Automatic Generation of R2RML Mappings

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

[1]Departament of Computing – UFC
Fortaleza – CE – Brazil

{vvidal,zemaria,luiseufrasio}@lia.ufc.br

Abstract. *This paper proposes an approach to generate RDF views of relational data, using SQL views. The paper first specify the conditions for a set of correspondence assertions to fully specify the RDF view in terms of the relational schema and, if so, we show that the mappings defined by the view correspondence assertions can be transformed in R2RML mapping. This paper focuses on an algorithm that automatically generates the SQL query from the view correspondence assertions.*

1. Introduction

R2RML [1] is a language for expressing customized mappings from relational databases to RDF [2] datasets. Such mappings provide the ability to view existing relational data in the RDF data model, expressed in a structure and target vocabulary of the mapping author's choice.

Every R2RML mapping is tailored to a specific database schema and target vocabulary. The input to an R2RML mapping is a relational database that conforms to that schema. The output is an RDF dataset, as defined in SPARQL [3], that uses predicates and types from the target vocabulary. The mapping is conceptual; R2RML processors are free to materialize the output data, or to offer virtual access through an interface that queries the underlying database, or to offer any other means of providing access to the output RDF dataset.

The mapping induces a RDF view that is exported from the data sources.

However, creating R2RML mapping demands advanced knowledge of R2RML language and is time consuming. Moreover, users will have to redefine the R2RML mapping whenever the base relational schema changes. Therefore, tools that facilitate the task of R2RML mapping creation and the maintenance should be developed.

We propose in this paper an approach where the R2RML mapping is derived from correspondence assertions, which specify relationships between the target ontology schema and the relational database schema. In the case of materialized views, as we shown in [], all rules required to maintain the view can be automatic generated based on the view correspondence assertions.

This paper has three major contributions. First, we propose the use of correspondence assertions [4] for specifying the mapping between an RDF view schema and a base relational schema. We formally specify the conditions under which a set of correspondence assertions fully specifies the RDF view in terms of the relational source and, if so, we show that the mappings defined by the view correspondence assertions can be transformed in R2RML mapping. Second, we propose an algorithm that, based on the view correspondence assertions, generates the R2RML mappings. Third, we

propose the R2RML Mappings-By-Assertions tool that facilitates the task of R2RML Mappings creation and maintenance. We note that the mapping formalisms used by other schema mapping tools are either ambiguous [5] or require the user to declare complex logical mapping [6].

This article is organized as follows. Section 2 discusses RDF Views. Section 3 presents our mapping formalism. Section 4 discusses how to specify RDF view using correspondence assertions. Section 5 presents the algorithm that automatically generates the SQL view definition from the correspondence assertions. Finally, Section 6 presents the conclusions.

2. RDF Views

Na especificação R2RML os mapeamentos referenciam *tabelas lógicas* para recuperar dados do banco relacional. Um tabela lógica pode ser: uma tabela, uma *visão* ou uma consulta SQL válida. Nesse artigo adotamos a estratégia de criar visões relacionais que representam esquemas RDF nos termos de esquemas relacionais.

Considere, por exemplo, o esquema relacional *Amazon_DB* e o esquema RDF *Sales*, cujas representações gráficas são mostradas nas Figuras 1 e 2 respectivamente. Para gerar triplas do esquema Sales a partir do esquema *Amazon_DB*, nós exportamos a ontologia de aplicação mostrada na figura 3 e criamos as visões relacionais e o mapeamento R2RML mostrados nas figuras 4 e 5 respectivamente. Como ilustrado nas figuras 6 e 7, para cada tupla nas tabelas do esquema relacional *Amazon_DB*, o arquivo de mapeamento usa as visões para gerar um conjunto de triplas do esquema RDF Sales.

Em mais detalhes, observe na figura 4 que para cada classe da Ontologia de Aplicação mostrada na figura 3 é criada uma Visão em *Amazon_DB*. Cada visão publica as propriedades de sua classe correspondente. O arquivo R2RML mostrado na figura 5 é uma mapeamento direto das visões. As visões utilizam subconsultas para fazer os joins com as tabelas relacionadas. Isso pode ser observado na visão *s_book* que possui um relacionamento 1:1 com *Product* e na visão *s_music* que também possui um relacionamento 1:1 com *Product* e outro n:1 com *Recorder*.

As IRIs de cada *TriplesMap* do mapeamento são geradas usando as colunas que compoem a chave primária da tabela que originou a visão. Por exemplo, na linha 8 do mapeamento, a IRI da classe *s:Product* é formada usando a coluna *s_id*, que corresponde à coluna *id* da tabela origem *Product*.

Na próxima seção apresentamos o formalismo de mapeamento usado para criar as assertivas de correspondência que mapeiam os conceitos do esquema RDF nos termos dos conceitos do esquema relacional. Veremos que a Ontologia de Aplicação é gerada de forma intuitiva a partir das assertivas criadas.

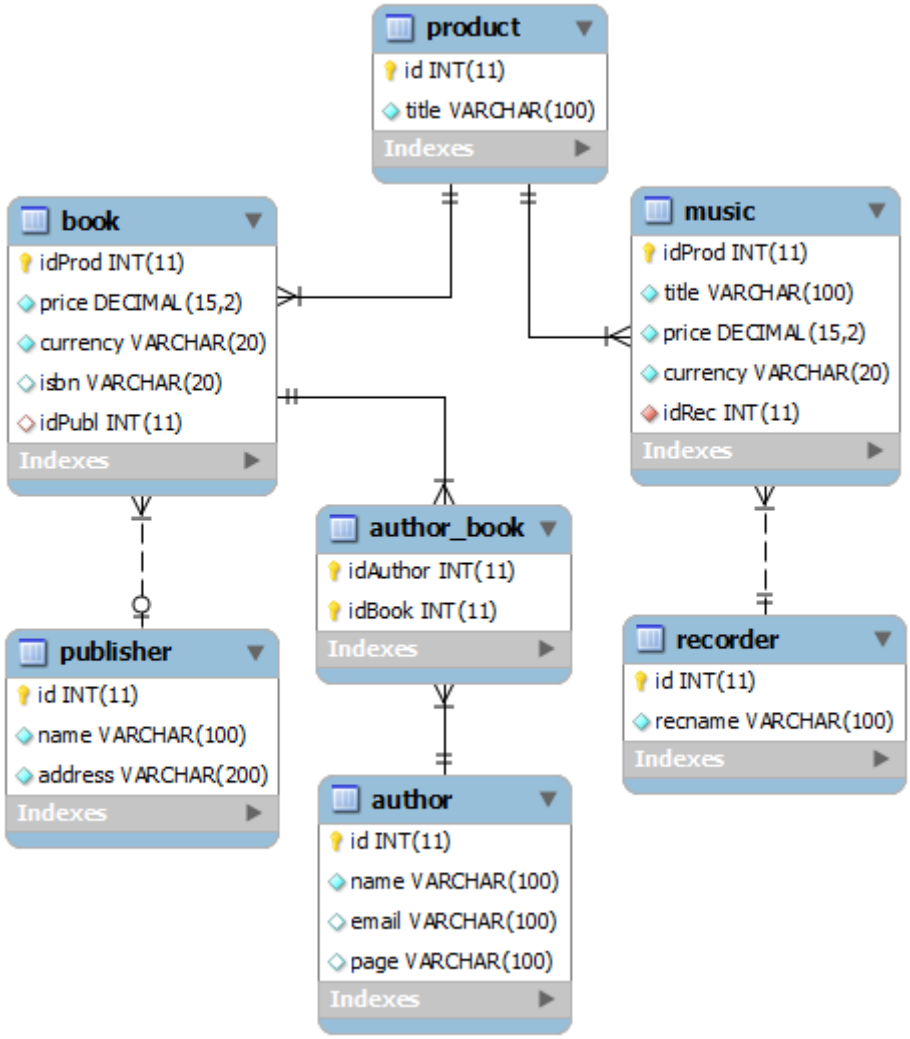

Figure 1 – Relational Schema Amazon

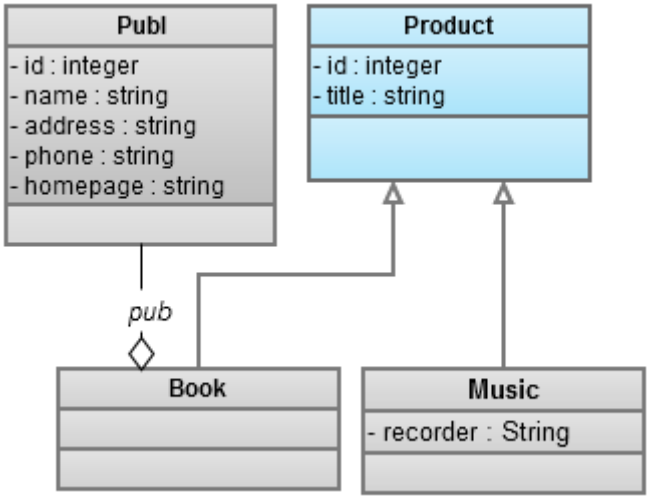

Figure 2 – Domain Ontology Sales

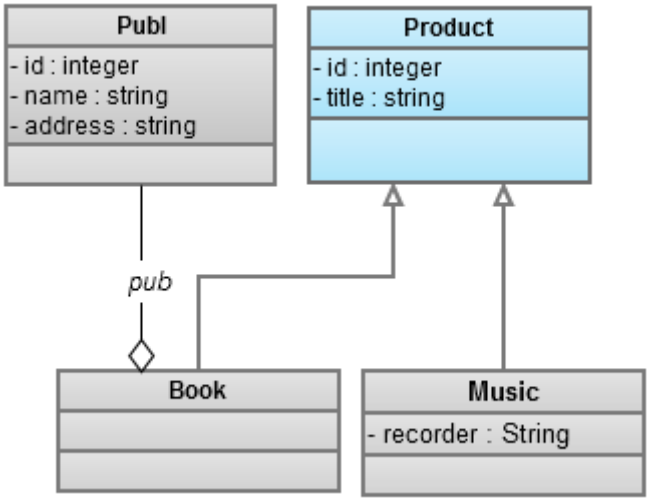

Figure 3 – Application Ontology

```
CREATE VIEW s_product AS
SELECT
    product.id AS s_id, product.title AS s_title
FROM
    product
```

```
CREATE VIEW s_book AS
SELECT b.idProd as s_id,
    b.idPubl as s_pub,
    (SELECT p.title
     FROM Product p
     WHERE p.id = b.idProd) as s_title
FROM Book b
```

```
CREATE VIEW s_music AS
SELECT
    m.idProd AS s_id,
    (SELECT p.title
     FROM product p
     WHERE (p.id = m.idProd)) AS s_title,
    (SELECT r.recname
     FROM recorder r
     WHERE (r.id = m.idRec)) AS s_recorder
FROM
    music m;
```

```
CREATE VIEW s_publ AS
SELECT
    p.id as s_id, p.name as s_name, p.address as s_address
FROM
    publisher p
```

Figure 4 – Database Views

```
@base <http://example.org/sales/>
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix s: <http://example.org/Sales#> .

# View s_product
<#ProductTriplesMap>
    rr:logicalTable [ rr:tableName "s_product" ];
    rr:subjectMap [
        rr:template "product/{s_id}";
        rr:class s:Product;
    ];
    rr:predicateObjectMap [
        rr:predicate s:id;
        rr:objectMap [ rr:column "s_id" ];
    ];
    rr:predicateObjectMap [
        rr:predicate s:title;
        rr:objectMap [ rr:column "s_title" ];
    ].

# View s_book
<#BookTriplesMap>
    rr:logicalTable [ rr:tableName "s_book" ];
    rr:subjectMap [
        rr:template "book/{s_id}";
        rr:class s:Book;
    ];
    rr:predicateObjectMap [
        rr:predicate s:id;
        rr:objectMap [ rr:column "s_id" ];
    ];
    rr:predicateObjectMap [
        rr:predicate s:title;
        rr:objectMap [ rr:column "s_title" ];
    ];
    rr:predicateObjectMap [
        rr:predicate s:pub;
        rr:objectMap [
            rr:parentTriplesMap <#PublTriplesMap>;
            rr:joinCondition [
                rr:child "s_pub";
                rr:parent "s_name";
            ];
        ];
    ].

# View s_publ
<#PublTriplesMap>
    rr:logicalTable [ rr:tableName "s_publ" ];
    rr:subjectMap [
        rr:template "publ/{s_id}";
        rr:class s:Publ;
    ];
    rr:predicateObjectMap [
        rr:predicate s:id;
        rr:objectMap [ rr:column "s_id" ];
    ];
    rr:predicateObjectMap [
        rr:predicate s:name;
        rr:objectMap [ rr:column "s_name" ];
    ];
    rr:predicateObjectMap [
        rr:predicate s:address;
        rr:objectMap [ rr:column "s_address" ];
    ].
```

```
63. # View s_music
64. <#MusicTriplesMap>
65.     rr:logicalTable [ rr:tableName "s_music" ];
66.     rr:subjectMap [
67.         rr:template "music/{s_id}";
68.         rr:class s:Music;
69.     ];
70.     rr:predicateObjectMap [
71.         rr:predicate s:id;
72.         rr:objectMap [ rr:column "s_id" ];
73.     ];
74.     rr:predicateObjectMap [
75.         rr:predicate s:title;
76.         rr:objectMap [ rr:column "s_title" ];
77.     ];
78.     rr:predicateObjectMap [
79.         rr:predicate s:recorder;
80.         rr:objectMap [ rr:column "s_recorder" ];
81.     ].
```

Figure 5 – R2RML Mapping

Product

id	title
1	Android in Action
2	Hello! Python
3	MongoDB in Action
4	One In A Million
5	Time Of The Season

Book

idProd	price	currency	isbn	idPubl
1	49.99	USD	9781935182726	1
2	34.99	USD	9781935182085	1
3	44.99	USD	9781935182870	1

Music

idProd	price	currency	idRec
4	1.99	USD	2
5	0.99	USD	1

Author

id	name	email	page
1	W. Frank Ableson	frank@ableson.com	NULL
2	Kyle Banker	kyle@banker.com	NULL
3	Anthony Briggs	anthony@briggs.com	NULL

Author_Book

idAuthor	idBook
1	1
3	2
2	3

Recorder

id	recname
1	Sony Music Entertainment
2	Universal Music Group

Publisher

id	name	address
1	Manning Publications Co.	20 Baldwin Road PO Box 261 Shelter Island, NY 11964

Figure 6 – An instance of Amazon

```
@base <http://example.org/sales/> .
@prefix s: <http://example.org/sales#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .


<product/1> s:id "1"^^xsd:integer .
<product/1> s:title "Android in Action" .
<product/1> rdf:type s:Product .
<product/2> s:id "2"^^xsd:integer .
<product/2> s:title "Hello! Python" .
<product/2> rdf:type s:Product .
<product/3> s:id "3"^^xsd:integer .
<product/3> s:title "MongoDB in Action" .
<product/3> rdf:type s:Product .
<product/4> s:id "4"^^xsd:integer .
<product/4> s:title "One In A Million" .
<product/4> rdf:type s:Product .
<product/5> s:id "5"^^xsd:integer .
<product/5> s:title "Time Of The Season" .
<product/5> rdf:type s:Product .


<book/1> s:pub <publ/1> .
<book/1> s:id "1"^^xsd:integer .
<book/1> s:title "Android in Action" .
<book/1> rdf:type s:Book .
<book/2> s:pub <publ/1> .
<book/2> s:id "2"^^xsd:integer .
<book/2> s:title "Hello! Python" .
<book/2> rdf:type s:Book .
<book/3> s:pub <publ/1> .
<book/3> s:id "3"^^xsd:integer .
<book/3> s:title "MongoDB in Action" .
<book/3> rdf:type s:Book .


<music/4> s:recorder "Universal Music Group" .
<music/4> s:id "4"^^xsd:integer .
<music/4> s:title "One In A Million" .
<music/4> rdf:type s:Music .
<music/5> s:recorder "Sony Music Entertainment" .
<music/5> s:id "5"^^xsd:integer .
<music/5> s:title "Time Of The Season" .
<music/5> rdf:type s:Music .


<publ/1> s:name "Manning Publications Co." .
<publ/1> s:address "20 Baldwin Road PO Box 261 Shelter Island, NY 11964" .
<publ/1> s:id "1"^^xsd:integer .
<publ/1> rdf:type s:Publ .
```

Figure 7 – An instance of Sales

3. Mapping Formalism

Nesta seção, descreveremos a lei de formação dos mapeamentos entre a base relacional e a ontologia de domínio. Em geral, um mapeamento entre um esquema relacional S_A *(Schema Amazon_DB)* e uma ontologia alvo O_S *(Ontology Sales)* é caracterizado por um conjunto de expressões que definem os conceitos de O_S em termos dos conceitos de S_A, de forma que estes sejam semanticamente correspondentes [7].

Neste artigo, os mapeamentos são expressos através de um conjunto de regras, as quais são obtidas adaptando a estratégia proposta em [4] para que a Ontologia Fonte seja um Esquema Relacional. Tal estratégia divide-se em duas etapas principais: (i) *matching de conceitos*, que define um modelo de *matching* para representação do alinhamento entre os termos do vocabulário da ontologia alvo e os conceitos do esquema relacional fonte; e (ii) geração das regras de mapeamento, que induz um conjunto de regras obtido na etapa (i).

Antes de prosseguirmos com a definição dos mapeamentos, apresentamos algumas definições.

Definition 1: *OWL Extralite* é um dialeto de OWL que contém os construtores fundamentais presentes na maioria das linguagens para representação de ontologias, bem como nos modelos de dados conceituais, tais como UML. Assim, uma ontologia *OWL Extralite* é um par $O = (V, A)$, tal que:

i. ***V*** é o vocabulário da ontologia, composto de termos que podem ser classes, propriedades de tipo de dado e propriedades de objeto.

ii. ***A*** é o conjunto de axiomas que correspondem às restrições definidas sobre ***O*** que podem ser:

- **Restrições sobre classes**: subsunção (*rdfs:subclassOf*), disjunção (*owl:disjointWith*) e equivalência (*owl:equivalentClass*);
- **Restrições sobre propriedades**: cardinalidade mínima (*owl:minCardinality*) e máxima (*owl:maxCardinality*), domínio (rdfs*:domain*) e equivalência (*owl:equivalentClass*);

Como característica das propriedades, *OWL Extralite* admite a utilização dos seguintes construtores: *owl:inverseFunctionalProperty*, que captura a representação de chaves simples; *owl:inverseOf*, que indica o relacionamento inverso entre propriedades; e *owl:functionalProperty*, que determina que uma propriedade deve possuir, no máximo, um valor para cada objeto. Além disso, este dialeto suporta a definição de indivíduos (instâncias na ontologia).

In this section, let R, R_1,...,R_n be relation schemes of a relational schema S. Let $\mathbf{R}$, $\mathbf{R_1}$,...,$\mathbf{R_n}$ be relations over R_1,...,R_n, respectively.

Definition 2: Let *fk* be a foreign key of R_1 that references R_2. Then, we say that:

i. *fk* is a link from R_1 to R_2.

ii. fk^{-1}, the inverse of a *fk* , is a link from R_2 to R_1.

Definition 3: Sejam l_1, ... , l_n links entre as relações (R, R_1), (R_1, R_2), … (R_{n-1}, R_n), respectivamente. Assim, podemos definir: c = (R, l_1, ... , l_n) como sendo um caminho

referencial de R para R_n.

3.1. Concepts Matching

Na abordagem usada neste artigo, o matching entre os conceitos do esquema *Amazon_DB* e do esquema Sales é feita pelo usuário através de uma interface gráfica como mostrado na figura _.

No entanto, podemos representar o matching como feito em [4] adaptando para substituir a ontologia local pelo esquema relacional. Dessa forma, definimos um matching contextualizado de conceitos entre S_A e O_S como um matching cujas correspondências criadas podem ser representadas através de um conjunto finito S de sêxtuplas (v_1, e_1, r_1, v_2, e_2, r_2), tal que:

- Se v_1 é uma relação de S_A e v_2 é uma classe de O_S, então e_1 e e_2 são conceitos topo $\top$ e r_1 e r_2 são restrições que delimitam um subconjunto dos conceitos v_1 e v_2, respectivamente;
- Se v_1 é uma coluna de uma relação de S_A e v_2 é uma propriedade de O_S, então e_1 é uma relação de S_A e e_2 é uma classe de O_S, onde e_1 é a relação cujo esquema define v_1 como coluna e e_2 é subclasse do domínio, domínio próprio ou domínio restrito da propriedade v_2. Já r_1 e r_2 são restrições que delimitam um subconjunto das tuplas de e_1 e um subconjunto dos indivíduos de e_2, respectivamente.

No primeiro caso, (v_1, $\top$, r_1, v_2, $\top$, r_2) indica que uma tupla x de v_1 que satisfaz r_1 será reinterpretada como um indivíduo de v_2 que satisfaz r_2. No segundo caso, (v_1, e_1, r_1, v_2, e_2, r_2) indica que um tupla da relação e_1 que satisfaz a restrição r_1 e cujo valor da coluna v_1 é x, será interpretada como uma tripla (y, v_2, x) se y é um indivíduo de e_2 que satisfaz r_2, v_2 é uma propriedade de e_2 e x é o valor de v_2.

No que diz respeito à definição das restrições, consideramos que uma restrição r é uma tripla do tipo (p, o, v), onde: $\boldsymbol{p}$ é uma propriedade de tipo de dados de uma classe ou uma coluna de uma relação, $\boldsymbol{o}$ é uma operação de comparação e $\boldsymbol{v}$ é um valor constante atribuído a $\boldsymbol{p}$. Tal valor é utilizado, juntamente com o operador de comparação o, com o intuito de estabelecer um filtro que determine um subconjunto do conceito sobre o qual a restrição está sendo aplicada. Os operadores de comparação considerados neste artigo são: igual (=), maior (>), maior ou igual (>=), menor (<), menor ou igual (<=) e diferente (≠). O operador de igualdade pode ser aplicado tanto sobre strings quanto sobre valores numéricos, enquanto os demais operadores são permitidos somente sobre valores numéricos.

Finalmente, definimos que uma relação v_1 (em S_A) está *semanticamente relacionada* a uma classe v_2 (em O_S), tal relação é representada por (v_1 C^{sr} v_2), se [4]:

- (v_1, $\top$, r_1, v_2, $\top$, r_2) $\in S$;
- (v_1, $\top$, r_1, v_2', $\top$, r_2) $\in S$, onde v_2' é uma subclasse de v_2.

Caso contrário, definimos que v_1 não é semanticamente relacionada a v_2, representado por (v_1 $\not C^{sr}$ v_2).

Na Tabela 1, a linha 1 indica que a relação *a:Product* e a classe *s:Product* são correspondentes. Isso significa que uma tupla da relação será interpretada como um indivíduo da classe. A linha 6 indica que a coluna *a:title* da relação *a:Product* e a

propriedade de tipo de dado *s:title* da classe *s:Product* são correspondentes. Dessa forma, o valor da propriedade em *Sales* será o mesmo que está atribuído à coluna. Na linha 14 temos que a coluna *a:recname* e a propriedade s:recorder são correspondentes, porém suas respectivas relação e classe não estão semanticamente relacionadas. Esse último exemplo mostra um caso que necessita de uma condição especial de *matching*: um caminho referencial para relacionar corretamente os conceitos correspondentes. Como veremos nas próximas seções, todas essas correspondências poderão ser validadas e ajustadas.

	Amazon_DB			***Sales Ontology***		
#	v_1	e_1	r_1	v_2	e_2	r_2
1	*a:Product*	⊤	⊤	*s:Product*	⊤	⊤
2	*a:Book*	⊤	⊤	*s:Book*	⊤	⊤
3	*a:Music*	⊤	⊤	*s:Music*	⊤	⊤
4	*a:Publisher*	⊤	⊤	*s:Publ*	⊤	⊤
5	*a:id*	*a:Product*	⊤	*s:id*	*s:Product*	⊤
6	*a:title*	*a:Product*	⊤	*s:title*	*s:Product*	⊤
7	*a:idProd*	*a:Book*	⊤	*s:id*	*s:Book*	⊤
8	*a:title*	*a:Product*	⊤	*s:title*	*s:Book*	⊤
9	*a:idPubl*	*a:Book*	⊤	*s:pub*	*s:Book*	⊤
10	*a:idProd*	*a:Music*	⊤	*s:id*	*s:Music*	⊤
11	*a:title*	*a:Product*	⊤	*s:title*	*s:Music*	⊤
12	*a:name*	*a:Publ*	⊤	*s:name*	*s:Publ*	⊤
13	*a:address*	*a:Publ*	⊤	*s:address*	*s:Publ*	⊤
14	*a:recname*	*a:Recorder*	⊤	*s:recorder*	*s:Music*	⊤

Table 1 Concept Matching from *Amazon_DB* to *Sales* Ontology

3.2. Rule-based Mapping Formalism

Seja um átomo uma expressão do tipo $p(t_1,..., t_n)$, onde p é um símbolo de predicado e t_i são termos, tal que $i=1$ até n. Um termo pode ser uma constante, uma variável ou uma função n-ária aplicada sobre outros termos. Seja R uma linguagem de regras. Um mapeamento entre um esquema relacional e uma ontologia é especificado através de um conjunto de regras de mapeamento, cada uma das quais com a seguinte forma:

$$\Psi(w) \leftarrow \varphi_1(t_1), \ldots, \varphi_n(t_n), \text{ onde:}$$

- $\varphi_1(t_1), \ldots, \varphi_n(t_n)$, chamado de *corpo* da regra, é um átomo ou uma conjunção de átomos, onde $\varphi_i(t_i)$ é um conceito ou um relacionamento pertencente ao esquema fonte S_A ou, ainda, uma operação de comparação e t_i é uma seqüência de termos.
- $\Psi(w)$, chamado de cabeça da regra, é um átomo, onde pode ser um conceito ou um relacionamento na ontologia alvo O_S e w é uma seqüência de termos.

Mais especificamente, o formalismo de regras que utilizamos é uma extensão de *Datalog* que possibilita a construção explícita de *OIDs* [8]. Com isso, é possível a definição de *Skolem functions*, as quais são termos da forma $f(c_1,...,c_n)$, onde f é um símbolo funcional de aridade $n>0$ e $c_1,...,c_n$ são conceitos pertencentes a S_A. Neste trabalho, estas funções são utilizadas para a geração de *IRIs*, correspondentes aos objetos criados nas classes de O_S, a partir de uma ou mais colunas de S_A.

3.3. Generation of Mapping Rules

Nesta seção, será explanado como as regras de mapeamento são derivadas a partir das sêxtuplas discutidas anteriormente. De maneira geral, a geração de tais regras é guiada por uma série de casos, apresentados em [4], que podem ser divididos em dois grupos principais: geração de regras para as classes e geração de regras para as propriedades. Em ambas as situações, para cada entidade mapeada, consideramos as implicações em duas circunstâncias: (i) quando seus respectivos contextos estão alinhados e (ii) quando tais contextos não estão alinhados. Além disso, em cada caso enunciado, há um exemplo ilustrativo que referencia o esquema relacional, a ontologia e os alinhamentos entre eles.

Seja S um matching contextualizado de vocabulários entre um esquema relacional fonte S_S e uma ontologia alvo O_T. Seja ainda $s = (v_1, e_1, r_1, v_2, e_2, r_2)$ uma sêxtupla pertencente a S. O conjunto M das regras de mapeamento entre S_S e O_T é derivado de acordo com os casos definidos a seguir.

Case 1. Se $ss{:}v_1$ é uma relação e $ot{:}v_2$ é uma classe e $ot{:}r_2$ é a classe topo $\top$, ou seja, não há restrição aplicada sobre a classe $ot{:}v_2$ da ontologia alvo, então:

$M = \{ot{:}v_2(x) \leftarrow ss{:}v_1(x), ss{:}r_1(x)\} \cup$

$\{ot{:}u(x) \leftarrow ss{:}v_1(x), ss{:}r_1(x),$ para cada superclasse u de $v_2\}$

Case 2. Se $ss{:}v_1$ é uma relação e $ot{:}v_2$ é uma classe e $ot{:}r_2$ é uma restrição da forma $ot{:}p_2\ op\hat{}ot{:}c_2$, onde $ot{:}p_2$ é uma propriedade de tipo de dados, $ot{:}c_2$ é uma constante e $op\hat{}$ é um dos seguintes operadores de comparação: =, >, >=, <, <=, ≠, então:

$M = \{ot{:}v_2(x) \leftarrow ss{:}v_1(x), ss{:}r_1(x)\} \cup$

$\{ot{:}u(x) \leftarrow ss{:}v_1(x), ss{:}r_1(x),$ para cada superclasse u de $v_2\} \cup$

$\{ot{:}p_2(x,y) \leftarrow ss{:}v_1(x), ss{:}r_1(x),$ y $op\ ss{:}c_2\}$

Case 3. Se $ss{:}v_1$ é uma coluna e $ot{:}v_2$ é uma propriedade de tipo de dado ou $ss{:}v_1$ é uma fk e $ot{:}v_2$ é uma propriedade de objeto tal que $ss{:}e_1\ \mathsf{C}^{sr}\ ot{:}e_2$, isto é, e_1 é semanticamente relacionada a e_2, conforme definido na Seção 3.1, então:

$M = \{ot{:}v_2(x,y) \leftarrow ss{:}v_1(x,y), ss{:}e_1(x), ss{:}r_1(x)\}$

Case 4. Se $ss{:}v_1$ é uma coluna e $ot{:}v_2$ é uma propriedade de tipo de dado ou $ss{:}v_1$ é uma fk e $ot{:}v_2$ é uma propriedade de objeto tal que $ss{:}e_1\ \not\subset^{sr}\ ot{:}e_2$ e há um caminho referencial $c = ss{:}fk_1(x, x_1), ss{:}fk_2(x_1, x_2), \ldots, ss{:}fk_n(x_{n-1}, z)$, no esquema fonte S_S, então:

$M = \{ot{:}v_2(x,y) \leftarrow ss{:}fk_1(x, x_1), ss{:}fk_2(x_1, x_2), \ldots, ss{:}fk_n(x_{n-1}, z), ss{:}v_1(z,y), ss{:}e_1(z), ss{:}r_1(z)\}$

Case 5. Se $ss{:}v_1$ é uma coluna e $ot{:}v_2$ é uma propriedade de tipo de dado ou uma propriedade de objeto tal que $ss{:}e_1\ \not\subset^{sr}\ ot{:}e_2$, mas há um caminho de propriedades θ, na ontologia alvo O_T, então:

$M = \{ot{:}v_2(f(y),y) \leftarrow ss{:}v_1(x,y), ss{:}e_1(x), ss{:}r_1(x),$ onde $ot{:}v_2$ é uma propriedade de tipo de dados $\} \cup$

$\{ot{:}e_2(f(y)) \leftarrow ss{:}v_1(x,y), ss{:}e_1(x), ss{:}r_1(x),$ onde $ot{:}e_2$ é uma classe$\} \cup$

$\{ot{:}p_2(x,f(y)) \leftarrow ss{:}v_1(x,y), ss{:}e_1(x), ss{:}r_1(x),$ onde $ot{:}p_2$ é uma propriedade de objeto$\}$

Case 6. Se $ss{:}v_1$ é uma coluna e $ot{:}v_2$ é uma propriedade de tipo de dado ou $ss{:}v_1$ é uma *fk* e $ot{:}v_2$ é uma propriedade de objeto tal que $ss{:}e_1 \not\subset^{sr} ot{:}e_2$ e há um caminho referencial $c = ss{:}fk_1(x, x_1), ss{:}fk_2(x_1, x_2), \ldots, ss{:}fk_n(x_{n-1}, z)$, no esquema fonte S_S e um caminho de propriedades θ, na ontologia alvo O_T, então:

M = {$ot{:}v_2$(f(y),y) ← $ss{:}v_1$(z,y), $ss{:}e_1$(z), $ss{:}r_1$(z), onde $ot{:}v_2$ é uma propriedade de tipo de dados } U

{$ot{:}e_2$(f(y)) ← $ss{:}v_1$(z,y), $ss{:}e_1$(z), $ss{:}r_1$(z), onde $ot{:}e_2$ é uma classe} U

{$ot{:}p_2$(x,f(y)) ← $ss{:}fk_1(x, x_1), ss{:}fk_2(x_1, x_2), \ldots, ss{:}fk_n(x_{n-1}, z), ss{:}v_1$(z,y), $ss{:}e_1$(z), $ss{:}r_1$(z), onde $ot{:}p_2$ é uma propriedade de objeto}

Este caso corresponde a uma combinação dos casos 4 e 5.

4. Specifying RDF Views

5. Mapping Assertions to RDF Views

6. R2RML Mapping Generation

7. Conclusions

8. References

1. Das Souripriya, Sundara Seema, Cyganiak Richard, R2RML: RDB to RDF Mapping Language. http://www.w3.org/TR/r2rml/, 2012.
2. Lassila Ora, Swick Ralph R., Resource Description Framework (RDF) Model and Syntax Specification. http://www.w3.org/TR/PR-rdf-syntax/, 1999.
3. Prud'hommeaux Eric, Seaborne Andy, SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2008.
4. Sacramento, E., Vidal, V. M., Macêdo, J. A., Lóscio, B. F., Lopes, F. L. R., Casanova, M. A., and Lemos, F. (2010a): Towards automatic generation of application ontologies, Journal of Information and Data Management (JIDM), 1(3):535–551..
5. Hernandez, M.A., Miller, R.J., Haas, L., Yan, L., Ho, C.T.H., Tian, X., Clio: A semi-automatic tool for schema mapping. In: International Conference on Management of Data, Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Page 607, 2001.
6. Yu, C., Popa, L., Constraint-Based XML Query Rewriting For Data Integration. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of data, SESSION: Research sessions: Data Integration, Pages 371–382, 2004.
7. Leme, L. A. P. P. (2009). Conceptual Schema Matching based on Similarity Heuristics. PhD thesis, PUC - Rio, Brazil.
8. Hull, R. and Yoshikawa, M. (1990). ILOG: Declarative creation and manipulation of object identifiers. In Proceedings of the 16th International Conference on Very Large Databases (VLDB), pages 455–468, Brisbane, Australia.