Construction of RDF(S) from UML Class Diagrams

Qiang Tong¹, Fu Zhang²∗ and Jingwei Cheng²

¹ Software College, Northeastern University, Shenyang, China
² College of Information Science and Engineering, Northeastern University, Shenyang, China

RDF (Resource Description Framework) and RDF Schema (collectively called RDF(S)) are the normative language to describe the Web resource information. How to construct RDF(S) from the existing data sources is becoming an important research issue. In particular, UML (Unified Modeling Language) is being widely applied to data modeling in many application domains, and how to construct RDF(S) from the existing UML models becomes an important issue to be solved in the context of Semantic Web. By comparing and analyzing the characteristics of UML and RDF(S), this paper proposes an approach for constructing RDF(S) from UML and implements a prototype construction tool. First, we give the formal definitions of UML and RDF(S). After that, a construction approach from UML to RDF(S) is proposed, a construction example is provided, and the analyses and discussions about the approach are done. Further, based on the proposed approach, a prototype construction tool is implemented, and the experiment shows that the approach and the tool are feasible.

Keywords: RDF (resource description framework), RDF Schema, UML (Unified Modeling Language), construction

1. Introduction

Semantic Web is an extension of the current Web, in which data is given a well-defined meaning by representing it in RDF (Resource Description Framework), RDF vocabulary description language RDF Schema, and OWL (Web Ontology Language) (Berners-Lee et al., 2001; Horrocks et al., 2003; RDF-W3C, 2014). RDF and RDF Schema (collectively called RDF(S)) are the W3C recommendation normative language to describe the Web resource information and their semantics (RDF-W3C, 2014). This semantic enrichment allows data to be shared, exchanged or integrated from different sources and enables applications to use data in different contexts. Currently, formal adoption of RDF(S) by W3C stimulates their utilization in many areas and by many organizations. In spite of an increasing acceptance of RDF(S), this is still a new technology, and most information resources are not available in RDF(S)-format. Therefore, how to construct RDF(S) from the existing information resources became an important research issue.

To this end, many approaches have been developed to construct RDF(S) from some data sources. Approaches for constructing RDF(S) from relational databases were proposed in (Sequeda et al., 2012; Mallede et al., 2013; Korotkiy & Top, 2004; Krishna, 2006; Michel et al., 2013). How to transform XML documents into RDF-format data was investigated in (Thuy et al., 2007; Klein, 2002; Kumar & Babu, 2013). Moreover, a proposal for building RDF from semi-structured legal documents was presented in (Amato et al., 2008). The rules of constructing RDF from spreadsheets were proposed in (Han et al., 2008).

It can be found that these existing approaches focus on establishing the direct mappings from the data sources to RDF(S), e.g., translating tables and columns of relational databases into classes and properties of RDF(S), respectively. As we all know, a relational schema often includes details that have nothing to do with the domain expressed: e.g., optimization sometimes require adaptations or lead to technical tricks like de-
normalized forms, many-to-many relations are expressed by a table with several foreign keys, etc. Thus, the relational schema is often a bad representation of the domain. Conversely, we can expect an UML (Unified Modeling Language developed by UML Object Management Group) model to express nothing else but actual concepts of the domain being modeled. Therefore, mapping of UML-to-RDF(S) may be a good way to derive an RDF(S) model that is as close as possible to the domain. In particular, to overcome the limitations of classical database models (e.g., relational models and ER models (Brown, 2001; Yeung & Brent Hall, 2007)), the Unified Modeling Language (UML), which is developed in response to the complex data modeling requirements and integration with object-oriented programming systems, has been the de facto standard and is extensively used in the design phase of data modeling in many areas of software and knowledge engineering. Consequently, many CASE tools are available on the market, and such tools provide a user-friendly environment for editing, storing, and accessing multiple UML models (Engels et al., 2000; Aguilar-Saven, 2004; UML Object Management Group). Therefore, to be able to use this data in a semantic context, it has to be mapped into RDF(S).

Currently, some proposals establish correspondences between UML and the Semantic Web (Cranefield, 2001; Baclawski et al., 2001; Guizzardi et al., 2004; Cranefield et al., 2001). But, to our best knowledge, little research has been done in constructing RDF(S) from UML. Only a brief discussion of the relationships between RDF Schema and UML was done in Chang (1998), but the approach for constructing RDF(S) from UML is missed. Moreover, Kim et al. (2005) investigated translations from RDF Schema to UML, but our work is to investigate how to translate UML into RDF(S). The purpose of our work is different from the work in (Kim et al., 2005).

Based on the observations above, how to construct RDF(S) from UML becomes an important issue to be solved in the context of the Semantic Web. In this paper, we propose a complete approach and develop an automated tool for constructing RDF(S) from UML. First, by comparing and analyzing the characteristics of UML and RDF(S), we propose formal definitions of UML and RDF(S), respectively. On this basis, an approach for constructing RDF(S) from UML is proposed, a full construction example is provided, and the analyses and discussions about the approach are done. Further, based on the proposed approach, a prototype construction tool is implemented, and the experiment shows that the approach is realistic.

The remainder of this paper is organized as follows. Section 2 gives formal definitions of UML and RDF(S). Section 3 proposes an approach for constructing RDF(S) from UML, provides a construction example, and makes the analyses and discussions about the approach. Section 4 describes an implementation. Section 5 shows conclusions and future works.

2. Formalizations of UML and RDF(S)

In order to well establish correspondences between UML and RDF(S), it is necessary to give formal definitions of UML and RDF(S). In particular, the formal definition of UML should include major notions of UML, which will then be transformed into corresponding elements of RDF(S). In this section, we propose formal definitions of UML and RDF(S), respectively.

2.1. Formalization of UML

Unified Modeling Language (UML) is a language initially proposed as a unification of several different visual notations and modeling techniques used for systems design, which has become a de-facto industry standard for modeling applications in data and software engineering communities (UML Object Management Group). From the data modeling point of view, the UML class diagram, which is used to describe the static structure of the information of an application domain, will be mainly considered in our work.

In general, the main notions in an UML class diagram include class, attribute, association, generalization, aggregation, and dependency as follows:

- **Class and Attribute**: Objects having the same attributes are gathered into classes that are organized into hierarchies. A class is defined by a set of attributes and their admissible values.
• **Association**: An association represents the relationship between \( n \) classes. The participation of a class in an association is called a *role* and has a unique name. Moreover, an association may have a related *association class* that describes attributes of the association. Names of associations or association classes are unique in an UML class diagram.

• **Generalization**: A generalization is a taxonomic relationship between a more general classifier named *superclass* and a more specific classifier named *subclass*. The subclass inherits all attributes and methods of the superclass, overrides some attributes and methods of the superclass, and defines some new attributes and methods.

• **Aggregation**: A particular kind of binary associations are aggregation. In an UML class diagram, an aggregation represents the *part-whole* relationship between a class named *aggregate* and a group of classes named constituent parts. The constituent parts can exist independently. In addition, an aggregation has no associated class.

• **Dependency**: A dependency is a relationship between a source class and a target class, which denotes that existence of the target class is dependent of the source class. In particular, dependency between the source class and the target class is only related to the classes themselves and does not require a set of instances for its meaning.

Figure 1 shows notations of the notions mentioned in an UML class diagram as mentioned above.

On the basis of the notions mentioned above, in the following we propose a formal definition of UML class diagrams.

Firstly, for two finite sets \( X \) and \( Y \), we call a function from a subset of \( X \) to \( Y \) an *\( X \)-labeled tuple over \( Y \)*. The labeled tuple \( T \) that maps \( x_i \in X \) to \( y_i \in Y \), for \( i \in \{1, \ldots, n\} \), is denoted \( T(X, Y) = [x_1 : y_1, \ldots, x_n : y_n] \).

**Definition 1 (UML class diagrams).** An UML class diagram is a tuple \( F_{UML} = (L, attC, attSC, ass, agg, gene, dep) \), where:

- \( L = C \cup A \cup S \cup Sc \cup R \cup D \) is a finite alphabet partitioned into a set \( C \) of *class* identifiers, a set \( A \) of *attribute* identifiers, a set \( S \) of *association* identifiers, a set \( Sc \) of *association class* identifiers, a set \( R \) of *role* identifiers, and a set \( D \) of *datatype* identifiers;

- \( attC : C \rightarrow T(A, D) \) is a function that maps each class identifier \( c \in C \) to an \( A \)-labeled tuple over \( D \), i.e., \( attC(c) = [\ldots, a_k : d_k, \ldots] \), where \( a_k \in A, d_k \in D \);

- \( attSC : Sc \rightarrow T(A, D) \) is a function that maps each association class identifier \( sc \in Sc \) to

![Figure 1. Notations of the notions in UML class diagrams.](image-url)
an A-labeled tuple over D, i.e., \( att_{SC}(s_c) = [..., d'_k : d_k, ...] \), where \( d'_k, d_k \in A, d'_k \in D \);  

- \( ass \) is a function that maps each association identifier \( s \in S \) and association class identifier \( s_c \in S_c \) to an \( R \)-labeled tuple over \( C \), i.e., \( ass(s) = [..., r_k : c_k, ...] \) and \( ass(s_c) = [..., r'_k : c'_k, ...] \), where \( r_k, r'_k \in R, c_k, c'_k \in C \); 

- \( agg \subseteq c \times (c_1, c_2, ..., c_n) \) is a relation that models the \textit{generalization} between a subclass \( c_1 \) and a superclass \( c_2 \), where \( c_1, c_2 \in C \);  

- \( gene \subseteq c_1 \times c_2 \) is a relation that models the \textit{generalization} between a target class \( c'_1 \) and a source class \( c'_2 \), where \( c'_1, c'_2 \in C \);  

- \( dep \subseteq c'_1 \times c'_2 \) is a relation that models the \textit{dependency} between a target class \( c'_1 \) and a source class \( c'_2 \), where \( c'_1, c'_2 \in C \).

### 2.2. Formalization of RDF(S)

\textit{Resource Description Framework (RDF) (RDF-W3C, 2014)} is a framework for expressing Web resource information. RDF is intended for situations in which information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications. The basic idea of RDF is:

- Anything is called \textit{“resource”}, which can be a class, a property or an individual. A \textit{resource} can be identified by a URI (Uniform Resource Identifier). For example, \texttt{http://purl.org/dc/elements/1.1/creator} is a URI of a resource, where \texttt{http://purl.org/dc/elements/1.1/} is a namespace, and \textit{creator} is the name of a term;  

- A resource may have some \textit{“properties”}, and these properties may have \textit{“values”}, which may be \textit{literal values} (e.g., string or integer) or \textit{other resources};  

- The relationships among resources, properties and values can be described by \textit{“statements”}, which always have the following triple structure: \texttt{<subject predicate object>}. Specifically, the part identifying the thing the statement is about is called the \textit{“subject”}. The part identifying the property or characteristic of the subject that the statement specifies is called the \textit{“predicate”}, and the part identifying the value of that property is called the \textit{“object”}. So, taking the English statement: \texttt{http://www.example.org/index.html} has a \textit{creator} whose value is \textit{John Smith} the RDF terms for the various parts of the statement are:  
  
  \begin{enumerate}
  
  \item the \textit{subject} is the URI \texttt{http://www.example.org/index.html}  
  \item the \textit{predicate} is the word \textit{“creator”}  
  \item the \textit{object} is the phrase \textit{“John Smith”}.
  \end{enumerate}

RDF provides a way to make statements about resources, but it cannot define semantic characteristics of data. For example, RDF cannot state that the URI \texttt{http://www.example.org/friendOf} can be used as a property and that the subjects and objects of \texttt{http://www.example.org/friendOf} triples must be resources of class \texttt{http://www.example.org/Person}. Instead, such constraints may be described as an RDF vocabulary, using extensions to RDF such as the RDF Vocabulary Description Language — RDF Schema.

\textit{RDF Schema} uses the notion of \textit{“class”} to specify categories that can be used to classify resources. The relation between an instance and its class is stated through the \textit{“type”} property. With RDF Schema one can create hierarchies of classes and \textit{“sub-classes”} and of properties and \textit{“sub-properties”}. Type restrictions on the subjects and objects of particular triples can be defined through \textit{“domain”} and \textit{“range”} restrictions. An example of a domain restriction was given above: subjects of \textit{“friendOf”} triples should be of class \textit{“Person”}. Also, we can define the class \textit{“GraduateStudent”} as a subclass of the class \textit{“Student”}, and Mary is an instance of the class \textit{“GraduateStudent”}, as illustrated in the following RDF Schema:

\begin{verbatim}
<rdfs:Class rdf:ID = "Student"/>
<rdfs:Class rdf:ID = "GraduateStudent">
  <rdfs:subClassOf rdf:resource = "#Student"/>
</rdfs:Class>
<rdfs:Description rdf:ID = "Mary">
  <rdfs:domain rdf:resource = "#GraduateStudent"/>
  <rdfs:range rdf:resource = "http://www.example.org/friendOf"/>
</rdfs:Description>
\end{verbatim}

RDF and RDF Schema are collectively called \textit{RDF(S)}. Detailed introductions about...
RDF(S) can be found in (RDF-W3C, 2014). By summarizing the elements of RDF(S), in the following we give a brief formal definition of RDF(S).

**Definition 2 (RDF(S)).** A set of RDF(S) elements $\mathcal{R}$ can be represented as $\mathcal{R} = \mathcal{R}_S, \mathcal{R}_T$, where:

- $\mathcal{R}_S = \mathcal{C} \cup \mathcal{P} \cup \mathcal{D} \cup \mathcal{I}$ is a set of identifiers partitioned into a set $\mathcal{C}$ of class identifiers, a set $\mathcal{P}$ of property identifiers, a set $\mathcal{D}$ of datatype identifiers, and a set $\mathcal{I}$ of individual identifiers;
- $\mathcal{R}_T$ is a set of triple statements defined over the set of identifiers $\mathcal{R}_S$.

On the basis of the formalizations of UML and RDF(S) above, in the following sections we investigate how to construct RDF(S) from UML class diagrams.

### 3. Construction of RDF(S) from UML

Based on the formalizations of UML and RDF(S) in Section 2, in this section, by comparing and analyzing the characteristics of UML and RDF(S), we first propose a formal approach for constructing RDF(S) from UML class diagrams (see Section 3.1). Then, we further provide a construction example, and make analyses and discussions about the approach (see Section 3.2).

#### 3.1. Approach for Constructing RDF(S) from UML

By comparing and analyzing the characteristics of UML and RDF(S), Table 1 first briefly summarizes the correspondences between UML and RDF(S) for providing readers with an initial understanding of the construction process.

Furthermore, in the following we propose detailed rules of constructing RDF(S) from UML. Given an UML class diagram $F_{UML} = (L, \text{att}_C, \text{attsC}, \text{ass}, \text{agg}, \text{gene}, \text{dep})$ in Definition 1, the corresponding RDF(S) $\mathcal{R} = (\mathcal{R}_S, \mathcal{R}_T)$ can be constructed as the following rules:

**Rule 1 (Mapping of identifiers):** For each identifier $l \in L$ in the set of identifiers $L = C \cup A \cup S \cup Sc \cup R \cup D$ in the UML class diagram $F_{UML}$, create a corresponding RDF(S) resource identifier URI $\varphi(l)$.

Comments: If two identifiers in the UML class diagram $F_{UML}$ have the same name, that can be tackled in the transformation process by renaming the identifiers.

**Rule 2 (Mapping of classes):** For each class $c \in C$ in the UML class diagram $F_{UML}$, create a RDF(S) class:

$$<\text{rdfs:Class} \text{rdf:ID} = \varphi(c)/.>$$

**Rule 3 (Mapping of associations or association classes):** For each association $s \in S$ or association class $sc \in Sc$ in the UML class diagram $F_{UML}$, create a RDF(S) class:

$$<\text{rdfs:Class} \text{rdf:ID} = \varphi(s)/>$$

or

$$<\text{rdfs:Class} \text{rdf:ID} = \varphi(sc)/>.$$

**Rule 4 (Mapping of attributes of classes):** For each function of attributes of classes $\text{att}_C(c) = [a_k : d_k, \ldots]$ in the UML class diagram $F_{UML}$, the attribute $a_k$ is mapped into a RDF(S) property $\varphi(a_k)$, the datatype $d_k$ is mapped into a RDF(S) datatype $\varphi(d_k)$, and created an RDF(S) triple statement to restrict the domain of the RDF(S) property $\varphi(a_k)$ to $\varphi(c)$, and the range of the RDF(S) property $\varphi(a_k)$ to $\varphi(d_k)$, are created as shown in the following RDF(S) triple statement:

$$<\text{rdf:Property} \text{rdf:ID} = \varphi(a_k)/>$$

$$<\text{rdfs:domain} \text{rdf:resource} = \varphi(c)/>$$

$$<\text{rdfs:range} \text{rdf:resource} = \varphi(d_k)/>$$

$$</\text{rdf:Property}>$$

<table>
<thead>
<tr>
<th>UML</th>
<th>RDF(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>Attribute</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>Association</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>Association Class</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>Role in Association (Class)</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>Generalization</td>
<td>rdfs:subClassOf</td>
</tr>
<tr>
<td>Aggregation</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>Dependency</td>
<td>rdf:Property</td>
</tr>
</tbody>
</table>

*Table 1. Correspondences of the main elements between UML and RDF(S).*
For example, given an UML class with attributes \( \text{att}(\text{Student}) = [\text{Name} : \text{string}, \text{Age} : \text{integer}] \), as to rules 2 and 4, the following RDF(S) class, properties, and triple statements will be created:

\[
\begin{align*}
&<\text{rdfs:Class} \text{ rdf:ID = "Student"/>} \\
&<\text{rdf:Property} \text{ rdf:ID = "Name"/>} \\
&\quad \text{rdf:domain rdf:resource = "#Student"/>} \\
&\quad \text{rdf:range rdf:resource = "&xsd:string"/>} \\
&</\text{rdf:Property}> \\
&<\text{rdf:Property} \text{ rdf:ID = "Age"/>} \\
&\quad \text{rdf:domain rdf:resource = "#Student"/>} \\
&\quad \text{rdf:range rdf:resource = "&xsd:integer"/>} \\
&</\text{rdf:Property}>
\end{align*}
\]

Moreover, each function of attributes of association classes \( \text{att}_{sc}(c) = [\ldots, a'_k : d'_k, \ldots] \) can be similarly translated into RDF(S), following the process in Rule 4.

**Rule 5 (Mapping of functions of associations):**

For each function of associations \( \text{ass}(s) = [\ldots, r_k : c_k, \ldots] \), the role \( r_k \) is mapped into a RDF(S) property \( \phi(r_k) \), and created an RDF(S) triple statement to restrict the domain of the RDF(S) property \( \phi(r_k) \) to \( \phi(s) \), and the range of the RDF(S) property \( \phi(r_k) \) to \( \phi(c_k) \), are created as shown in the following RDF(S) triple statement:

\[
\begin{align*}
&<\text{rdf:Property} \text{ rdf:ID = "}\phi(r_k)"/>} \\
&\quad \text{rdf:domain rdf:resource = "}\phi(s)"/>} \\
&\quad \text{rdf:range rdf:resource = "}\phi(c_k)"/>} \\
&</\text{rdf:Property}>
\end{align*}
\]

Moreover, each function of association classes \( \text{ass}(c) = [\ldots, r'_k : c'_k, \ldots] \) can be similarly translated into RDF(S), following the process in Rule 5.

**Rule 6 (Mapping of aggregation):**

For each aggregation \( agg \subseteq c \times (c_1, c_2, \ldots, c_n) \) in the UML class diagram \( F_{UML} \), create \( n \) special RDF(S) properties \( \text{part}_1, \ldots, \text{part}_n \), which are used to represent the relationships between the aggregation class \( c \) and several constituent classes \( c_i, i = 1, \ldots, n \), and create several RDF(S) triple statements to restrict the domain of the RDF(S) property \( \text{part}_i \) to \( \phi(c) \), and its range to \( \phi(c_i) \), as shown in the following RDF(S) triple statements:

\[
\begin{align*}
&<\text{rdf:Property} \text{ rdf:ID = "}\text{part}_1"/>} \\
&\quad \text{rdf:domain rdf:resource = "}\phi(c)"/>} \\
&\quad \text{rdf:range rdf:resource = "}\phi(c_1)"/>} \\
&</\text{rdf:Property}>
\end{align*}
\]

Note that, RDF(S) cannot represent directly the part-whole relationship of the aggregation relationship, thus it is translated into RDF(S) properties and constraints, as shown above.

For example, given an UML aggregation \( agg \subseteq \text{College} \times (\text{Institute} \times \text{Office}) \), as to rules 2 and 6, the following RDF(S) classes, properties, and triple statements will be created:

\[
\begin{align*}
&<\text{rdf:Class} \text{ rdf:ID = "College"/>} \\
&<\text{rdf:Class} \text{ rdf:ID = "Institute"/>} \\
&<\text{rdf:Class} \text{ rdf:ID = "Office"/>} \\
&<\text{rdf:Property} \text{ rdf:ID = "}\text{part}_1"/>} \\
&\quad \text{rdf:domain rdf:resource = "}\#\text{College}"/>} \\
&\quad \text{rdf:range rdf:resource = "}\#\text{Institute}"/>} \\
&</\text{rdf:Property}> \\
&<\text{rdf:Property} \text{ rdf:ID = "}\text{part}_2"/>} \\
&\quad \text{rdf:domain rdf:resource = "}\#\text{College}"/>} \\
&\quad \text{rdf:range rdf:resource = "}\#\text{Office}"/>} \\
&</\text{rdf:Property}>
\end{align*}
\]

**Rule 7 (Mapping of generalization):**

For each generalization \( gen \subseteq c_1 \times c_2 \) in the UML class diagram \( F_{UML} \), create the RDF(S) triple statements:

\[
\begin{align*}
&<\text{rdf:Class} \text{ rdf:ID = "}\phi(c_1)"/>} \\
&\quad \text{subClassOf rdf:resource = "}\phi(c_2)"/>} \\
&</\text{rdf:Class}>
\end{align*}
\]

For example, given an UML generalization gene \( \subseteq \text{Undergraduate} \times \text{Student} \) as
to rules 2 and 7, the following RDF(S) classes and triple statements will be created:

```xml
<rdfs:Class rdf:ID = "Student"/>
<rdfs:Class rdf:ID = "Undergraduate_Student"/>
<rdfs:subClassOf rdf:resource = "#Student"/>
</rdfs:Class>
```

**Rule 8 (Mapping of dependency):** For each dependency \( \text{dep} \subseteq c'_1 \times c'_2 \) in the UML class diagram \( F_{\text{UML}} \), create one special RDF(S) property \( c'_1 \text{dep}_c c'_2 \), which is used to represent the dependency relationship between the target class \( c'_1 \) and the source class \( c'_2 \), and create several RDF(S) triple statements to restrict the domain of the RDF(S) property \( c'_1 \text{dep}_c c'_2 \) to \( \varphi(c'_1) \), and its range to \( \varphi(c'_2) \), as shown in the following RDF(S) triple statements:

```xml
<rdfs:Property rdf:ID = "c'_1\text{dep}_c c'_2"/>
  <rdfs:domain rdf:resource = "\varphi(c'_1)"/>
  <rdfs:range rdf:resource = "\varphi(c'_2)"/>
</rdfs:Property>
```

Note that, the dependency in an UML class diagram denotes that the existence of a target class is dependent of a source class. RDF(S) cannot represent directly the part-whole relationship of the aggregation relationship, and thus when constructing RDF(S) from UML, several additional RDF(S) properties and constraints need to be added, as shown above.

**Rule 9 (Mapping of datatypes):** Each datatype \( d \in D \) in the UML class diagram \( F_{\text{UML}} \) can be mapped into a RDF(S) datatype \( \varphi(d) \). Table 2 gives the mapping rules from a part of UML datatypes to RDF(S) datatypes, and the other datatypes can be similarly handled.

<table>
<thead>
<tr>
<th>UML datatypes</th>
<th>RDF(S) datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>xsd:string</td>
</tr>
<tr>
<td>smallint</td>
<td>xsd:short</td>
</tr>
<tr>
<td>integer</td>
<td>xsd:integer</td>
</tr>
<tr>
<td>decimal</td>
<td>xsd:decimal</td>
</tr>
<tr>
<td>float</td>
<td>xsd:float</td>
</tr>
<tr>
<td>time</td>
<td>xsd:Time</td>
</tr>
<tr>
<td>date</td>
<td>xsd:Date</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

*Table 2. Mapping UML datatypes to RDF(S) datatypes.*

It should be noted that RDF(S) uses the XML Schema datatypes (XML Schema, 2004).

In order to illustrate the construction rules from UML to RDF(S) above, the following section will further provide a construction example.

### 3.2. A Construction Example

In the following, we provide a complete construction example to well explain the construction rules from UML to RDF(S) proposed in Section 3.1. Figure 2 shows an UML class diagram, which includes some UML classes, attributes, and relationships. Here, Faculty and adminStaff are the subclasses of Employee; there is an association **use** between adminStaff and Computer, and **use** is an association class; **Computer** is the aggregation of Monitor, Box, and **Keyboard**; the target class **Employee** is dependent of the source class Employee; there is an association **supervise** between Faculty and Student; there is an association **choose** between Student and Course.

According to the construction rules proposed in Section 3.1, Figure 3 shows the constructed RDF(S), where the URIs of the RDF(S) resources are omitted.

Note that Figure 2 gave a common UML class diagram, which basically includes the main elements of an UML class diagram. Further, Figure 3 shows that the UML class diagram can be transformed into RDF(S), and it can be seen from Figure 3 that the constructed RDF(S) can represent classes, attributes, and relationships (such as association, generalization, aggregation, and dependency) in an UML class diagram. All of these show that the approach for constructing RDF(S) from UML is feasible.

Moreover, it should be noted that since limitation of the expression of RDF(S), some semantic information of UML cannot be directly represented as RDF(S) in the construction process, including:

- An optional constraint **disjointness** can be enforced on a subclass/superclass relationship. The **disjointness** means that all the subclasses are disjoint. As we know, RDF(S) cannot represent such disjoint constraint. Therefore, in the process of constructing RDF(S) from UML, the constraint **disjointness** cannot be represented by RDF(S) directly, and it may be represented by adding
construction of RDF(S) from UML class diagrams

Figure 2. An UML class diagram modeling part of a university.

For example, given a generalization relationship with disjoint constraint gene ⊆ (Required_course × Optional_course) × Course in an UML class diagram, where Course is the superclass, and two subclasses Required_course and Optional_course are disjoint. In the process of constructing RDF(S) from the generalization, in order to represent the disjoint constraint in the generalization, an additional special property named "P_disjointness" need to be added in the RDF(S) used to represent the disjointness of classes as shown in the following RDF(S) triples:

```xml
<rdf:Class rdf:ID = "Course"/>
<rdf:Class rdf:ID = "Required_course">
  <rdfs:subClassOf rdf:resource = "#Course"/>
</rdf:Class>
<rdf:Class rdf:ID = "Optional_course">
  <rdfs:subClassOf rdf:resource = "#Course"/>
</rdf:Class>
<rdf:Property rdf:ID = "P_disjointness">
  <rdfs:domain rdf:resource = "#Required_course"/>
  <rdfs:range rdf:resource = "#Optional_course"/>
</rdf:Property>
```

- An optional cardinality constraint [m, n] can be enforced on an association relationship and is used to specify that each instance of the class can participate at least m times and at most n times to the association relationship. Since RDF(S) cannot represent the cardinality constraint, in the process of constructing RDF(S) from UML, the cardinality constraint cannot be represented by RDF(S) directly. Such cardinality constraint can only be represented by a further RDF(S) extension.

Based on the observations above, the approach proposed in the previous sections can construct RDF(S) from UML class diagrams. Furthermore, in order to implement the automated construction, in the following section we will develop a prototype construction tool.

4. Prototype Construction Tool

In this section, as a proof-of-concept for the proposed construction approach in Section 3, we developed a prototype tool called UML2RDFS, which can construct RDF(S) from UML class diagrams. In the following, we briefly introduce the design and implementation of the prototype tool UML2RDFS.
The core of **UML2RDFS** is that it can first read in an XML-coded UML class diagram file (a conceptual data model file produced from CASE tool PowerDesigner), and then transform it automatically into RDF(S). Implementation of **UML2RDFS** is based on Java 2 JDK 1.5 platform, and the Graphical User Interface (GUI) is exploited by using the java.awt and javax.swing packages. The overall architecture of **UML2RDFS** is briefly shown in Figure 4.
It is shown in Figure 4 that UML2RDFS includes several main modules, i.e., parse module, construction module, and output module:

- The parse module parses an input file (an XML-coded UML class diagram file produced from case tool PowerDesigner) and stores the parsed information. Features of the UML class diagram in the XML-coded file (such as classes, attributes, roles, associations, and several relationships as mentioned in Section 2.1) can be extracted and represented as the formalization of the UML class diagram as proposed in Section 2.1;

- The construction module transforms the parsed results into the corresponding RDF(S) according to the following algorithm UML2RDFS_Const in Figure 5, which is given based on the approach proposed in Section 3. Here, the algorithm briefly describes the construction process from UML to RDF(S), and does not contain the detailed construction steps that have been given in the approach proposed in Section 3.1. In brief, the algorithm performs two kinds of construction operations, i.e., the construction from UML symbols to RDF(S) identifies and the construction from UML constraints to RDF(S) triples;

- The output module finally produces the resulting RDF(S) which is saved as a text file and displayed on the tool screen. Also, the input XML-coded UML class diagram file and the parsing results are displayed on the tool screen.

![UML2RDFS software architecture.](image)

**Figure 4.** UML2RDFS software architecture.

**Algorithm** UML2RDFS\textsubscript{Const}  

//the algorithm briefly describes the construction process from an UML class diagram to RDF(S), and omits the preprocessing operations (i.e., UML parsing and element extraction as mentioned above).

**Input:** An UML class diagram \( F_{\text{UML}} = \{L, \ att_c, \ att_{\text{SC}}, \ ass, \ agg, \ gene, \ dep\} \).

**Output:** RDF(S) \( R = (R_S, R_T) \) that is defined by the transformation function \( \phi \) mentioned in Section 3.1.

1. Transformation from UML symbols \( L \) to RDF(S) resource identifiers. \( R_S = \phi(L) \):
   - For each UML class symbol \( c \in C \in L \), create a RDF(S) class identifier \( \phi(c) \);  
   - For each UML attribute symbol \( a \in A \in L \), create a RDF(S) property identifier \( \phi(a) \);  
   - For each UML association symbol \( s \in S \in L \), create a RDF(S) class identifier \( \phi(s) \);  
   - For each UML association class symbol \( s_c \in S_c \in L \), create a RDF(S) class identifier \( \phi(s_c) \);  
   - For each UML role symbol \( r \in R \in L \), create a RDF(S) property identifier \( \phi(r) \);  
   - For each UML datatype symbol \( d \in D \in L \), create a RDF(S) datatype identifier \( \phi(d) \).

2. Transformation from UML constraints to RDF(S) triples \( R_T = \phi(\text{att}_{\text{C}}, \text{att}_{\text{SC}}, \text{ass}, \text{agg}, \text{gene}, \text{dep}) \):
   - For each UML class function \( \text{att}_{\text{C}} \) or association class function \( \text{att}_{\text{SC}} \), create the corresponding RDF(S) triples according to rule 4 in Section 3.1;  
   - For each UML association function \( \text{ass} \), create the corresponding RDF(S) triples according to rule 5 in Section 3.1;  
   - For each UML aggregation function \( \text{agg} \), create the corresponding RDF(S) triples according to rule 6 in Section 3.1;  
   - For each UML generalization function \( \text{gene} \), create the corresponding RDF(S) triples according to rule 7 in Section 3.1;  
   - For each UML dependency function \( \text{dep} \), create the corresponding RDF(S) triples according to rule 8 in Section 3.1.

![The construction algorithm UML2RDFS_Const from UML to RDF(S).](image)

**Figure 5.** The construction algorithm UML2RDFS\textsubscript{Const} from UML to RDF(S).
We carried out construction experiments using the implemented tool UML2RDFS, with a PC (CPU P4/3.0GHz, RAM 3.0GB and Windows XP system). We choose more than thirty UML class diagrams including all features of UML mentioned in Section 2.1. Many more complex UML diagrams which consist of these features can be converted into RDF(S) by jointly using our approach and tool. The UML class diagrams used in our tests are mainly from the following parts: Some come from the existing UML diagrams from the website (http://www.uml-diagrams.org/index-examples.html), e.g., Library domain model, Online shopping domain, etc.; Some are created manually by us, with the CASE tool PowerDesigner, e.g., one of the UML diagrams mentioned in Section 3.2. Their sizes range from 40 to 3000 (here the scale of an UML class diagram denotes the numbers of classes, attributes, roles, associations, and relations in the UML diagram). The test results show that our approach and tool actually work, and the time complexity of the conversion is linear with the scales of UML diagrams, which is consistent with the theoretical analysis. Here we briefly analyze the time complexity of the algorithm UML2RDFS_Const in Figure 5. Since the conversion of UML symbols to RDF(S) resource identifiers (i.e., Step 1 of the algorithm) can be simultaneously made as sub-operations in creating RDF(S) triples (i.e., Step 2 of the algorithm), we can ignore the amount of work done in the first step and consider only the creation of triples in the second step. Also, we consider the conversion operations and ignore the preprocessing operations (i.e., the parsing and extracting of the XML-coded file of an UML diagram), that is, we exclude the amount of work done by an XML parser (e.g., the DOM API for Java in our implementation) that parses the UML diagram (i.e., an XML-coded file) and extracts and prepares the element data in computer memory for the usage in the conversion procedure of the algorithm. In this case, the time complexity of the algorithm mainly depends on the structure of an UML diagram. Suppose the scale of an UML diagram is \( N = N_C + N_A + N_S + N_R + N_D + N_{AGG} + N_{ISA} + N_{DEP} \), where \( N_C \), \( N_A \), \( N_S \), \( N_R \), \( N_D \), \( N_{AGG} \), \( N_{ISA} \), and \( N_{DEP} \) denotes the cardinality of the sets of classes, attributes, associations, roles, datatype domains, aggregation relations, ISA relations, and dependency relations, respectively. Then, the creating times of the corresponding RDF(S) triples of the cases \( att_C \) and \( att_{SC} \) are \( N_C + N_S + N_A + N_D \) at most, the case \( ass \) are \( N_S + N_C + N_R \) at most, the case \( agg \) are \( N_{AGG} + 2(N_C - 1) \), the case \( gene \) are \( N_C + N_{ISA} - 1 \) at most, and the case \( dep \) are \( N_{DEP} + N_C + 1 \). Therefore, in the worst case, the total running times \( T = 6N_C + 2N_S + N_A + N_D + N_R + N_{AGG} + N_{ISA} + N_{DEP} - 2 < 6N - 2 \), that is, the time complexity of the algorithm is \( O(N) \) at most. Figure 6 shows the execution time routines in the UML2RDFS tool running several UML class diagrams, where preprocessing denotes the operations of parsing and storing UML class diagrams, i.e., parsing the UML class diagrams and preparing the element data in computer memory for the usage in the construction procedure.

![Figure 6. The execution time of the tool UML2RDFS routine on several UML class diagrams.](image)

In the following, we provide an example of UML2RDFS. Figure 7 shows the screen snapshot of UML2RDFS tool running one of the case studies, which displays the construction from an UML class diagram (including the information of the UML class diagram in Figure 2 as mentioned in Section 3.2) to RDF(S). In Figure 7, the XML-coded UML class diagram file, the parsed results (i.e., the tree representation of the formalization of the UML class diagram), and the constructed RDF(S) are displayed in the left, middle and right areas, respectively, as annotated in the displayed parts of the graphical user interface. Moreover, it should be noted that our current version of the tool does not provide the function of checking the error itself. Currently, after an UML diagram is converted into RDF(S) by our approach and tool, we can test the validity of the conversion by constructing some SPARQL queries on the converted RDF(S) data (SPARQL is the standard query language for RDF (RDF-W3C,
If the query results include all of the information in the UML diagram, then the conversion is correct and valid. For example, we can construct a query “SELECT ?x WHERE { ?x rdfs:subClassOf Employee.}” for querying subclasses of a given class Employee as shown in Figure 3. The query returns two classes Faculty and adminStaff, which are consistent with the information in the UML diagram in Figure 2. In our near future work, we will enhance our tool to provide the function of checking the error itself. In addition, in the experiments, we also find how sensitive the algorithm is to possible errors in the input, e.g., an association relationship with cardinality constraint or an UML diagram containing dynamic behavior, which cannot be converted into RDF(S) as also mentioned in Section 3.2.

5. Conclusions and Future Works

In this paper, by comparing and analyzing the characteristics of UML and RDF(S), we proposed an approach and implemented a prototype tool for constructing RDF(S) from UML. We first gave formal definitions of UML and RDF(S). Then, we proposed a construction approach from UML to RDF(S), and gave detailed construction rules. Also, a construction example was provided, and the analyses and discussions about the approach were done. Further, based on the proposed approach, we implemented a prototype construction tool, and the experiment shows that the approach and the tool are feasible. The work in the paper may act as a gap-bridge between the existing UML applications and the Semantic Web.

As far as our future work, we will concern the following aspects: (i) After mapping UML to RDF(S), some reasoning tasks of UML may be done by means of the reasoning mechanism of RDF(S). For example, checking whether an UML class $C_1$ is a subclass of another UML class $C_2$ can be reduced to checking whether the RDF(S) class $\phi(C_1)$ (resp. the UML class $C_1$) is a subclass of the RDF(S) class $\phi(C_2)$ (resp. the UML class $C_2$). Further, the latter can be automatically handled by means of the existing inference systems (e.g., Jena and RACER (Vidy & Punitha, 2012; Horrocks et al., 2003)). In our near future work, we will further discuss how the constructed RDF(S) may be useful for reasoning on UML based on the reasoning mechanism of RDF(S). (ii) We may also further discuss how our work can be applied in a special domain. As shown in our work, an UML model expresses concepts of a domain being modelled, and some actual data from a database of the domain is not included in the UML model. In this case, for a special domain, we first can construct RDFS from the UML using our ap-
proach and tool, and then complying with the RDF(S) derived from the UML, some actual data from a database can be directly represented by RDF statements. For example, a student instance identified by its primary key attribute SID in a table Student of a database can be represented by a RDF statement: `<rdf:Description rdf:ID="φ(SID)"/> <rdf:type rdf:resource="#φ(Student)"/> </rdf:Description>`, where φ(Student) is the RDFS class derived from the UML. Such applications will be further discussed in detail in the future work. (iii) We will also test more and larger scale UML to further evaluate the tool. Moreover, as we mentioned at the end of Section 3.2, more features in data modelling may be represented by more expressive knowledge representation languages. On the basis of the work presented in this paper, we may further investigate UML-to-OWL approaches mentioned briefly at the end of Section 3.2. In addition, an UML from a specific domain complies with the ad-hoc vocabularies of UML in Section 2.1, and thus a complex domain-specific mapping from an UML model to an existing ontology may be realized by jointly using our approach. (iv) We will also strive to extend our algorithms to other types of UML diagrams besides class diagrams, and work toward unification of the RDFS language with other similar endeavors in the field. Further, we may test the retrieval of converted diagrams within a suitable expert system environment. In this paper, our main focus is the conversion of UML-to-RDF(S). All these perspectives will be investigated and discussed in depth in our future work.

Acknowledgments

The work is supported by the Fundamental Science Research Funds for the Education Department of Liaoning (L2013098), the National Natural Science Foundation of China (61202260) and the Fundamental Research Funds for the Central Universities (N120404005).

References


