Usando Visões SQL para Facilitar a Publicação de Visões RDF de Dados Relacionais

Luís Eufrasio Teixeira Neto[1], Vânia Maria P. Vidal[1], José Maria Monteiro[1]

[1] Universidade Federal do Ceará
{luiseufrasio,vvidal,zemaria}@lia.ufc.br

Abstract. This paper proposes an approach to easily publish relational data as RDF views. For this, the solution proposed uses SQL Views and defines a 3-tier architecture: data layer, SQL views layer and RDF views layer. In addiction, to enable the creation of endpoints SPARQL from R2RML mappings, this work describes an implementation that extends the D2RQ platform to provide support to R2RML language.

Resumo. Este artigo propõe uma abordagem para facilitar a tarefa de publicar dados relacionais na forma de visões RDF. A solução proposta baseia-se na utilização de visões SQL e em uma arquitetura em 3 camadas: de dados, de visões SQL e de visões RDF. Adicionalmente, para possibilitar a criação de endpoints SPARQL a partir de mapeamentos R2RML, esse trabalho descreve uma implementação que estende a plataforma D2RQ para fornecer suporte a linguagem R2RML.

Categories and Subject Descriptors: H. Information Systems [**H.m. Miscellaneous**]: Databases

Keywords: RDB2RDF, Linked Data, R2RML, RDF

1. INTRODUÇÃO

O padrão Linked Data foi estabelecido como a melhor prática para expor, compartilhar e conectar partes de dados, informações e conhecimentos na Web Semântica usando URIs e RDF [Lassila et al. 1999]. No entanto, a grande maioria dos dados corporativos, inclusive dados da Web, permanecem armazenados em SGBDs relacionais. Para tornar as informações armazenadas em bancos de dados relacionais disponíveis na Web de dados é preciso publicá-las no modelo RDF. Uma das principais ferramentas utilizadas com esta finalidade é o D2R Server [Bizer et al. 2006], o qual possibilita a publicação virtual de dados relacionais por meio de *endpoints* SPARQL/RDF [Prud'hommeaux et al. 2008].

O D2R Server permite que clientes HTML e RDF naveguem no conteúdo da bancos de dados não-RDF e possibilita que aplicações consultem um banco de dados usando a linguagem de consultas SPARQL sobre o protocolo SPARQL. O servidor recebe essas requisições da Web e as reescreve na forma de consultas SQL. Isso é realizado por meio de mapeamentos entre esquemas de banco de dados e esquemas RDFS ou ontologias OWL [McGuinness et al. 2004].

R2RML [Das et al. 2012] é a linguagem recomendada pela W3C (Consórcio World Wide Web) para expressar mapeamentos customizados de bancos de dados relacionais para datasets RDF. Tais mapeamentos permitem a visualização de dados relacionais existentes na forma de um modelo de dados RDF, cuja estrutura e vocabulário alvo são escolhidos pelo autor do mapeamento.

Por outro lado, a publicação dos dados relacionais em RDF usando a ferramenta D2R Server e a linguagem R2RML envolve alguns desafios:

1. O usuário necessita de conhecimentos avançados acerca da linguagem R2RML;

2. Existem diversos problemas relacionados à heterogeneidade entre um esquema relacional fonte e uma ontologia alvo, geralmente especificada em OWL (*Web Ontology Language*). Como exemplos desses problemas podemos citar: i) uma tabela do esquema relacional pode ser mapeada para uma única propriedade de dados na ontologia alvo; e ii) um caminho entre tabelas relacionais definido pelas chaves estrangeiras pode ser mapeado em uma propriedade de objetos na ontologia alvo.
3. Sempre que ocorrerem alterações no esquema relacional fonte isso implica na necessidade de se alterar os mapeamentos R2RML;
4. Até o presente momento nenhuma ferramenta implementa por completo a especificação R2RML.
5. Apesar do D2R Server ser uma das ferramentas mais utilizadas para a publicação de dados relacionais na Web Semântica, este utiliza uma linguagem de mapeamento proprietária (*D2RQ Mapping Language* - D2RM) ao invés da linguagem R2RML.

Para superar estes desafios, propomos neste artigo uma extensão do D2R Server a fim de fornecer suporte para a utilização da linguagem R2RML e uma arquitetura de três camadas para publicação de visões RDF de dados relacionais e que facilita a geração de mapeamentos R2RML.

O restante deste artigo está organizado como segue. A Seção 2 apresenta uma introdução à linguagem R2RML, discute os tipos de visões que podem ser utilizadas na linguagem, a utilização de mapeamentos diretos e como tornar o D2RQ compatível com essa linguagem. A seção 3 detalha a arquitetura em 3 camadas. A Seção 4 apresenta um estudo de caso da arquitetura proposta. A Seção 5 explica como geramos o mapeamento R2RML direto das Visões SQL para a Visão RDF. Na seção 6 temos a implementação que fizemos na plataforma D2RQ para que esta ferramenta suporte mapeamentos que usam a linguagem R2RML. A Seção 7 apresenta as conclusões.

2. INTRODUÇÃO AO R2RML

Uma linguagem para expressar mapeamentos customizados de bancos de dados relacionais para *datasets* RDF. Tais mapeamentos possibilitam a visualização de dados relacionais em um modelo RDF, cuja estrutura e o vocabulário alvo são escolhidos pelo autor dos mapeamentos [Das et al. 2012].

Um mapeamento R2RML faz referência a tabelas lógicas para obter dados do banco de dados de entrada. Uma tabela lógica pode ser dos seguintes tipos: i) uma tabela do esquema do banco relacional; ii) uma visão do esquema do banco relacional; e iii) uma consulta SQL válida chamada de "Visão R2RML" definida no mapeamento.

Mapeamentos customizados são importantes para solucionar os problemas de heterogeneidade os quais foram discutidos na seção anterior. Contudo, caso não hajam problemas de heterogeneidade, podemos criar mapeamentos diretos [Arenas et al. 2012]. Em um mapeamento direto de um banco de dados, o vocabulário RDF alvo reflete os nomes dos elementos do esquema do banco.

A Tabela I mostra dois exemplos de mapeamentos R2RML: um mapeamento direto e um mapeamento customizado com uma visão R2RML. No mapeamento direto o vocabulário reflete os nomes dos conceitos do banco: o nome da classe é o mesmo nome da tabela (*DVD*) e cada propriedade da classe é nomeada com o mesmo nome da coluna correspondente precedido pelo nome da tabela (*dvd#descricao*). Em contrapartida, o mapeamento customizado utiliza um vocabulário escolhido. Assim a tabela DVD passa a ser mapeada em uma classe com outro nome (*Video*). De forma similar, as colunas da tabela passam a ser mapeadas em termos do vocabulário escolhido: a coluna *descricao* é mapeada na propriedade *titulo*, por exemplo.

Adicionalmente, o mapeamento customizado utiliza uma visão R2RML para tratar um problema de heterogeneidade: a tabela *DVD* possui uma coluna denominada *temDiretor* que é uma chave estrangeira para a tabela *Diretor*, enquanto que a classe *Video* possui uma propriedade de dados denominada *diretor* que representa o nome do diretor. O problema envolvido nessa situação é que o valor do nome do diretor não está na tabela de origem (*DVD*), mas em uma tabela que se relaciona com ela (*Diretor*).

Mapeamentos diretos podem ser feitos de forma automática usando ferramentas que implementam a especificação desse tipo de mapeamento. O D2RQ é uma ferramenta que implementa mapeamentos diretos de forma automática seguindo a a especificação da W3C. No entanto, existe o desafio de tornar o D2R Server compatível com a linguagem R2RML.

Tabela I. Exemplos de Mapeamento Direto e de Mapeamento Customizado

Mapeamento direto da Tabela *DVD*

```
# Tabela dvd
<#DvdTriplesMap>
    rr:logicalTable [ rr:tableName "dvd" ];
    rr:subjectMap [
        rr:template "dvd/descricao-{descricao}";
        rr:class <dvd>;
    ];
    rr:predicateObjectMap [
        rr:predicate <dvd#descricao>;
        rr:objectMap [ rr:column "descricao" ];
    ];
    rr:predicateObjectMap [
        rr:predicate <dvd#valor>;
        rr:objectMap [ rr:column "valor" ];
    ];
    rr:predicateObjectMap [
        rr:predicate <dvd#temDiretor>;
        rr:objectMap [ rr:column "temDiretor" ];
    ].
```

Mapeamento Customizado: Tabela *DVD* para Classe *Video*

```
# Tabela DVD para Classe Video
<#VideoTriplesMap>
    rr:logicalTable [ rr:sqlQuery """
        SELECT d.descricao, d.valor,
        (SELECT i.nomeDiretor
         FROM diretor i
         WHERE d.temDiretor = i.nomeDiretor)
        FROM dvd d
        """];
    rr:subjectMap [
        rr:template "video/{descricao}";
        rr:class v:Video;
    ];
    rr:predicateObjectMap [
        rr:predicate v:titulo;
        rr:objectMap [ rr:column "descricao" ];
    ];
    rr:predicateObjectMap [
        rr:predicate v:preco;
        rr:objectMap [ rr:column "valor" ];
    ];
    rr:predicateObjectMap [
        rr:predicate v:diretor;
        rr:objectMap [ rr:column "nomeDiretor" ];
    ].
```

O D2R Server implementa internamente uma estrutura de objetos que representam os termos da sua linguagem proprietária D2RM. A *Engine* do D2R Server utiliza esses objetos para traduzir consultas SPARQL em consultas SQL no banco de dados [Bizer et al. 2006].

Para tornar o D2R Server compatível com R2RML sem alterarmos a sua *Engine* precisamos criar a mesma estrutura de objetos já existente a partir de mapeamentos R2RML. Essa tradução é simples quando o mapeamento é direto e não possui visões R2RML, haja visto que nesse caso podemos fazer uma correspondência direta entre os termos da linguagem R2RML usados no mapeamento com os objetos do D2R Server que implementam a linguagem D2RM. Entretanto, quando o mapeamento não é direto utilizando visões R2RML esta tradução é de alta complexidade, pois nesse caso teremos que validar as consultas SQL das visões R2RML e interpretá-las para identificar quais objetos do D2R Server precisam ser criados. A arquitetura que adotamos nesse trabalho utiliza mapeamentos R2RML diretos de visões SQL para visões RDF. Essa abordagem permite a utilização da extensão que criamos para o D2R Server na publicação das visões RDF, uma vez que não utiliza visões R2RML.

3. ARQUITETURA DE 3 CAMADAS

Em uma arquitetura convencional de duas camadas escrevemos mapeamentos do esquema do banco de dados relacional para o esquema da visão RDF, denominada ontologia alvo. Devido aos problemas de heterogeneidade entre os esquemas, nessa arquitetura devem ser criados mapeamentos customizados.

Na arquitetura de três camadas que estamos propondo serão criadas visões SQL no banco de dados relacional e essas visões serão mapeadas para o esquema da visão RDF. Nessa abordagem definimos uma visão SQL para cada classe OWL na ontologia alvo, caraterizando uma correspondência do tipo um-para-um.

A arquitetura proposta é ilustrada na Figura 1. A seguir discutiremos em detalhes cada uma das camadas que compõem essa arquitetura.

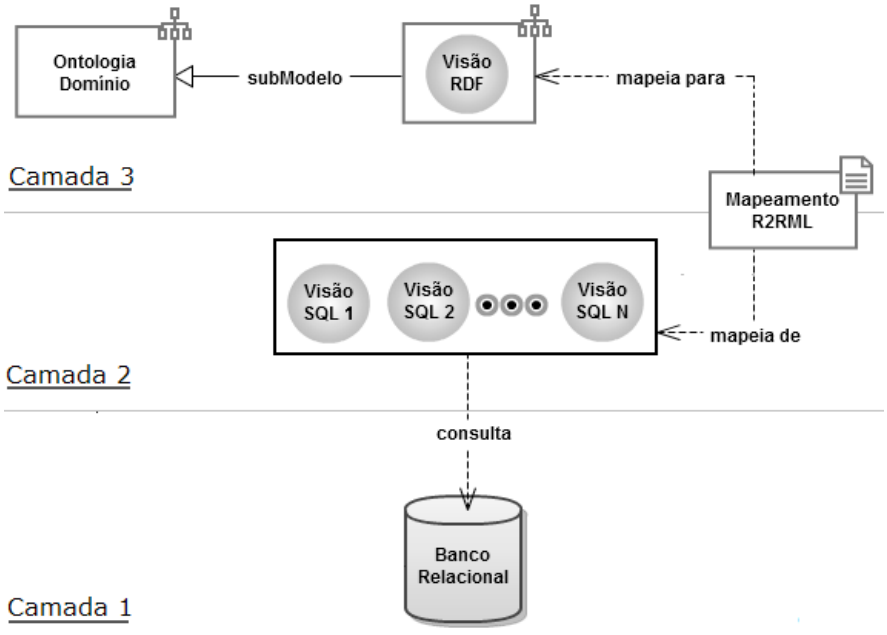

Fig. 1. Arquitetura em 3 camadas para publicação de Visões RDF de Dados Relacionais

A Camada Um é composta pelo banco de dados relacional. Nesta camada temos o esquema relacional do banco e os dados a serem publicados na Web Semântica.

Na Camada Três existem duas ontologias: a ontologia de domínio e a ontologia alvo denominada Visão RDF. O vocabulário de termos da ontologia alvo é um subconjunto do vocabulário de termos da ontologia de domínio. A ontologia alvo reflete as classes e propriedades da ontologia de domínio cujos valores podem ser exportados do banco de dados da Camada Um.

A Camada Dois tem por objetivo facilitar a geração de mapeamentos R2RML. Com essa finalidade, ela é constituída de Visões SQL do banco relacional da Camada Um. Os problemas de heterogeneidade são tratados nas consultas SQL desta camada, possibilitando a geração do mapeamento direto das visões SQL para a ontologia alvo da Camada Três.

Esta arquitetura faz parte de um processo de publicação de dados relacionais o qual engloba os seguintes etapas: i) Seleção do banco de dados fonte e da ontologia de domínio; ii) Geração da ontologia alvo (Visão RDF) [Sacramento et al. 2010B]; iii) Criação das Visões SQL no banco de dados fonte; e iv) Geração do mapeamento R2RML direto das Visões SQL para a visão RDF.

4. ESTUDO DE CASO

Para ilustrar o funcionamento da abordagem proposta iremos utilizar um estudo de caso que contempla as três camadas da arquitetura descrita na seção anterior.

Considere o esquema relacional *Amazon_DB* da Figura 2 com 5 tabelas (*Livro*, *Musica*, *Diretor*, *DVD* e *Estudio*) e dois relacionamentos (*temDiretor* e *produzidaPor*). O relacionamento *temDiretor* faz a ligação de uma tupla da tabela *DVD* com uma única tupla da tabela *Diretor*. O relacionamento *produzidaPor* vincula uma tupla da tabela *Musica* com uma única tupla da tabela *Estudio*. As colunas *descricao*, *valor* e *condicao* se repetem nas tabelas *Livro*, *Musica* e *DVD*, haja visto que essa foi a abordagem utilizada neste estudo de caso para modelar um relacionamento de herança dessas entidades com uma entidade mais genérica.

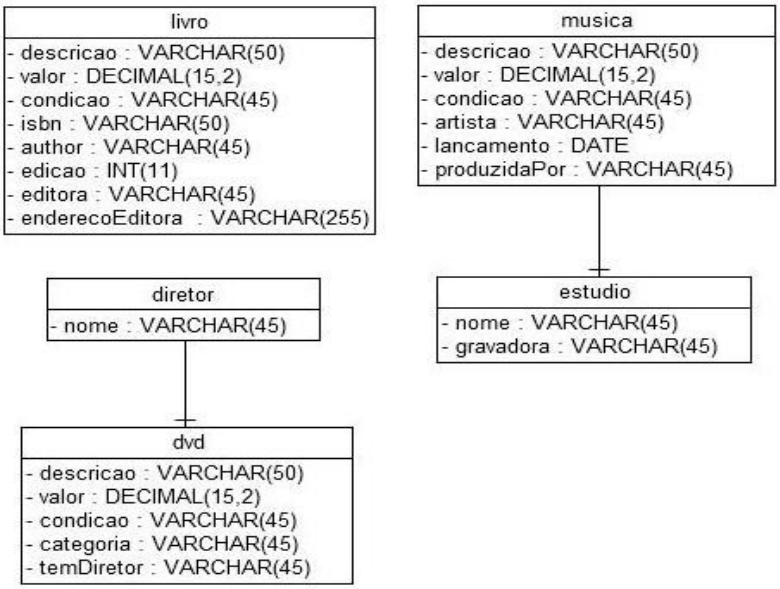

Fig. 2. Esquema Relacional *Amazon_DB*

Na Figura 3 está a ontologia alvo que contém um subconjunto do vocabulário *Vendas* escolhido para publicarmos os dados do banco *Amazon_DB*. Essa ontologia inclui 6 classes (*Gravadora, Musica, Produto, Video, Publicacao* e *Editora*) e 5 relacionamentos (3 de herança e 2 de associação). As classes *Musica*, *Video* e *Publicacao* herdam da classe *Produto* as propriedades *titulo* e *preco*. A classe

Musica possui uma propriedade de objeto denominada *distribuidaPor* que tem como imagem a classe *Gravadora*. A classe *Publicacao* é domínio da propriedade *publicadaPor* que tem como imagem a classe *Editora*.

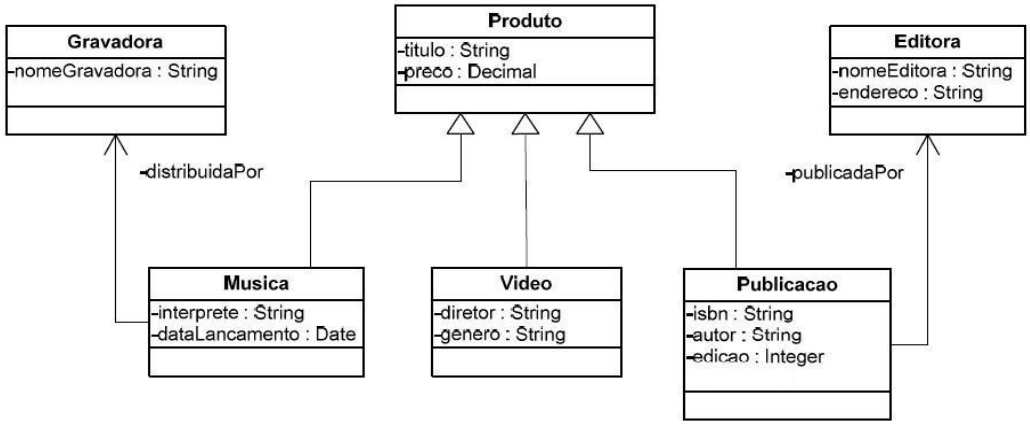

Fig. 3. Visão RDF dos dados relacionais exportados

O esquema relacional do banco *Amazon_DB* e os seus dados formam a camada Um. A ontologia de domínio Vendas e as classes e propriedades da visão RDF da Figura 3 compõem a camada Três.

A camada Dois será constituída por 6 visões SQL, uma para cada classe da Ontologia Alvo. Como a especificação de mapeamentos diretos define que o vocabulário alvo reflete os conceitos do banco de dados fonte, essas visões deverão ter o mesmo nome da classe correspondente e os atributos que as visões projetam em suas consultas SQL deverão ser nomeadas com o mesmo nome das propriedades correspondentes de cada atributo.

A Tabela II mostra as visões SQL geradas para as classes *Musica* e *Produto* do estudo de caso. O *subselect* incluído na consulta SQL da visão *Musica* foi gerado para criar a propriedade de objeto *distribuidaPor* que referencia a classe gravadora na ontologia alvo. Nas duas visões todos os atributos selecionados foram renomeados usando a cláusula **AS** para refletirem os nomes de suas propriedades correspondentes na ontologia alvo. A classe *Produto* engloba as propriedades *titulo* e *preco* que são herdadas pelas classes *Musica*, *Publicacao* e *Video*. Como não existe uma tabela que represente a classe *Produto*, esta visão precisa consultar os atributos *descricao* e *valor* que estão replicados nas tabelas Musica, Livro e DVD.

Tabela II. Uma Visão SQL para as classes Musica e Produto

Classe RDF	**Visão SQL**
Musica	**CREATE VIEW** *Musica* **AS** **SELECT** m.artista **AS** interprete, m.lancamento **AS** dataLancamento, m.descricao **AS** titulo, m.valor **AS** preco, (**SELECT** e.gravadora **FROM** Estudio e **WHERE** e.nome = m.produzidaPor) **AS** distribuidaPor **FROM** Musica m
Produto	**CREATE VIEW** *Produto* **AS** (**SELECT** descricao **AS** titulo, valor **AS** preco **FROM** Musica) **UNION** (**SELECT** descricao **AS** titulo, valor **AS** preco **FROM** Livro) **UNION** (**SELECT** descricao **AS** titulo, valor **AS** preco **FROM** Dvd d)

As vantagens da criação dessa camada são: i) alterações na estrutura do banco de dados não implicam em alterações no arquivo de mapeamento, caracterizando uma Independência Lógica. Por exemplo, caso seja adicionada uma tabela *Produto* no banco *Amazon_DB* para modelar a herança, basta alterarmos a consulta SQL da visão *Produto* e o mapeamento R2RML continuará o mesmo; ii) a geração do arquivo de mapeamento R2RML torna-se viável e de fácil implementação. O mapeamento

é direto, ou seja, para cada visão SQL é criada uma classe RDF com as propriedades exportadas na consulta; e iii) a publicação dos dados relacionais é facilitada pelo fato de não precisarmos tratar consultas SQL dentro do arquivo de mapeamento simplificando a tarefa dos processadores R2RML que não terão que validar essas consultas nem otimizá-las. Essas tarefas complexas serão realizadas pelo próprio SGBD.

5. GERAÇÃO DOS MAPEAMENTOS R2RML

Esta seção descreve a etapa 4 do processo de publicação de dados relacionais como visões RDF. Os mapeamentos R2RML diretos serão gerados das visões SQL para a ontologia alvo. Assumimos como premissa que para cada classe OWL da ontologia alvo foi criada uma visão SQL correspondente no banco de dados relacional.

Considerando que as visões SQL solucionam os problemas de heterogeneidade, esse passo da nossa abordagem é bastante simples: serão aplicados os *templates* de geração de mapeamentos da Tabela III para cada Visão SQL. Os termos em negrito e sublinhados dos *templates* são parâmetros de entrada cujos valores serão substituídos no momento da geração do mapeamento. Em seguida explicaremos a função de cada *template* e de seus respectivos parâmetros.

Tabela III. Templates para geração do mapeamento R2RML

#	**Template**	**Exemplo de Mapeamento da Classe *Musica***
1	<#**nomeDaVisao**TriplesMap> rr:logicalTable [rr:tableName "**nomeDaVisao**"]; rr:subjectMap [rr:template '**nomeDaVisao**/{**colunasChave**}'; rr:class **nomeDaVisao**;];	<#**Musica**TriplesMap> rr:logicalTable [rr:tableName "**Musica**"]; rr:subjectMap [rr:template '**musica**/{**descricao**}'; rr:class **Musica**;];
2	rr:predicateObjectMap [rr:predicate **nomeDaColuna**; rr:objectMap [rr:column '**nomeDaColuna**'];];	rr:predicateObjectMap [rr:predicate **interprete**; rr:objectMap [rr:column '**interprete**'];];
3	rr:predicateObjectMap [rr:predicate **nomeDaColuna**; rr:objectMap [rr:parentTriplesMap <#**classeRef**TriplesMap>; rr:joinCondition [rr:child '**nomeDaColuna**'; rr:parent '**nomeDaColunaPai**';];];];	rr:predicateObjectMap [rr:predicate v:**distribuidaPor**; rr:objectMap [rr:parentTriplesMap <#**Gravadora**TriplesMap>; rr:joinCondition [rr:child '**distribuidaPor**'; rr:parent '**gravadora**';];];];

O *Template* 1 é usado para criar o cabeçalho do mapeamento de triplas (*TriplesMap*) de uma classe OWL da ontologia alvo. Como nossas visões são definidas para ter o mesmo nome da classe OWL, então o parâmetro utilizado nesta parte do *template* é o nome da visão. Neste caso, além no nome da visão é utilizado um outro parâmetro denominado *colunasChave* que representa uma ou mais colunas que identificam unicamente uma tupla da visão.

Aplicamos o *Template* 2 para definir as propriedades de dados da classe OWL mapeada. Como o nome da coluna da visão SQL é igual ao nome da sua propriedade correspondente somente esse parâmetro será necessário.

Por fim usamos aplicamos o *Template* 3 para mapear as propriedades de objeto da classe OWL mapeada. No exemplo, a classe *Musica* possui apenas a propriedade *distribuidaPor* que cria um relacionamento com a classe *Gravadora*.

6. ESTENDENDO O D2RQ PARA FORNECER SUPORTE A MAPEAMENTOS R2RML

A plataforma D2RQ não oferece suporte a mapeamentos R2RML para criação do *endpoints* SPARQL no momento que este trabalho foi escrito. No entanto, estamos contribuindo no desenvolvimento de uma nova versão do D2R Server. Essa versão implementa um parser para mapeamentos R2RML. O objetivo dessa implementação é que o D2R Server suporte não somente arquivos de mapeamento na linguagem D2RM, mas também arquivos escritos na linguagem R2RML.

O Parser atual chamado *MapParser* percorre o arquivo D2RM e cria a estrutura de objetos que implementa os termos definidos no vocabulário D2RM. O novo parser denominado *R2RMLMapParser* percorre o arquivo R2RML e cria a mesma estrutura de objetos já existente. Dessa forma o parser traduz os termos da linguagem R2RML nos objetos que implementam os termos da linguagem D2RM de forma transparente para o usuário da ferramenta.

A vantagem dessa solução é que ela aproveita a *Engine* atual do D2R Server para criação de endpoints SPARQL. A implementação dessa *Engine* é complexa, pois envolve a reescrita de consultas SPARQL em SQL usando a estrutura de objetos criada para a linguagem D2RM. A desvantagem é que o novo parser não suporta mapeamentos que usem visões R2RML como tabelas lógicas.

Contudo, a falta desse suporte não impossibilita a utilização do D2R Estendido no nosso processo de publicação, haja visto que os mapeamentos definem somente visões SQL do banco de dados como tabelas lógicas. Assim, o componente *R2RMLMapParser* pode ser prontamente utilizado na nossa metodologia sem a necessidade de uma nova versão do D2R Server que suporte visões R2RML.

7. CONCLUSÕES

Neste trabalho foi proposta uma metodologia para facilitar a publicação de dados relacionais na forma de visões RDF. Definimos uma arquitetura em 3 camadas dentro de um processo de publicação dos dados e implementamos uma extensão para a ferramenta D2RQ oferecer suporte a mapeamentos R2RML. Trabalhos futuros definirão os algoritmos necessários para criação automática das visões SQL e dos mapeamentos R2RML a partir de assertivas de correspondência definidos pelo usuário. Além de uma ferramenta gráfica para suportar a criação dessas assertivas. Com a definição dessa ferramenta teremos um ganho de produtividade na geração e manutenção dos mapeamentos.

REFERENCES

BIZER CHRISTIAN, CYGANIAK RICHARD, D2R Server – Publishing Relational Databases on the Semantic Web, 2006.

ARENAS MARCELO, BERTAILS ALEXANDRE, PRUD'HOMMEAUX ERIC, SEQUEDA JUAN, A Direct Mapping of Relational Data to RDF, W3C Working Draft. http://www.w3.org/TR/rdb-direct-mapping/, 2012.

DAS SOURIPRIYA, SUNDARA SEEMA, CYGANIAK RICHARD, R2RML: RDB to RDF Mapping Language, W3C Working Draft. http://www.w3.org/TR/r2rml/, 2012.

LASSILA ORA, SWICK RALPH R., Resource Description Framework (RDF) Model and Syntax Specification, W3C Proposed Recommendation. http://www.w3.org/TR/PR-rdf-syntax/, 1999.

SACRAMENTO, E. R., VIDAL, V. M., MACÊDO, J. A., LÓSCIO, B. F., LOPES, F. L. R., CASANOVA, M. A., LEMOS, F. (2010b). Towards automatic generation of application ontologies. In Proceeding of the 12th International Conference on Enterprise Information Systems - ICEIS, Funchal, Madeira – Portugal.

PRUD'HOMMEAUX ERIC, SEABORNE ANDY, SPARQL Query Language for RDF, W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query/, 2008.

MCGUINNESS DEBORAH, HARMELEN FRANK, OWL Web Ontology Language, W3C Recommendation. http://www.w3.org/TR/owl-features/, 2004.