

# TRANSFORMATION FROM SEMANTIC DATA MODEL TO RDF

**Daniel Siahaan**

Jurusan Teknik Informatika,  
Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember  
Kampus ITS, Jl. Raya ITS, Sukolilo – Surabaya 60111, Telp. + 62 31 5939214, Fax. + 62 31 5913804  
Email: daniel@its-sby.edu

## ABSTRAK

*There have been several efforts to use relational model and database to store and manipulate Resource Description Framework (RDF). They have one general disadvantage, i.e. one is forced to map the model of semantics of RDF into relational model, which will end up in constraints and additional properties, such as, validating each assertion against the RDF schema which also stored as a triplets table. In this paper, we introduce Semantic Data Model as a proposed data model language to store and manipulate Resource Description Framework. This study also tries to prescribe the procedure on transforming a semantic data model into a RDF data model.*

**Keywords:** *Semantic Data Model, Resource Description Framework.*

## 1. INTRODUCTION

Until recently, several efforts have been taken to use relational model and database to store and manipulate Resource Description Framework [6], namely specs loyal, explicit models, hashed with origin, and the naïve approach. The "specs loyal" approach [2], which was proposed by Jonas Liljegren, attempts to provide a compact way of implementing every detail in the RDF model and schema specifications. Its database schema is implemented in Postgres. The "explicit models" approach [3], which was proposed by Brian McBride, treats models explicitly and makes use of views. Its database schema is implemented on Oracle. The "hashed with origin" approach [4], was proposed by Sergey Melnik, where it make used of CRC64 hash values to treat models explicitly. Its database schema is implemented in MySQL. Or the "naïve" approach, where all triplets are stored in one table that has three fields: Property, Resource, and Value.

These approaches have one general disadvantage, i.e. one is forced to map the model of semantics of RDF into relational model, which will end up in constraints and additional properties, such as, validating each assertion against the RDF schema which also stored as a triplets table.

The Semantic Data Modeling (SDM) [8] is built on the concept of semantics, which is also the concept used in RDF. This similarity enables them to be mapped into each other. Both, SDM and RDF, make use of semantics concept. Therefore the mapping will be done more smooth and with less constraints and additional properties. Another advantage of this approach is we can use the Xplain

system [8], which built on SDM, as the storage system for the RDF resources, since Xplain has advantages over relational databases.

The main purpose of this paper is described how a data model, i.e. semantic data model, can be mapped into RDF and whether both are adequate to represent each other. This paper is organized as follows. Section 2 gives a brief overview of the SDM. Section 3 describes the transformation from SDM into RDF. Finally, the last section gives a summary and lists several conclusions.

## 2. SEMANTIC DATA MODELING

The concept of semantics is the main issue in Semantic Data Modeling (SDM) [8]. It is all about interrelationships between formal definitions and their relationships with the real world that being modeled. But in SDM, only the interrelationships between formal definitions (data), which form information, are formalized in the conceptual model. The following are the basic concepts behind the SDM:

- A conceptual model consists only of positive statements (assertions). It means that a statement must be true since it should correspond to the reality.  
Therefore, for example, data of a person who is not working a company will never be stored in the table employee.
- Each type definition is unique, meaning that there is no different type definition with the same name of the same collection of attributes.
- An attribute is related to one and only one type, and a type is related to at least one attribute. An

attribute value is related to one and only one instance in the related type.

- An object can be either a type or an instance of a type, depending on the point of view.

A **type** is a set of objects that have definite properties.

**Attributes** of a type are the properties that aggregate that type.

An **instance** of a type is an object that has the properties of that type.

For example:

*type* employee = name, sex, department

*type* department = name, location

The definition of the model has not yet contained information about any **base type**, which is the type of which the attributes are no longer relevant. The base types appeared in the above model can be defined as the following:

*base* name (A20)

*base* sex (A1)

*base* location (A40)

### 2.1. AGGREGATION

A type (e.g. employee) is defined as a collection (aggregation) of characteristics (name, birth\_date, address, department, etc) called attributes. It also can be stated that an attribute is part of a type definition.

The semantic model shown in Figure 2 can be written as the following type definitions:

*type* employee = name, birth\_date, address, department

*type* department = ...

### 2.2. SPECIALIZATION AND GENERALIZATION

Type specialized\_A is a specialization of type A, if type specialized\_A is type A with at least one additional attribute. And the counterpart of specialization is generalization. The semantic model shown in Figure 3 can be written as the following type definitions:

*type* A = name, starting\_date

*type* specialized\_A = [A], ending\_date

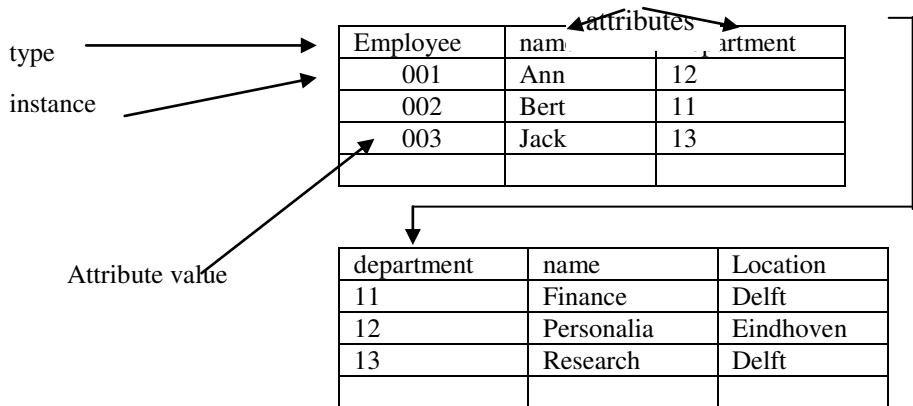


Figure 1. Type employee and department.

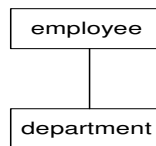


Figure 2. Aggregation

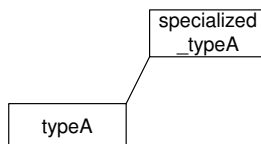







Figure 3. Specialization - Generalization

### 3. TRANSFORMATION TO RDF

The main purpose of this section is to describe the mechanism to transform a semantic data model into data models with the standard Semantic Web languages, i.e. Resource Description Framework (RDF). In this section we use the following conventions for describing the RDF Graph:

	instance of rdfs:Literal
	rdfs:subClassOf (Class) rdfs:subPropertyOf (Property)
	rdf:type
	rdfs:domain
	rdfs:range

#### 3.1. SEMANTIC DATA MODELING TO RESOURCE DESCRIPTION FRAMEWORK

##### 3.1.1. Aggregation

A type (i.e. employee) is defined as a combination (aggregation) of a number of characteristics (name, address, department, etc) called attributes. It also can be stated that an attribute is a part of a type definition.

The semantic data model shown in Figure 4.a. can be written as the following type definition:

*type* employee = department,...

The above type definition can be written in RDF as follows:

```
<rdf:Description
rdf:about="http://a.b.c/type#employee">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#department">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#its_department">
  <rdf:type rdf:resource="&22-rdf-syntax-
ns;#Property"/>
  <rdfs:domain
rdf:resource="http://a.b.c/type#employee"/>
  <rdfs:range
rdf:resource="http://a.b.c/type#department"/>
</rdf:Description>.
```

The RDF data model in Figure 4.b. cannot represent the semantic data model in Figure 4.a

flawlessly. The property *its\_department* cannot model the N-to-1 relation between type *employee* and department as viewed in the semantic data model shown in Figure 4.a. This is because in RDF, any instance of *rdf:Property* represents an M-to-N relation, and RDF does not provide any mechanism to define cardinality of a property.

Consider that the *employee* has a base type name, as shown in the following semantic data model definitions:

*base* name (A20)  
*type* employee = name,...

As shown in Figure 5, it can be defined as the following RDF data model:

```
<rdf:Description
rdf:about="http://a.b.c/type#employee">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#its_name">
  <rdf:type rdf:resource="&22-rdf-syntax-
ns;#Property"/>
  <rdfs:domain
rdf:resource="http://a.b.c/type#employee"/>
  <rdfs:range rdf:resource="http://.../22-rdf-
ns#Literal"/>
</rdf:Description>.
```

The above definition shows that an attribute of a type is defined in the same manner regardless whether it is a *base* attribute or not.

Consider a system where an *employee* works on some projects, and a project is done by several employees. This means that type *employee* and type *project* have M-to-N relation. The system is defined as the following:

*type* workon = employee, project  
*type* employee = name, ..  
*type* project = name,...

The above semantic data model can be transformed into two different RDF data models as shown in Figure 7 and Figure 8.

##### Alternative 1.

```
<rdf:Description
rdf:about="http://a.b.c/type#employee">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#project">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
```

```

<rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#workon">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#its_employee">
  <rdf:type rdf:resource="&22-rdf-syntax-
ns;#Property"/>
  <rdfs:domain
rdf:resource="http://a.b.c/type#workon"/>

```

```

<rdfs:range
rdf:resource="http://a.b.c/type#employee"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#its_project">
  <rdf:type rdf:resource="&22-rdf-syntax-
ns;#Property"/>
  <rdfs:domain
rdf:resource="http://a.b.c/type#workon"/>
  <rdfs:range
rdf:resource="http://a.b.c/type#project"/>
</rdf:Description>.

```

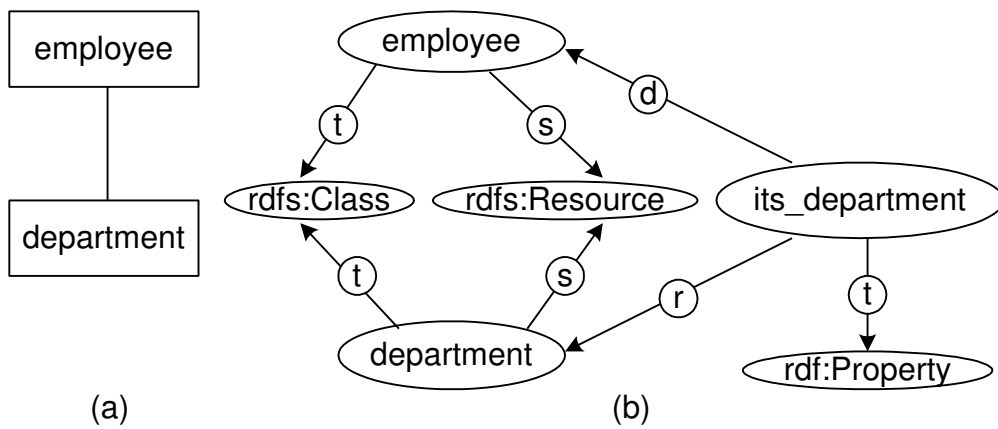


Figure 4. An aggregation (a) describe in RDF (b).

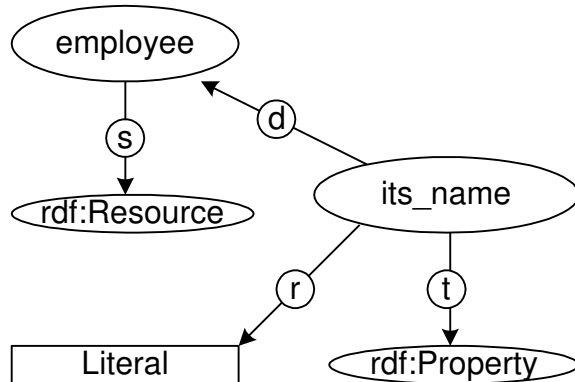


Figure 5. A base type in RDF/RDF Schema

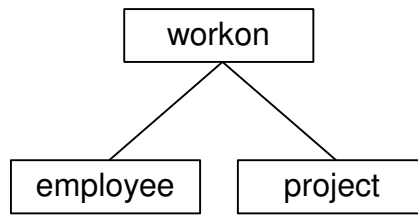


Figure 6. Employee and Project: M-to-N relation

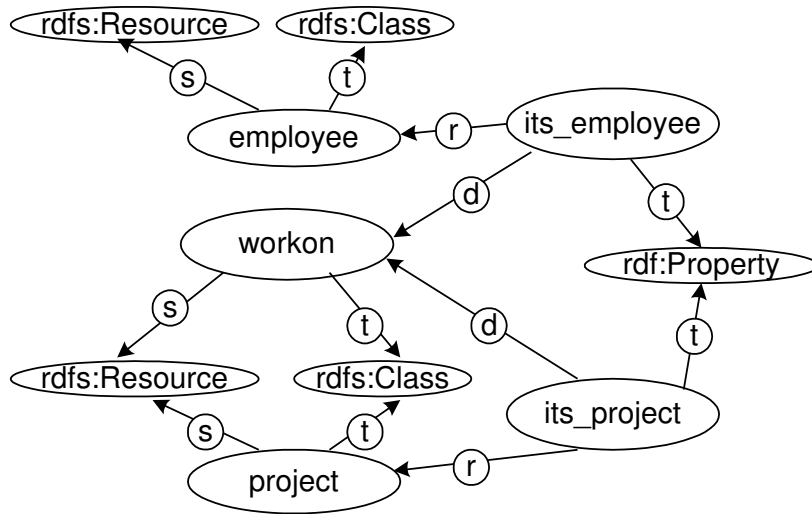


Figure 7. First alternative

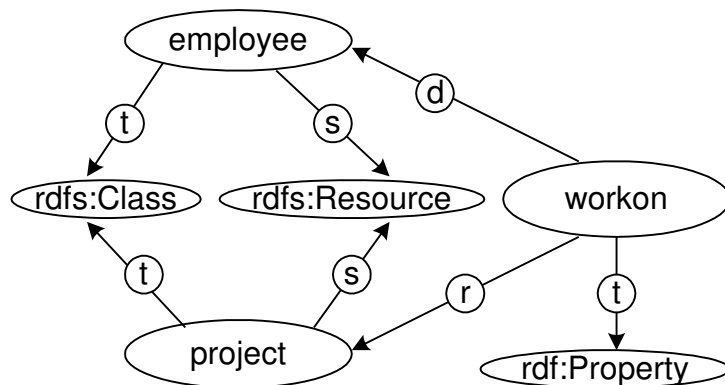


Figure 8. Second alternative

**Alternative 2.**

```

<rdf:Description
rdf:about="http://a.b.c/type#employee">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>
  
```

```

<rdf:Description
rdf:about="http://a.b.c/type#project">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#workon">
  
```

```
<rdf:type rdf:resource="&22-rdf-syntax-ns:#Property"/>
<rdfs:domain
rdf:resource="http://a.b.c/type#employee"/>
<rdfs:range
rdf:resource="http://a.b.c/type#project"/>
</rdf:Description>
```

The first RDF data model alternative has the following characteristics:

- The N-to-1 relation between resource workon and other resources (employee, project, and status) can not be satisfied, since any instance of rdf:Property represents M-to-N relation.
- It assumes that every type in the semantic data model is transformed as an instance of rdfs:Class.

The second RDF data model alternative has the following characteristics:

- It really represents the M-to-N relation between *employee* and *project* and at the same time reducing the need to create bigger model in RDF.
- Since the model is smaller than the first alternative, therefore the data will also be more compact. It also means the data is easier to manage and the query construction is simpler.
- It assumes that every type in the semantic data model is transformed as an instance of rdfs:Class, except those types that represent M-to-N relations between other two types. These types are represented as instances of rdf:Property.

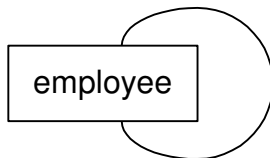


Figure 9. Recursive type

Consider a recursive type shown in Figure 9, which defines as the following type definitions:

type employee = [manager\_employee], ...

In RDF/RDF Schema, the type manager\_employee is defined in the same way as the other attributes are, with one significant difference: the range and the domain values of its\_manager\_employee property point to the same resource, which is the employee. Therefore the semantic data model can be represents in the following RDF data model:

```
<rdf:Description
rdf:about="http://a.b.c/type#employee">
<rdf:type rdf:resource="&rdf-schema:#Class"/>
<rdfs:subClassOf rdf:resource="&rdf-schema:#Resource"/>
</rdf:Description>
```

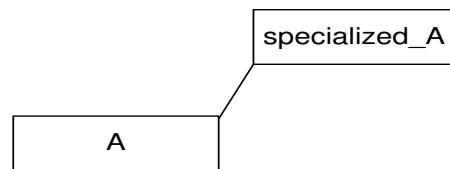
```
<rdf:Description
rdf:about="http://a.b.c/type#its_manager_employee"
>
<rdf:type rdf:resource="&22-rdf-syntax-ns:#Property"/>
<rdfs:domain
rdf:resource="http://a.b.c/type#employee"/>
<rdfs:range
rdf:resource="http://a.b.c/type#employee"/>
</rdf:Description>
```

This solution also rises a problem of incorrect relation cardinality with the fact that rdf:Property represents M-to-N relation instead of N-to-1 relation.

### 1.1.1. Specialization and Generalization

Type specialized\_A is a specialisation of type A, if type specialized\_A is a type A with one or more additional attributes. And the counterpart of specialization is generalization.

Figure 10. Specialization - Generalization



RDF provides rdfs:subClassOf property to model the specialization of semantic data model as shown in the Figure 11.

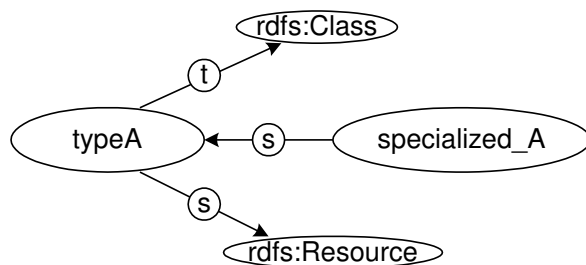


Figure 11. RDF diagram of Specialization- Generalization

Consider that type hotel is a specialization of type publichouse. This statement can be modeled as the following type definitions:

type publichouse = name...  
type hotel = [publichouse], stars

And the above type definitions can be described as the following RDF data model:

```
<rdf:Description
rdf:about="http://a.b.c/type#publichouse">
```

```

<rdf:type rdf:resource="&rdf-schema;#Class"/>
<rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#its_name">
  <rdf:type rdf:resource="&22-rdf-syntax-
ns;#Property"/>
  <rdfs:domain
rdf:resource="http://a.b.c/type#publichouse"/>
  <rdfs:range rdf:resource="http://a.b.c/type#Name"/>
</rdf:Description>

<rdf:Description rdf:about="http://a.b.c/type#hotel">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf
rdf:resource="a.b.c/type#publichouse"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#its_stars">
  <rdf:type rdf:resource="&22-rdf-syntax-
ns;#Property"/>
  <rdfs:domain
rdf:resource="http://a.b.c/type#hotel"/>
  <rdfs:range rdf:resource="http://.../22-rdf-syntax-
ns;#Literal"/>
</rdf:Description>.

```

Consider the following type definitions:

```

type human = ...
type male = [human], ...
type female = [human]...

```

By nature, a human can only be a male or a female. In semantic data modeling, the above definitions clearly restrict the possibility that an instance of type male is also an instance of type female. In RDF/RDF Schema, it is possible that a resource is instances of more than one class. But there has not yet a property that can define that a class, e.g. class male, is a disjoint of another class, e.g. class female. Therefore the previous model cannot be described in RDF flawlessly. In RDF, that model can be described as the following RDF data model:

```

<rdf:Description
rdf:about="http://a.b.c/type#human">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf rdf:resource="&rdf-
schema;#Resource"/>
</rdf:Description>

```

```

<rdf:Description rdf:about="http://a.b.c/type#male">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf
rdf:resource="a.b.c/type#human"/>
</rdf:Description>
<rdf:Description
rdf:about="http://a.b.c/type#female">
  <rdf:type rdf:resource="&rdf-schema;#Class"/>
  <rdfs:subClassOf
rdf:resource="a.b.c/type#human"/>
</rdf:Description>.

```

#### 4. CONCLUSION

As can be seen already, RDF data models cannot flawlessly represent Semantic Data Model. There are three reasons why this is the case. First, the properties in RDF can only represent M-to-N relations. Second and third, a resource can be instances of more than one class, and there is no mechanism available to constraint it.

The solution would be to have a mechanism to define cardinality of the property in RDF, and to have richer (algebra) primitives that allow more expressive class expressions, such as disjoint, union, and complement. And these are what DAML+OIL [1] and OWL [5] are doing. The future work will study whether DAML+OIL indeed provide enough data modeling primitives to express the semantic data model and how scalable is the DAML+OIL data model.

#### 5. REFERENCE

1. DAML+OIL, "<http://www.daml.org/2001/03/daml+oil-index.html>", March 2001.
2. Jonas Liljgren, "Description of the Database Implementation", <http://www-db.stanford.edu/~melnik/rdf/db-jonas.html>, 2000.
3. Brian McBride, "RDF Database Tables", <http://lists.w3.org/Archives/Public/www-rdf-interest/2000May/0094.html>, 2000.
3. Sergey Melnik, "RDF Resource", <http://www-db.stanford.edu/~melnik/rdf/index.html>, 2001.
4. "Requirements for a Web Ontology Language", W3C WebOnt Working Group. <http://www.w3.org/TR/webont-req>, 2002.
5. "Resource Description Framework", W3C, <http://www.w3.org/RDF>, 2002.
6. "Resource Description Framework Vocabulary Description Language version 1.0: RDF Schema", W3C, <http://www.w3.org/TR/rdf-schema>, 2002.