
**UNIVERSIDADE FEDERAL DO CEARÁ**
**CENTRO DE CIÊNCIAS**
**DEPARTAMENTO DE COMPUTAÇÃO**
**MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO**

**LUÍS EUFRASIO TEIXEIRA NETO**

# UMA ABORDAGEM PARA PUBLICAÇÃO DE VISÕES RDF DE DADOS RELACIONAIS

**FORTALEZA, CEARÁ**

**2014**

**LUÍS EUFRASIO TEIXEIRA NETO**

# UMA ABORDAGEM PARA PUBLICAÇÃO DE VISÕES RDF DE DADOS RELACIONAIS

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Banco de Dados.

Orientadora: Profa. Dra. Vânia Maria Ponte Vidal.

Co-orientador: Prof. Dr. José Maria da Silva Monteiro Filho.

**FORTALEZA, CEARÁ**

**2014**

**LUÍS EUFRASIO TEIXEIRA NETO**

# UMA ABORDAGEM PARA PUBLICAÇÃO DE VISÕES RDF DE DADOS RELACIONAIS

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Aprovada em: ____ / ____ / _____

## BANCA EXAMINADORA

_____

Profa. Dra. Vânia Maria Ponte Vidal (Orientadora)
Universidade Federal do Ceará (UFC)

_____

Dra. Valéria Magalhães Pequeno
INESC-ID, Lisboa, Portugal

_____

Prof. Dr. Marco Antonio Casanova
Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ)

## AGRADECIMENTOS

Agradeço, primeiramente, a Deus, a quem sou grato por seu Amor infinito demonstrado todos os dias em minha vida.

À Clarissa, minha esposa, por todo o apoio, incentivo, compreensão e cumplicidade na superação de todas as dificuldades e desafios encontrados, e aos meus queridos filhos Luís Miguel e Rafael pelo enorme carinho e por ser fonte de inspiração para mim.

Aos meus pais, Djalma Peres e Lana Mara, e aos meus irmãos Everton e Daji que sempre me apoiaram na busca dos valores mais importantes, e que sempre me deram todo o suporte necessário para o alcance dos meus objetivos.

Aos Professores Vânia Vidal e José Maria Monteiro pela orientação e co-orientação, respectivamente, pela paciência, ensinamentos, valiosas discussões e por contribuirem para o meu amadurecimento profissional. Sou muito grato pela oportunidade de concluir esse desafio.

Ao Professor Marco Antônio Casanova, pela valiosa colaboração para o desenvolvimento desse trabalho e por sua presença na banca examinadora.

À pesquisadora Valéria Pequeno pelas valiosas contribuições na revisão do texto e por sua presença na banca examinadora.

A todos os professores do Departamento de Computação e, principalmente, membros do grupo ARIDA (*Advanced Research In DAtabase*).

Aos amigos de mestrado Diego Sá, Regis Pires, Diego Victor, Mônica Regina, Manoel Siqueira, Carlos Eduardo (Cadu), Arlino Henrique, Igo Brilhante, Henrique Viana e Fabrício Lemos, pelos excelentes momentos de convívio e aprendizado que pudemos compartilhar ao longo dessa caminhada.

À Secretaria da Fazenda, na pessoa do orientador de célula Francisco das Chagas Pordeus, pela flexibilidade de horário dispensada para que eu pudesse atender os compromissos acadêmicos.

# RESUMO

A iniciativa *Linked Data* trouxe novas oportunidades para a construção da nova geração de aplicações Web. Entretanto, a utilização das melhores práticas estabelecidas por este padrão depende de mecanismos que facilitem a transformação dos dados armazenados em bancos relacionais em triplas RDF. Recentemente, o grupo de trabalho *W3C RDB2RDF* propôs uma linguagem de mapeamento padrão, denominada R2RML, para especificar mapeamentos customizados entre esquemas relacionais e vocabulários RDF. No entanto, a geração de mapeamentos R2RML não é uma tarefa fácil. É imperativo, então, definir: (a) uma solução para mapear os conceitos de um esquema relacional em termos de um esquema RDF; (b) um processo que suporte a publicação dos dados relacionais no formato RDF; e (c) uma ferramenta para facilitar a aplicação deste processo. Assertivas de correspondência são propostas para formalizar mapeamentos entre esquemas relacionais e esquemas RDF. Visões são usadas para publicar dados de uma base de dados em uma nova estrutura ou esquema. A definição de visões RDF sobre dados relacionais permite que esses dados possam ser disponibilizados em uma estrutura de termos de uma ontologia OWL, sem que seja necessário alterar o esquema da base de dados. Neste trabalho, propomos uma arquitetura em três camadas – de dados, de visões SQL e de visões RDF – onde a camada de visões SQL mapeia os conceitos da camada de dados nos termos da camada de visões RDF. A criação desta camada intermediária de visões facilita a geração dos mapeamentos R2RML e evita que alterações na camada de dados impliquem em alterações destes mapeamentos. Adicionalmente, definimos um processo em três etapas para geração das visões RDF. Na primeira etapa, o usuário define o esquema do banco de dados relacional e a ontologia OWL alvo e cria assertivas de correspondência que mapeiam os conceitos do esquema relacional nos termos da ontologia alvo. A partir destas assertivas, uma ontologia exportada é gerada automaticamente. O segundo passo produz um esquema de visões SQL gerado a partir da ontologia exportada e um mapeamento R2RML do esquema de visões para a ontologia exportada, de forma automatizada. Por fim, no terceiro passo, as visões RDF são publicadas em um *SPARQL endpoint*. Neste trabalho são detalhados as assertivas de correspondência, a arquitetura, o processo, os algoritmos necessários, uma ferramenta que suporta o processo e um estudo de caso para validação dos resultados obtidos.

***Palavras-Chave: Bancos de Dados Relacionais, Web Semântica, Linked Data.***

# ABSTRACT

The Linked Data initiative brought new opportunities for building the next generation of Web applications. However, the full potential of linked data depends on how easy it is to transform data stored in conventional, relational databases into RDF triples. Recently, the W3C RDB2RDF Working Group proposed a standard mapping language, called R2RML, to specify customized mappings between relational schemas and target RDF vocabularies. However, the generation of customized R2RML mappings is not an easy task. Thus, it is mandatory to define: (a) a solution that maps concepts from a relational schema to terms from a RDF schema; (b) a process to support the publication of relational data into RDF, and (c) a tool that implements this process. Correspondence assertions are proposed to formalize the mappings between relational schemas and RDF schemas. Views are created to publish data from a database to a new structure or schema. The definition of RDF views over relational data allows providing this data in terms of an OWL ontology structure without having to change the database schema. In this work, we propose a three-tier architecture – database, SQL views and RDF views – where the SQL views layer maps the database concepts into RDF terms. The creation of this intermediate layer facilitates the generation of R2RML mappings and prevents that changes in the data layer result in changes on R2RML mappings. Additionally, we define a three-step process to generate the RDF views of relational data. First, the user defines the schema of the relational database and the target OWL ontology. Then, he defines correspondence assertions that formally specify the relational database in terms of the target ontology. Using these assertions, an exported ontology is generated automatically. The second step produces the SQL views that perform the mapping defined by the assertions and a R2RML mapping between these views and the exported ontology. This dissertation describes a formalization of the correspondence assertions, the three-tier architecture, the publishing process steps, the algorithms needed, a tool that supports the entire process and a case study to validate the results obtained.

***Keywords: Relational Databases, Semantic Web, Linked Data.***

# LISTA DE FIGURAS

Figura 2.1 - Camadas da Web Semântica ..... 17
Figura 2.2 - Grafo RDF com uma única tripla ..... 19
Figura 2.3 - Infraestrutura atual de Linked Data ..... 25
Figura 3.1 - Arquitetura geral da plataforma D2RQ extraído de http://d2rq.org ..... 31
Figura 4.1 - BDR de Entrada ..... 36
Figura 4.2 - Esquema do Banco de Dados ISWC_REL ..... 39
Figura 4.3 - Ontologia de Domínio CONF_OWL ..... 40
Figura 5.1 - Visão RDF *ISWC_RDF* gerada a partir das ACs ..... 51
Figura 6.1 - Arquitetura em Três Camadas ..... 56
Figura 6.2 - Passos do Processo de Publicação ..... 57
Figura 6.3 - Esquema das Visões *ISWC_Views* saída do Algoritmo da Tabela 6.1 ..... 62
Figura 7.1 - Principais Componentes da Ferramenta RBA ..... 74
Figura 7.2 - Configuração para acesso à base *ISWC* ..... 77
Figura 7.3 - Configuração da Ontologia de Domínio ..... 78
Figura 7.4 - Tela Principal para Criação das ACs ..... 79
Figura 7.5 - Criação de uma ACO para a propriedade *dc:creator* ..... 80
Figura 7.6 - Lista com as assertivas criadas até o momento ..... 80
Figura 7.7 - Exemplo de criação de um filtro de seleção ..... 81
Figura 7.8 - Lista de assertivas atualizada ..... 81
Figura 7.9 - Lista de assertivas do estudo de caso ..... 82
Figura 7.10 - Ontologia Exportada a partir das ACs do estudo de caso ..... 83
Figura 7.11 - SQL das visões relacionais criadas ..... 85
Figura 7.12 - Mapeamento R2RML gerado ..... 90
Figura 7.13 - Consultas SPARQL submetida com seu resultado ..... 91
Figura 7.14 - Consulta SQL retornando pessoas e seus assuntos de interesse ..... 92
Figura 7.15 - Consulta SQL traduzida a partir da consulta SPARQL pelo D2RQ ..... 92
Figura 7.16 - Consulta SQL para verificar o resultado da consulta SPARQL da Tabela 7.3 ..93
Figura 7.17 - Consulta SQL para verificar o resultado da consulta SPARQL da Tabela 7.4 ..93

# LISTA DE TABELAS

Tabela 4.1 - Prefixos e namespaces utilizados ..... 35
Tabela 4.2 - Triplas RDF exportadas do BDR ..... 36
Tabela 4.3 - Mapeamento R2RML que produz as triplas da relação Empregados ..... 36
Tabela 4.4 - Triplas geradas pelo mapeamento da Tabela 4.3 ..... 36
Tabela 4.5 - Visão R2RML de Departamentos para a calcular a quantidade de empregados .. 37
Tabela 4.6 - Mapeamento R2RML de Departamentos ..... 37
Tabela 4.7 - Triplas geradas pelo mapeamento da Tabela 4.6 ..... 37
Tabela 4.8 - Alteração no mapeamento <*#TriplesMap1*> da Tabela 4.3 ..... 38
Tabela 4.9 - Tripla gerada pelo mapeamento da Tabela 4.8 ..... 38
Tabela 4.10 - Mapeamento R2RML da classe *foaf:Person* ..... 40
Tabela 4.11 - Mapeamento R2RML da classe *conf:Organization* ..... 41
Tabela 4.12 - Mapeamento R2RML da propriedade de objeto *conf:hasAffiliation* ..... 42
Tabela 4.13 - Mapeamento R2RML da propriedade de objeto *conf:researchInterests* ..... 42
Tabela 5.1 - Predicados embutidos ..... 48
Tabela 5.2 - Regras de Mapeamento ..... 48
Tabela 5.3 - Assertivas de Correspondência entre *ISWC_REL* e *CONF_OWL* ..... 50
Tabela 5.4 - Regras de Transformação induzidas das ACs da Tabela 5.3 ..... 51
Tabela 5.5 - Triplas geradas da aplicação das regras de transformação da Tabela 5.4 no estado da Figura 5.1 ..... 53
Tabela 6.1 - Algoritmo para gerar *EV* e *M_R2RML* ..... 60
Tabela 6.2 - Templates para traduzir assertivas de correspondência simples para mapeamentos R2RML ..... 61
Tabela 6.3 - Mapeamentos R2RML saída do Algoritmo da Tabela 6.1 ..... 62
Tabela 6.4 - Algoritmo para gerar o SQL das Visões Relacionais ..... 70
Tabela 6.5 - Templates para geração das cláusulas SQL ..... 71
Tabela 6.6 - Visões SQL geradas pelo Algoritmo da Tabela 6.4 ..... 71
Tabela 7.1 - Visões Relacionais geradas após a execução do Algoritmo da Tabela 5.4 ..... 84
Tabela 7.2 - Mapeamento R2RML gerado pelo componente GM-R2RML ..... 86
Tabela 7.3 - Consulta SPARQL para descobrir a homepage das empresas do Reino Unido ... 92
Tabela 7.4 - Consulta SPARQL para recuperar as pessoas com a quantidade de tópicos diferentes relacionados ..... 93

# SUMÁRIO


**1 INTRODUÇÃO ..... 11**

1.1 MOTIVAÇÃO ..... 11
1.2 CARACTERIZAÇÃO DO PROBLEMA ..... 13
1.3 CONTRIBUIÇÕES ..... 13
1.4 ORGANIZAÇÃO DA DISSERTAÇÃO ..... 14

**2 FUNDAMENTAÇÃO TEÓRICA ..... 16**

2.1 WEB SEMÂNTICA ..... 16
2.2 RESOURCE DESCRIPTION FRAMEWORK (RDF) ..... 18
*2.2.1 Conceitos e Sintaxe ..... 19*
*2.2.2 Serialização de RDF ..... 20*
*2.2.3 RDF Schema (RDFS) ..... 21*
2.3 WEB ONTOLOGY LANGUAGE (OWL) ..... 22
2.4 SPARQL ..... 23
2.5 LINKED DATA ..... 23
2.6 CONCLUSÕES ..... 27

**3 TRABALHOS RELACIONADOS ..... 28**

3.1 TRIPLIFY ..... 28
3.2 VIRTUOSO ..... 29
3.3 JENA ..... 29
3.4 SESAME ..... 30
3.5 PLATAFORMA D2RQ ..... 30
3.6 LDIF - LINKED DATA INTEGRATION FRAMEWORK ..... 31
3.7 ODCS - ODCLEANSTORE ..... 32
3.8 CONCLUSÕES ..... 33

**4 RDB TO RDF MAPPING LANGUAGE (R2RML) ..... 34**

4.1 LINGUAGEM R2RML ..... 34
4.2 ESTUDO DE CASO ..... 38
4.3 CONCLUSÕES ..... 43

**5 USANDO ASSERTIVAS DE CORRESPONDÊNCIA PARA DEFINIR VISÕES RDF DE DADOS RELACIONAIS ..... 44**

5.1 CONCEITOS BÁSICOS E NOTAÇÃO ADOTADA ..... 44
5.2 DEFINIÇÃO DE ASSERTIVAS DE CORRESPONDÊNCIA ..... 45
5.3 REGRAS DE TRANSFORMAÇÃO INDUZIDAS PELAS ASSERTIVAS DE CORRESPONDÊNCIA ..... 47
5.4 ASSERTIVAS DE CORRESPONDÊNCIA ENTRE *ISWC_REL* E *CONF_OWL* ..... 49

5.5 CONCLUSÕES ..... 54

**6 ABORDAGEM PARA GERAÇÃO DE MAPEAMENTOS R2RML ..... 55**

6.1 VISÃO GERAL ..... 55

*6.1.1 Arquitetura Adotada ..... 55*

*6.1.2 Processo para publicação de dados RDF ..... 56*

6.2 GERAÇÃO DA ONTOLOGIA EXPORTADA ..... 57

6.3 GERAÇÃO DA CAMADA DE VISÕES E DO MAPEAMENTO R2RML ..... 58

*6.3.1 Algoritmo para gerar EV e M_R2RML ..... 58*

*6.3.2 Algoritmo para gerar os SQLs das visões em EV ..... 65*

6.4 PUBLICAÇÃO DA *OE* EM UM SPARQL ENDPOINT ..... 72

6.5 CONCLUSÕES ..... 72

**7 RBA – R2RML BY ASSERTIONS ..... 73**

7.1 PRINCIPAIS COMPONENTES ..... 73

*7.1.1 GUI (Graphical User Interface) ..... 74*

*7.1.2 GEO (Generate Exported Ontology) ..... 75*

*7.1.3 GVS (Generate Views Schema) ..... 75*

*7.1.4 GM-R2RML (Generate Mapping – R2RML) ..... 76*

*7.1.5 D2RQ-Extension ..... 76*

7.2 IMPLEMENTAÇÃO DO ESTUDO DE CASO ..... 77

*7.2.1 Configuração Inicial para Criação de um Mapeamento ..... 77*

*7.2.2 Criação das Assertivas de Correspondência ..... 79*

*7.2.3 Geração da Ontologia Exportada ..... 82*

*7.2.4 Geração das Visões SQL ..... 83*

*7.2.5 Geração do Mapeamento R2RML ..... 85*

*7.2.6 Publicação dos Dados RDF e Testes ..... 90*

7.3 CONCLUSÕES ..... 94

**8 CONCLUSÃO E TRABALHOS FUTUROS ..... 95**

**REFERÊNCIAS BIBLIOGRÁFICAS ..... 96**

# 1 INTRODUÇÃO

A Web é atualmente um enorme espaço global de documentos e dados distribuídos em múltiplas fontes autônomas e heterogêneas, que se expande a cada dia, o que dificulta a recuperação de informações de forma integrada. Neste contexto, surgiu uma nova abordagem, denominada Web de Dados, capaz de viabilizar a integração de dados na Web.

A Web de Dados se baseia em um conjunto de melhores práticas para publicação e consumo de dados estruturados na Web, conhecido como *Linked Data*, que possibilita a interligação de itens entre diferentes fontes de dados e, portanto, a conexão dessas fontes em um único espaço de dados global (HEATH; BIZER, 2011). *Linked Data* fundamenta-se nas tecnologias da Web Semântica e permite reduzir a complexidade de integração de dados devido às ligações estabelecidas e descritas entre os conjuntos de dados. Além disso, a adoção de um modelo de dados padronizado (modelo RDF) e de um mecanismo padronizado de acesso aos dados (linguagem SPARQL), juntamente com a natureza autodescritiva dos dados também simplificam sua integração. A arquitetura aberta de *Linked Data* ainda permite a descoberta de novos dados e mesmo de conjuntos de dados em tempo de execução. Desse modo, *Linked Data* tem o potencial de facilitar o acesso aos dados semanticamente relacionados, estabelecendo conexões explícitas entre conjuntos de dados distintos a fim de facilitar sua integração.

## 1.1 Motivação

A grande maioria dos dados corporativos, inclusive dados da Web, permanece armazenada em Bancos de Dados Relacionais (BDRs). Embora BDRs sejam adequados para tratar grandes quantidades de dados, eles não são projetados para preservar a semântica destes dados. Esta semântica está implícita no nível de aplicação e não explicitamente codificada no modelo relacional. Ontologias e tecnologias da Web Semântica provêem explicitamente esta semântica para compartilhamento dos dados entre aplicações, empresas e comunidades. Resumidamente essas tecnologias são:

- **RDF** (*Resource Description Framework*), um modelo de dados simples, expressivo, extensível e que permite interligar itens entre diferentes fontes de dados;
- **URI** (ou **IRI**), usado como mecanismo de nome global;
- Linguagem **SPARQL** (PRUD'HOMMEAUX; SEABORNE, 2008), a linguagem de consulta recomendada pela W3C (*World Wide Web Consortium*) para recuperar e manipular dados armazenados no formato RDF.

Para tornar as informações armazenadas em BDRs disponíveis na Web de dados é preciso publicá-las no modelo RDF. Essa publicação pode utilizar o enfoque virtual ou o enfoque materializado. Na primeira abordagem são criados *endpoints* SPARQL/RDF que recebem consultas SPARQL referenciando termos de uma ontologia, essas consultas são traduzidas em consultas SQL e executadas no BDR. O *dataset* retornado do BDR é transformado em um grafo RDF que é o resultado final da consulta SPARQL original. Na segunda abordagem as tuplas do BDR são exportadas para triplas de uma ontologia alvo. O resultado desta conversão pode ser armazenado em um banco de dados de triplas RDF e pode ser consultado utilizando a linguagem SPARQL diretamente.

Os dois enfoques de publicação podem ser viabilizados pela utilização de um mapeamento formal entre os conceitos do BDR e os termos da ontologia. Este mapeamento permite a visualização de dados relacionais existentes na forma de um modelo de dados RDF. Neste trabalho adotamos o enfoque virtual que tem como vantagem a não necessidade de manutenção das triplas RDF quando os dados relacionais são atualizados. Portanto, o enfoque materializado está fora do escopo desta dissertação.

Motivados pelo sucesso da iniciativa *Linking Open Data*[1] e pelo grande crescimento das fontes de dados disponíveis na Web, novas ferramentas de publicação de dados relacionais em RDF estão surgindo. Estas ferramentas são descritas no Capítulo 3 que trata dos trabalhos relacionados. Adicionalmente, uma linguagem de mapeamentos relacional para RDF padrão (R2RML) foi especificada pela W3C. R2RML será explicada com exemplos no Capítulo 4.

[1] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

## 1.2 Caracterização do Problema

Ferramentas de mapeamento existentes utilizam linguagens próprias para construção dos mapeamentos. Além disso, estas ferramentas não disponibilizam interfaces de fácil utilização para construção dos mapeamentos. Assim, os mapeamentos são criados manualmente, o que dificulta sua manutenção quando o esquema da base de dados sofre alterações. Adicionalmente, podem existir problemas de heterogeneidade entre o esquema do BDR Fonte e o esquema da ontologia alvo, dificultando a criação dos mapeamentos. As ferramentas não auxiliam na identificação destes problemas e não possuem mecanismos para tratá-los. Outro problema encontrado é que muitas publicações não utilizam as melhores práticas, pois não seguem nenhum processo formal, e acabam criando mapeamentos diretos do esquema relacional, criando um novo vocabulário ao invés de reutilizar termos de vocabulários conhecidos.

## 1.3 Contribuições

As principais contribuições deste trabalho são:

1. A definição e validação, através de estudo de caso, de uma arquitetura de três camadas para facilitar a geração de mapeamentos R2RML.
2. A definição de um processo em três etapas para publicação dos dados relacionais seguindo as melhores práticas de *Linked Data*.
3. A formalização dos mapeamentos customizados por meio de Assertivas de Correspondência.
4. A especificação e implementação de uma ferramenta gráfica para apoiar a execução do processo de publicação a qual possui funcionalidades importantes como: a construção das assertivas de correspondência entre esquemas relacionais e esquemas RDF, a geração automática das visões SQL e a geração de um arquivo de mapeamentos R2RML.

As três primeiras contribuições deram origem a um artigo a ser publicado no *29th Symposium On Applied Computing* (VIDAL et al., 2014) e a quarta contribuição deu origem a um artigo *demo* apresentado no *10th ESWC* (NETO et al., 2013).

## 1.4 Organização da Dissertação

Esta dissertação possui oito capítulos. Não sendo mais necessário tratar deste capítulo introdutório, os demais são delineados a seguir.

O Capítulo 2 – Fundamentação Teórica – apresenta uma síntese dos assuntos mais relevantes que servem de fundamentação para o entendimento dos demais capítulos desta dissertação. Ele expõe os principais conceitos relativos à publicação de dados relacionais em RDF: Web Semântica, RDF, OWL (*Web Ontology Language*), SPARQL; além de tratar sobre *Linked Data*.

O Capítulo 3 – Trabalhos Relacionados – apresenta os trabalhos relacionados ao contexto de mapeamento e publicação de dados relacionais em triplas RDF. Neste capítulo é apresentada uma síntese das principais ferramentas: Triplify, Virtuoso, Jena, Sesame e D2RQ. Dentre elas destacamos o D2RQ por ter integração com a ferramenta sugerida no Capítulo 7.

O Capítulo 4 – RDB to RDF Mapping Language (R2RML) – apresenta a linguagem R2RML e o estudo de caso utilizado ao longo deste trabalho. Este capítulo serve de motivação para justificar a adoção da abordagem proposta.

O Capítulo 5 – Usando Assertivas de Correspondência para definir Visões RDF de Dados Relacionais – contém a definição do formalismo de regras sugerido para criação de mapeamentos entre um esquema relacional fonte e uma ontologia alvo. Estas regras são aplicadas ao estudo de caso e a visão RDF resultante é apresentada.

O Capítulo 6 – Abordagem Proposta – propõe uma arquitetura de três camadas baseada no uso de visões SQL. Adicionalmente, é proposto um processo para criação da arquitetura e de um *SPARQL endpoint*. Neste capítulo são apresentados os detalhes da arquitetura, os passos do processo para publicação de dados relacionais em RDF, os algoritmos responsáveis por criar as visões SQL e gerar o mapeamento R2RML.

O Capítulo 7 – RBA (R2RML By Assertions) – apresenta nossa ferramenta gráfica para criação dos mapeamentos e publicação da visão RDF dos dados relacionais. Usamos o estudo de caso introduzido no Capítulo 4 para validação da ferramenta.

Por fim, o Capítulo 8 – Conclusão – tece as considerações finais sobre o trabalho e apresenta possíveis trabalhos futuros para dar prosseguimento ao que obtivemos até o momento.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo tratamos de alguns conceitos cujo entendimento é essencial para a compreensão do contexto em que esta dissertação está inserida, bem como da fundamentação necessária ao entendimento dos capítulos seguintes.

A seção 2.1 dá uma visão geral da Web Semântica desde a sua origem até as camadas da sua arquitetura. A seção 2.2 explica os conceitos, a sintaxe e exemplifica o uso do *framework* RDF. A seção 2.3 apresenta a linguagem OWL e suas sublinguagens. A seção 2.4 introduz a linguagem SPARQL para consultas em grafos RDF. A seção 2.5 apresenta os principais componentes da infraestrutura de *Linked Data*. Para finalizar, a seção 2.6 apresenta as conclusões.

## 2.1 Web Semântica

A visão da Web Semântica é a de fazer com que as informações na Web possam ser processadas por máquinas e não somente compreendida por humanos. Embora a origem da Web Semântica esteja tipicamente ligada ao artigo publicado por Tim Berners Lee, James Hendler e Ora Lassila em Maio de 2001 (BERNERS-LEE et al., 2001), sua história iniciou nos anos 90 quando Tim Berners-Lee começava a desenvolver a WWW (*World Wide Web*). A ideia de adicionar semântica às páginas Web e *links* já fazia parte de seus planos e projetos originais. Porém a ideia de combinar RDF com lógicas descritivas para implementar a Web Semântica só foi formulada dez anos mais tarde.

A Figura 2.1 apresenta a arquitetura da Web Semântica (HORROCKS et al., 2005), a qual ilustra os diferentes conceitos que compõem o núcleo da Web Semântica organizados na forma de uma pilha.

A camada base é constituída de dois conceitos básicos *Unique Resource Identifiers* (URIs) e o padrão UNICODE de codificação de caracteres para garantir compatibilidade com todas as línguas mundialmente conhecidas. Devemos notar que existe uma abreviação mais recente IRI que é usada para denotar *Internationalized Unicode-encoded URI*.

A camada “XML” faz parte da arquitetura, pois RDF é originalmente serializado em XML. Isto ocorre principalmente porque XML tornou-se um padrão amplamente difundido para troca de dados. Mais tarde, foram propostos formatos de serialização mais compactos para representar grafos RDF como, por exemplo, *Turtle*, *Notation 3* e *N-Triples*. Na seção 2.3.2 mostraremos exemplos de serialização de grafos RDF nas notações Turtle e RDF/XML. Todo recurso ou nó descrito em um grafo RDF é unicamente identificado por uma URI. Isto permite que descrições de um mesmo recurso distribuídas na Web possam ser mescladas em um único grafo RDF.

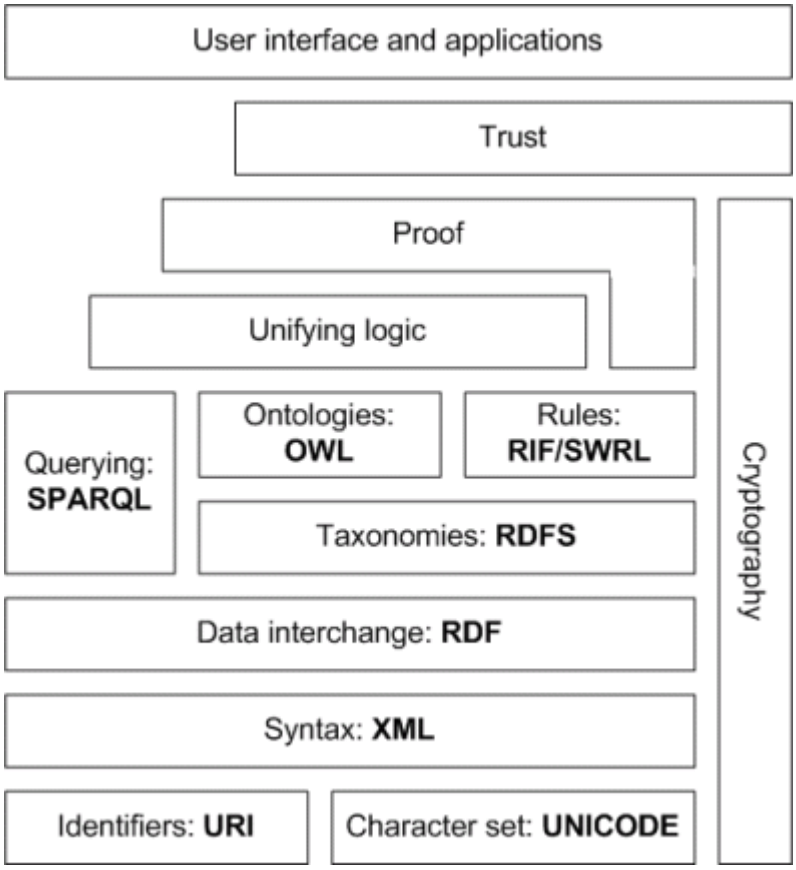


**Figura 2.1 - Camadas da Web Semântica**

Acima da camada “RDF”, RDF *Schema* (RDFS) pode ser usada para declarar classes pré-definidas. Estas classes podem ser utilizadas para adicionar restrições de tipo a recursos RDF. RDF *Schema* será descrito na seção 2.3.3. Enquanto RDFS trata da tipagem dos recursos, OWL pode ser usada para classificar recursos por meio de lógicas descritivas. OWL é o sucessor do DAML+OIL e tem três variantes com diferentes níveis de expressividade: OWL Light, OWL-DL e OWL Full. OWL e suas variantes serão introduzidas na seção 2.4.

Como restrições em OWL-DL são baseadas em lógica descritiva, não é possível criar restrições que envolvam várias classes. Por exemplo, com OWL não podemos expressar:

um ônibus pode transportar uma quantidade dez vezes maior de passageiros que um carro de passeio. Para suportar esse tipo de regra foi proposta a linguagem SWRL (HORROCKS et al., 2004).

Além do *framework* para lidar com ontologias, existe a linguagem de consultas SPARQL (apresentada na seção 2.5) que está empilhada acima da camada "RDF". SPARQL é a linguagem padrão da Web Semântica para recuperação de informações contidas em grafos RDF.

A camada *Proof* indica que deve haver meios para detectar e explicar os processos lógicos utilizados pelos *reasoners* semânticos. Por fim, existe o *framework Trust* no topo da pilha. Geralmente, a confiabilidade pode ser assegurada pelo uso de uma infraestrutura de chaves públicas. Em geral, dados RDF podem ser assinados e encriptados por meio do uso de certificados digitais emitidos por autoridades certificadoras. Isto está previsto na camada *Cryptography* que é apresentada na Figura 2.1 na vertical englobando todas as demais camadas.

## 2.2 Resource Description Framework (RDF)

RDF é um *framework* para representar informações na Web de uma forma flexível, com o mínimo de restrições e cujo vocabulário é conhecido como RDF *Core*. RDF pode ser utilizado em aplicações isoladas com formato de dados proprietários. No entanto, seu modelo de dados genérico agrega mais valor quando utilizado para compartilhamento de informações. Dessa forma, o valor da informação cresce à medida que ela se torna acessível a um maior número de aplicações na Web. RDF é o principal modelo de dados utilizado nas aplicações da Web Semântica. Toda informação adicionada por componentes de mais alto nível como RDF *Schema* e OWL é modelada em RDF. Dentre os documentos que especificam o RDF, o mais importante deles é o *RDF Primer*, pois este introduz os conceitos básicos do *framework* RDF. Ele é um resumo dos demais documentos e contém as informações básicas que são necessárias para o uso efetivo do RDF. O *RDF Primer* também explica como definir vocabulários usando a linguagem RDF *Schema*.

Nas seções que seguem discutimos os conceitos básicos do RDF, os principais formatos de serialização e a linguagem RDF *Schema*.

### *2.2.1 Conceitos e Sintaxe*

***Grafos como Modelo de dados***

A estrutura por trás de qualquer expressão RDF é uma coleção de triplas, onde cada tripla consiste de *sujeito*, *predicado (ou propriedade)* e *objeto* (s, p, o). Um conjunto de triplas é denominado grafo RDF. O poder do RDF está na simplicidade desse modelo de dados. Uma sentença do tipo: "A página Web https://sites.google.com/site/luiseufrasio/ foi criada por Luís Eufrasio." pode ser representada pelo grafo RDF mostrado na Figura 2.2. Neste exemplo, o sujeito é a URI *<https://sites.google.com/site/luiseufrasio/>*, o predicado é *dc:creator* (propriedade definida no vocabulário Dublin Core) e o literal *"Luís Eufrasio"* é o objeto. Este exemplo mostra como uma tripla RDF pode ser ilustrada na forma "nó-arco-nó". O sujeito e o objeto são os nós e o predicado é o arco que os liga. A direção do arco é muito significante, pois sempre parte do sujeito e aponta para o objeto.


**Figura 2.2 - Grafo RDF com uma única tripla**

***Vocabulário baseado em URIs e Identificadores de Nós***

URI é uma sequência compacta de caracteres que identifica um recurso físico ou abstrato. A utilização de URIs (BERNERS-LEE et al., 2005) é um dos princípios básicos: tudo que pode ser descrito por alguém na Web deve ter uma URI e, para possibilitar a recuperação da informação sobre um recurso, clientes HTTP devem ser capazes de resolverem tais URIs. Um nó pode ser uma URI, um literal ou *blank* (sem identificação). Predicados são sempre URIs. Uma URI ou um literal usados como nós identificam o que o nó representa. Uma URI usada como predicado identifica um relacionamento entre os nós conectados. Um *blank* é um nó que não é uma URI e também não é um literal, ou seja, é um nó que pode ser utilizado em sentenças RDF mas não possui nenhum nome associado.

***Tipos de Dados***

Um tipo de dados é usado em RDF na representação de valores como *inteiros*, *numéricos de ponto flutuante* e *datas*. Os tipos de dados pré-definidos no *XML Schema* (xsd) e identificados

através de URIs são utilizados pelo RDF, pois este pré-define somente um tipo de dado *rdf:XMLLiteral*. No entanto, novos tipos de dados podem ser definidos usando *XML Schema* como, por exemplo, *xsd:integer* para definir o tipo inteiro.

***Literais***

São utilizados para identificar valores como números e datas por meio de suas representações léxicas. Tudo que pode ser representado como um literal também pode ser representado como uma URI, porém em alguns casos é mais conveniente ou intuitivo usar literais. Um literal pode ser um objeto de uma tripla RDF, mas não pode ser o sujeito ou o predicado. Seu valor pode ser um texto livre ou tipado.

### *2.2.2 Serialização de RDF*

Existem muitos formatos de serialização para representar grafos RDF. O primeiro formato definido pela W3C foi o RDF/XML. Ele ainda é a sintaxe padrão para publicação de vocabulários RDF e para troca de dados na Web. Como a sintaxe XML é muito volumosa e difícil de ser lida por pessoas, sintaxes alternativas foram propostas. Como foram os casos da *Notation 3* (N3) proposta por Tim Berners-Lee (1998) e dos seus subconjuntos, dos quais a sintaxe *Turtle* (Beckett 2007) tornou-se a mais popular. A sintaxe de Turtle é mais legível, intuitiva e compacta quando comparada à sintaxe RDF/XML. Turtle é usada nos exemplos do estudo de caso dessa dissertação e na linguagem R2RML que será apresentada na seção 2.7.

O grafo da Figura 2.2 pode ser representado usando a sintaxe Turtle da seguinte forma:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
<https://sites.google.com/site/luiseufrasio/> dc:creator “Luís Eufrasio” .
```

O mesmo grafo quando representado usando a sintaxe RDF/XML:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:dc="http://purl.org/dc/elements/1.1/">
      <rdf:Description
            rdf:about="https://sites.google.com/site/luiseufrasio/">
            <dc:creator>Luis Eufrasio</dc:creator>
      </rdf:Description>
```

```
</rdf:RDF>
```

### *2.2.3 RDF Schema (RDFS)*

RDF *Schema* estende o vocabulário RDF *Core*. Ele contém vários conceitos pré-definidos como *rdfs:Class*, *rdfs:Property*, etc. que são usados para definir novas classes e propriedades.

***Classes e Propriedades***

No RDF, basicamente todo recurso pode ser utilizado como um predicado ou como uma classe (usando a propriedade *rdf:type*). Por exemplo:

```
<https://sites.google.com/site/luiseufrasio/foaf.rdf>
      rdf:type foaf:Person .
<https://sites.google.com/site/luiseufrasio/> dc:creator
      <https://sites.google.com/site/luiseufrasio/foaf.rdf> .
```

Entretanto, para entendermos a semântica de *foaf:Person* e *dc:creator*, estes recursos precisam estar descritos em algum lugar. A definição de *foaf:Person* é parte do vocabulário Friend-of-a-Friend (BRICKLEY; MILLER 2007) publicado em http://xmlns.com/foaf/0.1/. Para criar uma nova classe RDF basta declará-la como membro da classe *rdfs:Class*. Similarmente à classe *foaf:Person*, a definição da propriedade RDF *dc:creator* é parte do vocabulário *Dublin Core* publicado em http://purl.org/dc/elements/1.1/. Neste vocabulário, *dc:creator* é definido como sendo do tipo *rdfs:Property*.

***Domínio e Imagem de Propriedades***

Dada uma propriedade específica *p*, o conjunto de triplas RDF (*s*, *p*, *o*) pode ser interpretada como uma relação binária *p*(*s*, *o*) que associa um objeto *o* a um sujeito *s*. Usando esta notação, o domínio $D_p$ é o conjunto de todos os possíveis valores de *s* e a imagem $I_p$ é o conjunto de todos os possíveis valores de *o*. RDFS define duas propriedades *rdfs:domain* e *rdfs:range* usadas respectivamente para definir o domínio e a imagem de uma propriedade RDF. As propriedades RDF são definidas globalmente. Dessa forma, se uma propriedade não define domínio e imagem então ela pode ser usada com qualquer recurso independente da classe

deste recurso. Esta é a maior diferença do esquema RDF para um esquema relacional, onde atributos precisam ser definidos dentro do contexto de uma relação específica.

## 2.3 Web Ontology Language (OWL)

A W3C recomenda a utilização da linguagem OWL por aplicações que precisam processar o conteúdo de documentos, e não somente disponibilizar este conteúdo para que pessoas possam lê-lo. OWL pode ser usada para tornar explícito o significado de termos de vocabulários e os relacionamentos entre estes termos. Esta representação de termos e seus relacionamentos é chamada de ontologia.

OWL é mais abrangente que XML, RDF e RDF-S, pois provê vocabulários adicionais com uma semântica formal. OWL pode ser considerada uma versão revisada da linguagem DAM+OIL e possui três sublinguagens apresentadas em ordem crescente de expressividade:

- *OWL Lite*: suporta classificação hierárquica e restrições simples. Por exemplo, embora suporte restrições de cardinalidade, somente são permitidos os valores de cardinalidade 0 e 1. Sua formalização é menos complexa que a da OWL DL.
- *OWL DL*: suporta o máximo de expressividade que possa ser computado e decidido em um tempo finito. Inclui todas as estruturas da linguagem OWL, porém elas devem ser utilizadas respeitando certas restrições. Por exemplo, embora uma classe possa ser subclasse de mais de uma classe, uma classe não pode ser instância de uma outra classe. A sigla DL significa *Description Logics*, uma área de pesquisa dedicada a estudar as lógicas utilizadas na formalização da OWL.
- *OWL Full*: suporta o máximo de expressividade com a liberdade sintática do RDF, porém sem garantias computacionais. Por exemplo, em OWL Full uma classe pode ser tratada simultaneamente como uma coleção de indivíduos ou como um único indivíduo dentro do seu contexto. Com OWL Full uma ontologia pode aumentar o vocabulário (RDF ou OWL) pré-definido.

Mais recentemente, surgiu a *OWL 2* que tornou-se a nova recomendação da *W3C*. *OWL 2* é compatível com a *OWL* (agora denominada *OWL 1*). Embora a *OWL 1* tenha sido

um sucesso, alguns problemas foram identificados. Nenhum destes problemas são críticos, mas somados indicaram a necessidade de uma revisão da *OWL 1*. Uma das limitações da *OWL 1* é a carência de tipos de dados adequados, pois baseia-se em XML esquema (xsd) para definição destes tipos. *OWL 2* melhora consideravelmente a definição dos tipos de dados além de fornecer uma plataforma robusta para desenvolvimento futuro.

## 2.4 SPARQL

Grafos RDF podem ser consultados utilizando a linguagem SPARQL (PRUD'HOMMEAUX; SEABORNE, 2008), que é a linguagem padrão de consultas da Web Semântica. Entretanto, SPARQL não é somente uma linguagem declarativa, mas também um protocolo que envia consultas e recupera os resultados via HTTP.

Tipicamente os dados RDF são disponibilizados na Web por meio da criação de um *SPARQL endpoint*, um serviço Web com suporte ao protocolo SPARQL. Este serviço é acessado através de uma URI específica, a qual é criada para receber requisições HTTP com consultas SPARQL passadas como parâmetro e retorna os resultados de forma estruturada. Estes resultados podem vir em diferentes formatos dependendo dos comandos SPARQL enviados. Por exemplo, consultas usando os comandos SELECT e ASK geralmente terão retornos nos formatos XML, JSON ou texto plano. Já quando utilizamos o comando CONSTRUCT, os resultados normalmente usam os formatos RDF/XML, Turtle ou N3.

## 2.5 Linked Data

Em Julho de 2006, Tim Berners-Lee publicou um artigo na Web sobre a ideia de *Linked Data* (BERNERS-LEE T., 2006). Enquanto a WWW consiste de documentos HTML conectados por *hiperlinks*, a Web Semântica consiste de recursos de diversos tipos conectados por propriedades (ou *links* semânticos). Para que possamos acessar e navegar em grafos RDF de uma forma mais intuitiva, Berners-Lee propôs quatro regras de *design*:

1. Use URIs para nomear coisas.

2. Use URIs HTTP para que as pessoas façam buscas pelos nomes das coisas.
3. Se alguém busca uma URI, informações úteis devem ser fornecidas usando os padrões (RDF e SPARQL).
4. Links semânticos devem apontar para outras URIs para que as pessoas possam descobrir mais coisas.

Em 2007, muitos projetos foram lançados para dar suporte a esta ideia e promover a adoção dos paradigmas de *Linked Data* (AUER et al., 2007). Uma grande quantidade de dados disponíveis publicamente, dados gerados por usuários, e dados abertos governamentais foi publicada desde então. Por outro lado, existe a ideia de uma *Web de Dados* pública para adicioná-la à tradicional WWW. Os princípios de *Linked Data* também podem ser adaptados em soluções empresariais para integração de dados corporativos.

Na Figura 2.3 temos a infraestrutura atual da Web de Dados. A figura mostra no centro uma aplicação cliente que normalmente será um navegador comum ou um navegador especializado em *Linked Data* como o *Tabulator* (BERNERS-LEE et al., 2006), por exemplo. Também exibe: (i) alguns recursos RDF distribuídos na Web, (ii) alguns *datasets* RDF com um *SPARQL endpoint* e uma interface *Linked Data*, (iii) um motor de buscas da Web Semântica, (iv) um localizador de serviços, e.g. *Sindice* (TUMMARELLO et al., 2007), e (v) um mediador, p.e. *SemWIQ* (LANGEGGER, 2010). Todos estes componentes são descritos em seguida.

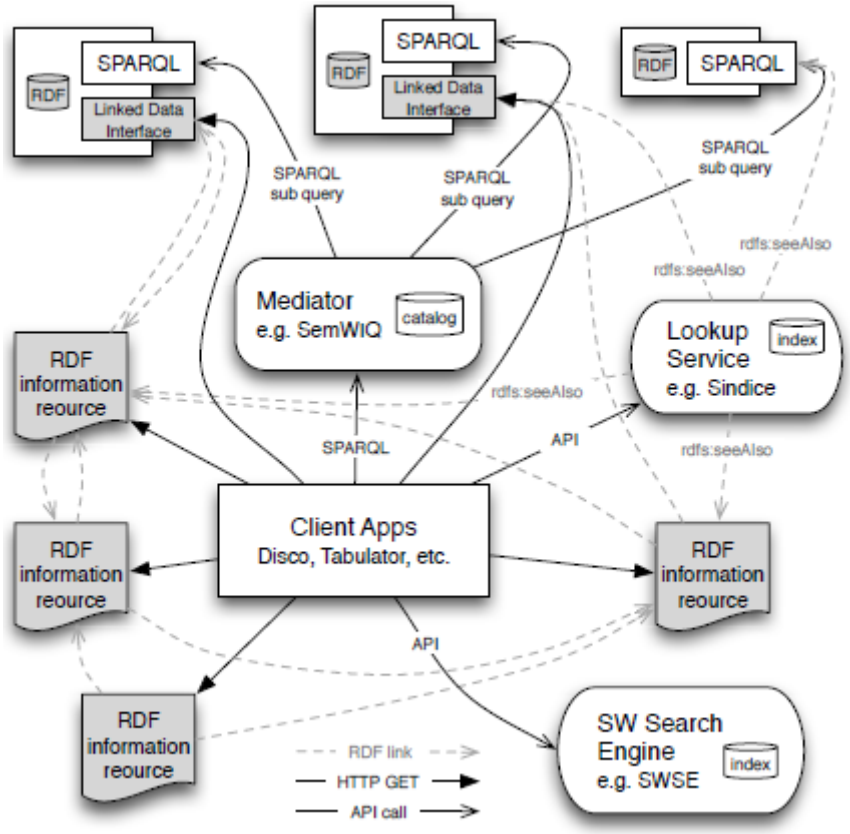


**Figura 2.3 - Infraestrutura atual de Linked Data**

***Aplicações Cliente***

Atualmente, os clientes mais populares são navegadores *Linked Data*. Eles interpretam dados RDF e disponibilizam visões estruturadas e bem definidas destes dados, normalmente na forma de grupos de sujeitos, propriedades e valores (literais ou *links* para recursos).

***Recursos RDF***

De acordo com a W3C *Technical Architecture Group* (TAG), um recurso RDF é um documento Web contendo dados RDF. Ele é identificado por uma URI HTTP completa e pode ser dinamicamente gerado a partir de um banco de dados. Entretanto, sua aparência é de um simples documento Web. Um recurso RDF contém um conjunto de sentenças RDF descrevendo vários recursos, onde alguns deles podem ser referências a *links* RDF externos.

***SPARQL Endpoints***

SPARQL *Endpoints* fornecem acesso a *datasets* RDF maiores através do uso da linguagem de consultas declarativa SPARQL e de um protocolo baseado em HTTP-REST. Para evitar ataques do tipo *deny of service* (DoS), alguns *endpoints* possuem restrições sobre o tamanho dos conjuntos de dados retornados. Podemos assumir que uma boa quantidade dos dados disponíveis será provida através de *endpoints* SPARQL.

***Linked Data Interfaces***

Como os SPARQL *endpoints* fornecem apenas acesso declarativo, não é possível navegar sobre os recursos encapsulados pelo *endpoint* sem ter que empacotar a informação executando consultas do tipo DESCRIBE, por exemplo. Uma *Linked Data interface* é um *wrapper* RDF-para-HTML anexado ao *endpoint*, e.g *Pubby* (CYGANIAK; BIZER 2009). Ela resolve URIs de recursos, executa as consultas, e retorna uma resposta HTTP em tempo de execução.

***Motor de Buscas da Web Semântica – em inglês: Semantic Web Search Engine (SWSE)***

SWSEs utilizam rastreadores especializados que procuram por dados RDF na Web, os quais usualmente são identificados no campo *Content-type* do cabeçalho de resposta HTTP. Eles também possuem uma interface Web e/ou uma API com várias opções de busca. Em geral, um SWSE possui busca por palavras chave que retorna uma lista de recursos RDF. Em um segundo momento, o usuário pode restringir os recursos RDF exibidos criando filtros por tipos de recursos conhecidos.

***Localizador de Serviços – em inglês: Lookup Service***

Enquanto os motores de busca guardam todos os dados (especialmente arquivos RDF e OWL), a ideia dos localizadores de serviços é usar índices específicos para armazenar somente a informação que for mais relevante. Por exemplo, o *Sindice* só armazena um índice para as URIs, as propriedades que são inversa funcional e seus valores, e as palavras chave. *Sindice* também pode se integrar a navegadores *Linked Data* ou outros clientes semelhantes. Existe uma API que retorna uma lista de propriedades *rdf:seeAlso* cujos valores são URIs.

***Mediadores***

Um mediador pode ser utilizado para consulta a *datasets* RDF distribuídos a partir de um único ponto de acesso. Ele pode ser visto como um novo serviço que é parte da Web de Dados e que pode ser usado por outros clientes e aplicações.

## 2.6 Conclusões

Este capítulo apresentou uma síntese dos assuntos mais relevantes que servem de fundamentação para o entendimento dos demais capítulos desta dissertação. Foram expostos os principais conceitos da Web Semântica, do modelo RDF, do RDF *schema*, das linguagens OWL e SPARQL, além de tratar também sobre *Linked Data*.

# 3 TRABALHOS RELACIONADOS

Esta seção aborda ferramentas e tecnologias dedicadas a mapear dados relacionais em triplas RDF. Algumas delas, além de realizar a conversão dos dados entre os dois modelos (Relacional e *RDF Schema*), também possibilitam: a publicação das triplas utilizando *Linked Data interfaces* ou *SPARQL endpoints*; a persistência das triplas em memória ou em *RDF stores*. A seção 3.1 apresenta a ferramenta Triplify. A seção 3.2 introduz a plataforma Virtuoso. A seção 3.3 mostra as principais características do *framework* Jena. A seção 3.4 resume as funcionalidades do *framework* Sesame. A seção 3.5 apresenta a plataforma D2RQ. A seção 3.6 aborda o *framework* LDIF para integração de dados ligados. A seção 3.7 apresenta o *framework* ODCleanStore para gerenciamento de dados ligados. Por fim, a seção 3.8 mostra as conclusões deste capítulo.

## 3.1 Triplify

O Triplify (AUER et al., 2009) é um *plugin* para ser instalado em aplicações Web. Converte o conteúdo de bancos relacionais para os formatos RDF, JSON ou *Linked Data*. Esta ferramenta é muito leve, consistindo de pouco menos de 500 linhas de código. É de fácil configuração, em média leva-se uma hora para ser configurada e se a aplicação precisar ser implantada várias vezes esta configuração pode ser reutilizada sem modificações. Os mapeamentos são simples e criados através de consultas SQL (visões). Assim, o trabalho do Triplify é converter os resultados destas consultas em RDF, JSON ou Linked Data.

Algumas limitações da ferramenta no momento em que este trabalho foi escrito:

1. Ainda estava na versão beta.
2. O desempenho só é garantido para aplicações Web com uma base de dados de no máximo 100 megabytes.
3. Não há garantia de confidencialidade dos dados. Dados confidenciais devem ser omitidos das consultas SQL dos mapeamentos.

## 3.2 Virtuoso

A plataforma Virtuoso (ERLING; MIKHAILOV, 2006) é bastante abrangente. Sua proposta é trabalhar com vários modelos (servidor de dados multimodal) e realizar o gerenciamento, acesso, integração e conversão de seus conteúdos. Os modelos de dados utilizados são XML, SQL, RDF e texto livre. Além disso, também se propõe a ser: (a) um servidor Web de Documentos, (b) um servidor Linked Data, (c) um servidor de aplicações Web, e (d) um repositório de serviços Web (SOAP ou REST).

No contexto deste trabalho, que é o mapeamento de dados relacionais para RDF, o Virtuoso possui uma linguagem proprietária *Linked Data Views* que foi criada antes da existência do R2RML. No entanto, ele suporta a linguagem R2RML por meio da instalação de um pacote (R2RML VAD) que realiza a tradução da sintaxe do R2RML para visões em *Linked Data Views*. Desta forma, os dados relacionais podem ser publicados em RDF utilizando mapeamentos R2RML e acessados via consultas SPARQL.

## 3.3 Jena

O *Jena Semantic Web Framework for Java* (CARROLL et al., 2004) fornece APIs para que aplicações da Web Semântica possam manipular grafos RDF. Jena foi originalmente desenvolvido no *HP Labs* em Bristol entre os anos de 2000 e 2009, mas tornou-se um projeto de código aberto em meados de 2009 fazendo parte do projeto Apache até hoje.

As principais características do *Jena* são:

- Suporte aos formatos *XML*, *N3* e *N-Triples* para leitura e escrita de dados RDF tanto em memória (*Jena SDB*) como em disco (*Jena TDB*);
- Suporte a *RDF*, *RDFa*, *RDFS* e *OWL*;
- Uma *API* para consultas federadas utilizando a linguagem *SPARQL* (Jena ARQ);
- Disponibilização de um *SPARQL endpoint* (*Joseki* ou *Fuseki*);
- Possui motores de inferência embutidos e interfaces para motores de inferência externos.

## 3.4 Sesame

Similar ao Jena, o Sesame (BROEKSTRA; KAMPMAN, 2001) é um *framework* da Web Semântica para armazenamento e consulta de dados e esquemas RDF. Quando concebido em 2001 representou um enorme avanço por ter sido a primeira ferramenta pública que realizava estas consultas utilizando a linguagem SeRQL. Também pode ser configurado com relação à forma de armazenamento (em memória ou *RDF store*), possui mecanismos de inferência e suporta diferentes formatos de arquivo RDF.

Por ser extensível, foram desenvolvidos alguns *plugins* e extensões. Como por exemplo:

- **Virtuoso Sesame Provider**: com ele é possível consultar dados armazenados no banco de dados do Virtuoso utilizando a API do Sesame.
- **Adaptador do Sesame para o Oracle**: integra a API do Sesame com as tecnologias da Web Semântica desenvolvidas para os bancos de dados Oracle.
- **Mecanismo D2RQ**: possibilita a integração do D2RQ com a API do Sesame.

## 3.5 Plataforma D2RQ

O D2RQ é uma plataforma para acessar bancos de dados relacionais na forma de grafos RDF. O acesso aos dados relacionais é virtual e somente para leitura. Assim, não é necessário replicar estes dados dentro de uma base RDF. Entretanto, também é possível criar *dumps* dos dados relacionais em formato RDF e carregar estes *dumps* em bases RDF. A plataforma é formada pelos seguintes componentes:

- **Linguagem D2RM** (BIZER, 2003): uma linguagem de mapeamentos declarativa para criação de correspondências entre conceitos do modelo relacional em termos do modelo RDF. Mapeamentos nesta linguagem são documentos RDF escritos usando a sintaxe N3.
- **Servidor D2R** (BIZER; CYGANIAK, 2006): é um servidor HTTP que disponibiliza uma interface *Linked Data* e um *SPARQL endpoint* sobre o banco relacional.

- **Motor de Regras D2RQ** (em inglês D2RQ Engine): é o responsável por interpretar os mapeamentos D2RM. Utilizado pelo Servidor D2R para transformar consultas SPARQL em consultas SQL. No entanto, também pode ser usado como um *plugin* para os *frameworks Jena* e *Sesame*. Neste caso os mapeamentos D2RM são utilizados para converter chamadas às APIs desses *frameworks* em consultas SQL no banco relacional.

Estes componentes e a integração entre eles podem ser observados na Figura 3.1 que apresenta a arquitetura geral da plataforma D2RQ. *SPARQL Clients*, *Linked Data Clients* e *HTML Browsers* submetem consultas *SPARQL* para o *D2R Server* via protocolo HTTP. O *D2R Server* encaminha as consultas para o *D2RQ Engine* que utiliza o *D2RQ Mapping* para traduzí-las em consultas *SQL* no banco de dados (*Non-RDF Database*). A *D2RQ Engine* pode receber consultas diretamente a partir de *Local Java Applications* via *API Jena ou Sesame* e pode gerar *RDF dumps* para *Triple Stores*.

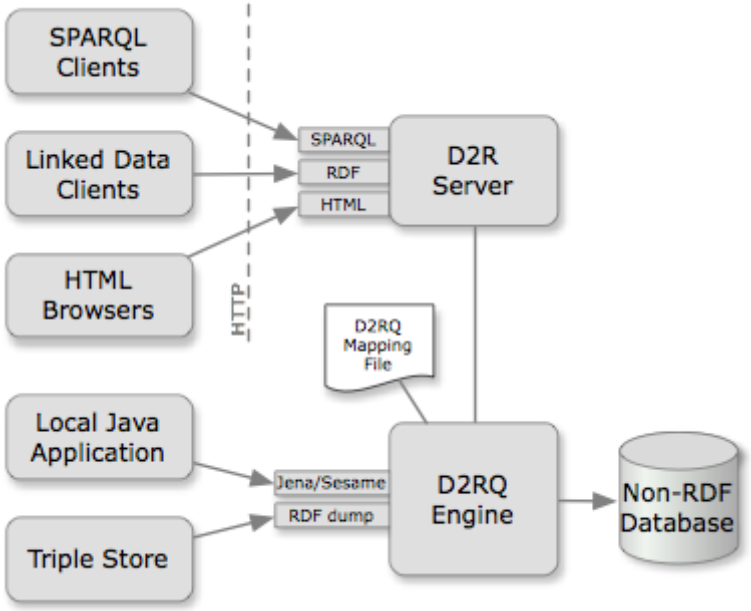


**Figura 3.1 - Arquitetura geral da plataforma D2RQ extraído de http://d2rq.org**

## 3.6 LDIF - Linked Data Integration Framework

O LDIF (SCHULTZ et al., 2011) é um *framework* para integração de dados ligados que se propõe a:

- Coletar dados: gerenciar *downloads* e atualizações;

- Traduzir os dados para um vocabulário alvo comum;
- Resolver os *aliases* dos identificadores em *URIs* locais;
- Gerar a saída dos dados.

A coleta dos dados pode ser feita através da criação de *RDF dumps* em vários formatos ou pela criação de *SPARQL Endpoints*. Os dados de entrada usam um amplo conjunto de vocabulários RDF. Funções de transformação são utilizadas para a tradução destes dados para o vocabulário alvo. As fontes de dados usam diferentes identificadores para nomear a mesma entidade. O *framework* utiliza *profiles XML* e suporta vários tipos de comparadores e transformações. A saída pode ser gerada em vários formatos: *N-Quads*, *N-Triples* ou *SPARQL Update Stream*.

## 3.7 ODCS - ODCleanStore

O *ODCS* (KNAP et al., 2012) é um *framework* para (1) possibilitar o gerenciamento de dados ligados – limpeza, ligação, transformação, e garantia da segurança dos dados – e (2) prover aos clientes a possibilidade de consumir os dados de forma integrada, o que reduz o custo de desenvolvimento das aplicações Web.

A ideia principal é que aplicações de terceiros possam se cadastrar no *ODCS* para alimentá-lo com dados RDF na forma de quádruplas (sujeito, predicado, objeto, grafo). De acordo com a origem dos dados são aplicadas regras de transformação e de limpeza. Em seguida os dados são armazenados em um *Clean Database*.

Os consumidores de dados podem consultar o *Clean Database*, via aplicações de terceiros, para obter dados sobre um determinado recurso. Como um mesmo recurso pode estar descrito em várias fontes, conflitos poderão surgir quando os dados forem integrados. Para resolver isto, *ODCS* aplica um *algoritmo de fusão de dados* que se baseia em políticas de resolução de conflitos. Estas políticas podem ser customizadas pelo consumidor.

## 3.8 Conclusões

Este capítulo apresentou algumas ferramentas destinadas a lidar com os problemas de como publicar dados relacionais na forma de grafos RDF e como consultar, persistir e manipular estes grafos. Foram destacadas as principais funcionalidades de cada uma destas ferramentas bem como a integração que pode existir entre elas. Pretendemos com esse estudo, obter subsídios para formular contribuições relevantes ao contexto de mapeamentos relacional para RDF.

# 4 RDB TO RDF MAPPING LANGUAGE (R2RML)

Neste capítulo introduzimos a linguagem de mapeamentos R2RML (seção 4.1) e um estudo de caso onde um esquema relacional fonte é mapeado para uma ontologia alvo utilizando mapeamentos R2RML (seção 4.2). Esta solução é utilizada para motivar a aplicação da abordagem proposta neste trabalho (seção 4.3).

## 4.1 Linguagem R2RML

R2RML (DAS et al., 2012) é uma linguagem para criação de mapeamentos customizados de bancos de dados relacionais para *datasets* RDF. Com esses mapeamentos podemos converter dados relacionais no modelo de dados RDF utilizando os vocabulários escolhidos pelo criador do mapeamento.

Existem duas formas de mapeamento de bancos relacionais para RDF: direto e customizado. No mapeamento direto (ARENAS et al., 2012) de um banco de dados, a estrutura do grafo RDF obtido reflete diretamente a estrutura do banco de dados, o vocabulário RDF reflete diretamente os nomes dos elementos do esquema do banco, e nenhuma estrutura ou vocabulário pode ser alterado. Por outro lado, com R2RML, o autor do mapeamento pode definir visões customizadas sobre os dados relacionais.

Todo mapeamento R2RML é feito para um esquema relacional específico e um vocabulário alvo. Dessa forma, a entrada para um mapeamento R2RML é um banco de dados relacional que está de acordo com este esquema. A saída é um *dataset* RDF que utiliza predicados e tipos definidos no vocabulário alvo. Os próprios mapeamentos R2RML são grafos RDF escritos na sintaxe *Turtle*. O mapeamento é conceitual. Assim, os processadores R2RML são livres para materializar os dados RDF ou para fornecer acesso virtual através de um SPARQL *endpoint* ou de uma *Linked Data interface*.

Um mapeamento R2RML referencia *tabelas lógicas* para recuperar dados da base de entrada. Uma tabela lógica pode ser:

1. Uma tabela relacional,
2. uma visão relacional, ou

3. uma consulta SQL válida (chamada de “visão R2RML”).

Cada tabela lógica é mapeada para RDF através de um *triples map*, que é uma regra para mapear cada tupla da tabela lógica em um conjunto de triplas RDF. A regra tem duas partes principais:

1. Um *subject map* que cria o sujeito de todas as triplas RDF geradas a partir de uma tupla da tabela lógica. Normalmente os sujeitos são URIs geradas a partir das colunas que compõem a chave primária da tabela.
2. Um conjunto de *predicate-object maps* que são compostas de *predicate maps* e *object maps.*

Triplas são produzidas pela combinação do *subject map* com um *predicate map* e um *object map* aplicados a cada tupla da tabela lógica. Em seguida, ilustramos estes conceitos com um exemplo prático. A Tabela 4.1 mostra os prefixos e seus respectivos *namespaces* utilizados no exemplo e ao longo deste trabalho.

**Tabela 4.1 - Prefixos e namespaces utilizados**

| ***Prefixo*** | ***IRI*** |
|---|---|
| rr: | http://www.w3.org/ns/r2rml# |
| rdf: | http://www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs: | http://www.w3.org/2000/01/rdf-schema# |
| xsd: | http://www.w3.org/2001/XMLSchema# |
| ex: | http://www.exemplo.com/ns# |

Considere o banco de dados dos profissionais da Universidade Federal do Ceará contendo duas tabelas: Empregado e Departamento cada uma com uma tupla como mostra a Figura 4.1.

As triplas RDF que são produzidas a partir dos dados de entrada apresentados na Figura 4.1 são mostradas na Tabela 4.2. Note quatro particularidades nestas triplas:

1. As URIs de *ex:Empregado* e *ex:Departamento* (como, por exemplo, aquelas mostradas nas Linhas 1 e 4) são formadas usando o *namespace* da classe concatenado com o valor do atributo que é chave primária.
2. Foi utilizado neste exemplo um vocabulário próprio “ex”.

3. A propriedade de dados *ex: quantidadeEmpregados* (Linha 7) representa o total de empregados que trabalham no departamento. Embora o valor desta propriedade não esteja armazenado diretamente no banco de dados, ele pode ser calculado.
4. A propriedade de objeto *ex:departamento* (Linha 3) relaciona um empregado ao seu departamento, utilizando os identificadores (URIs) das duas entidades.

**Empregados**

| **numEmpregado**<br>INT PRIMARY KEY | **eNome**<br>VARCHAR(100) | **cargo**<br>VARCHAR(20) | **numDepartamento**<br>INT REFS DEPT (NUM_DEPT) |
|---|---|---|---|
| 20 | “VANIA VIDAL” | “PROFESSORA” | 10 |

**Departamentos**

| **numEmpregado**<br>INT PRIMARY KEY | **eNome**<br>VARCHAR(100) | **cargo**<br>VARCHAR(20) |
|---|---|---|
| 20 | “VANIA VIDAL” | “PROFESSORA” |

**Figura 4.1 - BDR de Entrada**

**Tabela 4.2 - Triplas RDF exportadas do BDR**

```
1 <http://www.exemplo.com/empregado/20> rdf:type ex:Empregado .
2 <http://www.exemplo.com/empregado/20> ex:nome "VANIA VIDAL" .
3 <http://www.exemplo.com/empregado/20>
        ex:departamento <http://www.exemplo.com/departamento/10> .
4 <http://www.exemplo.com/departamento/10> rdf:type ex:Departmento .
5 <http://www.exemplo.com/departamento/10> ex:nome "COMPUTACAO" .
6 <http://www.exemplo.com/departamento/10> ex:local "FORTALEZA" .
7 <http://www.exemplo.com/departamento/10> ex:quantidadeEmpregados 1 .
```

A Tabela 4.3 mostra o mapeamento R2RML criado para produzir triplas a partir de uma única tabela (Empregados) e a Tabela 4.4 mostra as triplas produzidas por este mapeamento.

**Tabela 4.3 - Mapeamento R2RML que produz as triplas da relação Empregados**

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://www.exemplo.com/ns#>.

<#TriplesMap1>
    rr:logicalTable [ rr:tableName "Empregados" ];
    rr:subjectMap [
        rr:template "http://www.exemplo.com/empregado/{numEmpregado}";
        rr:class ex:Empregado;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:nome;
        rr:objectMap [ rr:column "eNome" ];
    ].
```

**Tabela 4.4 - Triplas geradas pelo mapeamento da** Ошибка! Источник ссылки не найден.

```
<http://www.exemplo.com/empregado/20> rdf:type ex:Empregado.
<http://www.exemplo.com/empregado/20> ex:nome "VANIA VIDAL".
```

Dando prosseguimento, precisamos mapear a tabela Departamentos. Neste caso, ao invés de utilizarmos o nome da tabela diretamente no mapeamento, criamos uma visão R2RML na forma de uma consulta SQL. Esta visão é necessária para calcularmos a quantidade de empregados de cada departamento. Ela é apresentada na Tabela 4.5. Uma alternativa seria criar uma visão diretamente no banco de dados e referenciá-la como uma tabela lógica.

**Tabela 4.5 - Visão R2RML de Departamentos para a calcular a quantidade de empregados**

```
<#DepartamentoTableView> rr:sqlQuery """
SELECT numDepartamento,
       dNome,
       cidade,
      (SELECT COUNT(*) FROM Empregados e
       WHERE e.numDepartamento = d.numDepartamento) AS qtdEmpregados
FROM Departamentos d;
""".
```

O uso de visões R2RML é indicado somente quando o criador do mapeamento não tiver permissão para criar visões no banco de dados, pois a utilização delas exige que os processadores R2RML avaliem consultas SQL e tal problema já está resolvido nos sistemas gerenciadores de bancos de dados.

A Tabela 4.6 mostra o mapeamento R2RML de Departamentos utilizando a visão *DepartamentoTableView* como tabela lógica, e a Tabela 4.7 apresenta as triplas geradas por este mapeamento.

**Tabela 4.6 - Mapeamento R2RML de Departamentos**

```
<#TriplesMap2>
    rr:logicalTable <#DepartamentoTableView>;
    rr:subjectMap [
        rr:template "http://www.exemplo.com/departamento/{numDepartamento}";
        rr:class ex:Departmento;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:nome;
        rr:objectMap [ rr:column "dNome" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:local;
        rr:objectMap [ rr:column "cidade" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:quantidadeEmpregados;
        rr:objectMap [ rr:column "qtdEmpregados" ];
    ].
```

**Tabela 4.7 - Triplas geradas pelo mapeamento da** Ошибка! Источник ссылки не найден.

```
<http://www.exemplo.com/departamento/10> rdf:type ex:Departmento.
<http://www.exemplo.com/departamento/10> ex:nome "COMPUTACAO".
<http://www.exemplo.com/departamento/10> ex:local "FORTALEZA".
```

```
<http://www.exemplo.com/departamento/10> ex:quantidadeEmpregados 1.
```

Até este ponto foram criados os mapeamentos para gerar as triplas mostradas nas Linhas 1, 2, 4, 5, 6 e 7 da Tabela 4.2. Para concluir nosso exemplo, precisamos gerar a tripla da Linha 3 que se refere à propriedade de objeto *ex:departamento*. O sujeito desta tripla é originado pelo *triples map <#TriplesMap1>* (Tabela 4.3) e o objeto é gerado pelo *triples map <#TriplesMap2>* (Tabela 4.6). Assim, a geração da tripla para *ex:departamento* é feita adicionando um novo *predicate object map* dentro *<#TriplesMap1>*. Este *predicate object map* referencia *<#TriplesMap2>* como *parent triples map* conforme mostrado em destaque na Tabela 4.8. Esta alteração no mapeamento cria um *Join* entre a tabela *Empregados* e a visão R2RML *DepartamentoTableView* sobre as colunas *numDepartamento* de cada relação. Os objetos das triplas são gerados a partir do *subject map* do *parent triples map*, o que resulta na tripla mostrada na Tabela 4.9.

**Tabela 4.8 - Alteração no mapeamento *<#TriplesMap1>* da Tabela 4.3**

```
<#TriplesMap1>
    rr:logicalTable [ rr:tableName "Empregados" ];
    rr:subjectMap [
        rr:template "http://www.exemplo.com/empregado/{numEmpregado}";
        rr:class ex:Empregado;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:nome;
        rr:objectMap [ rr:column "eNome" ];
    ].
    rr:predicateObjectMap [
        rr:predicate ex:departamento;
        rr:objectMap [
            rr:parentTriplesMap <#TriplesMap2>;
            rr:joinCondition [
                rr:child "numDepartamento";
                rr:parent "numDepartamento";
            ];
        ];
    ].
```

**Tabela 4.9 - Tripla gerada pelo mapeamento da Tabela 4.8**

```
<http://www.exemplo.com/empregado/20>
   ex:departamento <http://www.exemplo.com/departamento/10> .
```

## 4.2 Estudo de Caso

Nesta seção apresentamos o exemplo que será utilizado ao longo deste trabalho.

A Figura 4.1 mostra o esquema relacional *ISWC_REL*. Cada tabela possui uma chave primária sequencial cujo nome termina com a literal 'ID'. *Persons* e *Papers* são as relações principais do esquema. Elas representam autores e seus artigos, respectivamente. O atributo *conferences* de *Papers* é uma chave estrangeira para a relação *Conferences* e indica a conferência onde o artigo foi publicado. O relacionamento N:M entre *Persons* e *Papers* é realizado pela relação intermediária *Rel_Person_Paper*.

A relação *Topics* guarda os tópicos abordados nos artigos. *Rel_Paper_Topic* faz o relacionamento entre *Topics* e *Papers* e o atributo *parentID* de *Topics* cria um auto-relacionamento nesta relação.

A relação *Organizations* se refere às organizações onde os autores podem ter algum vínculo. Este vínculo é implementado pela relação intermediária *Rel_Person_Organization*. Os rótulos nos arcos de relacionamento, como *Fk_Papers*, são os nomes das chaves estrangeiras.

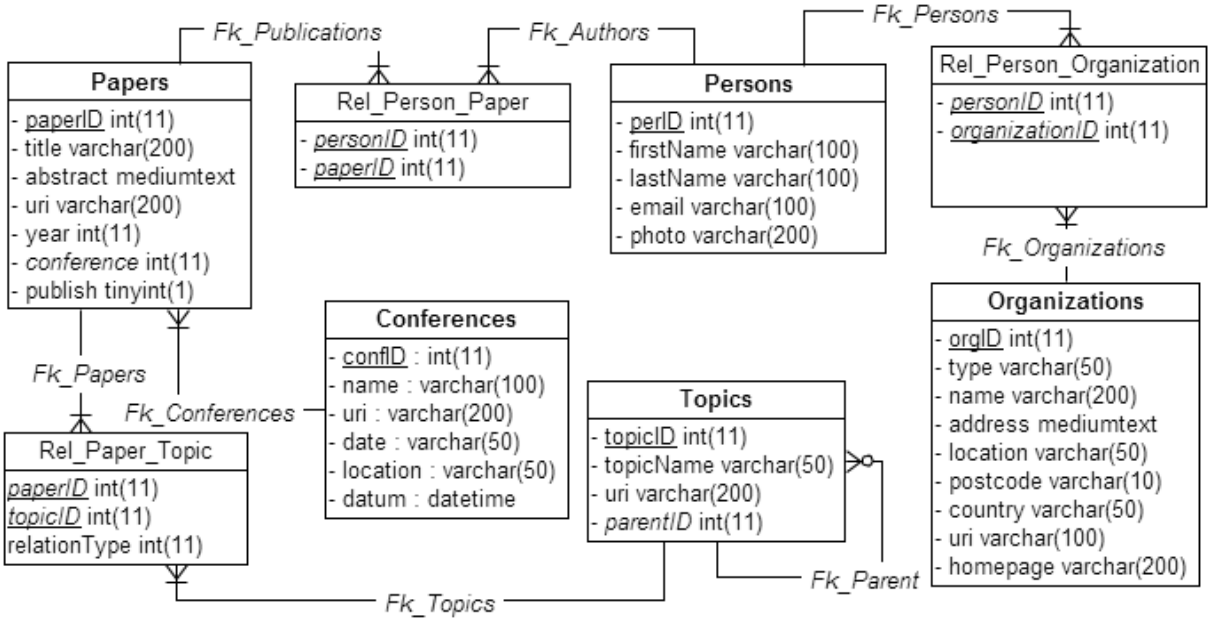


**Figura 4.2 - Esquema do Banco de Dados ISWC_REL**

A Figura 4.2 descreve a ontologia *CONF_OWL*. Esta ontologia reusa termos de quatro vocabulários conhecidos: *FOAF* (*Friend of a Friend*), *SKOS* (*Knowledge Organization System*), *VCARD* e *DC* (*Dublin Core*). Foi utilizado o prefixo *"conf"* para os novos termos definidos em CONF_OWL. As classes que representam os conceitos principais da ontologia são *foaf:Person* e *foaf:Document*. Os arcos entre as classes são rotulados com o nome da propriedade de objeto que as relaciona e a direção do arco parte do domínio para a imagem desta propriedade. Por exemplo, o arco entre *foaf:Document* e *foaf:Person* está rotulado com

a propriedade *dc:creator* que relaciona documentos com seus criadores. O domínio de *dc:creator* é *foaf:Document* e sua imagem é *foaf:Person*.

As classes *conf:Conference*, *skos:Concept*, *conf:Organization* e *conf:PostalAddress* representam, respectivamente, as *conferências* onde os documentos são publicados, os *assuntos* que são citados em documentos ou são temas de pesquisa de pessoas, as *organizações* onde pessoas atuam, e os *endereços* dessas organizações.

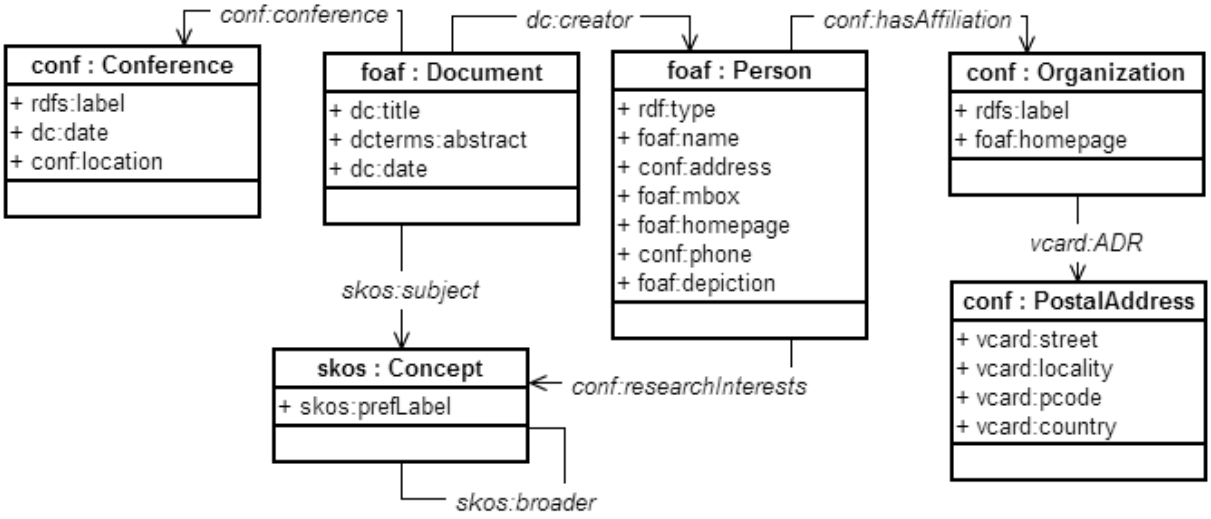


**Figura 4.3 - Ontologia de Domínio CONF_OWL**

Utilizamos os conceitos *foaf:Person*, *conf:Organization*, *conf:hasAffiliation* e *conf:researchInterests* da ontologia de domínio como exemplo para criação de triplas a partir das tuplas de *ISWC_Rel*, via R2RML.

O mapeamento apresentado na Tabela 4.10 referencia a relação *Persons* (linha 2). Este mapeamento especifica que cada tupla *t* em *Persons* produzirá três triplas RDF:

**(i)** <*http://example.com/person/t.perID*> *rdf:type foaf:Person* .
(Generado do *subject map* nas linhas 3-6)

**(ii)** <*http://example.com/person/t.perID*> *foaf:name "t.firstname t.lastname"* .
(Generado do *predicate object map* nas linhas 7-10)

**(iii)** <*http://example.com/person/t.perID*> *foaf:mbox "t.email"* .
(Generado do *predicate object map* nas linhas 11-14)

A tripla (i) define que cada indivíduo representado com este formato de URI será do tipo *foaf:Person*. A tripla (ii) significa que o valor literal da propriedade de dados *foaf:name* será a concatenação dos valores dos atributos *firstName* e *lastName* da relação *Persons*. Por último, a tripla (iii) atribui à propriedade de dados *foaf:mbox* o valor literal do atributo *email* da relação *Persons*.

**Tabela 4.10 - Mapeamento R2RML da classe *foaf:Person***

```
1  <#PersonTriplesMap>
2    rr:logicalTable [ rr:tableName "Persons" ];
3    rr:subjectMap [
4      rr:template "http://example.com/person/{perID}";
5      rr:class foaf:Person;
6    ];
7    rr:predicateObjectMap [
8      rr:predicate foaf:name;
9      rr:objectMap [rr:template "{firstName} {lastName}"];
10   ];
11   rr:predicateObjectMap [
12     rr:predicate foaf:mbox;
13     rr:objectMap [ rr:column "email" ];
14   ].
```

O mapeamento na Tabela 4.11 referencia a relação *Organizations* (linha 2). Este mapeamento especifica que cada tupla *t* em *Organizations* produzirá três triplas RDF:

**(i)** <*http://example.com/org/t.orgID> rdf:type conf:Organization* .
(Generado do *subject map* nas linhas 3-6)

**(ii)** <*http://example.com/org/t.orgID>rdfs:label "t.name"* .
(Generado do *predicate object map* nas linhas 7-10)

**(iii)** <*http://example.com/org/t.orgID> foaf:homepage "t.homepage"* .
(Generado do *predicate object map* nas linhas 11-14)

A tripla (i) define que cada indivíduo representado com este formato de URI será do tipo *conf:Organization*. A tripla (ii) significa que o valor literal da propriedade de dados *rdfs:label* será o valor do atributo *name* da relação *Organizations*. Por último, a tripla (iii) atribui à propriedade de dados *foaf:homepage* o valor literal do atributo *homepage* da relação *Organizations*.

**Tabela 4.11 - Mapeamento R2RML da classe *conf:Organization***

```
1  <#OrgTriplesMap>
2    rr:logicalTable [ rr:tableName "Organizations" ];
3      rr:subjectMap [
4      rr:template "http://example.com/org/{orgID}";
5      rr:class conf:Organization;
6    ];
7    rr:predicateObjectMap [
8      rr:predicate rdfs:label;
9      rr:objectMap [ rr: column "name" ];
10   ];
11   rr:predicateObjectMap [
12     rr:predicate foaf:homepage;
13     rr:objectMap [ rr:column "homepage" ];
14   ].
```

O mapeamento na Tabela 4.12 referencia a relação *Rel_Person_Organization* (linha 2) para ligar indivíduos da classe *foaf:Person* com indivíduos da classe *conf:Organization* via propriedade de objeto *conf:hasAffiliation*. O atributo *perID* é usado para compor a mesma URI de *PersonTriplesMap* (linha 4 da Tabela 4.12 é igual à linha 4 da Tabela 4.10). Este mapeamento realiza um *join* entre *Rel_Person_Organization* e

*Organizations* usando os atributos *organizationID* e *orgID* (linhas 11-14). O mapeamento especifica que, para cada par <*t, t'*>, onde *t* é uma tupla em *Rel_Person_Organization*, *t'* é uma tupla em *Organizations* e *t.organizationID = t'.orgID*, deve ser gerada uma tripla:

**(i)** <*http://example.com/person/t.personID*>
*conf:hasAffiliation* <*http://example.com/org/t'.orgID*> .

**Tabela 4.12 - Mapeamento R2RML da propriedade de objeto *conf:hasAffiliation***

```
1  <#HasAffiliationTriplesMap>
2    rr:logicalTable [ rr:tableName "Rel_Person_Organization" ];
3    rr:subjectMap [
4      rr:template "http://example.com/person/{personID}";
5      rr:class foaf:Person;
6    ];
7    rr:predicateObjectMap [
8      rr:predicate conf:hasAffiliation;
9      rr:objectMap [
10       rr:parentTriplesMap <#OrgTriplesMap>;
11       rr:joinCondition [
12         rr:child "organizationID";
13         rr:parent "orgID";
14       ];
15     ];
16   ].
```

O mapeamento R2RML da Tabela 4.13 referencia uma visão R2RML, *RV*, definida nas linhas 3-6. O *predicate object map* nas linhas 12-17 mapeia tuplas de *RV* para triplas da propriedade de objeto *conf:researchInterests*. Note que este mapeamento não pode ser definido diretamente sobre uma relação, pois um termo *rr:joinCondition* dentro de um *rr:objectMap* pode ter somente uma chave estrangeira referenciada. Neste caso, é preciso percorrer um caminho com três chaves estrangeiras para relacionar diretamente uma pessoa com seus tópicos de interesse. Portanto, para criar este tipo de mapeamento, é **obrigatório** definir uma visão com os joins que percorrem o caminho.

O mapeamento especifica que, para cada par <*t, t'*>, onde *t* é uma tupla em *RV*, *t'* é uma tupla em *Rel_Paper_Topic* e *t.idTopic = t'.topicID*, uma tripla deve ser gerada:

**(i)** <*http://example.com/person/t.idPerson*>
*conf:researchInterests* <*http://example.com/org/t'.topicID*> .

**Tabela 4.13 - Mapeamento R2RML da propriedade de objeto *conf:researchInterests***

```
1  <#ResearchInterestsTriplesMap>
2    rr:logicalTable [
3      rr:sqlQuery """
4      SELECT pe.perID as idPerson, rpt.topicID as idTopic
5      FROM Persons_Rel as pe, Rel_Person_Paper as rpp,
           Papers_Rel as pa, Rel_Paper_Topic as rpt
6      WHERE pe.perID = rpp.personID and rpp.paperID = pa.paperID
```

```
            and pa.paperID = rpt.paperID """;
  ];
  rr:subjectMap [
    rr:template "http://example.com/person/{idPerson}";
    rr:class foaf:Person;
  ];
  rr:predicateObjectMap [
    rr:predicate conf:researchInterests;
    rr:objectMap [
      rr:template "http://example.com/topic/{idTopic}"
    ];
  ].
```

É fácil notar, pelos exemplos apresentados, que a criação de mapeamentos R2RML é uma tarefa árdua. Percebemos a necessidade de criação de um método que auxilie a geração destes mapeamentos e das visões necessárias. Propomos então o uso de assertivas de correspondência para especificar mapeamentos de esquemas relacionais para vocabulários RDF. Estas assertivas serão detalhadas no capítulo 5. Em seguida, no capítulo 6, descrevemos uma abordagem para geração automática de mapeamentos R2RML a partir de um conjunto de assertivas de correspondência.

## 4.3 Conclusões

Este capítulo apresentou a linguagem de mapeamentos R2RML e o estudo de caso que será utilizado neste trabalho. Mostramos como mapear alguns conceitos do estudo de caso usando R2RML. Esta demonstração apresentou a complexidade da linguagem e a obrigatoriedade de criarmos visões em determinados tipos de mapeamento. Isto motiva a adoção da nossa abordagem, que se baseia na criação de visões relacionais para simplificar os mapeamentos R2RML gerados.

# 5 USANDO ASSERTIVAS DE CORRESPONDÊNCIA PARA DEFINIR VISÕES RDF DE DADOS RELACIONAIS

Neste capítulo, apresentamos o formalismo para especificação de visões RDF de dados relacionais. Este formalismo é uma forma conveniente para, manualmente, especificarmos os mapeamentos entre esquemas relacionais e vocabulários RDF. As definições aqui descritas são base para a solução do problema de criação de mapeamentos R2RML proposta neste trabalho. A seção 5.1 define conceitos básicos e a notação adotada. A seção 5.2 traz a definição de assertivas de correspondência. A seção 5.3 mostra como as regras de transformação são geradas a partir das assertivas de correspondência. A seção 5.4 mostra as assertivas de correspondência do estudo de caso, suas regras de transformação e as triplas resultantes. Finalmente, na seção 5.5 apresentamos nossas conclusões.

## 5.1 Conceitos Básicos e Notação Adotada

Ao longo deste trabalho usamos a notação $R[A_1,...,A_n]$ para representar um esquema de relação, e adotamos as restrições relacionais: atributos obrigatórios (ou *not null*), chaves, chaves primárias e chaves estrangeiras. Em particular, usamos $F(R{:}L,S{:}K)$ para denotar uma chave estrangeira $F$, onde $L$ e $K$ são as listas de atributos das relações $R$ e $S$, respectivamente. $L$ e $K$ possuem o mesmo tamanho. Dessa forma, $F$ relaciona $R$ e $S$.

Um esquema relacional é um par $\boldsymbol{S}=(\boldsymbol{R},\boldsymbol{\Omega})$, onde $\boldsymbol{R}$ é um conjunto de esquemas de relação e $\boldsymbol{\Omega}$ é um conjunto de restrições relacionais tais que:

(i) $\boldsymbol{\Omega}$ possui uma restrição de chave primária para cada esquema de relação em $\boldsymbol{R}$;

(ii) $\boldsymbol{\Omega}$ tem uma restrição de atributo obrigatório para cada atributo que é parte de uma chave única ou de uma chave primária;

(iii) Se $\boldsymbol{\Omega}$ tem uma chave estrangeira na forma $F(R{:}L,S{:}K)$, então $\boldsymbol{\Omega}$ também tem uma restrição indicando que $K$ é a chave primária de $S$.

O *vocabulário* de $\boldsymbol{S}$ é um conjunto de nomes de relação, nomes de atributos, etc, usados em $\boldsymbol{S}$. Dado um esquema de relação $R[A_1,....,A_n]$ e uma tupla $t$ de $R$, usamos $t.A_k$ para

indicar a projeção de $t$ sobre $A_k$. Usamos seleções sobre esquemas de relação como de costume.

Seja $\boldsymbol{S}=(\boldsymbol{R},\boldsymbol{\Omega})$ um esquema relacional e sejam $R$ e $T$ esquemas de relação de $\boldsymbol{S}$. Uma lista $\varphi=[F_1,...,F_{n-1}]$ de nomes de chaves estrangeiras de $\boldsymbol{S}$ é um *caminho* de $R$ para $T$ se, e somente se, existe uma lista $R_1,...,R_n$ de esquemas de relação de $\boldsymbol{S}$ tal que $R_1=R$, $R_n=T$ e $F_i$ relaciona $R_i$ e $R_{i+1}$. Dizemos que as tuplas de $R$ referenciam tuplas de $T$ através de $\varphi$. Um caminho $\varphi$ é um *caminho associativo* se, e somente se, $\varphi=[F_1,F_2]$, onde as chaves estrangeiras são das formas $F_1(R_2{:}L_2,R_1{:}K_1)$ e $F_2(R_2{:}M_2,R_3{:}K_3)$. Por exemplo, considere o esquema relacional *ISWC_REL* da Figura 4.1. A lista de nomes de chaves [*Fk_Publications*,*Fk_Authors*] é um caminho associativo de *Papers* para *Persons*, mas [*Fk_Publications*,*Fk_Persons*] não é nem mesmo um caminho.

Relembramos nesta seção alguns conceitos básicos relacionados a ontologias. Um vocabulário $\boldsymbol{V}$ é um conjunto de *classes*, *propriedades de objeto* e *propriedades de dados*. Uma *ontologia* é um par $\boldsymbol{O}=(\boldsymbol{V}, \Sigma)$ tal que $\boldsymbol{V}$ é um vocabulário e $\Sigma$ é um conjunto finito de fórmulas em $\boldsymbol{V}$, as *restrições* de $\boldsymbol{O}$. Dentre as restrições, consideramos aquelas que definem o *domínio* e a *imagem* de uma propriedade, além das *restrições de cardinalidade*, definidas de forma usual.

## 5.2 Definição de Assertivas de Correspondência

Nesta seção introduzimos a noção de *assertiva de correspondência* (VIDAL et al., 2005), deixando os exemplos para a Seção 4.5. Seja $\boldsymbol{S} = (\boldsymbol{R},\boldsymbol{\Omega})$ um esquema relacional e $\boldsymbol{O} = (\boldsymbol{V},\Sigma)$ uma ontologia. Assuma que $\Sigma$ possui restrições definindo o domínio e a imagem de cada propriedade.

**Definição 5.1:** Uma *assertiva de correspondência de classe (ACC)* é uma declaração em um dos seguintes formatos:

(i) $\Psi{:}\ C \equiv R[A_1,...,A_n]$

(ii) $\Psi{:}\ C \equiv R[A_1,...,A_n]\sigma$

onde $\Psi$ é o nome da ACC, $C$ é uma classe do vocabulário $\boldsymbol{V}$, $R$ é o nome de uma relação do esquema $\boldsymbol{S}$, $A_1,...,A_n$ são os atributos que compõem a chave primária de $R$, e $\sigma$ é um

filtro de seleção aplicado sobre *R*. Dessa forma, dizemos que *Ψ cria uma associação* da classe *C* com a relação *R*.

**Definição 5.2:** Uma *assertiva de correspondência de objeto (ACO)* é uma declaração em um dos seguintes formatos:

(i) $\Psi: P \equiv R$

(ii) $\Psi: P \equiv R / \varphi$

onde $\Psi$ é o nome da ACO, *P* é uma propriedade de objeto do vocabulário $\boldsymbol{V}$, *R* é o nome de uma relação do esquema $\boldsymbol{S}$ e $\varphi$ é um caminho a partir de *R*.

**Definição 5.3:** Uma *assertiva de correspondência de dados (ACD)* é uma declaração em um dos seguintes formatos:

(i) $\Psi: P \equiv R / A$

(ii) $\Psi: P \equiv R / \{A_1,...,A_n\}$

(iii) $\Psi: P \equiv R / \varphi / B$

(iv) $\Psi: P \equiv R / \varphi / \{B_1,...,B_n\}$

onde $\Psi$ é o nome da ACD, *P* é uma propriedade de dados do vocabulário $\boldsymbol{V}$, *R* é o nome de uma relação do esquema $\boldsymbol{S}$, *A* é um atributo de *R*, $A_1,...,A_n$ são atributos de *R*, $\varphi$ é um caminho de *R* para uma relação T do esquema $\boldsymbol{S}$, *B* é um atributo de *T*, e $B_1,...,B_n$ são atributos de *T*.

**Definição 5.4:** Um *mapeamento* entre $\boldsymbol{V}$ e $\boldsymbol{S}$ é um conjunto $\boldsymbol{A}$ de assertivas de correspondência tal que:

(i) Se $\boldsymbol{A}$ tem uma ACO na forma $P \equiv R$, então $\boldsymbol{A}$ deve ter também uma ACC que mapeia o domínio de *P* com *R*, e uma ACC que mapeia a imagem de *P* também com *R*.

(ii) Se $\boldsymbol{A}$ tem uma ACO na forma $P \equiv R / \varphi$, onde $\varphi$ é um caminho de *R* para *T*, então $\boldsymbol{A}$ deve ter uma ACC que mapeia o domínio de *P* com *R* e uma ACC que mapeia a imagem de *P* com *T*.

(iii) Se $\boldsymbol{A}$ tem uma ACD que mapeia uma propriedade de dados *P* do vocabulário $\boldsymbol{V}$ com uma relação *R* do esquema $\boldsymbol{S}$, então $\boldsymbol{A}$ deve ter uma ACC que mapeia o domínio de *P* com *R*.

**Definição 5.5:** Seja $\boldsymbol{M}$ um *mapeamento* entre $\boldsymbol{S}$ e $\boldsymbol{O}$. O esquema da visão RDF induzida por $\boldsymbol{M}$ é o vocabulário $\boldsymbol{V_M}$ consistindo de todos os nomes de classes e propriedades que ocorrem do lado esquerdo de uma Assertiva de Correspondência em $\boldsymbol{M}$. Podemos dizer que $\boldsymbol{M}$ é o conjunto de assertivas de correspondência de $\boldsymbol{V_M}$.

## 5.3 Regras de Transformação induzidas pelas Assertivas de Correspondência

Nesta seção introduzimos a noção de regra de transformação e mostramos como interpretar assertivas de correspondência na forma de regras de transformação. Sejam $\boldsymbol{O}=(\boldsymbol{V},\Sigma)$ uma ontologia e $\boldsymbol{S}=(\boldsymbol{R},\boldsymbol{\Omega})$ um esquema relacional, com vocabulário $\boldsymbol{U}$. Sejam $\boldsymbol{X}$ um conjunto de *variáveis escalares* e $\boldsymbol{T}$ um conjunto de *variáveis de tupla*, disjuntos entre eles e de $\boldsymbol{V}$ e $\boldsymbol{U}$.

Um *literal* é uma *expressão* na forma $R(t)$, onde $R$ é um nome de uma relação em $\boldsymbol{U}$ e $t$ é uma variável de tupla em $\boldsymbol{T}$, ou um *predicado embutido* de uma das formas mostradas na Tabela 5.1. Um *corpo de regra* $B$ é uma lista de literais. Quando necessário, usamos *"*$B[x_1,\ldots,x_k]$*"* para dizer que a tupla ou as variáveis escalares $x_1,\ldots,x_k$ ocorrem em $B$. Também usamos *"*$R(t)$, $B[t,x_1,\ldots,x_k]$*"* para dizer que o corpo de regra tem um literal na forma $R(t)$.

Uma *regra de transformação*, ou simplesmente uma *regra*, é uma expressão em uma das seguintes formas:

- $C(x) \leftarrow B[x]$, onde $C$ é uma classe em $\boldsymbol{V}$ e $B[x]$ é um corpo de regra.
- $P(x,y) \leftarrow B[x,y]$, onde $P$ é uma propriedade e $B[x,y]$ é um corpo de regra.

Seja $\boldsymbol{A}$ um conjunto de assertivas de correspondência que definem um mapeamento entre $\boldsymbol{V}$ e $\boldsymbol{S}$, ou seja, $\boldsymbol{A}$ satisfaz as condições declaradas na Definição 2.4. Assuma que toda classe $C$ em $\boldsymbol{V}$ está associada a um prefixo de um *namespace.*

A Tabela 5.2 mostra as regras de transformação *induzidas* das assertivas de correspondência de $\boldsymbol{A}$. Por exemplo, a regra de mapeamento da Linha 5 (coluna à direita) indica que, para cada tupla $t$ de $R$ tal que $t.A$ não é nulo, é possível:

- Gerar a URI $s$ do domínio $D$ de $P$ correspondente a $t$, usando a assertiva de correspondência de classe $\Psi_D$: $D(s) \leftarrow R(t)$, $B_D[t,s]$, onde:
  - $B_D[t,s]$ significa *"*$TemURI[\Psi](t,s)$*"*, se a ACC para $D$ for a da Linha 1 da Tabela 5.2; ou
  - $B_D[t,s]$ significa *"*$TemURI[\Psi](t,s)$, $\sigma(t)$*"*, se a ACC para $D$ for a da Linha 2 da Tabela 5.2;
- Traduzir o valor de $A$ na tupla $t$, gerando o literal $v$;
- Associar $v$ com o valor da propriedade $P$ de $s$.

**Tabela 5.1 - Predicados embutidos**

| Predicado embutido | Definição intuitiva |
|---|---|
| *naoNulo(v)* | *naoNulo(v)* é verdadeiro sse o valor de *v* não é nulo |
| *RDFLiteral(u, A, R, v)* | Dado um valor *u*, um atributo *A* de *R*, uma relação *R*, e um literal *v*, *RDFLiteral(u, A, R, v)* é válido sse *v* é a representação literal de *u*, considerando o tipo de *A* em *R* |
| *TemTuplasReferenciadas[φ](t,u)* onde $\varphi$ é um caminho de *R* para *T* | Dada uma tupla *t* de *R* e uma tupla *u* de *T*, *TemTuplasReferenciadas[φ](t,u)* é verdadeiro sse *u* é referenciado por *t* através do caminho $\varphi$ |
| *TemURI[Ψ](t,s)* onde $\Psi$ é uma ACC para a classe *C* de ***V***, usando os atributos $A_1,...,A_n$ de *R* | Dada uma tupla *t* de *R*, *TemURI[Ψ](t,s)* é verdadeiro sse *s* é a URI obtida pela concatenação do prefixo para o *namespace* de *C* com os valores de $t.A_1,...,t.A_n$ |
| *concat([$v_1$,...,$v_n$],v)* | Dada a lista [$v_1$,...,$v_n$] de valores *string*, concat([$v_1$,...,$v_n$], *v* é válido sse *v* é a string obtida concatenando $v_1$,...,$v_n$ |

**Tabela 5.2 - Regras de Mapeamento**

| | Assertiva de Correspondência | Regra |
|---|---|---|
| 1 | $\Psi$: $C \equiv R[A_1,...,A_n]$ | $C(s) \leftarrow R(t), TemURI[\Psi](t,s)$ |
| 2 | $\Psi$: $C \equiv R[A_1,...,A_n]\sigma$ | $C(s) \leftarrow R(t), TemURI[\Psi](t,s), \sigma(t)$ |
| 3 | $\Psi$: $O \equiv R$<br>onde:<br>- ***A*** tem uma ACC $\Psi_D$ que relaciona o domínio *D* de *O* com *R* e $\Psi_D$ tem uma regra $D(s) \leftarrow R(t), B_D[t,s]$<br>- ***A*** tem uma ACC $\Psi_N$ que relaciona a imagem *N* de *O* com *R* e $\Psi_N$ tem uma regra $N(o) \leftarrow R(t), B_N[t,o]$ | $P(s,o) \leftarrow R(t), B_D[t,s], B_N[t,o]$ |
| 4 | $\Psi$: $O \equiv R / \varphi$<br>onde:<br>- $\varphi$ é um caminho de *R* para *T*<br>- ***A*** tem uma ACC $\Psi_D$ que relaciona o domínio *D* de *O* com *R* e $\Psi_D$ tem uma regra $D(s) \leftarrow R(t), B_D[t,s]$<br>- ***A*** tem uma ACC $\Psi_N$ que relaciona a imagem *N* de *O* com *T* e $\Psi_N$ tem uma regra $N(o) \leftarrow T(u), B_N[u,o]$ | $P(s,o) \leftarrow$<br>$R(t), B_D[t,s],$<br>$TemTuplasReferenciadas[\varphi](t,u),$<br>$T(u), B_N[u,o]$ |
| 5 | $\Psi$: $P \equiv R / A$<br>onde:<br>- ***A*** tem uma ACC $\Psi_D$ que relaciona o domínio *D* de *P* com *R* e $\Psi_D$ tem uma regra $D(s) \leftarrow R(t), B_D[t,s]$<br>- *A* é um atributo de *R* | $P(s,v) \leftarrow R(t), B_D[t,s],$<br>$naoNulo(t.A),$<br>$RDFLiteral(t.A, "A", "R", v)$ |
| 6 | $\Psi$: $P \equiv R / \varphi / B$<br>onde:<br>- $\varphi$ é um caminho de *R* para *T*<br>- ***A*** tem uma ACC $\Psi_D$ que relaciona o domínio D de *P* com *R* e $\Psi_D$ tem uma regra $D(s) \leftarrow R(t), B_D[t,s]$<br>- *B* é um atributo de *T* | $P(s,v) \leftarrow R(t), B_D[t,s],$<br>$TemTuplasReferenciadas[\varphi](t,u),$<br>$naoNulo(u.B),$<br>$RDFLiteral(u.B, "B", "T", v)$ |
| 7 | $\Psi$: $P \equiv R / \{A_1,...,A_m\}$<br>onde:<br>- ***A*** tem uma ACC $\Psi_D$ que relaciona o domínio D de *P* com *R* e $\Psi_D$ tem uma regra $D(s) \leftarrow R(t), B_D[t,s]$<br>- $A_1,...,A_m$ são atributos de *R* | $P(s,v) \leftarrow R(t), B_D[t,s],$<br>$naoNulo(t.A_1),...,naoNulo(t.A_m),$<br>$RDFLiteral(t.A_1, "A_1", "R", v_1),$<br>$...,$<br>$RDFLiteral(t.A_m, "A_m", "R", v_m),$<br>$concat([v_1,...,v_m], v)$ |
| 8 | $\Psi$: $P \equiv R / \varphi / \{B_1,...,B_m\}$<br>onde:<br>- $\varphi$ é um caminho de *R* para *T*<br>- ***A*** tem uma ACC $\Psi_D$ que relaciona o domínio *D* de *P* com *R* e $\Psi_D$ tem uma regra $D(s) \leftarrow R(t), B_D[t,s]$<br>- $B_1,...,B_m$ são atributos de *T* | $P(s,v) \leftarrow D(t), B_D[t,s],$<br>$TemTuplasReferenciadas[\varphi](t,u),$<br>$naoNulo(u.B_1),..., naoNulo(u.B_m)$<br>$RDFLiteral(u.B_1, "B_1", "T", v_1),$<br>$...,$<br>$RDFLiteral(u.B_m, "B_m", "T", v_m),$<br>$concat([v_1,..,v_m], v)$ |

## 5.4 Assertivas de Correspondência entre *ISWC_REL* e *CONF_OWL*

A Tabela 5.3 mostra um conjunto de assertivas de correspondência *A* que especifica um mapeamento entre a ontologia de domínio *CONF_OWL* e o esquema relacional *ISWC_REL*, obtidas com o auxílio da ferramenta descrita no Capítulo 7.

As regras de transformação induzidas pelas assertivas $ACC_1$, $ACD_1$, $ACO_2$, $ACC_3$, $ACC_4$ e $ACO_3$ são:

- ***ACC₁*** especifica que cada tupla *t* em Persons produz uma tripla RDF:
  - *<http://example.com/person/t.perID> rdf:type foaf:Person.*

  Assim, o atributo *perID* que é chave primária da relação *Persons* é utilizado para compor a URI dos indivíduos da classe *foaf:Person*.
- ***ACD₁*** especifica que cada tupla *t* em Persons produz uma tripla RDF:
  - *<http://example.com/person/t.perID> foaf:name concat(t.firstname, t.lastname).*

  Desta forma o valor da propriedade *foaf:name* é obtido pela concatenação dos atributos *firstname* e *lastname*.
- ***ACO₂*** especifica que, para cada tupla *t* em *Persons*, para cada tupla *t'* em *Topics* tal que *t'* é referenciado por *t* através do caminho *[Fk_Authors, Fk_Publications, Fk_Papers, Fk_Topics]*, é gerada uma tripla RDF:
  - *<http://example.com/person/t.perID> conf:researchInterests <http://example.com/concept/t'.topicID>.*

  Como não há uma relação intermediária ligando diretamente as relações *Persons* e *Topics* é preciso percorrer um caminho de chaves estrangeiras partindo de *Persons* e terminando em *Topics*.
- ***ACC₃*** e ***ACC₄*** especificam que cada tupla *t* em Organizations produz, respectivamente, as triplas RDF:
  - *<http://example.com/organization/t.orgID> rdf:type conf:Organization*
  - *<http://example.com/organization/t.address/t.location/t.postcode/t.country> rdf:type conf:PostalAddress*

  Neste caso a relação *Organizations* é mapeada em duas classes: *conf:Organization* e *conf:PostalAddress*. A diferença entre elas consiste nos atributos selecionados para compor a URI. Para a classe *conf:Organization* foi escolhido o atributo *orgID* que é chave primária da relação, ao passo que para a classe *conf:PostalAddress*

foram escolhidos os atributos *address*, *location*, *postcode* e *country* que juntos compõem uma chave candidata.

- ***ACO₃*** especifica que cada tupla t em Organizations produz uma tripla RDF:
  - *<http://example.com/organization/t.orgID> vcard:ADR <http://example.com/organization/t.address/t.location/t.postcode/t.country>* .

**Tabela 5.3 - Assertivas de Correspondência entre *ISWC_REL* e *CONF_OWL***

| | |
|---|---|
| $ACC_1$ | *foaf:Person ≡ Persons[perID]* |
| $ACC_2$ | *foaf:Document ≡ Papers[paperID], σ [papers.Year > 2002]* |
| $ACC_3$ | *conf:Organization ≡ Organizations[orgID]* |
| $ACC_4$ | *conf:PostalAddress ≡ Organizations[address, location, postcode, country]* |
| $ACC_5$ | *conf:Conference ≡ Conferences[confID]* |
| $ACC_6$ | *skos:Concept ≡ Topics[topicID]* |
| $ACO_1$ | *conf:hasAffiliation ≡ Persons / [Fk_Persons, Fk_Organizations]* |
| $ACO_2$ | *conf:researchInterests ≡ Persons / [Fk_Authors, Fk_Publications, Fk_Papers, Fk_Topics ]* |
| $ACO_3$ | *vcard:ADR ≡ Organizations* |
| $ACO_4$ | *skos:subject ≡ Papers / [Fk_Papers, Fk_Topics]* |
| $ACO_5$ | *conf:conference ≡ Papers / Fk_Conferences* |
| $ACO_6$ | *skos:broader ≡ Topics / Fk_Parent* |
| $ACD_1$ | *foaf:name ≡ Persons / { firstName, lastName }* |
| $ACD_2$ | *foaf:mbox ≡ Persons / email* |
| $ACD_3$ | *rdfs:label ≡ Organizations / name* |
| $ACD_4$ | *foaf:homepage ≡ Organizations / homepage* |
| $ACD_5$ | *vcard:street ≡ Organizations / address* |
| $ACD_6$ | *vcard:locality ≡ Organizations / location* |
| $ACD_7$ | *vcard:pcode ≡ Organizations / postcode* |
| $ACD_8$ | *vcard:country ≡ Organizations /country* |
| $ACD_9$ | *dc:title ≡ Papers / title* |
| $ACD_{10}$ | *dcterms:abstract ≡ Papers / abstract* |
| $ACD_{11}$ | *dc:date ≡ Papers / year* |
| $ACD_{12}$ | *skos:prefLabel ≡ Topics / topicName* |
| $ACD_{13}$ | *rdfs:label ≡ Conferences / name* |
| $ACD_{14}$ | *dc:date ≡ Conferences / date* |
| $ACD_{15}$ | *conf:location ≡ Conferences / location* |

É importante salientar que nem todos os tipos de assertiva podem ser transformados em mapeamentos R2RML diretamente, ou seja, sem o uso de visões SQL. Por exemplo, a assertiva $ACO_2$ possui um caminho com quatro chaves. Como foi dito na seção 4.3, o R2RML possui uma limitação de só permitir referenciar uma chave estrangeira dentro do termo *rr:joinCondition*. Assim, para assertivas de correspondência com caminhos com mais de uma chave estrangeira, é **obrigatória** a criação de visões SQL.

A Figura 5.1 apresenta a visão RDF *ISWC_RDF* construída a partir do conjunto *A*. Conforme a Definição 5.5, o vocabulário de *ISWC_RDF* consiste de todos os nomes de

classes e propriedades que ocorrem do lado esquerdo de uma assertiva de correspondência em *A*. Portanto, o vocabulário de *ISWC_RDF* é um subconjunto do vocabulário de *CONF_OWL*.

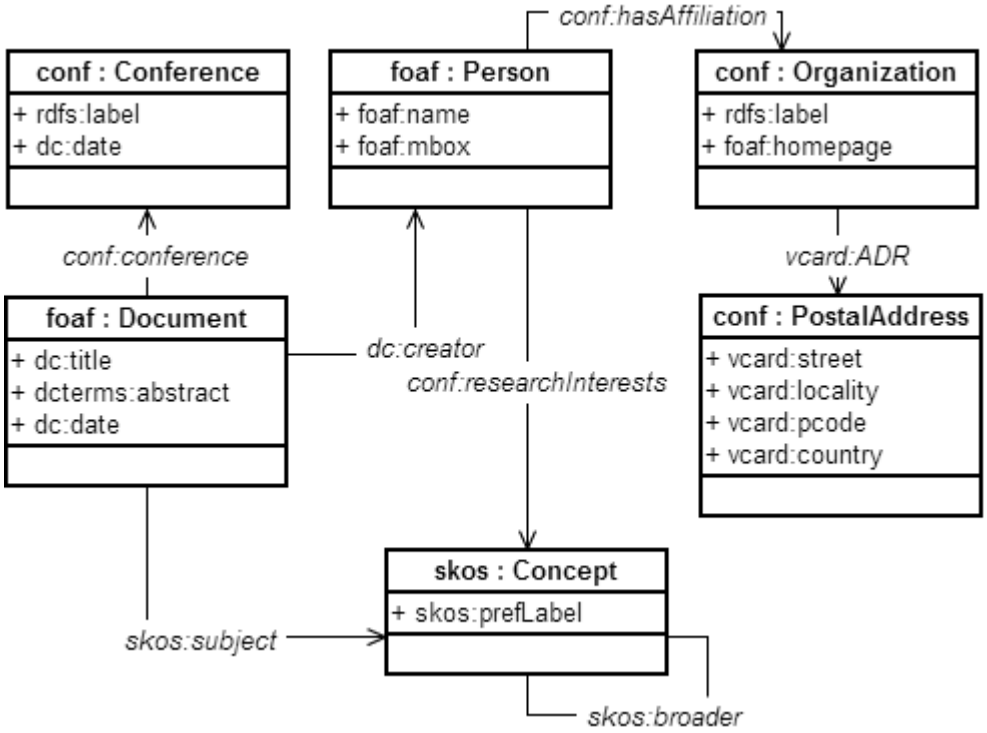


**Figura 5.1 - Visão RDF *ISWC_RDF* gerada a partir das ACs**

A Tabela 5.4 contém as regras de transformação induzidas pelas assertivas de correspondência da Tabela 5.3, onde a regra $RAC_i$ foi gerada a partir da assertiva $AC_i$. Por exemplo, a regra de transformação $RACO_1$, gerada a partir de $ACO_1$, indica que para cada tupla *p* da relação *Persons* que possua um caminho [*Fk_Persons, Fk_Organizations*] relacionando *p* com uma tupla *t* da relação *Organizations*, serão gerados:

- uma URI *s* da classe *foaf:Person* aplicando o predicato embutido *TemURI[$ACC_1$](p,s);*
- uma URI *o* da classe *conf:Organization* aplicando o predicato embutido *TemURI[$ACC_3$](t,o);*
- uma tripla *s conf:hasAffiliation o*;

**Tabela 5.4 - Regras de Transformação induzidas das ACs da Tabela 5.3**

| | |
|---|---|
| $RACC_1$ | *foaf:Person(s) ← Persons(t), TemURI[$ACC_1$](t,s)* |
| $RACC_2$ | *foaf:Document(s) ← Papers(t), TemURI[$ACC_2$](t,s)* |
| $RACC_3$ | *conf:Organization(s) ← Organizations (t), TemURI[$ACC_3$](t,s)* |
| $RACC_4$ | *conf:PostalAddress(s) ← Organizations (t), TemURI[$ACC_4$](t,s)* |
| $RACC_5$ | *conf:Conference(s) ← Conferences(t), TemURI[$ACC_5$](t,s)* |
| $RACC_6$ | *skos:Concept(s) ← Topics(t), TemURI[$ACC_6$](t,s)* |
| $RACO_1$ | *conf:hasAffiliation(s,o) ← Persons(p), TemURI[$ACC_1$](p,s), TemTuplasReferenciadas[Fk_Persons, Fk_Organizations](p,t), Organizations(t),* |

| | |
|---|---|
| | *TemURI[$ACC_3$](t,o)* |
| *$RACO_2$* | *conf:researchInterests(s,o) ← Persons(p), TemURI[$CCA_1$](p,s), TemTuplasReferenciadas[Fk_Authors, Fk_Publications, Fk_Papers, Fk_Topics](p,t), Topics(t), TemURI[$ACC_6$](t,o)* |
| *$RACO_3$* | *vcard:ADR(s,o) ← Organizations(t), TemURI[$ACC_3$](t,s), Organizations(t), TemURI[$ACC_4$](t,o)* |
| *$RACO_4$* | *skos:subject(s,o) ← Papers(p), TemURI[$ACC_2$](p,s), TemTuplasReferenciadas[Fk_Papers, Fk_Topics](p,t), Topics(t), TemURI[$ACC_6$](t,o)* |
| *$RACO_5$* | *conf:conference(s,o) ← Papers(p), TemURI[$ACC_2$](p,s), TemTuplasReferenciadas[Fk_Conferences](p,t), Conferences(t), TemURI[$ACC_5$](t,o)* |
| *$RACO_6$* | *skos:broader(s,o) ← Topics(t), TemURI[$ACC_6$](t,s), TemTuplasReferenciadas[Fk_Parent](t,u), Topics(u), TemURI[$ACC_6$](u, o)* |
| *$RACD_1$* | *foaf:name(s, v) ← Persons(p), TemURI[$ACC_1$](p, s), naoNulo(p.firstName), naoNulo(p.lastName), RDFLiteral(p.firstName, "firstName", "Persons", v1), RDFLiteral(p.lastName, "lastName", "Persons", v2), concat([v1, v2], v)* |
| *$RACD_2$* | *foaf:mbox(s, v) ← Persons(p), TemURI[$ACC_1$](p, s), naoNulo(p.email), RDFLiteral(p.email, "email", "Persons", v)* |
| *$RACD_3$* | *rdfs:label(s, v) ← Organizations(o), TemURI[$ACC_3$](o, s), naoNulo(o.name), RDFLiteral(o.name, "name", "Organizations", v)* |
| *$RACD_4$* | *foaf:homepage(s, v) ← Organizations(o), TemURI[$ACC_3$](o, s), naoNulo(o.homepage), RDFLiteral(o.homepage, "homepage", "Organizations", v)* |
| *$RACD_5$* | *vcard:street(s, v) ← Organizations(o), TemURI[$ACC_4$](o, s), naoNulo(o.address), RDFLiteral(o.address, "address", "Organizations", v)* |
| *$RACD_6$* | *vcard:locality (s, v) ← Organizations(o), TemURI[$ACC_4$](o, s), naoNulo(o.location), RDFLiteral(o.location, "location", "Organizations", v)* |
| *$RACD_7$* | *vcard:pcode (s, v) ← Organizations(o), TemURI[$ACC_4$](o, s), naoNulo(o.postcode), RDFLiteral(o.postcode, "postcode", "Organizations", v)* |
| *$RACD_8$* | *vcard:country (s, v) ← Organizations(o), TemURI[$ACC_4$](o, s), naoNulo(o.country), RDFLiteral(o.country, "country", "Organizations", v)* |
| *$RACD_9$* | *dc:title(s, v) ← Papers(p), TemURI[$ACC_2$](p, s), naoNulo(p.title), RDFLiteral(p.title, "title", "Papers", v)* |
| *$RACD_{10}$* | *dcterms:abstract(s, v) ← Papers(p), TemURI[$ACC_2$](p, s), naoNulo(p.abstract), RDFLiteral(p.abstract, "abstract", "Papers", v)* |
| *$RACD_{11}$* | *dc:date(s, v) ← Papers(p), TemURI[$ACC_2$](p, s), naoNulo(p.year), RDFLiteral(p.year, "year", "Papers", v)* |
| *$RACD_{12}$* | *skos:prefLabel(s, v) ← Topics(t), TemURI[$ACC_6$](t, s), naoNulo(t.topicName), RDFLiteral(t.topicName, "topicName", "Topics", v)* |
| *$RACD_{13}$* | *rdfs:label(s, v) ← Conferences(t), TemURI[$ACC_5$](t, s), naoNulo(t.name), RDFLiteral(t.name, "name", "Conferences", v)* |
| *$RACD_{14}$* | *dc:date(s, v) ← Conferences(t), TemURI[$ACC_5$](t, s), naoNulo(t.date), RDFLiteral(t.date, "date", "Conferences", v)* |
| *$RACD_{15}$* | *conf:location(s, v) ← Conferences(t), TemURI[$ACC_5$](t, s), naoNulo(t.location), RDFLiteral(t.location, "location", "Conferences", v)* |

Suponha o estado atual do banco de dados *ISWC_REL* como descrito na Figura 5.2.

*Persons*

| perID | FirstName | LastName | email |
|---|---|---|---|
| 1 | Yolanda | Gil | gil@isi.edu |
| 2 | Varun | Ratnakar | varunr@isi.edu |
| 3 | Jim | Blythe | blythe@isi.edu |

*Rel_Person_Organization*

| PersonID | OrganizationID |
|---|---|
| 1 | 7 |
| 2 | 7 |
| 3 | 7 |

*Rel_Person_Paper*

| PersonID | PaperID |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |

*Organizations*

| OrgID | Name | Address | Location | Postcode | Country | Homepage |
|---|---|---|---|---|---|---|
| 7 | Hewlett-Packard Laboratories, Bristol | Filton Road | Bristol | BS34 8QZ | UK | http://www.hpl.hp.com/ |

*Papers*

| paperID | title | abstract | year | conference |
|---|---|---|---|---|
| 1 | Trusting Information Sources One Citizen at a Ti... | This paper describes an approach to derive assessments ... | 2002 | 23541 |
| 2 | Automatic Generation of Java/SQL based Infere... | This paper describes two approaches for automatically co... | 2002 | 23541 |

*Conferences*

| ConfID | Name | Date | Location |
|---|---|---|---|
| 23541 | International Semantic Web Conference 2002 | June 9-12, 2002 | Sardinia |

*Topics*

| TopicID | TopicName | ParentID |
|---|---|---|
| 3 | Artificial Intelligence | 15 |
| 5 | Semantic Web | 15 |
| 15 | Knowledge Management | NULL |

*Rel_Paper_Topic*

| PaperID | TopicID |
|---|---|
| 1 | 5 |
| 1 | 15 |
| 2 | 3 |
| 2 | 5 |

**Figura 5.2 - Estado do Banco de Dados ISWC_REL**

A Tabela 5.5 mostra as triplas geradas a partir da aplicação das regras de transformação no estado do banco de dados. Estas triplas compõem a visão RDF gerada a partir do mapeamento que as assertivas de correspondência definem.

**Tabela 5.5 - Triplas geradas da aplicação das regras de transformação da Tabela 5.4 no estado da Figura 5.1**

| | |
|---|---|
| $ACC_1$ | *http://example.com/person/1 rdf:type foaf:Person*<br>*http://example.com/person/2 rdf:type foaf:Person*<br>*http://example.com/person/3 rdf:type foaf:Person* |
| $ACC_2$ | *http://example.com/document/1 rdf:type foaf:Document*<br>*http://example.com/document/2 rdf:type foaf:Document* |
| $ACC_3$ | *http://example.com/organization/7 rdf:type conf:Organization* |
| $ACC_4$ | *http://example.com/postaladdress/filton%20road/bristol/bs438qz/uk*<br>*rdf:type conf:PostalAddress* |
| $ACC_5$ | *http://example.com/conference/23541 rdf:type conf:Conference* |
| $ACC_6$ | *http://example.com/concept/3 rdf:type skos:Concept*<br>*http://example.com/concept/5 rdf:type skos:Concept*<br>*http://example.com/concept/15 rdf:type skos:Concept* |
| $ACO_1$ | *http://example.com/person/1 conf:hasAffiliation http://example.com/organization/7*<br>*http://example.com/person/2 conf:hasAffiliation http://example.com/organization/7*<br>*http://example.com/person/3 conf:hasAffiliation http://example.com/organization/7* |
| $ACO_2$ | *http://example.com/person/1 conf:researchInterests http://example.com/concept/5*<br>*http://example.com/person/1 conf:researchInterests http://example.com/concept/15*<br>*http://example.com/person/2 conf:researchInterests http://example.com/concept/5* |

| | |
|---|---|
| | *http://example.com/person/2 conf:researchInterests http://example.com/concept/15*<br>*http://example.com/person/3 conf:researchInterests http://example.com/concept/3*<br>*http://example.com/person/3 conf:researchInterests http://example.com/concept/5* |
| $ACO_3$ | *http://example.com/organization/7 vcard:ADR*<br>*http://example.com/postaladdress/filton%20road/bristol/bs438qz/uk* |
| $ACO_4$ | *http://example.com/document/1 skos:subject http://example.com/concept/5*<br>*http://example.com/document/1 skos:subject http://example.com/concept/15*<br>*http://example.com/document/2 skos:subject http://example.com/concept/3*<br>*http://example.com/document/2 skos:subject http://example.com/concept/5* |
| $ACO_5$ | *http://example.com/document/1*<br>*conf:conference http://example.com/conference/23541*<br>*http://example.com/document/2*<br>*conf:conference http://example.com/conference/23541* |
| $ACO_6$ | *http://example.com/concept/5 skos:broader http://example.com/concept/15*<br>*http://example.com/concept/3 skos:broader http://example.com/concept/15* |
| $ACD_1$ | *http://example.com/person/1 foaf:name "Yolanda Gil"*<br>*http://example.com/person/2 foaf:name "Varun Ratnakar"*<br>*http://example.com/person/3 foaf:name "Jim Blythe"* |
| $ACD_2$ | *http://example.com/person/1 foaf:mbox "gil@isi.edu"*<br>*http://example.com/person/2 foaf:mbox "varunr@isi.edu"*<br>*http://example.com/person/3 foaf:mbox "blythe@isi.edu"* |
| $ACD_3$ | *http://example.com/organization/7*<br>*rdfs:label "Hewlett-Packard Laboratories, Bristol"* |
| $ACD_4$ | *http://example.com/organization/7 foaf:homepage "http://www.hpl.hp.com/"* |
| $ACD_5$ | *http://example.com/postaladdress/filton%20road/bristol/bs438qz/uk*<br>*vcard:street "Filton Road"* |
| $ACD_6$ | *http://example.com/postaladdress/filton%20road/bristol/bs438qz/uk*<br>*vcard:locality "Bristol"* |
| $ACD_7$ | *http://example.com/postaladdress/filton%20road/bristol/bs438qz/uk*<br>*vcard:pcode "BS348QZ"* |
| $ACD_8$ | *http://example.com/postaladdress/filton%20road/bristol/bs438qz/uk*<br>*vcard:country "UK"* |
| $ACD_9$ | *http://example.com/document/1 dc:title "Trusting Information Sources One Citize…"*<br>*http://example.com/document/2 dc:title "Automatic Generation of Java/SQL base…"* |
| $ACD_{10}$ | *http://example.com/document/1 dcterms:abstract "This paper describes an appro…"*<br>*http://example.com/document/2 dcterms:abstract "This paper describes two appr…"* |
| $ACD_{11}$ | *http://example.com/document/1 dc:date 2002*<br>*http://example.com/document/2 dc:date 2002* |
| $ACD_{12}$ | *http://example.com/concept/3 skos:prefLabel "Artificial Intelligence"*<br>*http://example.com/concept/5 skos:prefLabel "Semantic Web"*<br>*http://example.com/concept/15 skos:prefLabel "Knowledge Management"* |
| $ACD_{13}$ | *http://example.com/conference/23541*<br>*rdfs:label "International Semantic Web Conference 2002"* |
| $ACD_{14}$ | *http://example.com/conference/23541 dc:date "June 9-12, 2002"* |
| $ACD_{15}$ | *http://example.com/conference/23541 conf:location "Sardinia"* |

## 5.5 Conclusões

Neste capítulo, foi proposto um formalismo para criação de mapeamentos entre esquemas relacionais e vocabulários RDF através de assertivas de correspondência e regras de transformação. Por fim, este formalismo foi aplicado ao estudo de caso e foi apresentada a visão RDF gerada.

# 6 ABORDAGEM PARA GERAÇÃO DE MAPEAMENTOS R2RML

Apresentamos neste capítulo nossa abordagem para geração de mapeamentos R2RML customizados a partir de assertivas de correspondência. Na seção 6.1 apresentamos a visão geral da nossa abordagem. Na seção 6.1.1 detalhamos a arquitetura adotada e na seção 6.1.2 introduzimos os passos do processo de publicação da visão RDF. A seção 6.2 descreve o passo 1, responsável pela geração da ontologia exportada *(OE)*. A seção 6.3 explica o passo 2, responsável pela criação da camada de visões relacionais e dos mapeamentos R2RML. Na seção 6.3.1 detalhamos o algoritmo para geração do esquema das visões relacionais *(EV)* e do mapeamento R2RML. Na seção 6.3.2 explicamos o algoritmo para gerar as cláusulas SQL de criação das visões em *EV*. A seção 6.4 mostra como é feita a publicação da *OE* em um *SPARQL endpoint*. Finalmente, na seção 6.5 apresentamos nossas conclusões e resultados.

## 6.1 Visão Geral

### *6.1.1 Arquitetura Adotada*

Nesta abordagem utilizamos uma arquitetura em três camadas para mapear bancos de dados relacionais em grafos RDF. Esta arquitetura está ilustrada na Figura 6.1.

A camada no topo da arquitetura contém a *OE* que é uma visão RDF dos dados relacionais exportados do *Esquema Fonte (EF)* na camada mais baixa. O vocabulário da *OE* é um subconjunto do vocabulário da *Ontologia Alvo*, pois contempla apenas os termos que foram mapeados do *EF*.

A camada intermediária consiste em um conjunto de visões denominado Esquema das Visões (*EV*), onde *EV* é gerado diretamente da *OE*. As visões em *EV* podem ser visões relacionais ou visões R2RML. Na nossa abordagem, o uso desta camada intermediária quebra o mapeamento em dois estágios: a definição de *mapeamentos SQL* e a definição de *mapeamentos R2RML*. As principais vantagens desta abordagem são:

**(i)** A criação do esquema de visões facilita a geração dos mapeamentos R2RML;

**(ii)** Alterações no *EF* não implicam em alterações nos mapeamentos R2RML.

Para construir esta arquitetura, propomos um processo que será apresentado na próxima seção.

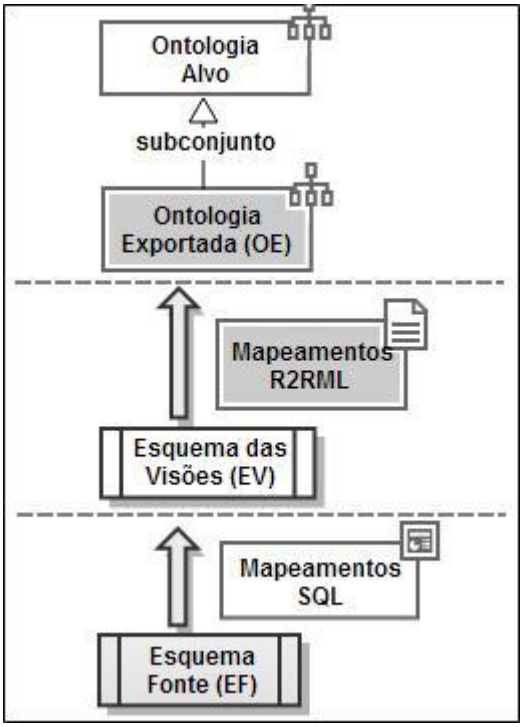


**Figura 6.1 - Arquitetura em Três Camadas**

### *6.1.2 Processo para publicação de dados RDF*

O processo para a publicação de dados RDF consiste de três passos principais, discutidos com maiores detalhes nas seções seguintes. Estes passos estão ilustrados na Figura 6.2. As entradas para o processo são:

- ***D*** = $(V_D, C_D)$ é a ontologia alvo.
- ***EF*** é o esquema da fonte de dados que precisa ser mapeado para ***D***.
- ***A*** é um mapeamento, ou seja, um conjunto de assertivas de correspondência entre $V_D$ e ***EF***.

Ao final do processo, as três camadas da arquitetura e seus componentes estarão criados. Portanto, será possível criar um SPARQL *endpoint* para disponibilizar os dados de ***EF*** na forma de um grafo RDF.

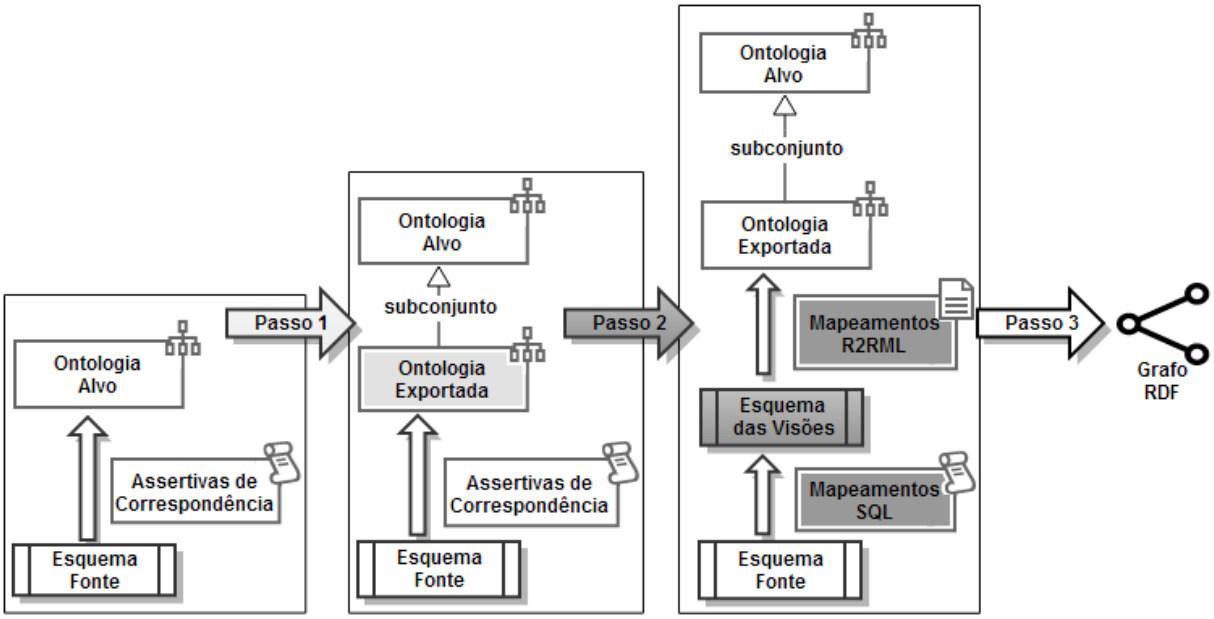


**Figura 6.2 - Passos do Processo de Publicação**

De forma simplificada, o primeiro passo envolve a geração da ontologia exportada ***OE*** (detalhado na seção 6.2). O segundo passo contempla a geração do esquema de visões relacionais ***EV***, do mapeamento R2RML ***M_R2RML*** que mapeia *EV* para *OE*, e dos mapeamentos SQL ***M_SQL*** que mapeiam ***EF*** em ***EV***. O algoritmo para criação de *EV* e de *M_R2RML* é apresentado na seção 6.3 e o algoritmo para gerar *M_SQL* é apresentado na seção 6.4. O terceiro e último passo consiste na publicação do grafo definido pela *OE*. Esta publicação possibilita a execução de consultas *SPARQL* nesta ontologia. Estas consultas podem ser traduzidas em consultas SQL nas visões de *EV* por meio do mapeamento *M_R2RML*.

## 6.2 Geração da Ontologia Exportada

Considere ***E*** = $(V_E, C_E)$ a ontologia exportada que será gerada ao final deste passo. Como proposto em (CASANOVA et al., 2011), é necessário que a ontologia exportada seja um fragmento aberto ou fechado da ontologia alvo, o que implica que $V_E$ é um subconjunto de $V_D$. Esta premissa implica que ***E*** pode ser automaticamente gerada a partir de ***A*** (CULLOT et al., 2007). Brevemente, um termo *T* (classe ou propriedade) de $V_D$ está em $V_E$ se e somente se existe uma assertiva de correspondência para *T* em ***A***. As restrições em $C_E$ são geradas usando a *procedure* ***closedFragment*** introduzida em (CASANOVA et al., 2011).

Considere, por exemplo, o esquema *ISWC_REL* da Figura 4.1, a ontologia de domínio *CONF_OWL* na Figura 4.2 e as assertivas de correspondência entre *ISWC*_REL e *CONF_OWL* da Tabela 5.3. A Figura 5.1 apresentada no capítulo 5 mostra a ontologia exportada *ISWC_RDF*. O vocabulário de *ISWC_RDF* contém todos os elementos da ontologia *CONF_OWL* que possuem um elemento correspondente em *ISWC_REL*. Por exemplo, a classe *foaf:Person* corresponde à relação *Persons*. Esta correspondência foi estabelecida pela assertiva de classe $ACC_1$. Apenas três propriedades desta classe foram mapeadas: as propriedades de dados *foaf:name* e *foaf:mbox* e a propriedade de objetos *conf:hasAffiliation*. As assertivas de correspondência responsáveis por estes três mapeamentos são, respectivamente: $ACD_1$, $ACD_2$ e $ACO_1$. Desta forma, as demais propriedades da classe *foaf:Person* (*rdf:type*, *conf:address*, *foaf:homepage*, *conf:phone* e *foaf:depiction*) não estão contempladas na ontologia exportada por não existir nenhuma assertiva de correspondência mapeando-as para algum elemento do esquema fonte.

## 6.3 Geração da Camada de Visões e do mapeamento R2RML

### *6.3.1 Algoritmo para gerar EV e M_R2RML*

A Tabela 6.1 mostra o algoritmo que, com base na ontologia exportada $\boldsymbol{E} = (V_E, C_E)$, gera automaticamente os esquemas das visões relacionais ***EV*** e o mapeamento R2RML ***M_R2RML*** de ***EV*** para ***E***. O algoritmo tem três passos principais que são explicados com maiores detalhes a seguir:

- **Passo 1**: Para cada classe em $V_E$, o algoritmo cria uma visão relacional e seus atributos chave (linhas 1.1 e 1.2) e o *subject map* para a assertiva de classe $\Psi_C$ usando o *template* $T_1$ da Tabela 6.2 (Linha 1.3). O *subject* é obtido pela concatenação do *namespace* da classe com os valores dos atributos da chave primária da visão relacional. Desta forma, é garantido que a URI gerada é única. Por exemplo, para a classe *foaf:Person* em *ISWC_RDF* são criados neste passo:

  **(i)** Uma visão em *EV* denominada *Person* (ver Figura 6.4) com um atributo *ID* como chave primária;

**(ii)** Um *subject map* em *M_R2RML* exibido na linha 1 da Tabela 6.3.

- **Passo 2**: As propriedades de dados em $V_E$ são processadas pelo algoritmo. Dois casos são possíveis:
  - Caso 2.1. A propriedade de dados tem cardinalidade igual a 1. Neste caso o algoritmo cria um novo atributo na visão (linha 2.1.1) e o *predicate object map* para a ACD usando o *template* $T_2$ da Tabela 6.2 (linha 2.1.2). Como exemplo, podemos citar a propriedade *foaf:mbox*, cuja cardinalidade é 1. Para esta propriedade, o algoritmo irá:

    **(i)** Alterar a visão *Person* para adicionar nela o atributo *mbox*, pois *foaf:Person* é a classe domínio de *foaf:mbox*;

    **(ii)** Incluir em *M_R2RML* o *predicate object map* visto na linha 2 da Tabela 6.3.
  - Caso 2.2. A propriedade de dados tem cardinalidade maior que 1. Neste caso o algoritmo cria uma nova visão relacional com uma chave estrangeira e um atributo (linhas 2.2.1 a 2.2.4) e o *subject map* e *predicate object map* para a ACD usando o *template* $T_3$ da Tabela 6.2 (linha 2.2.5).
- **Passo 3**: As propriedades de objeto em $V_E$ são processadas pelo algoritmo. Novamente dois casos são possíveis:
  - Caso 3.1. A propriedade de objeto tem cardinalidade igual a "1". Neste caso o algoritmo cria os atributos chave e a chave estrangeira (linhas 3.1.1 e 3.1.2) e o *predicate object map* para a ACO usando o *template* $T_4$ da Tabela 6.2 (linha 3.1.3). Para exemplificar este caso, temos a propriedade de objeto *conf:conference* com cardinalidade igual a 1, cujo domínio é *foaf:Document* e cuja imagem é *conf:Conference*. Assim, o algoritmo fará três modificações:

    **(i)** Alterar a visão *Document* para adicionar nela o atributo *ID_Conference*;

    **(ii)** Criar a chave estrangeira *FK(Document:{ID_Conference}, Conference:{ID})*;

    **(ii)** Incluir em *M_R2RML* o *predicate object map* como na linha 7 da Tabela 6.3.
  - Caso 3.2. A propriedade de objeto tem cardinalidade maior que 1. No segundo caso o algoritmo cria uma nova visão relacional com duas chaves estrangeiras (linhas 3.2.1 a 3.2.5) e o *subject map* e *predicate object map* para a ACO usando o *template* $T_5$ da Tabela 6.2 (linha 3.1.6). Para este caso podemos utilizar como exemplo a propriedade de objeto *dc:creator*. O domínio desta propriedade é *foaf:Document* e a imagem é *foaf:Person*. Como um documento pode ter mais de um criador, esta propriedade é definida com cardinalidade maior que 1. Logo, o algoritmo fará as seguintes alterações:

**(i)** Criar a visão *Document_Person* e adicionar os atributos *ID_Document* e *ID_Person* nela. Estes atributos serão chave primária;

**(ii)** Criar duas *chaves estrangeiras:*

- *$FK_1$(Document_Person:{ID_Document}, Conference:{ID});*
- *$FK_2$(Document_Person:{ID_Person }, Person:{ID});*

**(iii)** Incluir em *M_R2RML* o *subject map* e o *predicate object map* como na linha 9 da Tabela 6.3.

**Tabela 6.1 - Algoritmo para gerar *EV* e *M_R2RML***

| |
|---|
| **Entrada:** Ontologia Exportada (*E*)<br>**Saída:** Esquema das Visões (*EV*) e mapeamento R2RML (*M_R2RML*) |
| **PASSO 1**: Para cada classe *C* em $V_E$ onde $K_1,...,K_n$ são as propriedades de dados da URI de *C* Faça<br>1.1 Criar uma visão relacional também denominada *C*;<br>1.2 Criar os atributos $K_1, ..., K_n$ que formam a chave primária da visão *C*;<br>1.3 Criar o *subject map* para $\Psi_C$ usando o Template $T_1$ da Tabela 6.2.<br>**PASSO 2**: Para cada propriedade de dados *P* em $V_E$ Faça<br>Seja *D* a visão criada no Passo 1.1 que corresponde ao domínio de *P e sejam* $K_{D1},...,K_{Dn}$ os atributos que formam a chave primária de *D*;<br>Caso 2.1: *P* tem cardinalidade igual a 1.<br>2.1.1 Criar atributo *P* na visão *D* cujo tipo é definido de acordo com a imagem da propriedade *P*;<br>2.1.2 Criar o *predicate object map* para $\Psi_P$ usando o Template $T_2$ da Tabela 6.2 e adicionar ao *triples map* da visão *D* que foi criado no passo 1.3;<br>Caso 2.2: *P* tem cardinalidade maior que 1.<br>2.2.1 Criar a visão relacional *D_P;*<br>2.2.2 Criar os atributos $K_{D1},...,K_{Dn}$ em *D_P* cujos tipos são definidos como em *D*;<br>2.2.3 Criar a chave estrangeira *FK_D_P(D_P:{*$K_{D1},...,K_{Dn}$*}, D:{*$K_{D1},...,K_{Dn}$ *});*<br>2.2.4 Criar o atributo *P* em *D_P* cujo tipo é definido de acordo com a imagem da propriedade *P*;<br>2.2.5 Criar o *subject map* e o *predicate object map* para $\Psi_P$ usando o Template $T_3$ da Tabela 6.2.<br>**PASSO 3**: Para cada propriedade de objeto *P* em $V_E$ Faça<br>Sejam *D* e *R* as visões criadas no Passo 1.1 que correspondem ao domínio e à imagem de *P*, respectivamente, sejam $K_{D1},...,K_{Dn}$ os atributos da chave primária de *D* e sejam $K_{R1},...,K_{Rn}$ os atributos da chave primária de *R*;<br>Caso 3.1: *P* tem cardinalidade igual a 1.<br>3.1.1 Criar atributos $K_{R1}, ..., K_{Rn}$ em D cujos tipos são definidos como em *R*;<br>3.1.2 Criar a chave estrangeira *FK_D_R(D:{*$K_{R1}, ..., K_{Rn}$*}, R:{*$K_{R1}, ..., K_{Rn}$*})*<br>3.1.3 Criar o *predicate object map* para $\Psi_P$ usando o Template $T_4$ da Tabela 6.2 e adicionar ao *triples map* da visão *D* que<br>foi criado no passo 1.4;<br>Caso 3.2: *P* tem cardinalidade maior que 1.<br>3.2.1 Criar a visão relacional *D_P*;<br>3.2.2 Criar os atributos $K_{D1},...,K_{Dn}$ em *D_P* cujos tipos são definidos como em D;<br>3.2.3 Criar a chave estrangeira *FK_D_P_D(D_P:{*$K_{D1},...,K_{Dn}$*}, D:{*$K_{D1},...,K_{Dn}$ *});*<br>3.2.4 Criar os atributos $K_{R1}, ..., K_{Rn}$ em *D_P* cujos tipos são definidos como em *R*;<br>3.2.5 Criar a chave estrangeira *FK_D_P_R(D_P:{*$K_{R1}, ..., K_{Rn}$*}, R:{*$K_{R1}, ..., K_{Rn}$*});*<br>3.2.6 Criar o *subject map* e o *predicate object map* para $\Psi_P$ usando o Template $T_5$ da Tabela 6.2. |

**Tabela 6.2 - Templates para traduzir ACs simples para mapeamentos R2RML**

| | |
|---|---|
| $T_1$ | <#**C**_TriplesMap><br>rr:logicalTable [ rr:tableName "**C**" ];<br>rr:subjectMap [<br>rr:template "**namespaceDeC**/{$\mathbf{K_1}$}/{$\mathbf{K_2}$}/… /{$\mathbf{K_n}$}/";<br>rr:class **C**;<br>]; |
| $T_2$ | rr:predicateObjectMap [<br>rr:predicate **P**;<br>rr:objectMap [ rr:column "**P**" ];<br>]; |
| $T_3$ | <#**D_P_**TriplesMap><br>rr:logicalTable [ rr:tableName "**D_P**" ];<br>rr:subjectMap [<br>rr:template "**namespaceDeD**/{$\mathbf{K_{D1}}$}/{$\mathbf{K_{D2}}$}/... /{$\mathbf{K_{Dn}}$}/";<br>rr:class **D**;<br>];<br>rr:predicateObjectMap [<br>rr:predicate **P**;<br>rr:objectMap [ rr:column "**P**" ];<br>]; |
| $T_4$ | rr:predicateObjectMap [<br>rr:predicate **P**;<br>rr:objectMap [<br>rr:parentTriplesMap <**R_**TriplesMap>;<br>rr:joinCondition [<br>rr:child "$\mathbf{K_{R1}}$";<br>rr:parent "$\mathbf{K_{R1}}$";<br>];<br>…<br>rr:joinCondition [<br>rr:child "$\mathbf{K_{Rn}}$";<br>rr:parent "$\mathbf{K_{Rn}}$";<br>];<br>];<br>]; |
| $T_5$ | <#**D_P_**TriplesMap><br>rr:logicalTable [ rr:tableName "**D_P**" ];<br>rr:subjectMap [<br>rr:template "**namespaceDeD**/{$\mathbf{K_{D1}}$}/{$\mathbf{K_{D2}}$}/... /{$\mathbf{K_{Dn}}$}/";<br>rr:class **D**; ];<br>rr:predicateObjectMap [<br>rr:predicate **P**;<br>rr:objectMap [<br>rr:parentTriplesMap <**R_**TriplesMap>;<br>rr:joinCondition [<br>rr:child "$\mathbf{K_{R1}}$";<br>rr:parent "$\mathbf{K_{R1}}$"; ];<br>…<br>rr:joinCondition [<br>rr:child "$\mathbf{K_{Rn}}$";<br>rr:parent "$\mathbf{K_{Rn}}$"; ]; ]; ]; |

A Figura 6.3 e a Tabela 6.3 mostram as saídas do Algoritmo da Tabela 6.1 para o nosso estudo de caso. A Figura 6.3 apresenta o esquema de visões relacionais *ISWC_Views* para a ontologia exportada *ISWC_RDF* e a Tabela 6.3 mostra os mapeamentos R2RML gerados utilizando os templates da Tabela 6.2.

As visões: *Person*, *Document*, *Conference*, *Concept*, *Organization* e *PostalAddress* foram criadas a partir das classes em $V_E$. As outras visões foram criadas a partir de propriedades de objeto com cardinalidade maior que 1. É importante salientar que para cada visão gerada é criado um *triples map* no mapeamento R2RML.

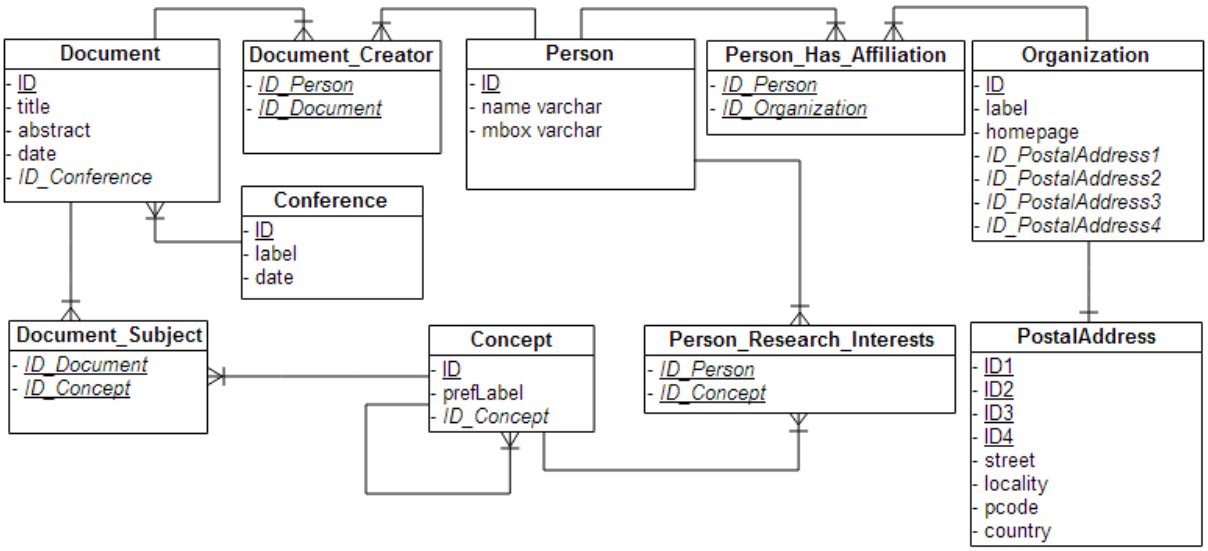


**Figura 6.3 - Esquema das Visões *ISWC_Views* saída do algoritmo da Tabela 6.1**

**Tabela 6.3 - Mapeamentos R2RML saída do algoritmo da Tabela 6.1**

| | |
|---|---|
| 1 | <#Person_TriplesMap><br>rr:logicalTable [ rr:tableName "**Person**" ];<br>rr:subjectMap [<br>rr:template "http://xmlns.com/foaf/0.1/person/{**ID**}";<br>rr:class foaf:Person;<br>]; |
| 2 | rr:predicateObjectMap [<br>rr:predicate foaf:mbox;<br>rr:objectMap [ rr:column "**mbox**" ];<br>]; |
| 3 | rr:predicateObjectMap [<br>rr:predicate foaf:name;<br>rr:objectMap [ rr:column "**name**" ];<br>]. |
| 4 | <#Person_Has_Affiliation_TriplesMap><br>rr:logicalTable [ rr:tableName "**Person_Has_Affiliation**" ];<br>rr:subjectMap [<br>rr:template "http://xmlns.com/foaf/0.1/person/{**ID_Person**}";<br>rr:class foaf:Person;<br>];<br>rr:predicateObjectMap [<br>rr:predicate conf:has_affiliation;<br>rr:objectMap [<br>rr:parentTriplesMap <#Organization_TriplesMap>;<br>rr:joinCondition [<br>rr:child "**ID_Organization**";<br>rr:parent "ID";<br>];<br>];<br>]. |
| 5 | <#Person_Research_Interests_TriplesMap ><br>rr:logicalTable [ rr:tableName "**Person_Research_Interests**" ];<br>rr:subjectMap [<br>rr:template "http://xmlns.com/foaf/0.1/person/{**ID_Person**}"; |

| | |
|---|---|
| | rr:class foaf:Person;<br>];<br>rr:predicateObjectMap [<br>rr:predicate conf:research_interests;<br>rr:objectMap [<br>rr:parentTriplesMap <#ConceptTriplesMap>;<br>rr:joinCondition [<br>rr:child "**ID_Concept**";<br>rr:parent "ID";<br>];<br>];<br>]. |
| 6 | <#Document_TriplesMap><br>rr:logicalTable [ rr:tableName "**Document**" ];<br>rr:subjectMap [<br>rr:template "http://xmlns.com/foaf/0.1/document/{**ID**}/";<br>rr:class foaf:Document;<br>];<br>rr:predicateObjectMap [<br>rr:predicate dc:title;<br>rr:objectMap [ rr:column "**title**" ];<br>];<br>rr:predicateObjectMap [<br>rr:predicate dcterms:abstract;<br>rr:objectMap [ rr:column "**abstract**" ];<br>];<br>rr:predicateObjectMap [<br>rr:predicate dc:date;<br>rr:objectMap [ rr:column "**date**" ];<br>]; |
| 7 | rr:predicateObjectMap [<br>rr:predicate conf:conference;<br>rr:objectMap [<br>rr:parentTriplesMap <#Conference_TriplesMap>;<br>rr:joinCondition [<br>rr:child "**ID_Conference**";<br>rr:parent "ID";<br>];<br>];<br>]. |
| 8 | <#Document_Subject_TriplesMap><br>rr:logicalTable [ rr:tableName "**Document_Subject**" ];<br>rr:subjectMap [<br>rr:template " http://xmlns.com/foaf/0.1/document/{**ID_Document**}";<br>rr:class foaf:Document;<br>];<br>rr:predicateObjectMap [<br>rr:predicate skos:subject;<br>rr:objectMap [<br>rr:parentTriplesMap <#Concept_TriplesMap>;<br>rr:joinCondition [<br>rr:child "**ID_Concept**";<br>rr:parent "ID";<br>];<br>];<br>]. |
| 9 | <#Document_Creator_TriplesMap><br>rr:logicalTable [ rr:tableName "**Document_Creator**" ];<br>rr:subjectMap [<br>rr:template " http://xmlns.com/foaf/0.1/document/{**ID_Document**}";<br>rr:class foaf:Document;<br>];<br>rr:predicateObjectMap [<br>rr:predicate dc:creator;<br>rr:objectMap [<br>rr:parentTriplesMap <#Person_TriplesMap>;<br>rr:joinCondition [ |

```
            rr:child "ID_Person";
            rr:parent "ID";
          ];
      ];
    ].
```

```
10  <#Organization_TriplesMap>
      rr:logicalTable [ rr:tableName "Organization" ];
      rr:subjectMap [
        rr:template "conf/organization/{ID}/";
        rr:class conf:Organization;
      ];
      rr:predicateObjectMap [
        rr:predicate rdfs:label;
        rr:objectMap [ rr:column "label" ];
      ];
      rr:predicateObjectMap [
        rr:predicate foaf:homepage;
        rr:objectMap [ rr:column "homepage" ];
      ];
      rr:predicateObjectMap [
        rr:predicate vcard:ADR;
        rr:objectMap [
          rr:parentTriplesMap <#PostalAddress_TriplesMap>;
          rr:joinCondition [
            rr:child "ID_PostalAddress1";
            rr:parent "ID1";
          ];
          rr:joinCondition [
            rr:child "ID_PostalAddress2";
            rr:parent "ID2";
          ];
          rr:joinCondition [
            rr:child "ID_PostalAddress3";
            rr:parent "ID3";
          ];
          rr:joinCondition [
            rr:child "ID_PostalAddress4";
            rr:parent "ID4";
          ];
        ];
      ].
```

```
11  <#PostalAddress_TriplesMap>
      rr:logicalTable [ rr:tableName "PostalAddress" ];
      rr:subjectMap [
        rr:template "conf/postaladdress/{ID1}/{ID2}/{ID3}/{ID4}";
        rr:class conf:PostalAddress;
      ];
      rr:predicateObjectMap [
        rr:predicate vcard:street;
        rr:objectMap [ rr:column "street" ];
      ];
      rr:predicateObjectMap [
        rr:predicate vcard:locality;
        rr:objectMap [ rr:column "locality" ];
      ];
      rr:predicateObjectMap [
        rr:predicate vcard:pcode;
        rr:objectMap [ rr:column "pcode" ];
      ];
      rr:predicateObjectMap [
        rr:predicate vcard:country;
        rr:objectMap [ rr:column "country" ];
      ].
```

```
12  <#Conference_TriplesMap>
      rr:logicalTable [ rr:tableName "Conference" ];
      rr:subjectMap [
        rr:template "conf/conference/{ID}/";
```

| | |
|---|---|
| | rr:class conf:Conference;<br>];<br>rr:predicateObjectMap [<br>rr:predicate rdfs:label;<br>rr:objectMap [ rr:column "**label**" ];<br>];<br>rr:predicateObjectMap [<br>rr:predicate dc:date;<br>rr:objectMap [ rr:column "**date**" ];<br>]. |
| 13 | <#Concept_TriplesMap><br>rr:logicalTable [ rr:tableName "**Concept**" ];<br>rr:subjectMap [<br>rr:template " http://www.w3.org/2004/02/skos/core/concept/{**ID**}";<br>rr:class skos:Concept;<br>];<br>rr:predicateObjectMap [<br>rr:predicate skos:prefLabel;<br>rr:objectMap [ rr:column "**prefLabel**" ];<br>];<br>rr:predicateObjectMap [<br>rr:predicate skos:broader;<br>rr:objectMap [<br>rr:parentTriplesMap <#Concept_TriplesMap>;<br>rr:joinCondition [<br>rr:child "**ID_Concept**";<br>rr:parent "ID";<br>];<br>];<br>]. |

### *6.3.2 Algoritmo para gerar os SQLs das visões em EV*

Tendo criado o esquema das visões relacionais, é preciso gerar as cláusulas SQL que definem cada visão individualmente. Estas cláusulas são automaticamente geradas com base na ontologia exportada (*E*) e nas assertivas de correspondência entre $V_D$ e *EF*. O algoritmo da Tabela 6.4 apresenta os passos para criação dos SQLs das visões usando os *templates* definidos na Tabela 6.5.

O algoritmo inicia percorrendo as classes do vocabulário da ontologia exportada ($V_E$). Para cada classe *C*, onde os atributos que compõem sua URI são $K_1$, ..., $K_n$. Os passos para conceber a definição da visão de *C* são descritos como segue:

- **Passo 1**: Cria a variável *N* e atribue o nome da classe C para ela. *N* será usada como nome da visão relacional. Por exemplo, para a classe *foaf:Document* em $V_E$, *N* recebe o valor “Document”.
- **Passo 2**: Cria três listas: *T* de nomes de tabelas a serem incluídas na cláusula *FROM* da Visão *N*, *J* de *joins* a serem incluídos na cláusula *WHERE* e *LA* dos

pares (nome do atributo, *alias* do atributo) que serão selecionados na cláusula *SELECT*. Logo na criação da lista *LA* são incluídos os atributos chave da visão ($K_1$,...,$K_n$) com seus aliases ID1,...,IDn respectivamente e na criação da lista *T* é incluída a relação *R* mapeada na classe *C* em $\Psi_C$.

Tomando, como exemplo, a ACC:

*foaf:Document* ≡ *Papers*[*paperID*], $\sigma$[*papers.Year* > *2002*].

Neste caso, temos que:

**(i)** A lista de atributos inicialmente será *LA*[(*"paperID", "ID"*)];

**(ii)** A lista de relações será criada como *T* [*"Papers"*].

**(iii)** O SQL inicial que define a visão *Document* será:

"*SELECT paperID AS ID FROM Papers*"

- **Passo 3**: Percorre as propriedades de dados do vocabulário $V_E$ cujo domínio é a classe *C*. A variável *p* recebe o nome da propriedade *P*. Considere que $\Psi_P$ é a ACD de *P*. Temos então dois casos possíveis:

  Caso 3.1: A cardinalidade de *P* é 1. Para cada tipo de ACD o algoritmo realiza um tratamento específico:

  - $\Psi_P$: *P* ≡ *R* / *A*: o atributo *A* é adicionado à lista *LA* com alias igual a *p*;
  - $\Psi_P$: *P* ≡ *R* / {$A_1$,...,$A_n$}: os atributos $A_1$,...,$A_n$ são adicionados à lista *LA* com aliases *p* + *"1"*, ... , *p* + *"n"*, respectivamente;
  - $\Psi_P$: *P* ≡ *R* /$\varphi$ / *B*: neste caso é preciso percorrer cada chave estrangeira (*fk* relacionando as tabelas $T_1$ e $T_2$) no caminho $\varphi$ para incluir $T_1$ e $T_2$ em *T* e os pares de *joins* em *J*. Por fim, o atributo *B* é adicionado à lista *LA*.
  - $\Psi_P$: *P* ≡ *R* /$\varphi$ / {$B_1$,...,$B_n$}: Idêntico ao caso anterior com um única diferença: no final os atributos $B_1$,...,$B_n$ são adicionados à lista *LA*.

Para exemplificar este caso, tomamos as propriedades de dados em $V_E$ cujo domínio é *foaf:Document*: *dc:title*, *dcterms:abstract* e *dc:date*. Todas estas propriedades têm cardinalidade igual a 1 e suas ACDs são do tipo $\Psi_P$: *P* ≡ *R* / *A*:

**(i)** *dc:title* ≡ *Papers / Title*;

**(ii)** *dcterms:abstract* ≡ *Papers / Abstract*; e

**(iii)** *dc:date* ≡ *Papers / Year* .

Portanto, o algoritmo irá incluir o nome de cada propriedade em *LA* e, ao final deste passo, teremos:

**(a)** *LA*[(*"paperID", "ID"*), (*"Title", "title"*), (*"Abstract", "abstract"*), (*"Year", "date"*)]; e

**(b)** o SQL da visão *Document*:

*"SELECT paperID AS ID, Title AS title, Abstract AS abstract,*
*Year AS date*
*FROM Papers".*

Caso 3.2: A cardinalidade de *P* é MAIOR que 1. Neste caso é preciso criar a visão específica de *P*. Assim:

- A variável *np* é definida para ser o nome desta nova visão e seu valor será a concatenação do nome da classe *C* com o nome da propriedade *P*;
- São criadas novas listas $T_p$, $J_p$ e $LA_p$ de tabelas, joins e atributos, respectivamente. Os atributos ($K_1$,...,$K_n$) são incluídos em $LA_p$, pois irão compor uma chave estrangeira para a visão da classe *C*.
- Neste caso teremos somente dois tipos de ACD: $\Psi_P$: $P \equiv R / \varphi / B$ ou $\Psi_P$: $P \equiv R / \varphi / \{B_1,...,B_n\}$. Logo, será preciso percorrer cada chave estrangeira (*fk* relacionando as tabelas $T_1$ e $T_2$) no caminho $\varphi$ para incluir $T_1$ e $T_2$ em $T_p$ e os pares de *joins* em $J_p$. Em seguida o atributo *B* ou os atributos $B_1,...,B_n$ são adicionados à lista $LA_p$ dependendo do tipo de ACD. Por fim, para criação da visão usando o template correto, é feita uma verificação para saber se a ACC da classe de domínio *C* possui um filtro de seleção. Caso não possua, é utilizado o template $TV_3$ da Tabela 6.5 passando os parâmetros: nome da visão *np* e as listas $T_p$, $J_p$ e $LA_p$. Caso contrário, é utilizado o template $TV_4$ da Tabela 6.5 e o filtro de seleção $\sigma$ é adicionado aos parâmetros já mencionados.

- **Passo 4**: Percorre as propriedades de objetos do vocabulário $V_E$ cujo domínio é a classe *C*. A variável *o* recebe o nome da propriedade *O* e a variável *I* recebe o nome da imagem da propriedade *O*. Considere que $\Psi_O$ é a ACO de *O*. Temos então dois casos possíveis:

  Caso 4.1: A cardinalidade de *O* é 1. Para cada tipo de ACO o algoritmo realiza um tratamento específico:

- $\Psi_O$: $O \equiv R$: considere $\Psi_I$: $I \equiv R[K_1,...,K_n]$ como sendo a ACC da imagem de *O*. Os atributos $K_1,...,K_n$ são adicionados à lista *LA* com *aliases o + "1", ... , o + "n"*, respectivamente;
- $\Psi_O$: $O \equiv R / \varphi$: Se $\varphi$ possui apenas uma chave estrangeira *fk*($T_1$ : $[a_1, ... ,a_n]$, $T_2$ : $[b_1, ... ,b_n]$) os atributos $a_1, ..., a_n$ são adicionados à lista *LA*. Caso contrário, é preciso percorrer cada chave estrangeira (*fk* relacionando as tabelas $T_1$ e $T_2$) no caminho $\varphi$ para incluir $T_1$ e $T_2$ em *T* e os pares de *joins* em *J*. Por fim, os atributos $a_1,...,a_n$ que compõem a última *fk* são adicionados à lista *LA*.

Por exemplo, a propriedade *conf:conference* em $V_E$ cujo domínio é *foaf:Document* e cuja imagem é *conf:Conference*, tem cardinalidade igual a 1 e sua ACO é do tipo $\Psi_O$: $O \equiv R / \varphi$ (*conf:conference* ≡ *Papers / Fk_Conferences*). Note que o caminho $\varphi$ possui apenas uma *fk*. Assim, o algoritmo irá incluir na lista de atributos *LA*, o atributo que compõe a chave *Fk_Conferences*. Portanto, teremos neste ponto:

**(i)** *LA*[(*"PaperID", "ID"*), (*"Title", "title"*), (*"Abstract", "abstract"*), (*"Year", "date"*), (*"Conference", "ID_Conference"*)]; e

**(ii)** o SQL da visão *Document* neste ponto será:

*"SELECT paperID AS ID, Title AS title, Abstract AS abstract,*
*Year AS date, Conference AS ID_Conference*
*FROM Papers".*

Caso 4.2: A cardinalidade de *O* é MAIOR que 1. Neste caso é preciso criar uma visão específica para *O*. Assim:

- A variável *no* é definida para ser o nome desta nova visão e seu valor será a concatenação do nome da classe *C* com o nome da propriedade *O*;
- São criadas novas listas $T_o$, $J_o$ e $LA_o$ de tabelas, joins e atributos, respectivamente. Os atributos ($K_1,...,K_n$) são incluídos em $LA_o$, pois irão compor uma chave estrangeira para a visão da classe *C*.
- Neste caso, teremos a ACO do tipo: $\Psi_P$: $O \equiv R / \varphi$. Logo, será preciso percorrer cada chave estrangeira (*fk* relacionando as tabelas $T_1$ e $T_2$) no caminho $\varphi$ para incluir $T_1$ e $T_2$ em $T_o$ e os pares de *joins* em $J_o$. Em seguida os atributos $a_1,...,a_n$ que compõem a última *fk* são adicionados à lista $LA_o$. Por fim, é feita uma verificação para saber se a ACC da classe

de domínio *C* possui um filtro de seleção. Caso não possua, é utilizado o template $TV_3$ da Tabela 6.5 passando os parâmetros: nome da visão *no* e as listas $T_o$, $J_o$ e $LA_o$. Caso contrário, é utilizado o template $TV_4$ da Tabela 6.5 e o filtro de seleção $\sigma$ é adicionado aos parâmetros já mencionados.

Por exemplo, a propriedade *skos:subject* em $V_E$ cujo domínio é *foaf:Document* e cuja imagem é a classe *skos:Concept*, tem cardinalidade maior que 1 e sua ACO é do tipo $\Psi_O$: $O \equiv R \,/\, \varphi$ (*skos:subject* $\equiv$ *Papers* / [*Fk_Papers, Fk_Topics*]). Dessa forma, o algoritmo irá:

**(i)** Criar uma nova visão denominada *Document_Subject*;

**(ii)** Criar $LA_o$ = [(*"PaperID", "ID_Document"*), (*"TopicID", "ID_Concept"*)];

**(iii)** Criar $J_o$ = [(*"papers.PaperID", "rel_paper_topic.PaperID"*), (*"rel_paper_topic.TopicID", "topics.TopicID"*)];

**(iv)** Criar $T_o$ = [*papers, rel_paper_topic, topics*].

**(v)** Incluir a condição de seleção definida na ACC da classe de domínio: *papers.Year > 2002*.

O SQL resultante é mostrado na linha $V_5$ da Tabela 6.6.

- **Passo 5**: No último passo será criada a visão da classe C. Considere $\Psi_C$ a ACC da classe *C*. Temos quatro situações possíveis:
  - A lista de joins *J* está vazia e $\Psi_C$ não possui filtro de seleção:
    A visão é criada utilizado o template $TV_1$ da Tabela 6.5 passando os parâmetros: nome da visão *N* e as listas *LA* e *T*.
  - A lista de joins está vazia e $\Psi_C$ possui filtro de seleção:
    A visão é criada utilizado o template $TV_2$ da Tabela 6.5 passando os parâmetros: nome da visão *N* e as listas *LA*, *T* e o filtro de seleção $\sigma$.
  - A lista de joins *J* não está vazia e $\Psi_C$ não possui filtro de seleção:
    A visão é criada utilizado o template $TV_3$ da Tabela 6.5 passando os parâmetros: nome da visão *N* e as listas *LA*, *T* e *J*.
  - A lista de joins *J* não está vazia e $\Psi_C$ possui filtro de seleção:
    A visão é criada utilizado o template $TV_4$ da Tabela 6.5 passando os parâmetros: nome da visão *N* e as listas *LA*, *T*, *J* e o filtro de seleção $\sigma$.

**Tabela 6.4 - Algoritmo para gerar o SQL das Visões Relacionais**

**Entrada:** Ontologia Exportada (*E*), Conjunto de Assertivas de Correspondência (A)
**Saída:** SQL para criação das Visões Relacionais em *EV*

Para cada classe *C* em $V_E$, onde $\Psi_C$: $C \equiv R[K_1, \ldots, K_n]$ é a ACC de *C* Faça
1. $N \leftarrow nome(C)$
2. Criar $T[R]$, $J[]$ e $LA[(K_1, \text{“ID1”}), \ldots, (K_n, \text{“IDn”})]$
3. Para cada propriedade de dados *P* em $V_E$, onde *domínio(P)* = *C* Faça
   $p \leftarrow nome(P)$
   Caso 3.1: *P* tem cardinalidade IGUAL a 1
      Se $\Psi_P$: $P \equiv R / A$ Então $LA \leftarrow LA + [(A, p)]$
      Se $\Psi_P$: $P \equiv R / \{A_1,\ldots,A_n\}$ Então $LA \leftarrow LA + [(A_1, p + \text{“1”})] + \ldots + [(A_n, p + \text{“n”})]$
      Se $\Psi_P$: $P \equiv R / \varphi / B$ ou $\Psi_P$: $P \equiv R / \varphi / \{B_1,\ldots,B_n\}$ Então
         Para cada $fk(T_1 : [a_1, \ldots, a_n], T_2 : [b_1, \ldots, b_n])$ em $\varphi$ Faça
            $T = T + [T_1, T_2]$
            $J = J + [T_1.a_1 = T_2.b_1, \ldots, T_1.a_n = T_2.b_n]$
         Se $\Psi_P$: $P \equiv R / \varphi / B$ *Então* $LA \leftarrow LA + [(B, p)]$
         *Senão* $LA \leftarrow LA + [(B_1, p + \text{“1”})] + \ldots + [(B_n, p + \text{“n”})]$
      Caso 3.2: *P* tem cardinalidade MAIOR que 1
         $np \leftarrow n + \text{“\_”} + p$
         Criar $T_p[]$, $J_p[]$ e $LA_p[(K_1, \text{“ID\_”} + n + \text{“1”}), \ldots, (K_n, \text{“ID\_”} + n + \text{“n”})]$
         Seja $\varphi$ o caminho de chaves em $\Psi_P$
         Para cada $fk(T_1 : [a_1, \ldots, a_n], T_2 : [b_1, \ldots, b_n])$ em $\varphi$ Faça
            $T_p = T_p + [T_1, T_2]$
            $J_p = J_p + [T_1.a_1 = T_2.b_1, \ldots, T_1.a_n = T_2.b_n]$
         Se $\Psi_P$: $P \equiv R / \varphi / B$ *Então* $LA_p \leftarrow LA_p + [(B, p)]$
         Senão $LA_p \leftarrow LA_p + [(B_1, p + \text{“1”})] + \ldots + [(B_n, p + \text{“n”})]$
         Se $\Psi_C$: $C \equiv R[A_1, \ldots, A_n]$ Então $TV_3$ ($np$, $T_p$, $J_p$, $LA_p$);
         Se $\Psi_C$: $C \equiv R[A_1, \ldots, A_n]\sigma$ Então $TV_4$ ($np$, $T_p$, $J_p$, $LA_p$, $\sigma$);
4. Para cada propriedade de objetos *O* em $V_E$ onde o *domínio(O)* = *C* Faça
   $o \leftarrow nome(O)$
   $I \leftarrow imagem(O)$
   Caso 4.1: *O* tem cardinalidade IGUAL a 1
      Se $\Psi_O$: $O \equiv R$ Então
         Seja $\Psi_I$: $I \equiv R[K_1,\ldots,K_n]$ a ACC de *I*;
         $LA \leftarrow LA + [(K_1, o + \text{“1”}), \ldots, (K_n, o + \text{“n”})]$
      Se $\Psi_O$: $O \equiv R / \varphi$ Então
         Se *tamanho*($\varphi$) = 1 Então
            *Seja* $fk(T_1 : [a_1, \ldots, a_n], T_2 : [b_1, \ldots, b_n])$ *a única chave em* $\varphi$
            $LA \leftarrow LA + [(a_1, \text{“ID\_”} + I + \text{“1”})] + \ldots + [(a_n, \text{“ID\_”} + I + \text{“n”})]$
         Se*não* Para cada $fk(T_1 : [a_1, \ldots, a_n], T_2 : [b_1, \ldots, b_n])$ em $\varphi$ Faça
            $T = T + [T_1, T_2]$
            $J = J + [T_1.a_1 = T_2.b_1, \ldots, T_1.a_n = T_2.b_n]$
            Se *fk* é a última chave em $\varphi$ Então
               $LA \leftarrow LA + [(a_1, \text{“ID\_”} + I + \text{“1”})] + \ldots + [(a_n, \text{“ID\_”} + I + \text{“n”})]$
      Caso 4.2: *O* tem cardinalidade MAIOR que 1, onde $\Psi_O$: $O \equiv R / \varphi$ é a ACO de *O*;
         $no \leftarrow n + \text{“\_”} + o$
         Criar $T_o[]$, $J_o[]$ e $LA_o[(K_1, \text{“ID\_”} + n + \text{“1”}), \ldots, (K_n, \text{“ID\_”} + n + \text{“n”})]$
         Para cada $fk(T_1 : [a_1, \ldots, a_n], T_2 : [b_1, \ldots, b_n])$ em $\varphi$ Faça
            $T_p = T_p + [T_1, T_2]$
            $J_p = J_p + [T_1.a_1 = T_2.b_1, \ldots, T_1.a_n = T_2.b_n]$
            Se *fk* é a última chave em $\varphi$ Então
               $LA_o \leftarrow LA_o + [(a_1, \text{“ID\_”} + I + \text{“1”})] + \ldots + [(a_n, \text{“ID\_”} + I + \text{“n”})]$
         Se $\Psi_C$: $C \equiv R[A_1, \ldots, A_n]$ Então $TV_3$ ($no$, $T_o$, $J_o$, $LA_o$);
         Se $\Psi_C$: $C \equiv R[A_1, \ldots, A_n]\sigma$ Então $TV_4$ ($no$, $T_o$, $J_o$, $LA_o$, $\sigma$);
5. Se *J* está vazia Então
      Se $\Psi_C$: $C \equiv R[A_1, \ldots, A_n]$ Então
         $TV_1$ ($N$, $LA$, $T$);
      Se $\Psi_C$: $C \equiv R[A_1, \ldots, A_n]\sigma$ Então
         $TV_2$ ($N$, $LA$, $T$, $\sigma$);
   Senão
      Se $\Psi_C$: $C \equiv R[A_1, \ldots, A_n]$ Então
         $TV_3$ ($N$, $LA$, $T$, $J$);
      Se $\Psi_C$: $C \equiv R[A_1, \ldots, A_n]\sigma$ Então
         $TV_4$ ($N$, $LA$, $T$, $J$, $\sigma$);

Finalizando o exemplo, no passo 5, último passo, a visão *Document* será criada. Chegamos neste passo com a lista de joins vazia (*J*=[]) e a ACC da classe *Document* possui filtro de seleção (*foaf:Document* ≡ *Papers*[*paperID*], ***σ [papers.Year > 2002]***). Portanto, o template $TV_2$ da Tabela 6.5 é aplicado e o resultado é apresentado na linha $V_4$ da Tabela 6.6.

**Tabela 6.5 - Templates para geração das cláusulas SQL**

| | |
|---|---|
| $TV_1$ | *CREATE OR REPLACE VIEW* **N** *AS*<br>*SELECT* **LA**[1], **LA**[2], … , **LA**[n]<br>*FROM* **T**; |
| $TV_2$ | *CREATE OR REPLACE VIEW* **N** *AS*<br>*SELECT* **LA**[1], **LA**[2], … , **LA**[n]<br>*FROM* **T**<br>*WHERE* $\sigma$; |
| $TV_3$ | *CREATE OR REPLACE VIEW* **N** *AS*<br>*SELECT* **LA**[1], **LA**[2], … , **LA**[n]<br>*FROM* **T**[1], **T**[2], … , **T**[m]<br>*WHERE* **J**[1] *AND* **J**[2] *AND* … *AND* **J**[o]; |
| $TV_4$ | *CREATE OR REPLACE VIEW* **N** *AS*<br>*SELECT* **LA**[1], **LA**[2], … , **LA**[n]<br>*FROM* **T**[1], **T**[2], … , **T**[m]<br>*WHERE* **J**[1] *AND* **J**[2] *AND* … *AND* **J**[o] *AND* $\sigma$; |

A Tabela 6.6 mostra a saída do algoritmo da Tabela 6.4 para o nosso estudo de caso. Nela estão listadas as cláusulas SQLs geradas para definir cada visão de *EV*.

**Tabela 6.6 - Visões SQL geradas pelo Algoritmo da Tabela 6.4**

| | |
|---|---|
| $V_1$ | *CREATE OR REPLACE VIEW* **Person** *AS*<br>*SELECT* persons.PerID *AS* **ID**, persons.FirstName *AS* **name**, persons.Email *AS* **mbox**<br>*FROM* persons |
| $V_2$ | *CREATE OR REPLACE VIEW* **Person_Has_Affiliation** *AS*<br>*SELECT* persons.PerID *AS* **ID_Person**, rel_person_organization.OrganizationID *AS* **ID_Organization**<br>*FROM* persons, rel_person_organization, organizations<br>*WHERE* persons.PerID = rel_person_organization.PersonID<br>*AND* rel_person_organization.OrganizationID = organizations.OrgID |
| $V_3$ | *CREATE OR REPLACE VIEW* **Person_Research_Interests** *AS*<br>*SELECT* persons.PerID AS **ID_Person**, rel_paper_topic.TopicID *AS* **ID_Concept**<br>*FROM* persons, rel_person_paper, papers, rel_paper_topic, topics<br>*WHERE* persons.PerID = rel_person_paper.PersonID *AND* rel_person_paper.PaperID = papers.PaperID<br>*AND* papers.PaperID = rel_paper_topic.PaperID *AND* rel_paper_topic.TopicID = topics.TopicID |
| $V_4$ | *CREATE OR REPLACE VIEW* **Document** *AS*<br>*SELECT* papers.PaperID *AS* **ID**, Title *AS* **title**, Abstract *AS* **abstract**, Year *AS* **date**, Conference *AS* **ID_Conference**<br>*FROM* papers<br>*WHERE* papers.Year > 2002 |
| $V_5$ | *CREATE OR REPLACE VIEW* **Document_Subject** *AS*<br>*SELECT* papers.PaperID *AS* **ID_Document**, rel_paper_topic.TopicID *AS* **ID_Concept**<br>*FROM* papers, rel_paper_topic, topics<br>*WHERE* papers.PaperID = rel_paper_topic.PaperID *AND* rel_paper_topic.TopicID = topics.TopicID<br>*AND* papers.Year > 2002 |
| $V_6$ | *CREATE OR REPLACE VIEW* **Document_Creator** *AS*<br>*SELECT* papers.PaperID *AS* **ID_Document**, rel_person_paper.PersonID *AS* **ID_Person**<br>*FROM* papers, rel_person_paper, persons<br>*WHERE* papers.PaperID = rel_person_paper.PaperID *AND* rel_person_paper.PersonID = persons.PerID<br>*AND* papers.Year > 2002 |
| $V_7$ | *CREATE OR REPLACE VIEW* **Organization** *AS* |

| | |
|---|---|
| | *SELECT* OrgID *AS* **ID**, Name *AS* **label**, Homepage *AS* **homepage**, Address *AS* **ID_PostalAdress1**, Location *AS* **ID_PostalAdress2**, Postcode *AS* **ID_PostalAdress3**, Country *AS* **ID_PostalAdress4**,<br>*FROM* organizations |
| $V_8$ | *CREATE OR REPLACE VIEW* **PostalAddress** *AS*<br>*SELECT* Address *AS* **ID1**, Location *AS* **ID2**, Postcode *AS* **ID3**, Country *AS* **ID4**, Address *AS* **street**, Location *AS* **locality**, Postcode *AS* **pcode**, Country *AS* **country**<br>*FROM* organizations |
| $V_9$ | *CREATE OR REPLACE VIEW* conf_Conference_view *AS*<br>*SELECT* ConfID *AS* **ID**, Name *AS* **label**, Date *AS* **date**<br>*FROM* conferences |
| $V_{10}$ | *CREATE OR REPLACE VIEW* Concept *AS*<br>*SELECT* TopicID *AS* **ID**, TopicName *AS* **prefLabel**, ParentID as **ID_Concept** *FROM* topics |

## 6.4 Publicação da *OE* em um SPARQL endpoint

O mapeamento R2RML foi criado no Passo 2 do processo com o objetivo de publicar os dados relacionais mapeados. Logo, um processador R2RML receberá como entrada este mapeamento e criará um serviço para atender consultas SPARQL sobre a ontologia exportada gerada no Passo 1 do processo. Este serviço utiliza o mapeamento para traduzir consultas SPARQL em consultas SQL sobre o esquema de visões relacionais criado no Passo 2 do processo. Por fim, o resultado desta consulta é convertido em um grafo RDF que é retornado como resposta à requisição.

## 6.5 Conclusões

Neste capítulo, foram apresentadas as camadas da arquitetura proposta e os passos do processo de publicação de dados relacionais no modelo RDF, onde foram detalhados os algoritmos para: construção dos esquemas das visões relacionais, geração dos mapeamentos R2RML e escrita do SQL de criação das visões. Para facilitar o entendimento dos algoritmos, utilizamos as assertivas de correspondência do nosso estudo de caso como entrada e mostramos a saída gerada.

# 7 RBA – R2RML BY ASSERTIONS

Neste capítulo mostramos a ferramenta RBA (Neto et al., 2013) que simplifica as tarefas de geração dos mapeamentos R2RML e de publicação dos dados relacionais em um *SPARQL endpoint* usando mapeamentos definidos por assertivas de correspondência. A seção 7.1 apresenta os cinco componentes principais da ferramenta: a interface gráfica (GUI), o gerador da ontologia exportada (GEO), o gerador do esquema das visões (GVS), o gerador de mapeamentos R2RML (GM-R2RML) e a extensão da plataforma D2RQ para suportar mapeamentos R2RML (D2RQ-Extension). Na seção 7.2 abordamos a utilização da ferramenta no estudo de caso adotado neste trabalho. Por fim, a seção 7.3 tece as conclusões deste capítulo.

## 7.1 Principais Componentes

RBA é uma ferramenta composto por cinco componentes. Nesta seção descrevemos as funções de cada componente com as entradas e saídas geradas. Cada componente é responsável pela implementação de uma etapa ou parte de uma etapa do processo apresentado no capítulo 6. A Figura 7.1 apresenta os componentes com suas entradas e saídas. Como eles estão interligados por meio do processo, a saída de um componente é entrada para o próximo componente a ser executado. Para melhor entendimento apresentamos os componentes na ordem em que são utilizados no processo, ou seja, desde a geração das assertivas de correspondência até a criação dos mapeamentos R2RML que são a saída final do processo.

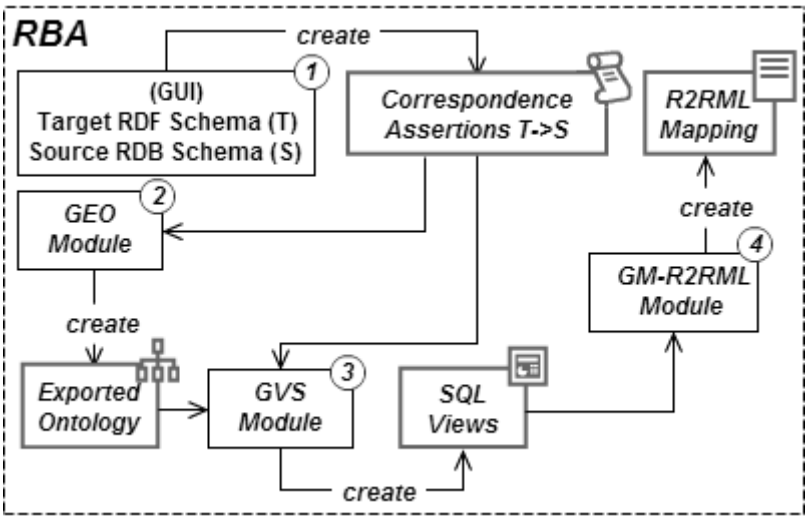


**Figura 7.1 - Principais Componentes da Ferramenta RBA**

### *7.1.1 GUI (Graphical User Interface)*

Uma sessão da ferramenta inicia com o usuário configurando os esquemas fonte e alvo, ou seja, configurando a conexão com o banco de dados fonte e escolhendo a ontologia alvo. Em seguida, o usuário pode carregar os dois esquemas em duas árvores de nós e criar as assertivas de correspondência associando nós do esquema da ontologia com nós do esquema do banco de dados. Assim, esta tarefa é feita de forma gráfica pelo uso do componente GUI.

As associações criadas especificam o mapeamento entre o esquema RDF alvo e o esquema relacional fonte e este componente facilita a geração das assertivas, uma vez que as cria a partir das correspondências gráficas que o usuário especifica. Desta forma, as entradas para este componente são o esquema relacional fonte, o esquema RDF alvo e as correspondências gráficas. A saída é o conjunto de assertivas de correspondências que será a entrada para o próximo componente.

### *7.1.2 GEO (Generate Exported Ontology)*

Este módulo é responsável pela implementação do Passo 1 do processo, ou seja, é o responsável pela geração da ontologia exportada. Sejam $\boldsymbol{O} = (\boldsymbol{V},\boldsymbol{\Sigma})$ a ontologia de domínio e $\boldsymbol{S} = (\boldsymbol{R},\boldsymbol{\Omega})$ o esquema relacional fonte. GEO recebe como entrada:

- O Conjunto de Assertivas de Correspondência ***A*** que mapeia ***V*** em ***S***.

  E gera como saída:

- A Ontologia Exportada: $\boldsymbol{E} = (\boldsymbol{V_E},\boldsymbol{\Sigma_E})$;

A tarefa do componente GEO é bem simples: ele percorre a lista ***A*** e inclui em ***E*** todas as classes e propriedades mapeadas. Assim, $\boldsymbol{V_E}$ é um subconjunto de ***V*** e $\boldsymbol{\Sigma_E}$ é o subconjunto de $\boldsymbol{\Sigma}$ que foi gerado pela *procedure* ***closedFragment***. Concluída esta tarefa, o esquema das visões pode ser gerado automaticamente pelo componente GVS a partir de ***E***.

### 7.1.3 GVS (Generate Views Schema)

Este componente implementa os algoritmos apresentados no Capítulo 5 para geração dos esquemas das visões e do SQL de cada uma delas. Os esquemas constituem uma transformação direta da Ontologia Exportada gerada pelo GEO. Os esquemas das visões podem ser implementados como visões SQL ou visões R2RML e a ferramenta ***RBA*** permite que o usuário escolha o tipo de visão a ser criada. O primeiro tipo cria as visões no banco de dados fonte e o segundo tipo cria as visões dentro do mapeamento R2RML. Na Tabela 6.4 apresentamos o algoritmo para gerar automaticamente o SQL das visões relacionais com base na ontologia exportada e nas assertivas de correspondências criadas. Estando criadas as visões, o componente GM-R2RML pode ser inicializado para gerar o mapeamento R2RML.

### 7.1.4 GM-R2RML (Generate Mapping – R2RML)

Na Tabela 6.1 mostramos o algoritmo que gera o esquema das visões *EV* juntamente com o mapeamento R2RML entre *EV* e a ontologia exportada. GM-R2RML implementa a geração deste mapeamento que é do tipo um-para-um e, é trivial, como veremos na implementação do estudo de caso apresentado na Seção 7.3. O componente aplica os *templates* da Tabela 6.2 que geram as declarações R2RML para cada caso descrito no algoritmo. Após a aplicação de um *template*, o resultado é adicionado ao *código* R2RML gerado.

### 7.1.5 D2RQ-Extension

Após a criação do mapeamento R2RML, o usuário precisa utilizá-lo para publicar os dados relacionais em um grafo RDF e fazer algumas consultas SPARQL sobre este grafo. RBA emula o *D2RQ Server* com algumas customizações. Como visto no Capítulo 3, a plataforma D2RQ possui uma linguagem declarativa própria denominada D2RM e até o momento de escrita deste trabalho não havia uma versão em produção que suportasse R2RML. Para solucionar este problema, desenvolvemos uma versão customizada do D2RQ (D2RQ-Ext) que suporta os principais termos da linguagem R2RML e está embutindo como um componente dentro do RBA. A limitação no momento é que D2RQ-Ext não suporta “Visões R2RML”, porém suporta o uso de “Visões Relacionais”, o que viabiliza a publicação dos mapeamentos quando o usuário escolhe este tipo de visão no componente GVS. Veremos esta publicação na próxima seção com o resultado das consultas SPARQL executadas sobre nosso estudo de caso.

## 7.2 Implementação do Estudo de Caso

Abordamos nesta seção a aplicação da ferramenta ao estudo de caso introduzido no capítulo 4. Desta forma, apresentamos passo a passo a interação do usuário com a ferramenta desde a configuração para acesso à base de dados fonte até a publicação dos dados RDF gerados pelo mapeamento.

### *7.2.1 Configuração Inicial para Criação de um Mapeamento*

O usuário inicia uma nova sessão na ferramenta utilizando o componente GUI para configurar o acesso à base relacional e escolher a ontologia exportada. No nosso exemplo utilizamos o banco de dados MySQL e a base de dados da conferência ISWC (apresentada na Figura 4.1) que vem como exemplo da plataforma D2RQ. A Figura 7.2 mostra a configuração de acesso à base que corresponde aos parâmetros de conexão com o banco.

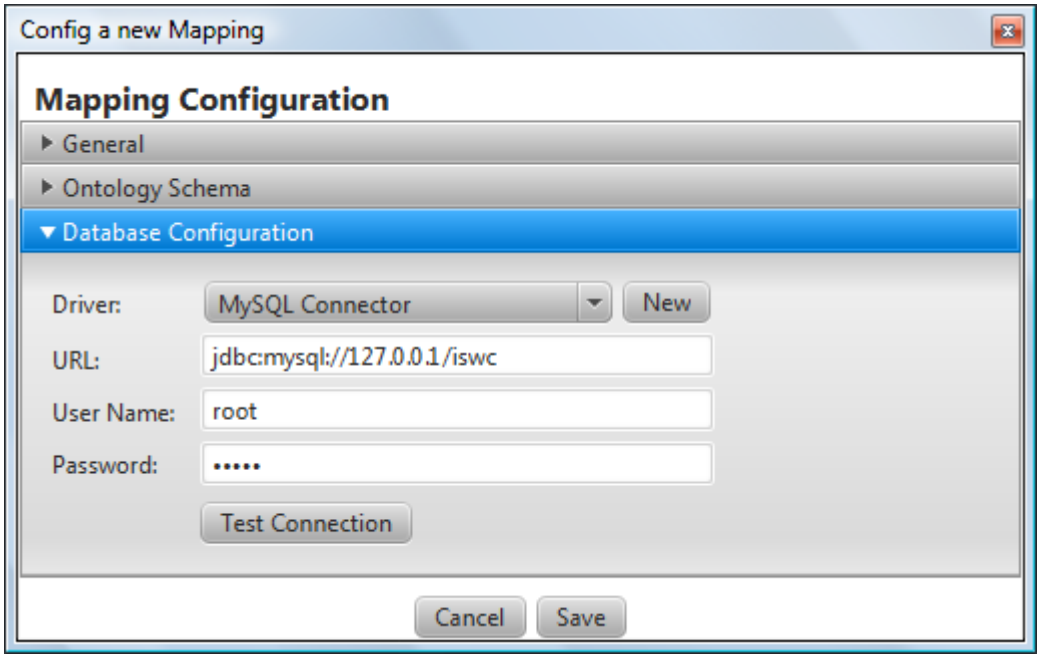


**Figura 7.2 - Configuração para acesso à base *ISWC***

Criamos para este exemplo uma ontologia de domínio denominada *CONF_OWL* apresentada na Figura 4.2. A descrição desta ontologia fica em um arquivo local na máquina

do usuário, porém a ferramenta também permite que seja escolhida uma URL como pode ser visto na Figura 7.3.

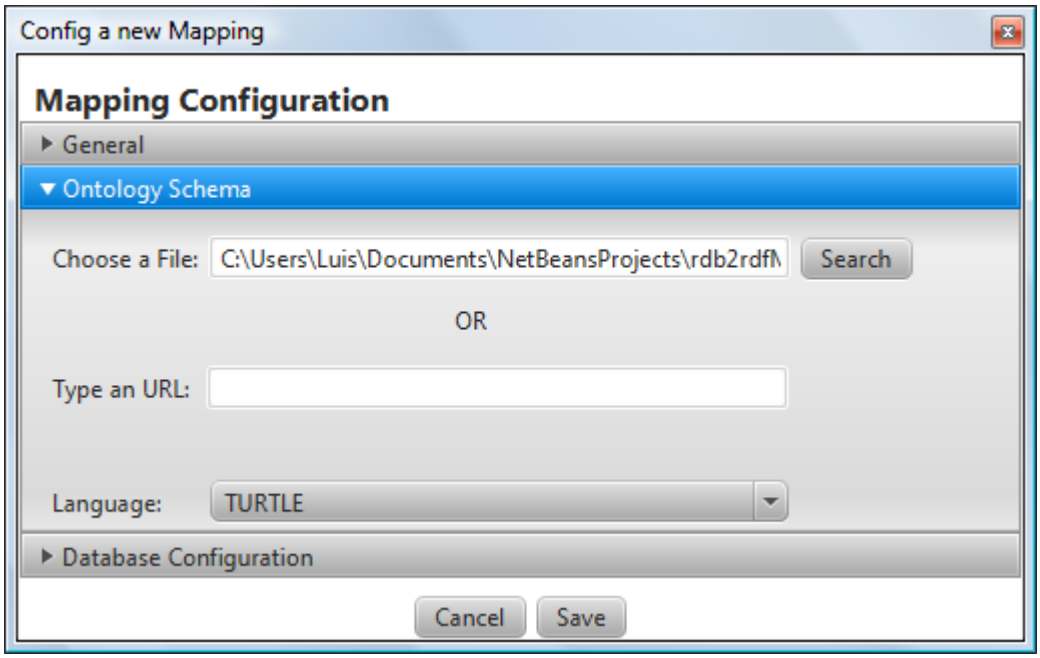


**Figura 7.3 - Configuração da Ontologia de Domínio**

Além de escolher a ontologia é obrigatório que o usuário escolha a linguagem na qual esta ontologia está descrita. Neste exemplo *CONF_OWL* foi escrita usando a linguagem *TURTLE* apresentada no Capítulo 2.

Com a conclusão desta etapa inicial, o componente GUI apresenta na *Grid* à esquerda da Tela Principal uma nova linha com os nomes da Ontologia Alvo (*RDF Target*) e do Banco Relacional Fonte (*RDB Source*). Ao executar um duplo clique nesta linha, as árvores dos dois esquemas são carregadas na aba "*Build Assertions*" da tela principal. À esquerda fica a árvore do esquema alvo (*Target Schema T*) e à direita fica a árvore do esquema fonte (*Source Schema S*). As demais abas estão inicialmente desabilitadas. Elas são habilitadas à medida que vamos executando na ferramenta os passos do processo descrito no Capítulo 6. O resultado desta configuração inicial está na Figura 7.4.

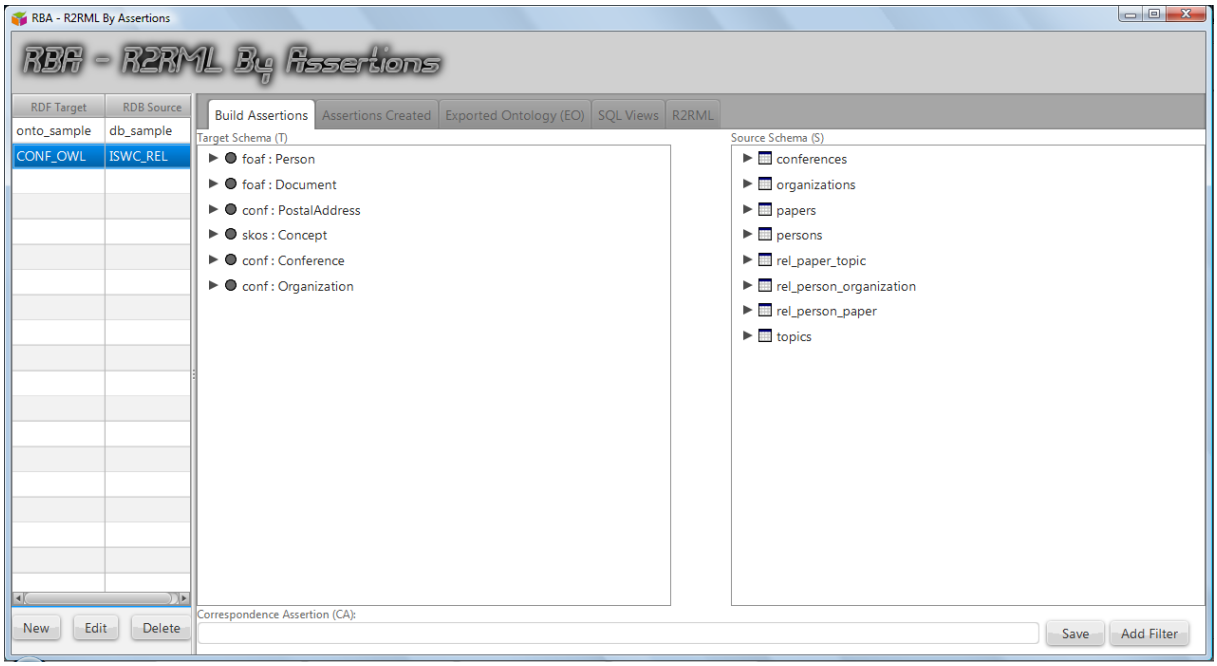


**Figura 7.4 - Tela Principal para Criação das ACs**

### *7.2.2 Criação das Assertivas de Correspondência*

Veremos nesta seção como RBA torna fácil a tarefa de criar as assertivas de correspondência. Basicamente, o usuário deverá selecionar elementos das duas árvores que sejam correspondentes e a ferramenta cria a declaração da assertiva na caixa de texto que fica na parte de baixo da aba "*Build Assertions*". Em seguida, o usuário clica em "Salvar" e a assertiva é incluída na lista da aba "*Assertions Created* " que é habilitada após a criação da primeira assertiva.

A Figura 7.5 mostra a criação da assertiva de correspondência de objeto para a propriedade *dc:creator* que relaciona indivíduos da classe *foaf:Document* com indivíduos da classe *foaf:Person*. Considere que o usuário já criou as assertivas para essas duas classes criando correspondências com as Tabelas *Papers* e *Persons* respectivamente. Primeiro o usuário navega na árvore *T* até localizar a propriedade de objetos *dc:creator* e selecioná-la. Em seguida ele navega na árvore *S* a partir da tabela *Papers* passando por um caminho com duas chaves estrangeiras para chegar à tabela *Persons* (*fk1_rel_person_paper2papers* e *fk0_rel_person_paper2persons*). Por fim, ele seleciona a opção Salvar (*Save*) e a nova assertiva é incluída na lista conforme a Figura 7.6.

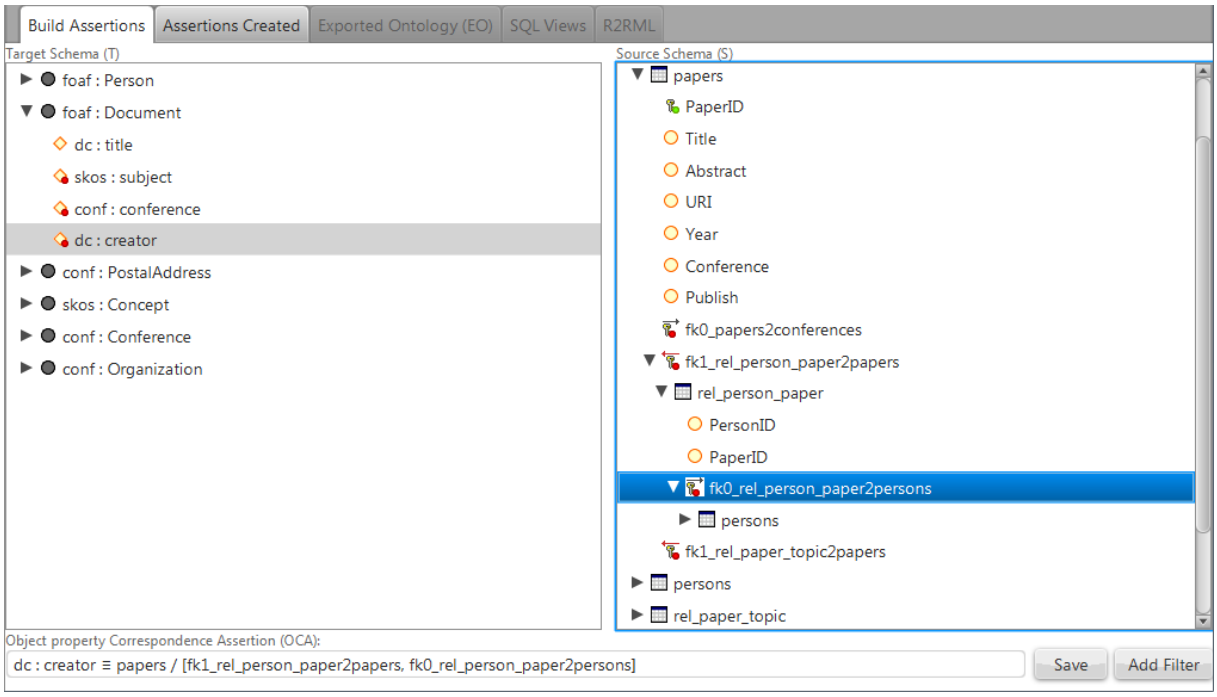


**Figura 7.5 - Criação de uma ACO para a propriedade *dc:creator***

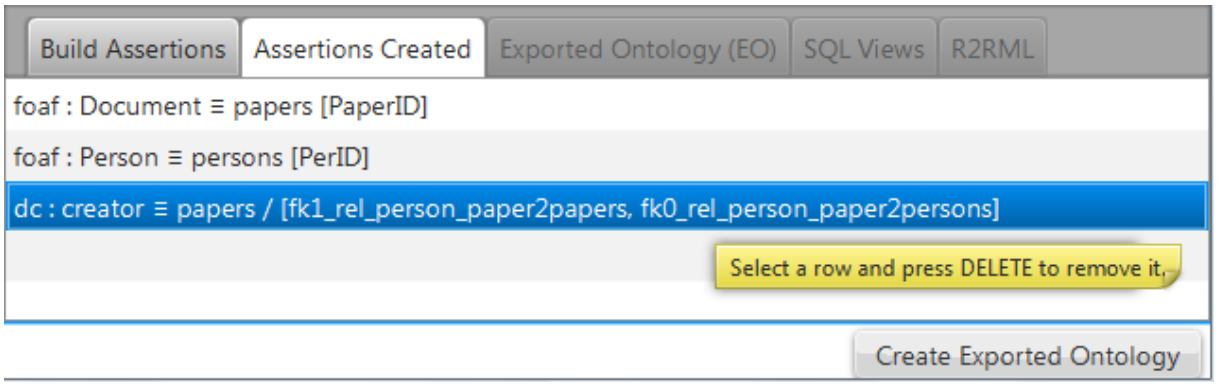


**Figura 7.6 - Lista com as assertivas criadas até o momento**

Opcionalmente, é possível criar um filtro de seleção em uma assertiva de correspondência de classe. Na Figura 7.5 podemos observar esta opção de Adicionar um Filtro (*Add Filter*). Quando o usuário seleciona esta opção o componente GUI apresenta uma nova tela para construção do filtro de seleção mostrada na Figura 7.7.

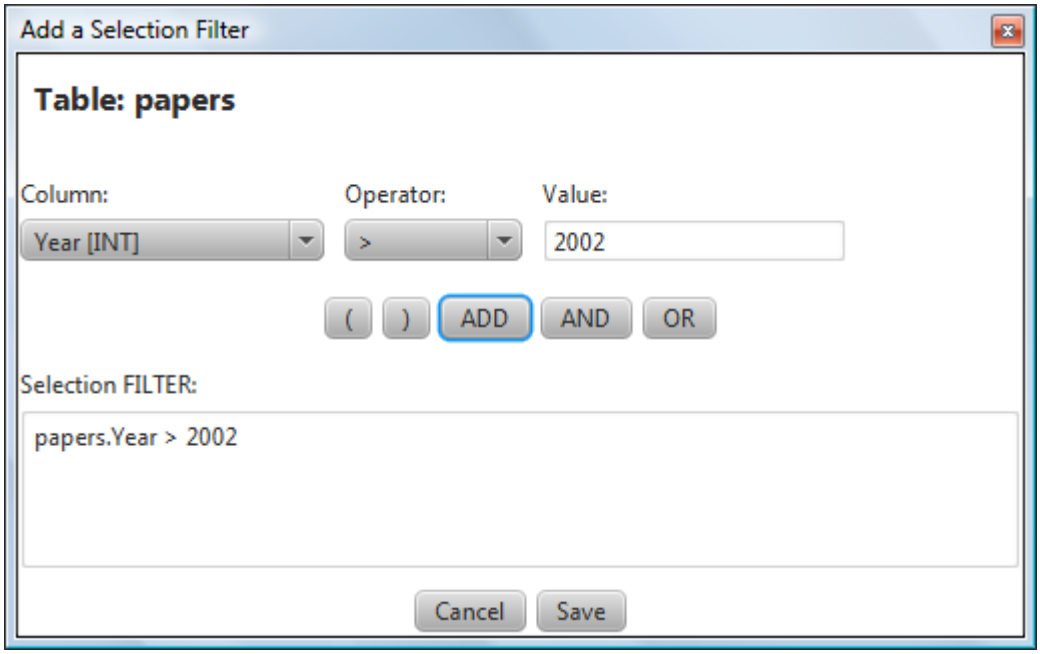


**Figura 7.7 - Exemplo de criação de um filtro de seleção**

No nosso exemplo criamos um filtro para selecionar somente os *papers* publicados após 2002. Ao selecionarmos a opção Salvar (*Save*) o filtro é incluído na assertiva de correspondência da classe *foaf:Document* gerando o seguinte resultado:

*foaf : Document* ≡ *papers* [*PaperID*], FILTER [*papers.Year* > 2002]

A Figura 7.8 mostra a lista de assertivas atualizada com a inclusão do filtro. Esta atualização ocorre após o usuário selecionar a opção "Salvar" novamente.

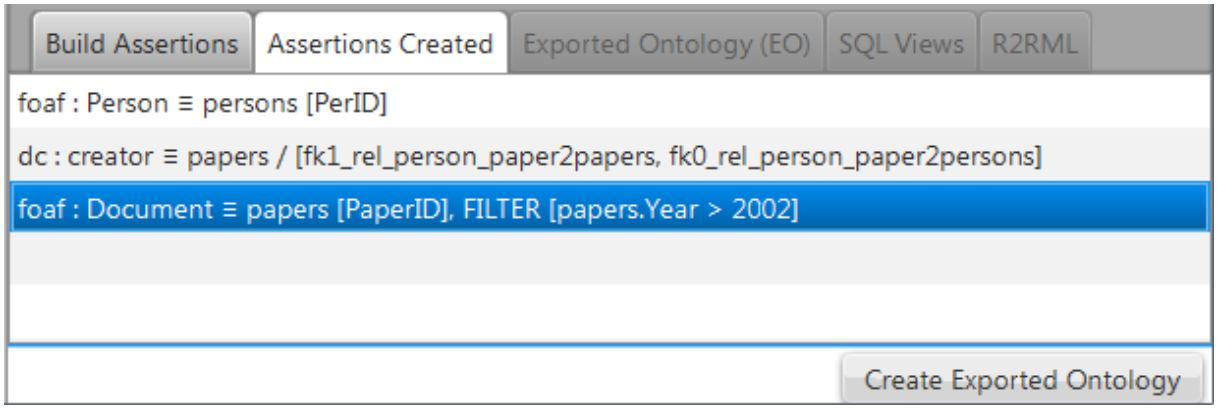


**Figura 7.8 - Lista de assertivas atualizada**

Desta forma, vemos que é possível criar facilmente todas as assertivas listadas na Tabela 5.3 do nosso exemplo. A Figura 7.9 mostra a lista destas assertivas criadas em RBA.

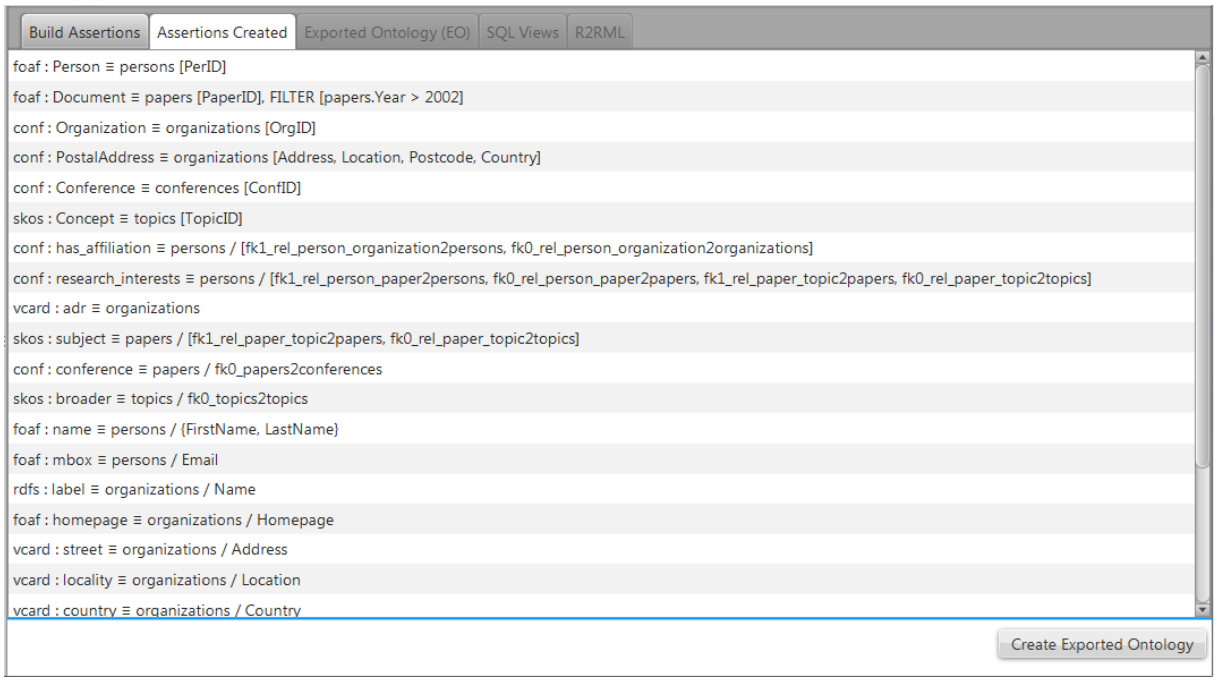


**Figura 7.9 - Lista de assertivas do estudo de caso**

### *7.2.3 Geração da Ontologia Exportada*

Criadas as assertivas, a geração da ontologia exportada ocorre de forma automática quando o usuário seleciona esta opção na lista de assertivas criadas como pode ser visto na Figura 7.9 (*Create Exported Ontology*).

O componente GEO irá percorrer a lista de assertivas e criar uma ontologia que contenha somente os termos mapeados. Esta ontologia é então exibida na forma de uma árvore na aba "*Exported Ontology*". A Ontologia exportada do nosso exemplo é mostrada na Figura 7.10 contendo os elementos que foram mapeados, ou seja, seis classes com suas respectivas propriedades que possuem assertivas associadas.

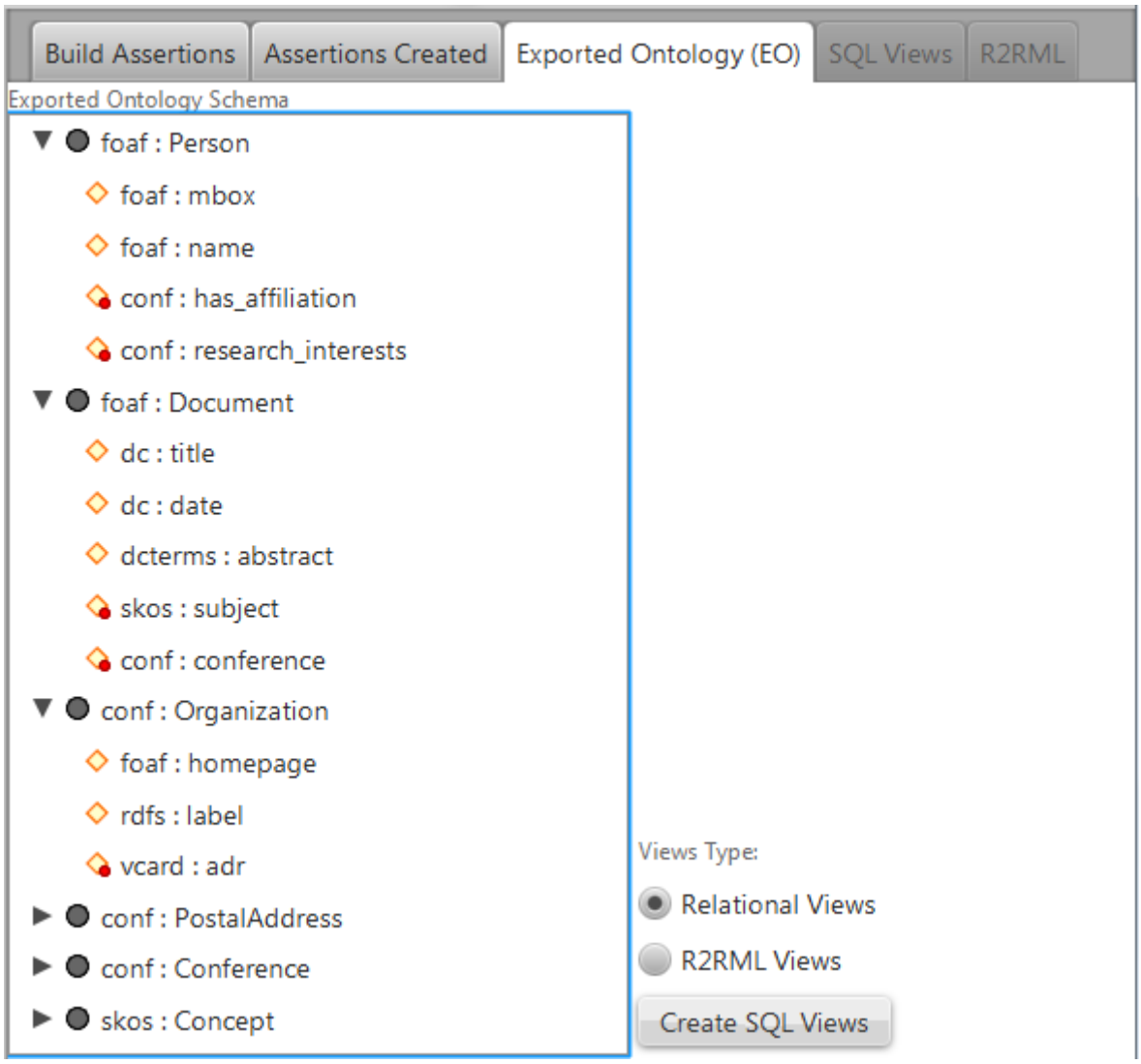


**Figura 7.10 - Ontologia Exportada a partir das ACs do estudo de caso**

Na aba "*Exported Ontology*" o usuário tem a opção de selecionar qual tipo de visão ele deseja criar: *Relational Views* ou *R2RML Views*. No nosso exemplo criamos as visões no banco (*Relational Views*) no lugar de criá-las dentro do próprio mapeamento R2RML (*R2RML Views*). Desta forma podemos publicar nossos dados no D2RQ-Ext, pois este ainda não suporta visões definidas dentro do próprio mapeamento.

### *7.2.4 Geração das Visões SQL*

A partir da ontologia exportada e das assertivas, o componente GVS executa os passos do algoritmo 6.4 gerando as visões apresentadas na Tabela 7.1. O algoritmo criou um

total de nove visões, onde as visões 1, 4, 6, 7, 8 e 9 correspondem respectivamente às classes *foaf:Person*, *foaf:Document*, *conf:Organization*, *conf:PostalAddress*, *conf:Conference* e *skos:Concept*. As outras três visões são relacionadas a propriedades de objeto e são necessárias porque estas propriedades possuem cardinalidade máxima maior que 1 na definição da ontologia exportada. Portanto, as visões 2, 3 e 5 correspondem respectivamente às propriedades *conf:hasAffiliation*, *conf:researchInterests* e *skos:subject*.

**Tabela 7.1 - Visões Relacionais geradas após a execução do Algoritmo da Tabela 6.4**

| | |
|---|---|
| 1 | **CREATE OR REPLACE VIEW** foaf_Person_view **AS**<br>**SELECT** PerID **AS** ID, Email **AS** foaf_mbox, FirstName **AS** foaf_name, LastName **AS** foaf_name2<br>**FROM** persons |
| 2 | **CREATE OR REPLACE VIEW** foaf_Person_conf_has_affiliation_view **AS**<br>**SELECT** persons.PerID **AS** ID_foaf_Person, rel_person_organization.OrganizationID **AS** conf_has_affiliation<br>**FROM** persons, rel_person_organization, organizations<br>**WHERE** persons.PerID = rel_person_organization.PersonID<br>**AND** rel_person_organization.OrganizationID = organizations.OrgID |
| 3 | **CREATE OR REPLACE VIEW** foaf_Person_conf_research_interests_view **AS**<br>**SELECT** persons.PerID **AS** ID_foaf_Person, rel_paper_topic.TopicID **AS** conf_research_interests<br>**FROM** persons, rel_person_paper, papers, rel_paper_topic, topics<br>**WHERE** persons.PerID = rel_person_paper.PersonID<br>**AND** rel_person_paper.PaperID = papers.PaperID<br>**AND** papers.PaperID = rel_paper_topic.PaperID<br>**AND** rel_paper_topic.TopicID = topics.TopicID |
| 4 | **CREATE OR REPLACE VIEW** foaf_Document_view **AS**<br>SELECT PaperID **AS** ID, Title **AS** dc_title, Year **AS** dc_date, Abstract **AS** dcterms_abstract, Conference **AS** conf_conference<br>**FROM** papers<br>**WHERE** papers.Year > 2002 |
| 5 | **CREATE OR REPLACE VIEW** foaf_Document_skos_subject_view **AS**<br>SELECT papers.PaperID **AS** ID_foaf_Document, rel_paper_topic.TopicID **AS** skos_subject<br>**FROM** papers, rel_paper_topic, topics<br>**WHERE** papers.PaperID = rel_paper_topic.PaperID<br>**AND** rel_paper_topic.TopicID = topics.TopicID<br>**AND** papers.Year > 2002 |
| 6 | **CREATE OR REPLACE VIEW** conf_Organization_view **AS**<br>**SELECT** OrgID **AS** ID, Homepage **AS** foaf_homepage, Name **AS** rdfs_label, Address **AS** vcard_ADR, Location **AS** vcard_ADR2, Postcode **AS** vcard_ADR3, Country **AS** vcard_ADR4<br>**FROM** organizations |
| 7 | **CREATE OR REPLACE VIEW** conf_PostalAddress_view **AS**<br>**SELECT** Address **AS** ID, Location **AS** ID2, Postcode **AS** ID3, Country **AS** ID4, Country **AS** vcard_country, Address **AS** vcard_street, Location **AS** vcard_locality, Postcode **AS** vcard_pcode<br>**FROM** organizations |
| 8 | **CREATE OR REPLACE VIEW** conf_Conference_view **AS**<br>**SELECT** ConfID **AS** ID, Date **AS** dc_date, Location **AS** conf_location, Name **AS** rdfs_label<br>**FROM** conferences |
| 9 | **CREATE OR REPLACE VIEW** skos_Concept_view **AS**<br>**SELECT** TopicID **AS** ID, TopicName **AS** skos_prefLabel, ParentID **AS** skos_broader<br>**FROM** topics |

### *7.2.5 Geração do Mapeamento R2RML*

Estando criadas as visões relacionais, o usuário então seleciona a opção de criar o mapeamento R2RML (*Create R2RML*) como pode ser visto na Figura 7.11. Neste passo, o componente GM-R2RML executa o algoritmo gerando o mapeamento mostrado na Tabela 7.2 que é a saída final do nosso processo.

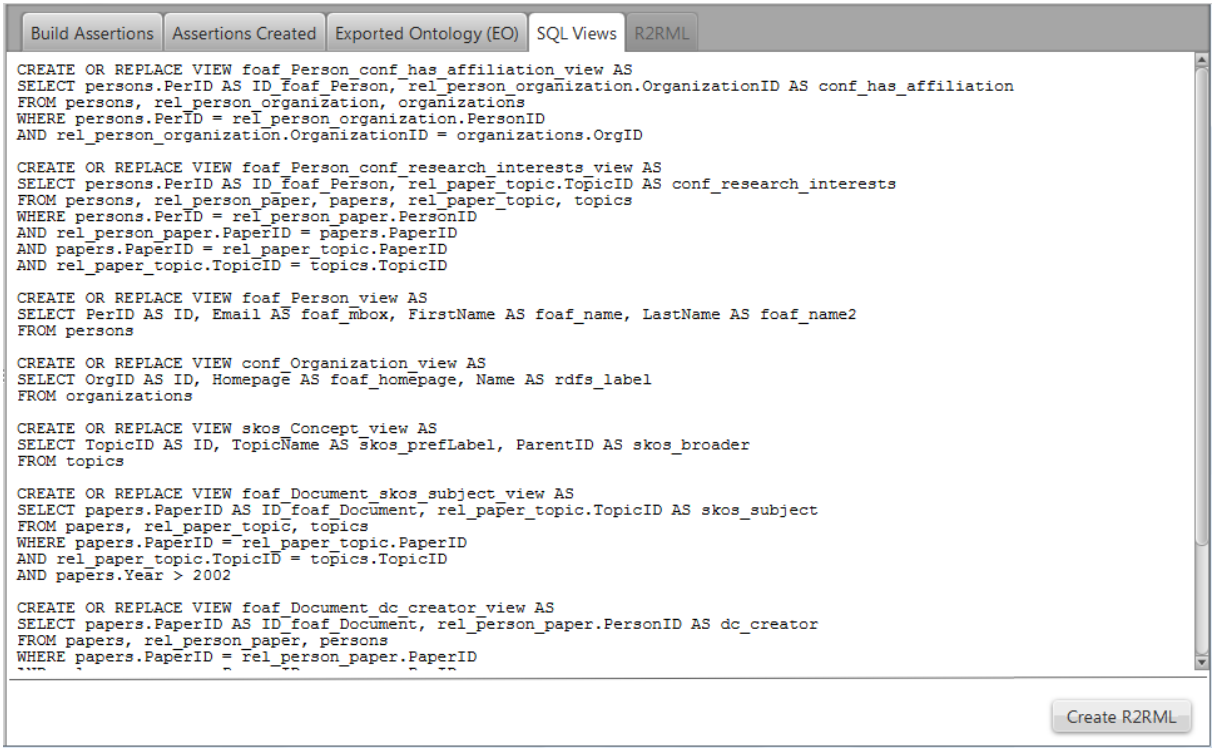


**Figura 7.11 - SQL das visões relacionais criadas**

A Linha 1 da Tabela 7.2 referencia todos os vocabulários utilizados no mapeamento com seus respectivos prefixos. A Linha 2 cria o mapeamento de triplas da classe *foaf:Person*, note que a tabela lógica deste mapeamento é a visão relacional *foaf_Person_view* criada no passo anterior do processo.

Na Linha 3 temos o mapeamento de triplas da propriedade de objeto *conf:has_affiliation*. A URI do sujeito é definida para ser a URI de *foaf:Person* que é o domínio da propriedade. O mapeamento de objetos referencia *TriplesMapOrganization* que é o mapeamento de triplas da classe *conf:Organization* (imagem de *conf:has_affiliation*). A criação de um mapeamento de triplas específico para *conf:has_ affiliation* é necessária devido a propriedade ter cardinalidade máxima maior que 1. Caso contrário, o mapeamento desta propriedade teria sido definido dentro do próprio mapeamento de triplas de *foaf:Person* na forma de um simples mapeamento de objetos.

Pelo mesmo motivo o algoritmo cria os mapeamentos de triplas das propriedades de objetos *research_interests* (Linha 4) e *skos_subject* (Linha 6). A Linha 5 declara o mapeamento de triplas da classe *foaf:Document*. Neste caso, temos um mapeamento de objetos para a propriedade de objetos *conf:conference* cuja cardinalidade máxima é igual a 1.

Por fim, as Linhas 7, 8, 9 e 10 apresentam os mapeamentos de triplas das classes *conf:Organization*, *conf:PostalAddress*, *conf:Conference* e *skos:Concept*, respectivamente.

**Tabela 7.2 - Mapeamento R2RML gerado pelo componente GM-R2RML**

| | |
|---|---|
| 1 | `@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .`<br>`@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .`<br>`@prefix skos: <http://www.w3.org/2004/02/skos/core#> .`<br>`@prefix dcterms: <http://purl.org/dc/terms/> .`<br>`@prefix dc: <http://purl.org/dc/elements/1.1/> .`<br>`@prefix conf: <http://ufc.br/rdb2rdfmb/conf/> .`<br>`@prefix foaf: <http://xmlns.com/foaf/0.1/> .`<br>`@prefix rr: <http://www.w3.org/ns/r2rml#> .`<br>`@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .`<br>`@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .`<br>`@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .` |
| 2 | `<#TriplesMapPerson>`<br>`  rr:logicalTable [ rr:tableName "foaf_Person_view" ];`<br>`  rr:subjectMap [`<br>`  rr:template "Person/{ID}/";`<br>`  rr:class foaf:Person;`<br>`  ];`<br>`  rr:predicateObjectMap [`<br>`  rr:predicate rdfs:label;`<br>`  rr:objectMap [ rr:template "Person #{ID}"; ];`<br>`  ];`<br>`  rr:predicateObjectMap [`<br>`  rr:predicate foaf:mbox;`<br>`  rr:objectMap [ rr:column "foaf_mbox" ];`<br>`  ];`<br>`  rr:predicateObjectMap [`<br>`  rr:predicate foaf:name;`<br>`  rr:objectMap [`<br>`  rr:template "{foaf_name} {foaf_name2}";`<br>`  rr:termType rr:Literal;`<br>`  ];`<br>`  ]. |
| 3 | `<#TriplesMapPerson_has_affiliation>`<br>`  rr:logicalTable [ rr:tableName "foaf_Person_conf_has_affiliation_view" ];`<br>`  rr:subjectMap [`<br>`  rr:template "Person/{ID_foaf_Person}/";`<br>`  rr:class foaf:Person;`<br>`  ];`<br>`  rr:predicateObjectMap [`<br>`  rr:predicate rdfs:label;`<br>`  rr:objectMap [ rr:template "Person #{ID_foaf_Person}"; ];`<br>`  ];`<br>`  rr:predicateObjectMap [`<br>`  rr:predicate conf:has_affiliation;`<br>`  rr:objectMap [`<br>`  rr:parentTriplesMap <#TriplesMapOrganization>;`<br>`  rr:joinCondition [`<br>`  rr:child "conf_has_affiliation";`<br>`  rr:parent "ID";`<br>`  ];`<br>`  ];`<br>`  ]. |

| | |
|---|---|
| 4 | <#TriplesMapPerson_research_interests><br>**rr:logicalTable** [<br>**rr:tableName** "foaf_Person_conf_research_interests_view"<br>];<br>**rr:subjectMap** [<br>**rr:template** "Person/{ID_foaf_Person}/";<br>**rr:class** foaf:Person;<br>];<br>**rr:predicateObjectMap** [<br>**rr:predicate** rdfs:label;<br>**rr:objectMap** [ **rr:template** "Person #{ID_foaf_Person}"; ];<br>];<br>**rr:predicateObjectMap** [<br>**rr:predicate** conf:research_interests;<br>**rr:objectMap** [<br>**rr:parentTriplesMap** <#TriplesMapConcept>;<br>**rr:joinCondition** [<br>**rr:child** "conf_research_interests";<br>**rr:parent** "ID";<br>];<br>];<br>]. |
| 5 | <#TriplesMapDocument><br>**rr:logicalTable** [ **rr:tableName** "foaf_Document_view" ];<br>**rr:subjectMap** [<br>**rr:template** "Document/{ID}/";<br>**rr:class** foaf:Document;<br>];<br>**rr:predicateObjectMap** [<br>**rr:predicate** rdfs:label;<br>**rr:objectMap** [ **rr:template** "Document #{ID}"; ];<br>];<br>**rr:predicateObjectMap** [<br>**rr:predicate** dc:title;<br>**rr:objectMap** [ **rr:column** "dc_title" ];<br>];<br>**rr:predicateObjectMap** [<br>**rr:predicate** dc:date;<br>**rr:objectMap** [ **rr:column** "dc_date" ];<br>];<br>**rr:predicateObjectMap** [<br>**rr:predicate** dcterms:abstract;<br>**rr:objectMap** [ **rr:column** "dcterms_abstract" ];<br>];<br>**rr:predicateObjectMap** [<br>**rr:predicate** conf:conference;<br>**rr:objectMap** [<br>**rr:parentTriplesMap** <#TriplesMapConference>;<br>**rr:joinCondition** [<br>**rr:child** "conf_conference";<br>**rr:parent** "ID";<br>];<br>];<br>]. |
| 6 | <#TriplesMapDocument_subject><br>**rr:logicalTable** [ **rr:tableName** "foaf_Document_skos_subject_view" ];<br>**rr:subjectMap** [<br>**rr:template** "Document/{ID_foaf_Document}/";<br>**rr:class** foaf:Document;<br>];<br>**rr:predicateObjectMap** [<br>**rr:predicate** rdfs:label;<br>**rr:objectMap** [ **rr:template** "Document #{ID_foaf_Document}"; ];<br>];<br>**rr:predicateObjectMap** [ |

```
        rr:predicate skos:subject;
        rr:objectMap [
            rr:parentTriplesMap <#TriplesMapConcept>;
            rr:joinCondition [
                rr:child "skos_subject";
                rr:parent "ID";
            ];
        ];
    ].
```

7

```
<#TriplesMapOrganization>
    rr:logicalTable [ rr:tableName "conf_Organization_view" ];
    rr:subjectMap [
        rr:template "Organization/{ID}/";
        rr:class conf:Organization;
    ];
    rr:predicateObjectMap [
        rr:predicate rdfs:label;
        rr:objectMap [ rr:template "Organization #{ID}"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate foaf:homepage;
        rr:objectMap [ rr:column "foaf_homepage" ];
    ];
    rr:predicateObjectMap [
        rr:predicate rdfs:label;
        rr:objectMap [ rr:column "rdfs_label" ];
    ];
    rr:predicateObjectMap [
        rr:predicate vcard:ADR;
        rr:objectMap [
            rr:parentTriplesMap <#TriplesMapPostalAddress>;
            rr:joinCondition [
                rr:child "vcard_ADR";
                rr:parent "ID";
            ];
            rr:joinCondition [
                rr:child "vcard_ADR2";
                rr:parent "ID2";
            ];
            rr:joinCondition [
                rr:child "vcard_ADR3";
                rr:parent "ID3";
            ];
            rr:joinCondition [
                rr:child "vcard_ADR4";
                rr:parent "ID4";
            ];
        ];
    ].
```

8

```
<#TriplesMapPostalAddress>
    rr:logicalTable [ rr:tableName "conf_PostalAddress_view" ];
    rr:subjectMap [
        rr:template "PostalAddress/{ID}/{ID2}/{ID3}/{ID4}/";
        rr:class conf:PostalAddress;
    ];
    rr:predicateObjectMap [
        rr:predicate rdfs:label;
        rr:objectMap [ rr:template "PostalAddress #{ID} {ID2} {ID3} {ID4}"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate vcard:country;
        rr:objectMap [ rr:column "vcard_country" ];
    ];
    rr:predicateObjectMap [
        rr:predicate vcard:street;
        rr:objectMap [ rr:column "vcard_street" ];
```

```
    ];
    rr:predicateObjectMap [
        rr:predicate vcard:locality;
        rr:objectMap [ rr:column "vcard_locality" ];
    ];
    rr:predicateObjectMap [
        rr:predicate vcard:pcode;
        rr:objectMap [ rr:column "vcard_pcode" ];
    ].
```

9

```
<#TriplesMapConference>
    rr:logicalTable [ rr:tableName "conf_Conference_view" ];
    rr:subjectMap [
        rr:template "Conference/{ID}/";
        rr:class conf:Conference;
    ];
    rr:predicateObjectMap [
        rr:predicate rdfs:label;
        rr:objectMap [ rr:template "Conference #{ID}"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate dc:date;
        rr:objectMap [ rr:column "dc_date" ];
    ];
    rr:predicateObjectMap [
        rr:predicate conf:location;
        rr:objectMap [ rr:column "conf_location" ];
    ];
    rr:predicateObjectMap [
        rr:predicate rdfs:label;
        rr:objectMap [ rr:column "rdfs_label" ];
    ].
```

10

```
<#TriplesMapConcept>
    rr:logicalTable [ rr:tableName "skos_Concept_view" ];
    rr:subjectMap [
        rr:template "Concept/{ID}/";
        rr:class skos:Concept;
    ];
    rr:predicateObjectMap [
        rr:predicate rdfs:label;
        rr:objectMap [ rr:template "Concept #{ID}"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate skos:prefLabel;
        rr:objectMap [ rr:column "skos_prefLabel" ];
    ];
    rr:predicateObjectMap [
        rr:predicate skos:broader;
        rr:objectMap [
            rr:parentTriplesMap <#TriplesMapConcept>;
            rr:joinCondition [
                rr:child "skos_broader";
                rr:parent "ID";
            ];
        ];
    ].
```

### *7.2.6 Publicação dos Dados RDF e Testes*

A publicação dos dados em triplas RDF irá ocorrer quando o usuário selecionar esta opção ("*Publish Data*" na Figura 7.12) após a geração do mapeamento R2RML. Como mencionado na seção 7.2.5, neste momento o componente D2RQ-Ext fará esta publicação.

D2RQ-Ext irá iniciar um serviço na porta 2020 que atenderá a consultas SPARQL via protocolo HTTP. A partir deste momento, o usuário poderá submeter consultas SPARQL utilizando um navegador Web convencional e o componente irá exibir o resultado destas consultas em HTML como visto na Figura 7.13.

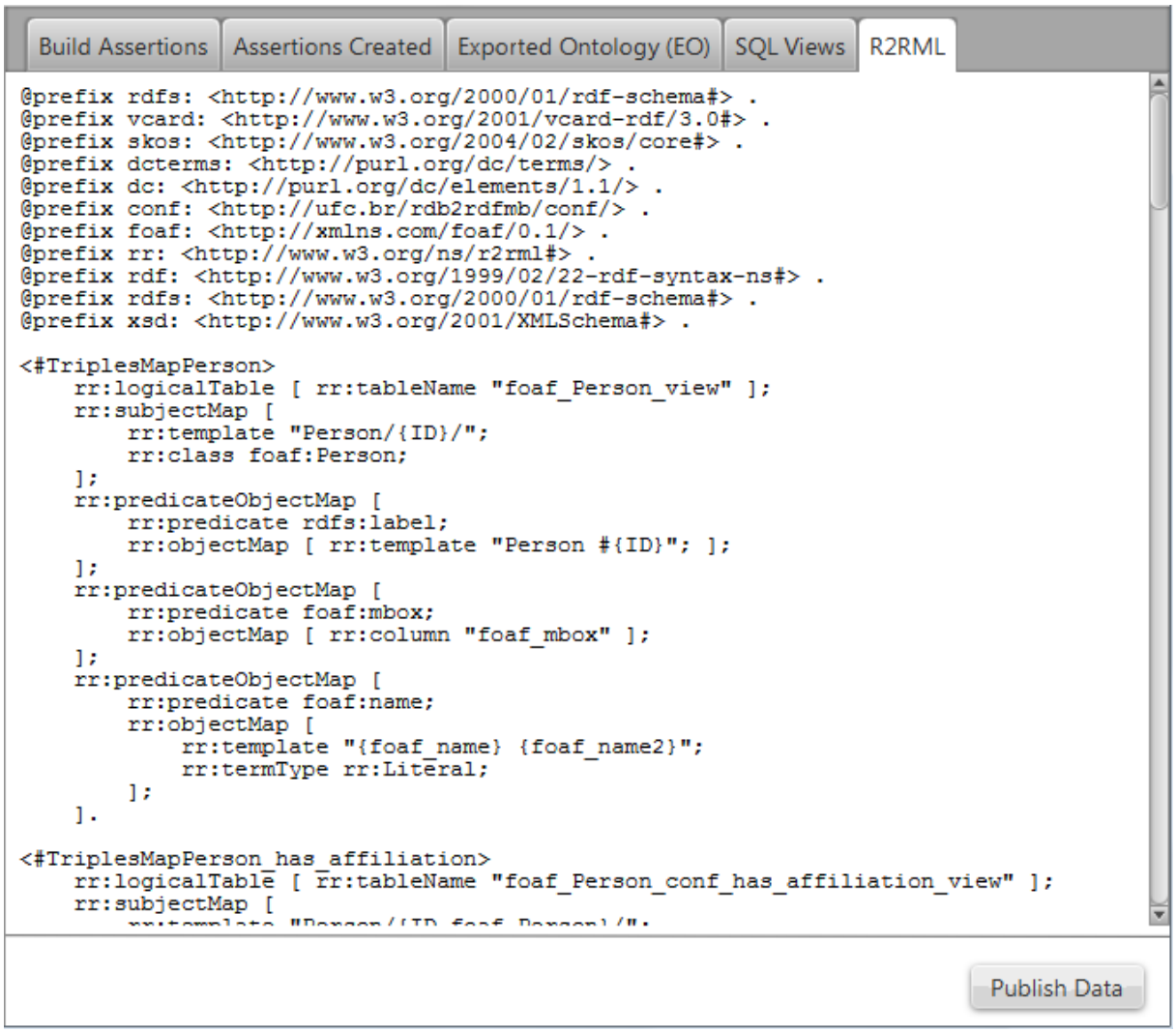


**Figura 7.12 - Mapeamento R2RML gerado**

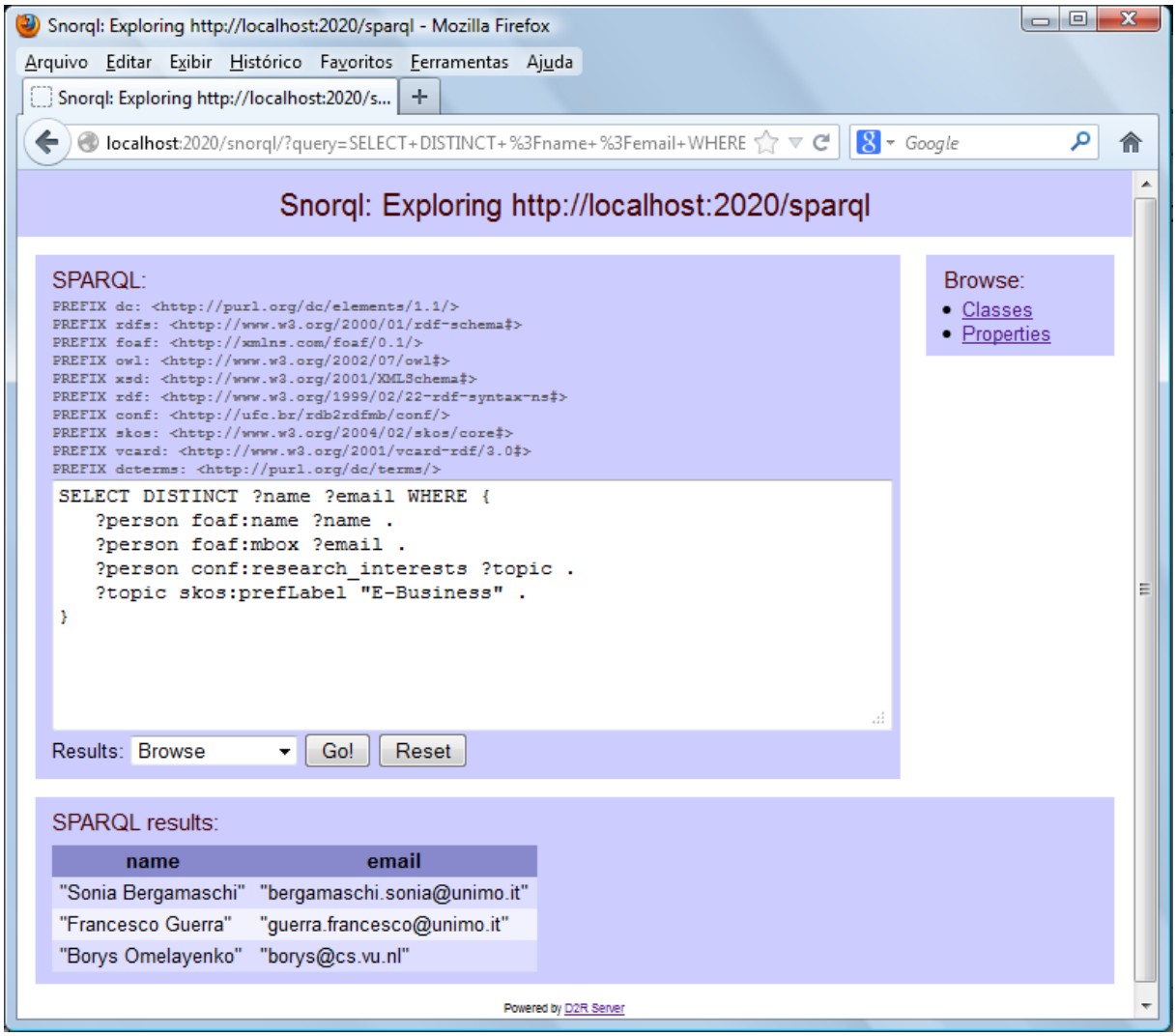


**Figura 7.13 - Consultas SPARQL submetida com seu resultado**

A Figura 7.13 mostra o SPARQL submetido para descobrir o nome e o email das pessoas que se interessam (propriedade de objetos *conf:research_interests* de *foaf:Person*) em pesquisar sobre "E-Business" (propriedade de dados *skos:prefLabel* de *skos:Concept*). Para comprovarmos que o resultado desta consulta está correto, fizemos uma consulta SQL diretamente no banco retornando todas as pessoas com seus respectivos assuntos de interesse. A Figura 7.14 mostra o resultado desta consulta ordenado pelo nome do assunto. Marcamos as três pessoas que se interessam em pesquisar sobre o assunto "E-Business" e vemos que são as mesmas retornadas pela consulta SPARQL.

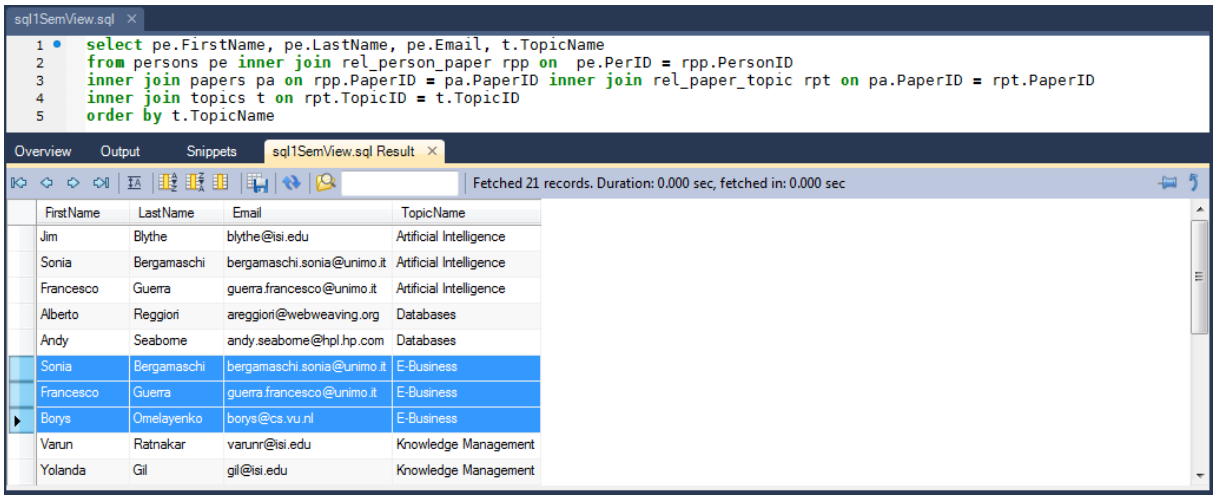


**Figura 7.14 - Consulta SQL retornando pessoas e seus assuntos de interesse**

Para chegar ao resultado correto, o componente D2RQ-Ext utiliza a solução já existente na plataforma D2RQ de reescrita de consulta. Assim, D2RQ-Ext utiliza o mapeamento R2RML e reescreve a consulta SPARQL na forma de uma consulta SQL em cima das visões relacionais. Esta consulta resultante é apresentada na Figura 7.15.

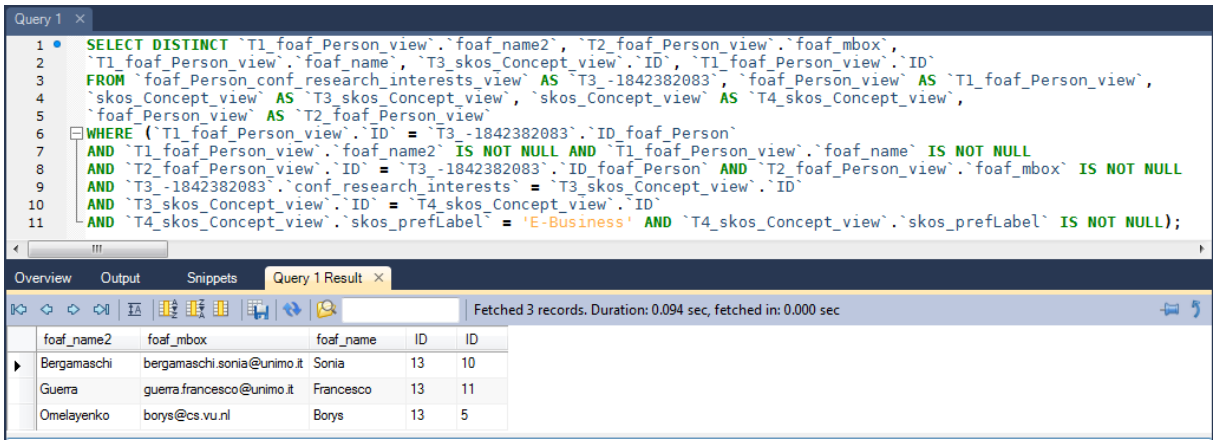


**Figura 7.15 - Consulta SQL traduzida a partir da consulta SPARQL pelo D2RQ**

Por fim mostramos mais duas consultas SPARQL feitas na ferramenta com a avaliação dos seus resultados em relação ao estado do banco de dados:

- Descobrir a *homepage* das empresas que ficam localizadas no Reino Unido (UK). A consulta SPARQL e o seu resultado são mostrados na Tabela 7.3.

**Tabela 7.3 - Consulta SPARQL para descobrir a homepage das empresas do Reino Unido**

| SPARQL | Resultado |
|---|---|
| **SELECT DISTINCT** ?homepage<br>**WHERE** {<br>?organization rdf:type conf:Organization .<br>?organization foaf:homepage ?homepage .<br>?organization vcard:ADR ?address .<br>?address vcard:country "UK" .<br>} | "http://www.hpl.hp.com/" |

A Figura 7.16 mostra a mesma consulta em SQL executada diretamente no banco de dados fonte e exibindo o mesmo resultado da consulta SPARQL.

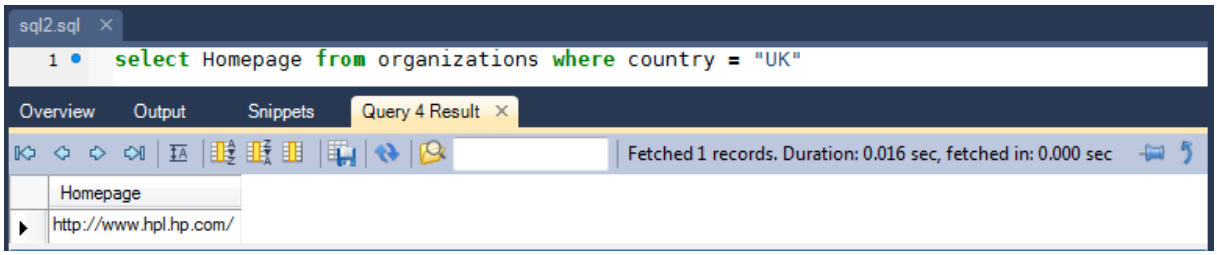


**Figura 7.16 - Consulta SQL para verificar o resultado da consulta SPARQL da Tabela 7.3**

- Nome das pessoas e a quantidade de tópicos diferentes que elas se interessam em pesquisar. A consulta SPARQL e o seu resultado são mostrados na Tabela 7.4.

**Tabela 7.4 - Consulta SPARQL para recuperar as pessoas com a quantidade de tópicos diferentes relacionados**

| | |
|---|---|
| **SPARQL:** | **SELECT DISTINCT**<br>?name (**COUNT**(**DISTINCT** ?t) **AS** ?qtdeTopics) **WHERE** {<br>?p foaf:name ?name .<br>?p conf:research_interests ?t .<br>}<br>**GROUP BY** ?name<br>**ORDER BY** ?name |
| **Resultado:** | **Name** — **qtdeTopics**<br>"Alberto Reggiori" — 3<br>"Andy Seaborne" — 3<br>"Borys Omelayenko" — 3<br>"Francesco Guerra" — 3<br>"Jim Blythe" — 2<br>"Sonia Bergamaschi" — 3<br>"Varun Ratnakar" — 2<br>"Yolanda Gil" — 2 |

A Figura 7.17 mostra a consulta em SQL correspondente executada no banco de dados fonte e exibindo o mesmo resultado da consulta SPARQL.

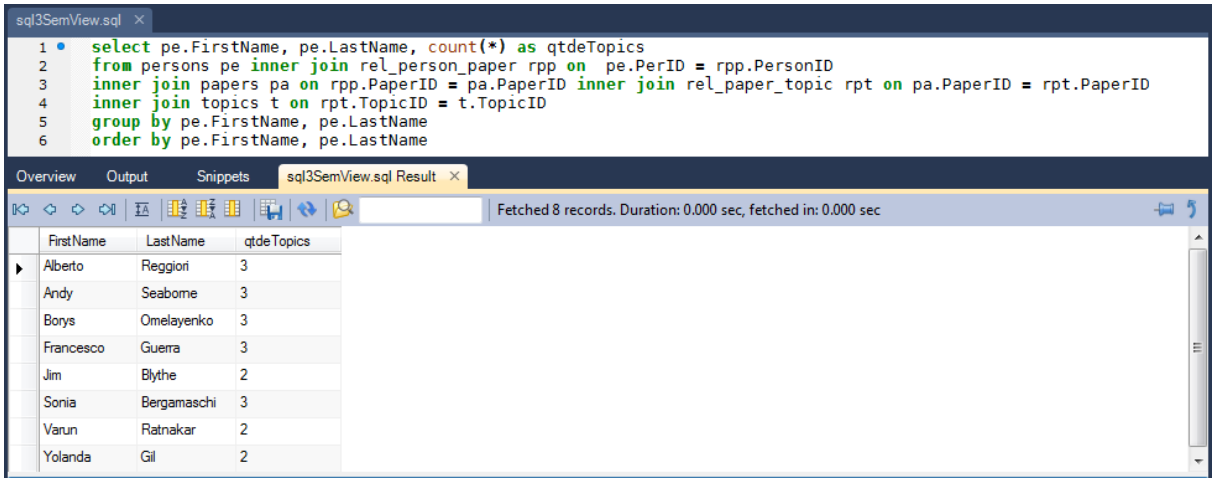


**Figura 7.17 - Consulta SQL para verificar o resultado da consulta SPARQL da Tabela 7.4**

## 7.3 Conclusões

Este capítulo apresentou as principais funcionalidades da ferramenta RBA, explicou sua arquitetura e o funcionamento de cada um dos seus componentes e aplicou sua utilização ao estudo de caso adotado. Por fim, foram apresentados testes de consultas onde comprovamos a exatidão dos resultados obtidos.

# 8 CONCLUSÃO E TRABALHOS FUTUROS

Motivados pela necessidade de facilitar a publicação de mapeamentos R2RML, primeiramente introduzimos assertivas de correspondência para especificar o mapeamento entre um vocabulário alvo e um esquema relacional fonte. Então propomos uma abordagem para gerar automaticamente mapeamentos R2RML com base no conjunto de assertivas de correspondência. A abordagem usa visões relacionais como uma camada intermediária, o que facilita o processo de criação do R2RML e melhora a manutenibilidade do mapeamento.

Assertivas de correspondência foram introduzidas em trabalhos anteriores (VIDAL et al., 2005) para investigar Visões XML. Portanto, seria natural adotar a mesma abordagem para o problema de criar mapeamentos de bancos de dados relacionais para RDF. De fato, este último problema é comprovadamente mais simples que o anterior, pois visões XML são bem mais complexas.

A abordagem proposta é suportada pela ferramenta gráfica (NETO et al., 2013) apresentada com detalhes no Capítulo 6. Esta ferramenta foi demonstrada passo a passo com a aplicação de um estudo de caso introduzido no Capítulo 4.

Esta dissertação resultou em dois trabalhos publicados em conferências importantes:

- Um demo da ferramenta foi aceito no European Semantic Web Conference (ESWC), 2013.
- Um artigo a ser publicado no Symposium On Applied Computing (SAC), 2014.

Atualmente, estamos estendendo a plataforma D2R Server para processar mapeamentos R2RML como base para publicação destes. Esta extensão suporta o Passo Três do processo de publicação descrito no Capítulo 6.

Futuramente, vamos desenvolver uma solução para validar mapeamentos R2RML com base nas restrições definidas pelos esquemas da Ontologia Alvo e do Banco Relacional Fonte.

Além disso, adaptaremos os algoritmos apresentados nas seções 6.3.1 e 6.3.2 para suportarem o caso em que uma classe da ontologia exportada possui mais de uma assertiva de correspondência de classe criada.

# REFERÊNCIAS BIBLIOGRÁFICAS

(ARENAS et al., 2012) ARENAS, M., BERTAILS, A., PRUD'HOMMEAUX, E., SEQUEDA, J. *A Direct Mapping of Relational Data to RDF*. W3C Recommendation 27 September 2012. Disponível em: http://www.w3.org/TR/rdb-direct-mapping/.

(AUER et al., 2007) AUER, S., BIZER, C., LEHMANN, J., KOBILAROV, G., CYGANIAK, R. AND IVES, Z. *DBpedia: A Nucleus for a Web of Open Data*, in K. e. a. Aberer (ed.), Proceedings of the 6th International Semantic Web Conference (ISWC'07), 2007. Vol. 4825 of LNCS, Springer, Berlin, Heidelberg, pp. 715–728.

(AUER et al., 2009) AUER, S. et al. *Triplify: Light-weight linked data publication from relational databases*. In: QUEMADA, J. et al. (Ed.). Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009. [S.l.]: ACM, 2009. p. 621–630. ISBN 978-1-60558-487-4.

(BERNERS-LEE T., 2006) BERNERS-LEE, T. *Linked Data*, 2006. Disponível em: http://www.w3.org/DesignIssues/LinkedData.html/.

(BERNERS-LEE et al., 2006) BERNERS-LEE, T., CHEN, Y., CHILTON, L. AND CONNOLLY, D. E. A. *Tabulator: Exploring and analyzing linked data on the Semantic Web*, Proceedings of the ISWC Workshop on Semantic Web User Interaction, CEUR Workshop Proceedings, 2006.

(BERNERS-LEE et al., 2005) BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. *RFC 3986 – Uniform Resource Identifier (URI): Generic Syntax*, 2005.
Disponível em: http://tools.ietf.org/html/rfc3986.

(BERNERS-LEE et al., 2001) BERNERS-LEE, T.; HENDLER J.; LASSILA O. *The Semantic Web – A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities*, Scientific American, 2001.

(BIZER, 2003) BIZER, C. *D2R MAP - A Database to RDF Mapping Language*. The Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary, May 2003.

(BIZER; CYGANIAK, 2006) BIZER, C., CYGANIAK, R. *D2R Server – Publishing Relational Databases on the Semantic Web*. In: 5th International Semantic Web Conference. [S.l.: s.n.], 2006.

(BRICKLEY; MILLER, 2007) BRICKLEY, D. AND MILLER, L. *FOAF Vocabulary Specification*, 2007. Disponível em: http://xmlns.com/foaf/spec/.

(BROEKSTRA; KAMPMAN, 2001) Broekstra, J., Kampman, A. *Sesame: A generic architecture for storing and querying RDF and RDF schema*, October 2001.

(CARROLL et al., 2004) Carroll, J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A. and Wilkinson, K. *Jena: Implementing the Semantic Web Recommendations*, Proceedings of the International World Wide Web Conference, Hewlett Packard Labs, p. 74, 2004.

(CASANOVA et al., 2011) CASANOVA, M., BREITMAN, K., FURTADO A., VIDAL, V., MACEDO, J., GOMES, R., SALAS, P. *The Role of Constraints in Linked Data*. Proc. 11th Int'l. Conf. on the move to meaningful internet systems - Volume Part II, pp. 781-799, 2011.

(CULLOT et al., 2007) CULLOT, N., GHAWI, R., YE ́TONGNON, K. *DB2OWL: A Tool for Automatic Database-to-Ontology Mapping*. Proc. SEBD, pp. 491-494, 2007.

(CYGANIAK; BIZER, 2009) CYGANIAK, R. AND BIZER, C. *Pubby – A Linked Data Frontend for SPARQL Endpoints*, 2009.
Disponível em: http://www4.wiwiss.fu-berlin.de/pubby/.

(DAS et al., 2012) DAS, S., SUNDARA, S., CYGANIAK, R. *R2RML: RDB to RDF Mapping Language*, 2012. Disponível em: http://www.w3.org/TR/r2rml/.

(ERLING; MIKHAILOV, 2006) Erling, O.; Mikhailov, I. *Mapping Relational Data to RDF in Virtuoso*, 2006.
Disponível em: http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSQLRDF .

(HEATH; BIZER, 2011) HEATH, T.; BIZER, C. *Linked Data: Evolving the Web into a Global Data Space*. 1st. ed. [S.l.]: Morgan & Claypool, 2011. 136 p. ISBN 9781608454303.

(HORROCKS et al., 2005) HORROCKS, I., PARSIA, B., SCHNEIDER, P. P. AND HENDLER, J. *Semantic Web Architecture: Stack or Two Towers?,* in F. Fages and S. Soliman (eds), *Principles and Practice of Semantic Web Reasoning* (PPSWR 2005), 2005. Vol. 3703 of LNCS, Springer, Berlin, Heidelberg, pp. 37–41.

(HORROCKS et al., 2004) HORROCKS, I., PATEL-SCHNEIDER, P. F., BOLEY, H., TABET, S., GROSOF, B. AND DEAN, M. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, 2004. Disponível em: http://www.w3.org/Submission/SWRL/.

(KNAP et al., 2012) KNAP, T., MICHELFEIT, J., DANIEL, J., JERMAN, P., RYCHNOVSKY, D., SOUKUP, T., NECASKY, M. (2012) *ODCleanStore: A Framework for Managing and Providing Integrated Linked Data on the Web*, Proc. Int'l. Semantic Web Conf. 2012, Posters & Demonstrations Track.

(LANGEGGER, 2010) LANGEGGER, A. *A Flexible Architecture for Virtual Information Integration based on Semantic Web Concepts*. Tese (Doutorado) — J. Kepler University Linz, 2010.

(NETO et al., 2013) NETO, L., VIDAL, V., CASANOVA, M., MONTEIRO, J. *R2RML by Assertion: A Semi-Automatic Tool for Generating Customized R2RML Mappings*. (Accepted demo – European Semantic Web Conference), 2013.

(PRUD'HOMMEAUX; SEABORNE, 2008) PRUD'HOMMEAUX, E., SEABORNE, A. *SPARQL Query Language for RDF*, 2008.
Disponível em: http://www.w3.org/TR/rdf-sparql-query/.

(SCHULTZ et al., 2011) SCHULTZ, A., MATTEINI, A., ISELE, R., BIZER, C., BECKER C. *LDIF - Linked Data Integration Framework*. Proc. 2nd Int'l. Workshop on Consuming Linked Data, pp. 1-6, 2011.

(TUMMARELLO et al., 2007) TUMMARELLO, G., DELBRU, R. AND OREN, E. *Sindice.com: Weaving the Open Linked Data*, Proceedings of the 6th International Semantic Web Conference (ISWC), 2007.

(VIDAL et al., 2005) VIDAL, V., ARAUJO, V., CASANOVA, M. *Towards Automatic Generation of Rules for Incremental Maintenance of XML Views of Relational Data*. Proc. Web Information Systems Engineering, pp. 189-202, 2005.

(VIDAL et al., 2014) VIDAL, V., CASANOVA, M., MONTEIRO, J., NETO, L. *A Semi-Automatic Approach for Generating Customized R2RML Mappings*. In proceedings of 29th Symposium On Applied Computing, Gyeongju, Korea, March, 2014 (to appear).