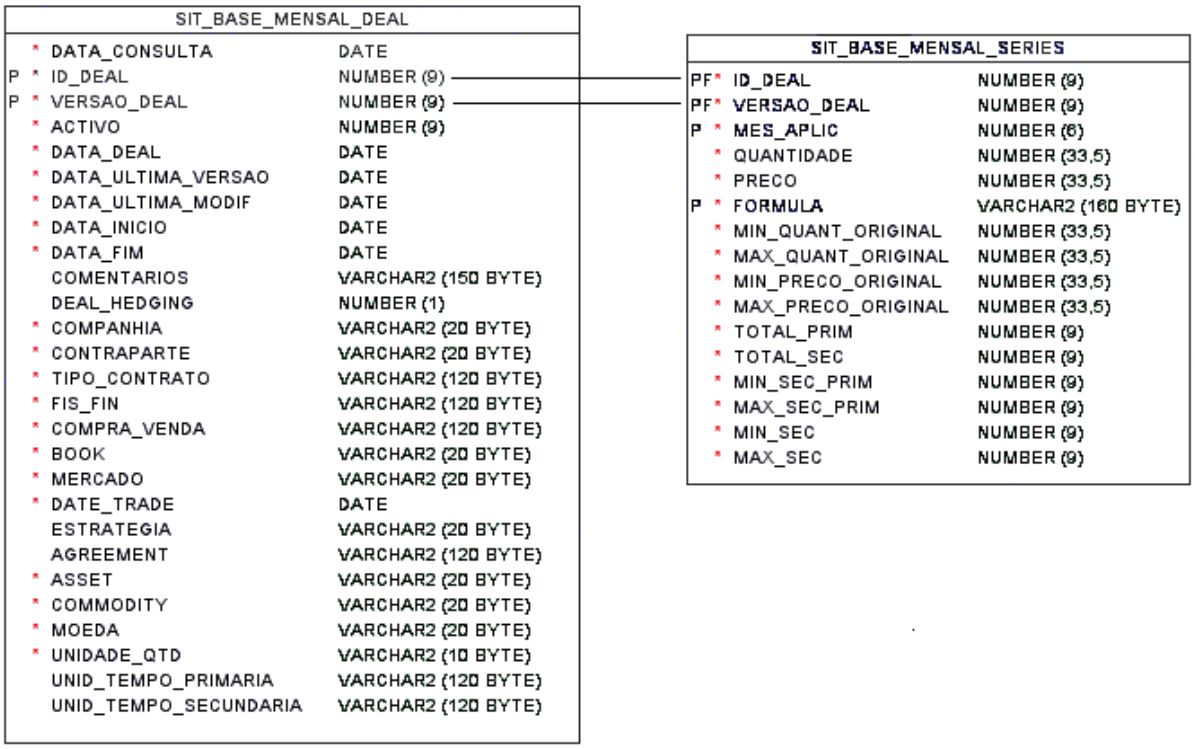


Figura 6.1: Modelo de dados exposto pelas vistas de acesso à fonte *contratos-no-SIT*

O *contratos-no-SIT* será acedido através de vistas sobre a base de dados SIT, de modo a realizar mais operações junto da fonte de dados, permitindo tirar partido das optimizações locais existentes. Neste cenário, o modelo parcial exposto pelas vistas utilizadas para aceder à fonte de dados é igual ao modelo global, tal como apresentado na figura 6.1.

O ficheiro excel da fonte *contratos-fora-SIT* é composto pelas colunas apresentadas na figura 6.2.

ID_registo	Tipo_de_ Operaçao	Produto	Status	ORIGEM/PAÍS	Empresa	Cod_Fornec.	Designação
Navio	N_Contrato	De_registo_no _sistema	Da_última_ actualização	CIF/FOB/Frete	Tipo_Contrato	Unidade	Fixo
índice_0	índice	Preço_Fixo	Formula	pzinicio	pzfim	factor_mult	factor_plus
Opção_ Fórmula_Preço	Moeda	Moeda_ Spread	Contratação	Descarga	Prevista_n	Prevista_n_plus _one	Quantidade
Opção_ Volumes	Singular	Outras_ empresas	Id Deal				

Figura 6.2: Colunas do modelo de dados da fonte *contratos-fora-SIT*

6.2 Modelos Parciais e Modelo Global

A criação dos modelos parciais e do modelo global é realizada recorrendo à aplicação descrita no capítulo 5. Os modelos parciais foram importados automaticamente usando a mesma aplicação (ver secção 5.2.1).

Para a importação da fonte de dados *contratos-no-SIT*, é utilizado o importador de fontes de dados *jdbc*. Neste caso particular, a base de dados é gerida pelo SGBD *mysql*, pelo que o *driver* a utilizar é o *com.mysql.jdbc.Driver*. A base de dados está acessível através da porta 3306 do servidor "homeServer" e a base de dados tem o nome "sit". Assim. os dados a introduzir no ecrã de importação do modelo parcial são:

- **provider** - com.mysql.jdbc.Driver
- **uri** - jdbc:mysql://homeServer:3306/sit
- **username** - user_FS_evaluation
- **password** - **********

A listagem 42 apresenta uma parcela do modelo gerado. É apresentada a criação da representação para a base de dados, para a tabela *sit_base_mensal_deal* e para a coluna *FIS_FIN*. A listagem 43 apresenta os mesmos elementos serializados em XML.

```
1  #Base de dados
2  dbs:Database(sit)
3
4  #Acesso à fonte de dados
5  dbs:provider(sit, ''com.mysql.jdbc.Driver'')
6  dbs:uri(sit, ''jdbc:mysql://homeServer:3306/sit'')
7  dbs:username(sit, ''user_FS_evaluation'')
8
9  #tabela sit_base_mensal_deal
10 dbs:Table(sit_base_mensal_deal)
11 dbs:hasTable(sit, sit_base_mensal_deal)
12
13 #coluna FIS_FIN
14 dbs:Column(sit_base_mensal_deal.FIS_FIN)
15 dbs:columnAccess(sit_base_mensal_deal.FIS_FIN, FIS_FIN)
16 dbs:hasColumn(sit_base_mensal_deal, sit_base_mensal_deal.FIS_FIN)
```

Listagem 42: Modelo parcial da base de dados "sit", representado em DL

```
<dbs:Database rdf:about="http://edu.ipl.isel.deetc.fis/#sit">
    <dbs:provider rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        com.mysql.jdbc.Driver
    </dbs:provider>
    <dbs:uri rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    jdbc:mysql://homeServer:3306/sit
  </dbs:uri>
  <dbs:username rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    user_FS_evaluation
  </dbs:username>

  <dbs:hasTable>
    <dbs:Table rdf:about = "http://edu.ipl.isel.deetc.fis/#sit_base_mensal_deal">
      <dbs:hasColumn>
        <dbs:Column rdf:about =
    "http://edu.ipl.isel.deetc.fis/#sit_base_mensal_deal.FIS_FIN">
          <dbs:columnAccess rdf:datatype = "http://www.w3.org/2001/XMLSchema#string">
            FIS_FIN
          </dbs:columnAccess>
        </dbs:Column>
      </dbs:hasColumn>
    </dbs:Table>
  </dbs:hasTable>
</dbs:database>
```

Listagem 43: Serialização XML do modelo parcial da base de dados "sit"

Para a importação da fonte de dados *contratos-fora-SIT*, apenas é necessário indicar o *url* através do qual é possível aceder ao ficheiro excel.

O modelo global dos dados é composto pelas tabelas *BASE_MENSAL_DEAL* e *BASE_MENSAL_SERIES*. Este modelo tem de ser criado manualmente através da aplicação de criação do modelo.

6.3 Mapeamento entre os Modelos

As vistas disponibilizadas na base de dados *contratos-no-SIT* já expõem o modelo de dados da visão global, pelo que o mapeamento entre o modelo global e este modelo parcial é realizado apenas com o mapeamento directo entre as colunas dos modelos.

Para a realização deste mapeamento, foram criadas as instâncias *deal_db* e *series_db*. As listagens 44 e 45 apresentam uma parte do mapeamento da instância *deal_db*.

```
#Instância
BASE_MENSAL_DEAL(deal_db)

#Mapeamento da coluna FIS_FIN
FIS_FIN(deal_db, sit_base_mensal_deal)
```

Listagem 44: Excerto do mapeamento da instância *deal_db*, em DL

```
 <BASE_MENSAL_DEAL rdf:about = "http://edu.ipl.isel.deetc.fis/#deal_db">
    <FIS_FIN rdf:resource =
    "http://edu.ipl.isel.deetc.fis/#sit_base_mensal_deal.FIS_FIN"/>
 </BASE_MENSAL_DEAL>
```

Listagem 45: Serialização XML de um excerto do mapeamento da instância *deal_db*

O mapeamento entre o modelo global e o ficheiro excel *Carvao.xlsx* é realizado nas instâncias *deal_xls* e *series_xls*. O mapeamento desta fonte de dados não é directo como no caso da base de dados. O mapeamento de cada coluna do modelo global está descrito na tabela 6.1.

Base-Mensal	**Carvão**	**Mapeamentos**
DATA_CONSULTA	2010-07-20	Constante
ID_DEAL	id_deal	Função
VERSAO_DEAL	1	Constante
ACTIVO	1	Constante
DATA_DEAL	contratação|2010-07-20 If Empty	Função
DATA_ULTIMA_VERSAO	contratação|2010-07-20 If Empty	Função
DATA_ULTIMA_MODIF	contratação|2010-07-20 If Empty	Função
DATA_INICIO	descarga	Directo
DATA_FIM	descarga	Directo
TIPO_CONTRATO	tipo_contrato	Directo
HEDGING	1	Constante
FIS_FIN	"Physical Supply"	Constante
COMPANHIA	Empresa	Directo
CONTRAPARTE	Designação	Directo
COMMODITY	"Coal"	Constante
ASSET	"Coal_month"	Constante
BOOK	"Carvão"	Constante
MERCADO	"FUEL GENERIC"	Constante
COMPRA_VENDA	"Purchase"	Constante
COMENTARIOS	""	Constante
AGREEMENT	""	Constante
ESTRATEGIA	"05 - Stock"	Constante
UNID_TEMPO_PRIMARIA	"Month"	Constante
UNID_TEMPO_SECUNDARIA	"Month"	Constante
FORMULA	indice	Directo
UNIDADE_QTD	"MT"	Constante
QUANTIDADE	quantidade	Directo
MOEDA	"US Dollar"	Constante
PRECO	preco_fixo	Directo
MES_APLIC	descarga	Directo
TOTAL_PRIM	1	Constante
TOTAL_SEC	1	Constante
MIN_SEC_PRIM	1	Constante
MAX_SEC_PRIM	1	Constante
MIN_SEC	1	Constante
MAX_SEC	1	Constante
DATE_TRADE	contratacao|2010-07-20 If Empty	Função
MIN_QUANT_ORIGINAL	quantidade	Directo
MAX_QUANT_ORIGINAL	quantidade	Directo
MIN_PRECO_ORIGINAL	preco_fixo	Directo
MAX_PRECO_ORIGINAL	preco_fixo	Directo
FONTE	"CAR_FIS"	Constante
FORMULA_INICIO	pzfim	Directo
FORMULA_FIM	pzinicio	Directo
FORMULA_FACTOR_MULT	factor_mult	Directo
FORMULA_FACTOR_ADD	factor_plus	Directo
CODE_STATE	"Liberated"	Constante

Tabela 6.1: Mapeamento entre o modelo global e a fonte de dados *excel*

6.4 Execução de Interrogações

A interrogação utilizada para exemplificar a execução de interrogações sobre o sistema é:

```
SELECT ID_DEAL, DATA_ULTIMA_MODIF FROM BASE_MENSAL_DEAL
```

Esta interrogação implica a consulta das duas fontes de dados (através de adaptadores para fontes *jdbc* e *excel*), a consulta de colunas de mapeamento directo, uma coluna calculada (a coluna *DATA_ULTIMA_MODIF* na instância *deals_xlsx*) e da utilização de uma partição vertical (na instância *deals_xls*, entre as tabelas *Registos* e a tabela *constantTable*).

A listagem apresentada no anexo B.1 contém código que executa esta interrogação, utilizando o modelo de dados atrás descrito e guardado no ficheiro *E:/cenario/exemplo.owl*.

Segue-se a descrição da execução da interrogação, nas fases de análise, planeamento e processamento.

6.4.1 Análise da Interrogação

A interrogação SQL é passada à API sobre a forma de uma *String*, dando origem aos objectos apresentados na figura 6.3. É criado um objecto do tipo *FisQuery* que contém dois objectos do tipo *FisQueryColumn* com a informação das colunas a apresentar e um objecto do tipo *FisQueryTable* com a informação da tabela a interrogar:

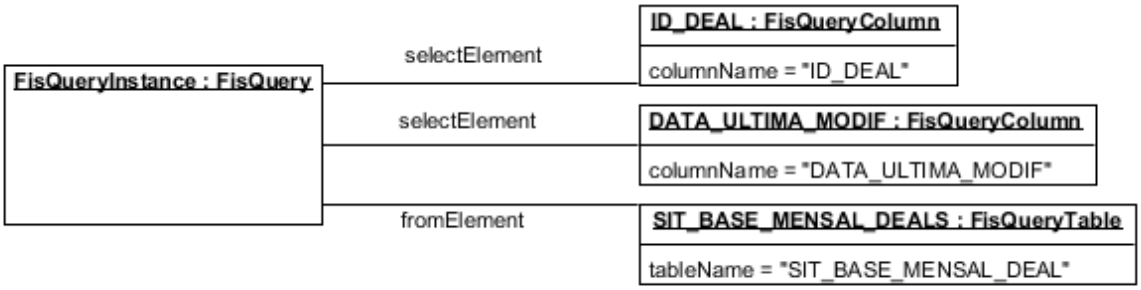


Figura 6.3: Diagrama dos objectos resultantes da análise da interrogação

6.4.2 Planeamento da Interrogação

O segundo passo na execução de uma interrogação é o planeamento. O plano gerado para a execução da interrogação é composto pelos objectos apresentados na figura 6.4.

Cada objecto do plano tem uma função específica e uma interrogação parcial associada. Os objectos que compõem o plano de execução da interrogação são apresentados de seguida.

FisQueryMasterPlan_1

Contém a interrogação inicial.

```
SELECT ID_DEAL,
       DATA_ULTIMA_MODIF
FROM   BASE_MENSAL_DEAL
```

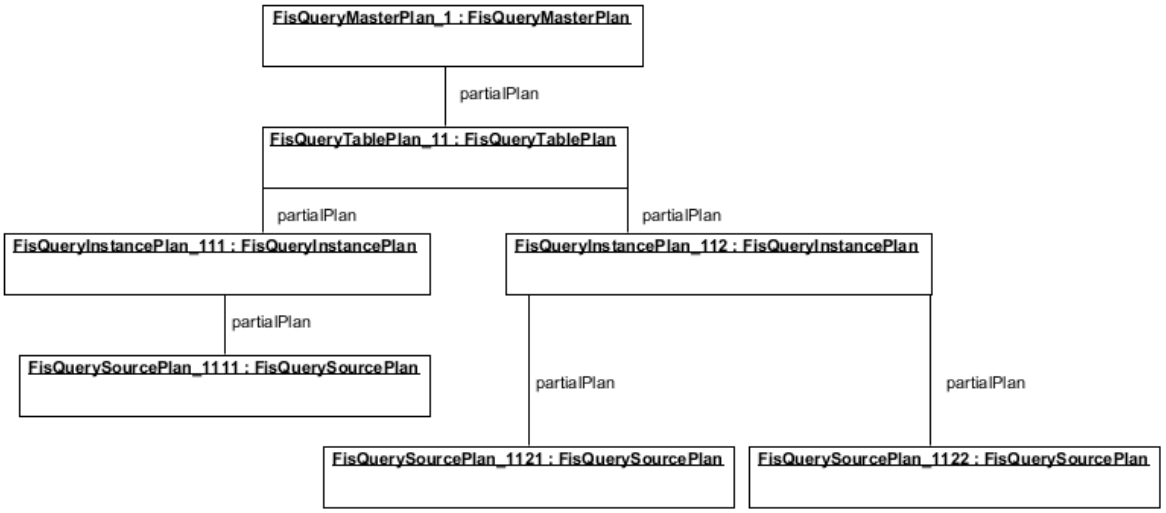


Figura 6.4: Diagrama dos objectos do plano de execução

FisQueryTablePlan_11

Contém a interrogação correspondente a uma tabela do modelo global. Neste caso, como apenas é consultada uma tabela global, a interrogação é igual à interrogação inicial.

```
SELECT  ID_DEAL,
        DATA_ULTIMA_MODIF
FROM    BASE_MENSAL_DEAL
```

FisQueryInstancePlan_111

Contém a interrogação correspondente à instância *deals_db*. As tabelas e colunas são traduzidas para os nomes correspondentes na fonte de dados.

```
SELECT  sit_base_mensal_deal.ID_DEAL AS ID_DEAL,
        sit_base_mensal_deal.DATA_ULTIMA_MODIF AS DATA_ULTIMA_MODIF
FROM    sit_base_mensal_deal
```

FisQuerySourcePlan_1111

Contém a interrogação executável na fonte de dados *sit*. Como os dados da instância *deals_db* são todos originados na mesma fonte de dados, não são realizadas alterações em relação à interrogação parcial de *FisQueryInstancePlan_111*.

```
SELECT  sit_base_mensal_deal.ID_DEAL,
        sit_base_mensal_deal.DATA_ULTIMA_MODIF
FROM    sit_base_mensal_deal
```

FisQueryInstancePlan_112

Contém a interrogação correspondente à instância *deals_xlsx*. Na construção desta interrogação são realizadas três transformações:

- As tabelas e colunas são traduzidas para os nomes correspondentes na fonte de dados;
- A coluna *DATA_ULTIMA_MODIF* é expandida para a função *IfEmpty(Registos.AB, constantTable.2010-07-20)*;

- É introduzida a junção correspondente à partição vertical estabelecida entre as tabelas *Registos* e *constantTable*. Neste caso particular, a partição vertical não tem condição de junção.

```
SELECT  Registos.AJ AS ID_DEAL,
        IfEmpty(Registos.AB, constantTable.2010-07-20)
                                    AS DATA_ULTIMA_MODIF
FROM    Registos JOIN constantTable
```

FisQuerySourcePlan_1121

Contém a interrogação executável na fonte de dados *contratos-fora-SIT*. As colunas não existentes na fonte de dados são retiradas.

```
SELECT  Registos.AJ,
        Registos.AB
FROM    Registos
```

FisQuerySourcePlan_1122

Contém a interrogação executável na fonte de dados especial *constantSource* (que contém as constantes). As colunas não existentes na fonte de dados são retiradas.

```
SELECT  constantTable.2010-07-20
FROM    constantTable
```

6.4.3 Processamento da Interrogação

A última fase da execução da interrogação é o processamento do plano de execução e obtenção dos resultados. O processamento realizado ao processar cada um dos objectos do plano de execução é apresentado de seguida.

FisQuerySourcePlan_1111

É realizada a interrogação através do adaptador para fontes *jdbc*.

Resultado: <sit_base_mensal_deal.ID_DEAL,

sit_base_mensal_deal.DATA_ULTIMA_MODIF>

FisQuerySourcePlan_1121

É realizada a interrogação através do adaptador para fontes *excel*.

Resultado: <Registos.AJ, Registos.AB>

FisQuerySourcePlan_1122

É realizada a interrogação através do adaptador para fontes *constant*.

Resultado: <constantTable.2010-07-20>

FisQueryInstancePlan_111

Realiza apenas a tradução dos nomes das colunas resultantes da execução de *FisQuerySourcePlan_1111* para os nomes das colunas correspondente no modelo global.

Resultado: <ID_DEAL, DATA_ULTIMA_MODIF>

FisQueryInstancePlan_112

Realiza a operação de junção dos resultados de *FisQuerySourcePlan_1121* e *FisQuerySourcePlan_1122*. Realiza a execução da função *IfEmpty* sobre as colunas *Registos.AB* e *constantTable.2010-07-20*.

Resultado: <ID_DEAL, DATA_ULTIMA_MODIF>

Operação	Tempo Médio (em *milisegundos*)
1	2062
2	3973
3	253
4	411

Tabela 6.2: Tempos de execução da interrogação

FisQueryTablePlan_11

Realiza a união dos dados de cada instância (objectos *FisQueryInstancePlan_111* e *FisQueryInstancePlan_112*).

Resultado: <ID_DEAL, DATA_ULTIMA_MODIF>

FisQueryMasterPlan_1

Não é realizada nenhuma operação, pelo que os dados resultantes do processamento de *FisQueryTablePlan_11* são retornados sem alterações.

Resultado: <ID_DEAL, DATA_ULTIMA_MODIF>

6.4.4 Resultado da Execução

No cenário apresentado, a execução retorna 265 resultados, sendo 193 originários da base de dados e 72 do ficheiro *excel*.

Foram medidos os tempos de execução das seguintes operações:

1. Criação do objecto de acesso à API, *FisDatabase*.
2. Primeira execução da interrogação. Alguns objectos de análise da interrogação e acesso aos metadados são inicializados neste momento.
3. Execuções de 5 repetições da mesma interrogação, utilizando a *cache* do plano, mas não dos resultados.
4. Primeira execução de outra interrogação, com o mesmo grau de complexidade e número de resultados. O plano não existe em *cache* mas os objectos de análise da interrogação e acesso aos metadados já estão inicializados.

A interrogação apresentada de seguida é em tudo semelhante à interrogação original, já que o número de resultados é o mesmo e a coluna *DATA_DEAL* é calculada da mesma forma que a coluna *DATA_ULTIMA_MODIF*.

```
SELECT  ID_DEAL,
        DATA_DEAL
FROM    BASE_MENSAL_DEAL
```

Os resultados apresentados na tabela 6.2 são os tempos médios de 10 execuções do teste completo. Na operação 3, é apresentado o tempo médio de uma execução da interrogação.

É possível verificar que o tempo gasto na inicialização de objectos auxiliares é muito elevado em comparação com o tempo de execução da interrogação. Em média, o tempo de execução da operação 4 (primeira execução de uma interrogação sem inicialização de objectos) é 10,34% do tempo de execução do operação 2 (primeira execução de uma interrogação com inicialização de objectos).

É também possível verificar a utilidade da *cache* de planos de execução. Nos testes realizados, o tempo de execução de interrogações com *cache* foi, em média, 61,56% do tempo de execução de interrogações

sem *cache*. Esta diferença resulta de não ser necessário realizar as fases de análise e planeamento da interrogação.

6.5 Aspectos a Evidenciar

O cenário apresentado foi extraído de um contexto real com elevado número de atributos e foi bastante simples de configurar e interrogar.

Uma operação comum é a realização de vários relatórios com a mesma base de informação, proveniente da base de dados e das folhas excel. A existência de uma visão uniformizada dos dados permite que esses relatórios possam ser gerados apenas sobre um modelo de dados. No contexto real, oferece grandes vantagens face à alternativa de gerar diversas versões do mesmo relatório para cada uma das fontes de dados distintas e heterogéneas.

A realização desta experiência permite consolidar a ideia de que as propostas e todo o trabalho desenvolvido são aplicáveis em contexto real e até em cenários com maior exigência (a nível de partições, réplicas e funções) do que aquele a que se teve acesso.

CAPÍTULO 7

CONCLUSÃO

Este trabalho explora uma alternativa pouco abordada - a utilização de formalismos que suportam descrição semântica para modelar sistemas federados - e aborda algumas lacunas nas soluções existentes - como o fraco suporte a fontes de dados e limitações na modelação dos dados.

Para além da validação da utilização dos formalismos que suportam descrição semântica como suporte ao modelo de dados e de um sistema de informação federado e da linguagem SQL como linguagem de interrogação, outras contribuições resultam desta dissertação. Relacionadas directamente com a validação das hipóteses colocadas, foi construído um modelo para a meta informação técnica, lógica e de meta modelos utilizando a linguagem OWL. Foi também implementado um executor de interrogações SQL sobre sistemas de informação federados descritos por este modelo e que permite interrogar fontes de dados *xlsx* e fontes de dados com interface *jdbc*, para além de ser facilmente adicionado suporte a outros tipos de fontes de dados.

Foram também contribuições desta dissertação a criação de uma versão alterada da ontologia *relational.owl* que introduz informação de acesso às fontes de dados e corrige uma falha no suporte a chaves estrangeiras. Foi criado um conjunto de classes sobre o analisador de interrogações do *Eclipse Data Tools* que torna a sua utilização bastante mais simples. O código do *plugin Datamaster* foi também extendido para dar suporte à versão alterada da ontologia *relational.owl* e para ser mais fácil a implementação de suporte a novas fontes de dados para além do suporte já implementado para fontes de dados *xlsx* e fontes com interface *jdbc*.

O modelo proposto, baseado em formalismos que suportam descrição semântica e concretizado em OWL, apresenta capacidades de modelação de dados superiores às dos produtos e projectos de código de fonte aberta analisados. Com os devidos cuidados no mapeamento de conceitos, foi possível verificar que o modelo tem capacidade para descrever fontes de dados de características heterogéneas, desde que estas contenham algum tipo de estrutura. Durante a concepção do modelo e a adição de novas funcionalidades, o modelo apresentou-se muito estável e fácil de expandir, apresentando várias possibilidades de trabalho futuro.

A utilização destes formalismos origina, no entanto, algumas dificuldades. O facto de o utilizador típico deste tipo de sistemas estar habitado aos conceitos do modelo relacional leva a que soluções baseadas em outros modelos não sejam tão facilmente aceites. A aplicação desenvolvida permite ultrapassar esta barreira e simplifica a adopção da solução. Outra dificuldade reside na necessidade de realizar inferência (*reasoning*) sobre o modelo em vez de aceder aos dados directamente. Este processo é tipicamente mais lento que o simples acesso aos dados mas é o preço a pagar pelas vantagens que essa inferência introduz em toda a solução.

Foi possível implementar uma solução de interrogação do sistema de informação federado sem ser necessário alterar a linguagem SQL. Sendo esta uma linguagem de interrogação familiar a quem trabalha com sistemas de informação baseados no modelo relacional, é uma boa forma de proporcionar transparência na interrogação das fontes. É, inclusive, possível utilizar mecanismos criados sobre a linguagem SQL como o mapeamento relacional-objecto (suportado por bibliotecas como o *hibernate*).

A arquitectura utilizada como referência neste trabalho é o resultado de anos de investigação, mantendo-se válida para responder aos desafios dos sistemas actuais. Nesta arquitectura é necessário realizar as operações de modo centralizado - na camada de federação - sobre os dados. Existe no entanto, a possibilidade de ver esta camada de federação como um de vários mediadores, que colaboram entre si, criando novas possibilidades de optimização e distribuição de trabalho.

Durante a realização deste trabalho, muitos temas surgiram como potenciais tópicos de trabalho futuro. Os tópicos de maior relevância identificados foram:

- Implementação de novos adaptadores para fontes de dados semi-estruturadas, como o *xml* ou sistemas de recuperação de informação.
- Introdução de semântica no modelo que permita, por exemplo, detectar informação repetida em fontes de dados distintas. É também necessário adaptar o executor de interrogações para processar esta semântica.
- Optimizações na análise, planeamento e execução de interrogações, utilizando as técnicas apresentadas na secção 2.3.
- Criação de sistemas de informação federados constituídos por diversas camadas de federação que colaboram entre si para responder a interrogações.
- Enriquecimento do modelo com regras que permitam validar a sua consistência e realizar mais inferências.

APÊNDICE A

INTERROGAÇÕES SPARQL

A.1 Obter todas as instâncias de uma tabela global

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4 PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
5
6 SELECT DISTINCT ?instance
7 WHERE {
8   ?instance rdf:type ?federatedTable
9 }
```

A.2 Obter informação de acesso a uma instância de uma tabela global

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4 PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
5 PREFIX dbs: <http://www.dbs.cs.uni-duesseldorf.de/RDF/relational.owl#>
6
7 SELECT ?federatedColumn ?column ?columnUri ?table ?source ?uri ?provider
8 WHERE {
9   ?instance rdf:type ?federatedTable .
10  ?federatedColumn rdfs:domain ?federatedTable .
11  ?instance ?federatedColumn ?column .
12  ?table dbs:hasColumn ?column .
13  ?source dbs:hasTable ?table .
14  ?source dbs:uri ?uri .
15  ?source dbs:provider ?provider .
16  ?column dbs:columnAccess ?columnUri
17 }
```

A.3 Obter informação para juntar partições verticais de uma instância de uma tabela global

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
PREFIX dbs: <http://www.dbs.cs.uni-duesseldorf.de/RDF/relational.owl#>
PREFIX fm: <http://edu.ipl.isel.deetc.fis/#>

SELECT ?join ?left ?right ?cLeft ?cRight
WHERE {
  ?entity fm:implicitJoin ?join .
  ?join fm:TableLeft ?left .
  ?join fm:TableRight ?right .
  OPTIONAL
  {
    ?join dbs:relatedColumns ?columnList .
    ?columnList dbs:fromColumn ?instLeft .
    ?instLeft dbs:columnAccess ?cLeft .
    ?columnList dbs:toColumn ?instRight .
    ?instRight dbs:columnAccess ?cRight
  }
}
ORDER BY ?join
```

A.4 Obter informações de mapeamento de uma coluna global para uma função

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
PREFIX dbs: <http://www.dbs.cs.uni-duesseldorf.de/RDF/relational.owl#>
PREFIX fm: <http://edu.ipl.isel.deetc.fis/#>

SELECT ?op ?args ?columnUri ?table ?source ?uri ?provider
WHERE {
  ?instance ?federatedColumn ?column .
  ?column fm:operation ?op .
  ?column fm:arguments ?args .
  ?table dbs:hasColumn ?args .
  ?source dbs:hasTable ?table .
  ?source dbs:uri ?uri .
  ?source dbs:provider ?provider .
  ?args dbs:columnAccess ?columnUri
}
```

A.5 Obter réplicas de uma instância

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4 PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
5 PREFIX fm: <http://edu.ipl.isel.deetc.fis/#>
6
7 SELECT ?replic
8 WHERE {
9   ?instance fm:replic ?replic
10 }
```

APÊNDICE B

LISTAGENS DE CÓDIGO

B.1 Execução de uma interrogação sobre o sistema construído para o cenário de avaliação

```
String configurationFile = "E:\\cenario\\exemplo.owl";

try
{
    //create object for access
    FisDatabase db = new FisDatabase(configurationFile);

    //execute and get results
    CachedRowSet res = (CachedRowSet)db.execute("SELECT ID_DEAL, DATA_ULTIMA_MODIF FROM
        SIT_BASE_MENSAL_DEAL");

    //show results
    res.beforeFirst();

    System.out.print("\n");
    for (int i = 1; i <= res.getMetaData().getColumnCount(); ++i) {
        System.out.print(res.getMetaData().getColumnName(i) + "\t");
    }
    System.out.print("\n");
    while (res.next()) {
        for (int i = 1; i <= res.getMetaData().getColumnCount(); ++i) {
            System.out.print(res.getObject(i) + "\t");
        }
        System.out.print("\n");
    }
}
catch (FisUnknownException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
catch (FisConfigurationException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
catch (FisCredentialsException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
catch (FISInaccessibleDataException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
catch (FISExecutionException ex) {
```

```
39      Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
40  }
41  catch (SQLException ex) {
42      Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
43  }
44  catch (FISWrapperException ex) {
45      Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
46  }
47  catch (FISParserException ex) {
48      Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
49  }
```

REFERÊNCIAS

[1] Susanne Busse, Ralf-Detlef Kutsche, Ulf Leser, and Herbert Weber. Federated information systems: Concepts, terminology and architectures. Technical Report 99-9, Technische Universitat Berlin, 1999.

[2] Cristian Pérez de Laborda and Stefan Conrad. Relational.owl: a data and schema representation format based on owl. In *APCCM '05: Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling*, pages 89–96, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[3] Eclipse data tools platform (dtp) project. `http://www.eclipse.org/datatools/`.

[4] Datamaster - protégé wiki. `http://protegewiki.stanford.edu/wiki/DataMaster`.

[5] The protégé ontology editor and knowledge acquisition system. `http://protege.stanford.edu/`.

[6] Dennis McLeod and Dennis Heimbigner. A federated architecture for database systems. In *AFIPS '80: Proceedings of the May 19-22, 1980, national computer conference*, pages 283–289, New York, NY, USA, 1980. ACM.

[7] Thierry Barsalou, Niki Siambela, Arthur M. Keller, and Gio Wiederhold. Updating relational databases through object-based views. *SIGMOD Rec.*, 20:248–257, April 1991.

[8] Ravi Krishnamurthy, Witold Litwin, and William Kent. Language features for interoperability of databases with schematic discrepancies. *SIGMOD Rec.*, 20:40–49, April 1991.

[9] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. Schemasql: An extension to sql for multidatabase interoperability. *ACM Trans. Database Syst.*, 26:476–519, December 2001.

[10] Diplom Informatiker, Ulf Leser, Vom Fachbereich Informatik, Dr. Ing, and Prof Dr. Weber. Query planning in mediator based information systems, 2000.

[11] F. Mougin, A. Burgun, O. Loreal, and P. Le Beux. Towards the automatic generation of biomedical sources schema. *Studies in Health Technology and Informatics*, 107:783–787, 2005.

[12] Daniela Nicklas, Matthias Grossmann, Thomas Schwarz, Steffen Volz, and Bernhard Mitschang. A model-based, open architecture for mobile, spatially aware applications. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 117–135, London, UK, 2001. Springer-Verlag.

[13] Hai Zhuge, Yunpeng Xing, and Peng Shi. Resource space model, owl and database: Mapping and integration. *ACM Trans. Internet Technol.*, 8:20:1–20:31, October 2008.

[14] Boris Motik, Bernardo Cuenca Grau, and Ulrike Sattler. Structured objects in owl: representation and reasoning. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 555–564, New York, NY, USA, 2008. ACM.

[15] Agustina Buccella, Alejandra Cechich, Ra Cechich, and Nieves R. Brisaboa. An ontology approach to data integration. *Journal of Computer Science and Technology*, 3:62–68, 2003.

[16] L. Haas and E. Lin. Ibm federated database technology. `http://www.ibm.com/developerworks/db2/library/techarticle/0203haas/0203haas.html`.

[17] Enterprise data integration: The case for federated systems solutions using sybase® adaptive server® enterprise 12.5. White paper, Sybase, 2001. Available online.

[18] Apache derby. `http://db.apache.org/derby/`.

[19] Vera Goebel. Heterogeneous / federated / multi-database systems. Technical report, Department of Informatics, University of Oslo, 2009.

[20] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *In Proc. of the 2004 Description Logic Workshop (DL 2004*, pages 41–50, 2004.

[21] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database Systems Concepts*. McGraw-Hill Higher Education, 2001.

[22] Michael L. Rupley Jr. Introduction to query processing and optimization. Technical report, Indiana University, 2008.

[23] Cem Evrendilek and Asuman Dogac. Query decomposition, optimization and processing in multidatabase systems. In *In Proc. of Workshop on Next Generation Information Technologies and Systems, Naharia*, 1995.

[24] Graham Bent, Patrick Dantressangle, David Vyvyan, Abbe Mowshowitz, and Valia Mitsou. A dynamic distributed federated database. *ACITA*, September 2008.

[25] Microsoft. Sql server 2008 books online: Understanding federated database servers. `http://msdn.microsoft.com/en-us/library/ms187467.aspx`, October 2009.

[26] org.apache.derby.vti (apache derby v10.6 internals). `http://db.apache.org/derby/javadoc/engine/org/apache/derby/vti/package-summary.html`.

[27] Chaitanya Baru, Amarnath Gupta, Bertram Ludäscher, Richard Marciano, Yannis Papakonstantinou, and Pavel Velikhov. Xml-based information mediation with mix, 1999.

[28] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. In *IPSJ Conference*, pages 7–18, October 1994.

[29] ISO. *Information technology – Syntactic metalanguage – Extended BNF*. ISO - International Organization for Standardization, 1996.

[30] Pellet: The open-source owl reasoner. `http://clarkparsia.com/pellet`.

[31] ISO. *Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)*. ISO - International Organization for Standardization, 1999.

[32] Antlr parser generator. `http://www.antlr.org/`.

[33] Sqljep - java sql expression parser. `http://sqljep.sourceforge.net/`.

[34] Javacc home. `https://javacc.dev.java.net/`.

[35] Apache derby. `http://db.apache.org/derby/`.

[36] Eclipse public license 1.0. `http://www.eclipse.org/legal/epl-v10.html`.

[37] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 16. Cambridge University Press, 2003.

[38] Gnu affero general public license, version 3. `http://www.gnu.org/licenses/agpl-3.0.html`.

[39] The owl api. `http://owlapi.sourceforge.net/`.

[40] Jena – a semantic web framework for java. `http://jena.sourceforge.net/`.

[41] Evren Sirin and Bijan Parsia. Sparql-dl: Sparql query for owl-dl. In *In 3rd OWL Experiences and Directions Workshop (OWLED-2007*, 2007.

[42] Apache license, version 2.0. `http://www.apache.org/licenses/LICENSE-2.0`.

[43] Apache poi - the java api for microsoft documents. `http://poi.apache.org/`.

[44] Merge sort - sorting algorithm animations. `http://www.sorting-algorithms.com/merge-sort`.

[45] Interface databasemetadata javadoc. `http://java.sun.com/j2se/1.3/docs/api/java/sql/DatabaseMetaData.html`.

[46] Mozilla public license version 1.1. `http://www.mozilla.org/MPL/MPL-1.1.html`.