RDF Declarative Description (RDD): A Language for Metadata

Chutiporn Anutariya and Vilas Wuwongse
Computer Science and Information Management Program
Asian Institute of Technology, Pathumtani 12120, Thailand
{ca, vw}@cs.ait.ac.th

Kiyoshi Akama
Center for Information and Multimedia Study
Hokkaido University, Sapporo 060, Japan
akama@cims.hokudai.ac.jp

Ekawit Nantajeewarawat
Information Technology Program
Sirindhorn International Institute of Technology
Thammasat University, Pathumtani 12120, Thailand
ekawit@siit.tu.ac.th

Abstract

RDF Declarative Description (RDD) is a metadata modeling language which extends RDF(S) expressiveness by provision of generic means for succinct and uniform representation of metadata, their relationships, rules and axioms. Through its expressive mechanism, RDD can directly represent all RDF-based languages such as OIL and DAML-family markup languages (e.g., DAML+OIL and DAML-S), and hence allows their intended meanings to be determined directly without employment of other formalisms. Therefore, RDD readily enables interchangeability, interoperability as well as integrability of metadata applications, developed independently by different communities and exploiting different schemas and languages. Moreover, RDD is also equipped with computation and query-processing mechanisms.

Keywords: *Metadata*, *RDF*, *RDF Schema*, *RDF Declarative Description*, *RDD language*.

1 Introduction

Resource Description Framework (*RDF*) [12] is a W3C's recommended framework for encoding, exchange and reuse of metadata, which offers foundations for syntactic and semantic interoperations among Web applications. Although *RDF Schema* (*RDFS*) [7] provides a simple ontological-modeling facility for descriptions of classes, properties and their hierarchies, its mechanism is still limited and lacks expressive power to describe Web resources and Web applications in that:

- it cannot represent ontological and domain axioms as well as relational algebraic properties, e.g., transitivity, symmetry of relations and inverse relations;
- it does not support an efficient and powerful computation mechanism; and
- it does not provide a query-processing mechanism.

In order to cope with such limitations, this paper develops *RDF Declarative Description* (*RDD*) language which employs RDF(S) and *Declarative Description* (*DD*) theory [1,2] as its underlying frameworks for description of metadata and for enhancement of RDF(S) expressive power. It allows ordinary RDF (metadata) elements, encoded in XML language and describing specific facts and relations among certain resources, to be directly represented. Moreover, it enhances the capability and expressiveness of ordinary RDF elements by additionally allowing expression of implicit complex resources as well as their relations, rules and axioms in terms of *RDF expressions*—an extension of RDF elements with variables—and *RDF clauses*.

An *RDD statement descriptor*, formulated as a set of ordinary RDF elements, RDF expressions with variables and RDF clauses, then provides sufficient, flexible and expressive means for describing and modeling a wide diversity of resources. Its semantics is formally defined as a set of ordinary RDF elements which are directly described by or derivable from the description itself.

https://doi.org/10.23106/dcmi.952106423

Table 1. Variable Types.

Variable Type	Variable Names Beginning with	Instantiation to
N-variables: *Name-variables*	\$N	Element types or attribute names
S-variables: *String-variables*	\$S	Strings
P-variables: *Attribute-value-pair-variables*	\$P	Sequences of zero or more attribute-value pairs
E-variables: *RDF-expression-variables*	\$E	Sequences of zero or more RDF expressions
I-variables: *Intermediate-expression-variables*	\$I	Parts of RDF expressions

Moreover, RDD also supports formulation and evaluation of queries which not only allows pattern matching and selective retrieval, but also supports inquiries about implicit information contained in RDF metadata.

Section 2 formalizes RDD language, Section 3 presents an RDD approach to metadata modeling, Section 4 demonstrates an example of modeling and querying a metadata application, Section 5 reviews current, related works, and Section 6 draws conclusions.

2 RDF Declarative Description (RDD)

By employment of DD theory [1,2] (cf. Appendix for its fundamental concepts), RDF—the framework for representation of metadata—will be extended with full-fledged modeling and reasoning services. First, a formal definition of RDF expressions will be given, followed by a formalization of an *RDF specialization system*—a mathematical abstraction characterizing the data structure of RDF expressions. Based on such a structure, RDD language will be developed.

2.1 RDF specialization system

Ordinary RDF elements are *ground* (variable-free) and take one of the forms:

- $<t\ \ a_1{=}v_1 \ldots a_m{=}v_m\ />$,
- $<t\ \ a_1{=}v_1 \ldots a_m{=}v_m>\ v_{m+1}\ </t>$,
- $<t\ \ a_1{=}v_1 \ldots a_m{=}v_m>\ e_1 \ldots e_n\ </t>$,

where m, $n \geq 0$, t is an element name, the a_i are distinct attribute names, the v_i are strings, and the e_i are RDF elements.

In order to represent implicit information and to enhance the expressiveness of RDF elements, their definition will be formally extended by incorporation of variables and then called *RDF expressions*. Every component of an RDF expression—the expression itself, its element name, attribute names and values, pairs of attribute-value, contents, sub-expressions as well as some partial structures—can contain variables. Table 1 defines all types of variables and their usages.

An *RDF expression alphabet* Σ_R then comprises the symbols in the sets of names N, characters C as well as the sets of those five types of variables.

An RDF expression on Σ_R takes formally one of the following forms:

- *evar*,
- $<t\ \ a_1{=}v_1 \ldots a_m{=}v_m\ \ pvar_1 \ldots pvar_k\ />$,
- $<t\ \ a_1{=}v_1 \ldots a_m{=}v_m\ \ pvar_1 \ldots pvar_k>\ v_{m+1}\ </t>$,
- $<t\ \ a_1{=}v_1 \ldots a_m{=}v_m\ \ pvar_1 \ldots pvar_k>\ e_1 \ldots e_n</t>$,
- $<ivar>\ e_1 \ldots e_n\ </ivar>$,

where - *evar* is an *E*-variable,
- k, m, $n \geq 0$,
- t, a_i are names or *N*-variables,
- $pvar_i$ is a *P*-variable,
- v_i is a string or an *S*-variable,
- *ivar* is an *I*-variable,
- e_i is an RDF expression on Σ_R.

The order of the attribute-value pairs $a_1{=}v_1 \ldots a_m{=}v_m$ and the order of the *P*-variables $pvar_1 \ldots pvar_k$ as well as the order of the expressions $e_1 \ldots e_n$ are immaterial. Only the order of the expressions $e_1 \ldots e_n$ contained in an rdf:Seq-expression is important. RDF expressions without variables are called *ground RDF expressions* or simply *RDF elements*, those with variables *non-ground RDF expressions*. An expression of the second, the third or the fourth form is referred to as a *t-expression*, while that of the fifth form as an *ivar-expression*. A ground *t*-expression will also be called a *t-element*. An *I*-variable is employed to represent an RDF expression when its structure or nesting pattern is not fully known. For example, the expression $<ivar>\ e_1 \ldots e_n\ </ivar>$, where e_i are expressions, represents a set of RDF expressions which contain the sub-expression sequence $e_1 \ldots e_n$ to an arbitrary depth.

Figure 1-a illustrates an example of a non-ground RDF expression which represents a class of Staff resources with some relation to the resource referred to by "staff_02"; however, the name of such a relation as well as the identifiers (URIs) and other properties of the resources in the class are unknown and are represented by an *N*-variable \$N:relation, an *S*-variable \$S:uri and an *E*-variable \$E:properties, respectively. It will be seen later in this sub-section that such a non-ground expression can be specialized into a ground RDF expression of Figure 1-b.

Instantiation of those various types of variables is defined by *basic specialization*, each of which has the form (v, w) where v specifies the name of the variable to be specialized and w the specializing

https://doi.org/10.23106/dcmi.952106423

```
<Staff  rdf:about=$S:uri>
   <$N:relation  rdf:resource="staff_02"/>
   $E:properties
</Staff>
```

(a) A non-ground RDF expression a.

Specialized into by $\mu_R(\theta)$

```
<Staff rdf:about="staff_03">
   <boss rdf:resource="staff_02"/>
   <name>Derek T.</name>
   <worksFor>Computer Dept.</worksFor>
</Staff>
```

(b) A ground RDF expression $g = \mu_R(\theta)(a) = a\theta$

```
($S:uri, "staff_03")
($N:relation, boss)
($E:properties, ($E:p1, $E:p2))
($E:p1, <name>Derek T.</name>)
($E:p2, <worksFor>Computer Dept.</worksFor>)
```

(c) Specialization of the non-ground expression a to the ground RDF expression g by

- **Instantiation of the S-variable $S:uri into the string "staff_03",**
- **Instantiation of the N-variable $N:relation into the property name boss,**
- **Expansion of the E-variable $E:p1 into the sequence of the E-variables $E:p1 and $E:p2,**
- **Instantiation of the E-variable $E:p1 into the RDF expression <name>Derek T.</name>,**
- **Instantiation of the E-variable $E:p2 into the RDF expression <worksFor>Computer Dept.</worksFor>.**

Figure 1. Specialization of a non-ground RDF expression in $\mathcal{A}_R$ into a ground RDF expression in $\mathcal{G}_R$ by the operator μ_R using a specialization θ in $\mathcal{S}_R$.

value. For example, ($S:s1, $S:s2), ($N:relation, boss) and ($E:properties, ($E:p1, $E:p2)) are basic specializations which rename the *S*-variable $S:s1 to $S:s2, instantiate the *N*-variable $N:relation into the element name boss, and expand the *E*-variable $E:properties into the sequence of the *E*-variables $E:p1 and $E:p2, respectively.

There are four types of basic specializations:

1. Rename variables.
2. Expand a *P*- or an *E*-variable into a sequence of variables of their respective types.
3. Remove *P*- or *E*-variables.
4. Instantiate variables to RDF expressions or components of RDF expressions which correspond to the types of the variables.

Denote a sequence of zero or more basic specializations by a *specialization*.

The data structure and the specialization of RDF expressions are characterized by a mathematical abstraction

$$\Gamma_R = \langle \mathcal{A}_R, \mathcal{G}_R, \mathcal{S}_R, \mu_R \rangle,$$

called **RDF Specialization System**, where

- $\mathcal{A}_R$ is the set of all RDF expressions on Σ_R,
- $\mathcal{G}_R$ is the subset of $\mathcal{A}_R$ which comprises all ground RDF expressions in $\mathcal{A}_R$,
- $\mathcal{S}_R$ is the set of all *specializations* that reflect the data structure of the RDF expressions in $\mathcal{A}_R$, and
- μ_R is the *specialization operator* which determines, for each specialization s in $\mathcal{S}_R$, the change of each RDF expression a in $\mathcal{A}_R$ caused by s.

Note that a specialization in $\mathcal{S}_R$ will often be denoted by a Greek letter such as θ, and when μ_R is clear from the context, for $\theta \in \mathcal{S}_R$, $\mu_R(\theta)(a)$ will be written simply as $a\theta$.

Consider Figure 1-c for an example of a specialization in $\mathcal{S}_R$ and its application to an RDF expression by the specialization operator μ_R.

2.2 RDD language

After the RDF specialization system Γ_R has been developed, RDD language and its related concepts can now be defined.

A *constraint* on Γ_R, useful for defining a restriction on RDF expressions or components of RDF expressions, is a formula $q(a_1, \ldots, a_n)$, where $n > 0$, q is a *constraint predicate* and a_i an RDF expression in $\mathcal{A}_R$. Application of $\theta \in \mathcal{S}_R$ to a constraint $q(a_1, \ldots, a_n)$ yields $q(a_1\theta, \ldots, a_n\theta)$. Given a *ground constraint* $q(g_1, \ldots, g_n)$, $g_i \in \mathcal{G}_R$, its truth or falsity is predetermined.

Given two RDF expressions a_1 and a_2, define GT(a_1, a_2) as a constraint which will be true, iff a_1, a_2 are RDF elements of the forms <Num>v_1</Num>, and <Num>v_2</Num>, respectively, where v_1, v_2 are numbers and $v_1 > v_2$. Obviously, a constraint

GT(<Num>8</Num>,<Num>4</Num>)

is a true ground constraint in *Tcon*. In addition, for some $a_1, a_2, a_3 \in \mathcal{A}_R$, let Mul($a_1$, a_2, a_3) be a constraint which will be true iff a_1, a_2 and a_3 are RDF elements of the forms:

<Num>n_1</Num>,
<Multiplier>n_2</Multiplier>, and
<Result>n_3</Result>,

respectively, where n_1, n_2, n_3 are numbers, and n_3 is the result of multiplying n_1 and n_2, i.e., $n_3 = n_1 \times n_2$.

https://doi.org/10.23106/dcmi.952106423

An *RDF declarative description* on Γ_R, simply called an *RDD description*, is a set of *RDF clauses*, each of which has the form:

$$H \leftarrow B_1, B_2, ..., B_n$$

where $n \geq 0$, H is an RDF expression in $\mathcal{A}_R$, and B_i is an RDF expression in $\mathcal{A}_R$, or a *constraint* on Γ_R. The order of the B_i is immaterial. H is called the *head* and $(B_1, ..., B_n)$ the *body* of the clause. Such a clause, if $n = 0$, is called a *unit clause*, if $n > 0$, a *non-unit clause*. When it is clear from the context, a unit clause $(H \leftarrow)$ will be simply written as H. Therefore, an RDF document, containing a set of RDF elements and describing certain resources, is directly mapped onto an RDD description comprising solely ground RDF unit clauses.

Given an RDD description P, its meaning, denoted by $\mathcal{M}(P)$ (cf. Appendix), is the set of all RDF elements which are directly described by and are derivable from the unit and the non-unit clauses in P, respectively, i.e.:

- Given a unit clause $(H \leftarrow)$ in P, for $\theta \in \mathcal{S}_R$: $H\theta \in \mathcal{M}(P)$ if $H\theta$ is a ground RDF expression.
- Given a non-unit clause $$(H \leftarrow B_1, ..., B_i, B_{i+1}, ..., B_n)$$ in P, assuming without loss of generality that $B_1, ..., B_i$ are RDF expressions and $B_{i+1}, ..., B_n$ constraints, for $\theta \in \mathcal{S}_R$:

  $H\theta \in \mathcal{M}(P)$ if - $H\theta$ is a ground RDF expression,
  - $B_1\theta,..., B_i\theta \in \mathcal{M}(P)$, and
  - $B_{i+1}\theta, ... ,B_n\theta$ are true constraints.

3 RDD: a metadata modeling language

In RDD language, metadata expressed in terms of RDF elements are directly represented by ground RDF expressions in $\mathcal{G}_R$, while classes of RDF metadata which share certain similar components and structures are modeled by RDF expressions with variables as shown in Figure 1-a.

A collection of RDF metadata describing certain specific facts and relations among resources of a real-world domain is then modeled as a set of RDF expressions. Ontological and domain axioms as well as implicit relations and their algebraic properties, such as transitivity, symmetry and inverse, are expressed as RDF clauses.

Hence, a particular metadata application domain is readily modeled as an RDD description which comprises a set of RDF expressions, describing explicit, complex objects and their relations in the domain, together with a set of RDF clauses, representing the domain's axioms as well as certain implicit relations. Its semantics is a set of RDF elements, which are explicit in the domain, together with all the derived ones, which are inferred from the specified axioms and relations.

Besides provision of a simple, yet expressive mechanism to model metadata, RDD also provides a facility to formulate and evaluate queries. Basically, a query is represented by an RDD description containing one or more RDF non-unit clauses. The head of each clause describes the structure of the query result, while its body specifies the pattern as well as the selection condition of the query.

Evaluation of a query is carried out by employment of *Equivalent Transformation* (*ET*) [2,3] (cf. Appendix)—a new computational model for solving problem based on *semantics-preserving* transformations. Given a description P specifying a collection of metadata elements as well as a set of their relations and axioms within some particular domain, a query represented by a description Q is evaluated by transforming the description $(P \cup Q)$ successively into a simpler but equivalent description, from which the answers to the query can be obtained easily and directly. More precisely, such a description $(P \cup Q)$ will be successively transformed until it becomes the description $(P \cup Q')$, where Q' consists of only *ground unit clauses* and $\mathcal{M}(P \cup Q) = \mathcal{M}(P \cup Q')$. In order to guarantee the correctness of the computation, only equivalent transformations can be applied at every step. The unfolding transformation, a widely-used program transformation in conventional logic programming, is a kind of equivalent transformation. Other kinds of equivalent transformation can also be devised, especially for improvement of computation efficiency.

4 Example: metadata modeling

DAML+OIL [10] is an RDF-based language which extends RDFS by a richer set of modeling constructs for description of ontological axioms as well as algebraic properties of relations. This example demonstrates an RDD approach to representing and querying a metadata application by employment of DAML+OIL to model the application's schema and to describe the application's data and objects. The example will first show that by RDD language, instances of DAML+OIL can be directly expressed without a necessity for translation or modification and their semantics can be precisely determined. It will then demonstrate the expressive power of RDD language by modeling a particular domain axiom which is essential in the application but inexpressible by DAML+OIL. Finally, an example of a query about information implicit in the application will be given.

Modeling the application's schema and data: Figure 2 illustrates an example of a DAML+OIL document which defines the application's schema and describes certain explicit information about each data object of the application. Obviously, such a document is an RDD description which contains merely unit clauses and will be referred to as P_1.

https://doi.org/10.23106/dcmi.952106423

```
<daml:Class rdf:ID="Staff"/>
<daml:Class rdf:ID="JuniorStaff">
    <rdfs:subClassOf rdf:resource="Staff"/>
</daml:Class>
<daml:Class rdf:ID="SeniorStaff">
    <rdfs:subClassOf rdf:resource="Staff"/>
</daml:Class>
<daml:Property rdf:ID="name"/>
<daml:Property rdf:ID="worksFor"/>
<daml:Property rdf:ID="salary"/>
<daml:TransitiveProperty rdf:ID="boss"/>
<daml:Property rdf:ID="contractPeriod"/>
<SeniorStaff rdf:about="staff_01">
    <name>Somchai P.</name>
    <worksFor>Sales Dept.</worksFor>
    <salary>5000</salary>
</SeniorStaff>
<SeniorStaff about="staff_02">
    <name>Sawat K.</name>
    <worksFor>Computer Dept.</worksFor>
    <salary>7000</salary>
</SeniorStaff>
<SeniorStaff  about="staff_03">
    <name>Derek T.</name>
    <worksFor>Computer Dept.</worksFor>
    <salary>6000</salary>
    <boss rdf:resource="staff_02"/>
</SeniorStaff>
<JuniorStaff  about="staff_04">
    <name>Arunee I.</name>
    <worksFor>Computer Dept.</worksFor>
    <salary>3000</salary>
    <boss rdf:resource="staff_03"/>
</JuniorStaff>
```

Figure 2. Modeling of an application's schema and data.

Modeling relational algebraic properties and ontological axioms: In order to define the meanings of those DAML+OIL modeling constructs (i.e., subClassOf and transitiveProperty), which are employed by Figure 2 and include some notion of implication, the RDF clauses C_1–C_3 of Figure 3 are formulated. Denote the set of these clauses by an RDD description P_2, i.e., $P_2 = \{C_1, C_2, C_3\}$. Note that the meanings of other DAML+OIL modeling constructs such as domain, range, inverseOf, intersectionOf and equivalentTo can also be defined in terms of RDF clauses in a similar manner.

Modeling domain axioms: The RDF clause C_4 of Figure 4 illustrates an example of modeling a domain axiom. It defines additional relationships among the properties salary, worksFor, bonus and contractPeriod of the class SeniorStaff. Donote the set of the clause C_4 by an RDD description P_3, i.e., $P_3 = \{C_4\}$.

Let P be the union of the descriptions P_1, P_2 and P_3 which model (i) the application's schema and data, (ii) relational algebraic properties and ontological axioms, and (iii) domain axioms, respectively. Thus, P becomes immediately a model of the application, and the meaning of P, $\mathcal{M}(P)$, includes not only the information explicit in the application, i.e., those elements of P_1, but also the following implicit information, which is uncovered by the clauses of the descriptions P_2 and P_3:

- The resources referred to by staff_01, staff_02, staff_03 and staff_04 are instances of the class Staff.
- The Staff referred to by staff_02 is a boss of the Staff referred to by staff_04.
- The contractPeriod of the SeniorStaff referred to by staff_02 is a 2-year term and that SeniorStaff will receive a bonus of 14000.
- The contractPeriod of the SeniorStaff referred to by staff_03 is a 2-year term and that SeniorStaff will receive a bonus of 12000.

Formulating and evaluating a query: The clause C_q of Figure 5 represents a query, which selects all SeniorStaffs who get a bonus of more than 10000, and then lists their names and bonuses.

Using unfolding transformation, the description

$$P \cup \{C_q\}$$

can be transformed into the description

$$P \cup \{C_{q1}, C_{q2}\},$$

where C_{q1} and C_{q2} are the following unit clauses

C_{q1}:
```
<BigBonusStaff>
    <name>Sawat K.</name>
    <bonus>14000</bonus>
</BigBonusStaff>    ←  .
```

C_{q2}:
```
<BigBonusStaff>
    <name>Derek T.</name>
    <bonus>12000</bonus>
</BigBonusStaff>    ←  .
```

Thus, one can directly draw that the head elements of C_{q1} and C_{q2} are the answers to the given query. Moreover, since only unfolding transformation, which always preserves the equivalence of declarative descriptions, is used, the two obtained answers are guaranteed to be correct.

5 Related works

DAML+OIL [10] is the most improved ontology markup language, which has been defined on the basis of RDF(S) and *OIL* [6,9], in order to provide an expressive set of modeling constructs. However, its mechanism is insufficient to model metadata, since it can represent only a limited set of ontological axioms and relational algebraic properties, while lacking an ability to express arbitrary rules and domain axioms.

https://doi.org/10.23106/dcmi.952106423

```
C1: <daml:Class  rdf:ID=$S:classA>
        <rdfs:subClassOf  rdf:resource=$S:classC/>
        $E:A_properties
    </daml:Class>
            ← <daml:Class  rdf:ID=$S:classA>
                  <rdfs:subClassOf  rdf:resource=$S:classB/>
                  $E:A_properties
              </daml:Class>,
              <daml:Class  rdf:ID=$S:classB>
                  <rdfs:subClassOf  rdf:resource=$S:classC/>
                  $E:B_properties
              </daml:Class>.
```

% **subClassOf Transitivity**
% **Property:** If A is a subclass of B
% and B is a subclass of some broader
% class C, this implies that A is also a
% subclass of C.

```
C2: <$S:classB  rdf:about=$S:resourceX>
        $E:X_properties
    </$S:classB>
            ← <daml:Class  rdf:ID=$S:classA>
                  <rdfs:subClassOf  rdf:resource=$S:classB/>
                  $E:A_properties
              </daml:Class>,
              <$S:classA  rdf:about=$S:resourceX>
                  $E:X_properties
              </$N:classA>.
```

% **The meaning of subClassOf**
% **construct:** If a class A is declared
% as a subclass of another class B,
% then every resource which is an
% instance of the class A will also be
% an instance of the class B.

```
C3: <$N:classA  rdf:about=$S:resourceX>
        <$S:propertyP  rdf:resource=$S:resourceZ/>
    </$N:classA>
            ← <daml:TransitiveProperty  rdf:ID=$S:propertyP>,
              <$N:classA  rdf:about=$S:resourceX>
                  <$S:propertyP  rdf:resource=$S:resourceY/>
                  $E:X_properties
              </$N:classA>,
              <$N:classB  rdf:about=$S:resourceY>
                  <$S:propertyP  rdf:resource=$S:resourceZ/>
                  $E:Y_properties
              </$N:classB>.
```

% **The meaning of Transitive-**
% **Property construct:** For a
% TransitiveProperty P, if a property P
% of a resource X is a resource Y and
% that of the resource Y is a resource
% Z, then one can imply that a
% property P of a resource X is also
% the resource Z.

** **Note:** When an S-variable is used as an element name of an RDF expression, that variable can only be specialized into a valid element name, but not into any arbitrary string. For instance, an expression <**$S:classA** rdf:about="X"/> can be specialized into <**Staff** rdf:about="X"/> but not into <**"An arbitrary string"** rdf:about="X"/>.

Figure 3. Modeling of ontological axioms and relational algebraic properties.

```
C4: <SeniorStaff  rdf:about=$S:staff>
        <worksFor>Computer Dept.</worksFor>
        <salary>$S:salary</salary>
        <bonus>$S:bonus</bonus>
        <contractPeriod>2 years</contractPeriod>
        $E:staff_properties
    </SeniorStaff>
            ← <SeniorStaff  rdf:about=$S:staff>
                  <worksFor>Computer Dept.</worksFor>
                  <salary>$S:salary</salary>
                  $E:staff_properties
              </SeniorStaff>,
              Mul(<Num>$S:salary</Num>, <Multiplier>2</Multiplier>,
                  <Result>$S:bonus</Result>).
```

% The clause C_4 defines additional
% relationships among the properties
% worksFor, salary, bonus and
% contractPeriod of the class
% SeniorStaff, by asserting that for
% every SeniorStaff of the Computer
% Department, his/her contractPeriod
% is 2 years and he/she will receive a
% double-salary bonus.

Figure 4. Modeling of domain axioms.

```
Cq: <BigBonusStaff>
        <name>$S:name</name>
        <bonus>$S:bonus</bonus>
    </BigBonusStaff>
            ← <SeniorStaff  rdf:about=$S:staff>
                  <name>$S:name</name>
                  <bonus>$S:bonus</bonus>
                  $E:staff_properties
              </SeniorStaff>,
              GT(<Num>$S:bonus</Num>, <Num>10000</Num>).
```

% The clause C_q represents a query
% which lists the name and bonus of
% each SeniorStaff who gets a bonus
% of more than 10000.

Figure 5. Modeling of a query.

https://doi.org/10.23106/dcmi.952106423

As demonstrated by the example of Section 4, RDD language can be employed to enhance the expressiveness of DAML+OIL. Besides allowing the semantics of each DAML+OIL modeling construct to be precisely determined, RDD also provides sufficient means to describe additional rules and axioms in terms of RDF non-unit clauses.

DAML-S [4,14] is a recently proposed, DAML-family markup language for description of Web service properties, capabilities and functionalities. Instances of DAML-S, encoded in RDF/XML serialization, can be directly represented by RDD language as RDF unit clauses. Based on DAML-S syntax and constructs, an RDD approach to modeling and implementing Web services is being developed. In essence, such an approach will enable the automation of the following tasks:

- service advertisement and discovery,
- negotiation,
- service invocation and execution,
- service composition and integration, and
- service customization.

SquishQL [15]—the most recent, improved query language for RDF—is an SQL-like language with SELECT-FROM-WHERE-style syntax. Its query mechanism is based on subgraph matching, where patterns and query selection criteria are expressed in terms of RDF triples of subject, predicate and object. Based on SquishQL, several RDF query engines have been developed [15,8]. Apart from the simple ontological-modeling facility provided by RDFS, these engines do not allow additional descriptions of rules, axioms and relational algebraic properties. Thus, their sole inference service is based on class and property hierarchies.

Metalog [13] and *SiLRi* (*Simple Logic-based RDF Interpreter*) [5] employ logic programming and F-logic theories, respectively, in order to provide both query and reasoning services for RDF. In these two approaches, RDF metadata elements must be translated into sets of corresponding representation in their original frameworks, i.e., into sets of binary predicates and F-logic formulae. Querying and reasoning about RDF metadata are then performed on these corresponding translations instead of direct operation on RDF elements.

6 Conclusions

By integration of the RDF data model, DD theory and ET computational paradigm, this paper has developed a solid, practical framework for a uniform representation of and reasoning with RDF metadata. The developed framework derives metadata description facilities, exchangeability and interoperability from the RDF data model, expressiveness from DD theory and an efficient computational mechanism from ET paradigm.

In order to demonstrate its usefulness and practicability in real applications, the framework has been employed to model a resource discovery problem as well as to develop a unified foundation for *software configuration management* [11]. Their Web-based prototype systems have also been implemented, using *ETC* [3]—a compiler under the ET paradigm.

References

1. K. Akama. Declarative Semantics of Logic Programs on Parameterized Representation Systems. *Advances in Software Science and Technology*, 5: 45–63, 1993.
2. K. Akama, T. Shimitsu, and E. Miyamoto. Solving Problems by Equivalent Transformation of Declarative Programs. *J. of the Japanese Society of Artificial Intelligence*, 13(6): 944–952, 1998 (in Japanese)
3. K. Akama. ET Computational Paradigm. *Home Page*. Available at http://kr.cs.ait.ac.th/et
4. M. Burstein et al. DAML-S 0.5 Draft Release. May 2001. http://www.daml.org/services/ daml-s/2001/05
5. S. Decker, D. Brickley, J. Saarela and J. Angele. A Query and Inference Service for RDF. *Proc. Query Languages Workshop* (*QL'1998*), MA, 1998. http://www.w3.org/TandS/QL/QL98/pp/queryservice.html
6. S. Decker, S. Melnik, F.V. Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Harrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing*, 15(5): 63–74, Sep./Oct. 2000.
7. D. Brickley and R.V. Guha. RDF Schema Specification 1.0. *W3C Candidate Recommendation*. http://www.w3.org/TR/ rdf-schema/
8. R.V. Guha. rdfDB: An RDF Database. *Home Page*. http://web1.guha.com/rdfdb/
9. F. V. Harmelen and I. Harrocks. FAQs on OIL: The Ontology Inference Layer. *IEEE Intelligent Systems*, 15(6): 69–72, Nov./Dec. 2000.
10. J. Hendler, and D. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6): 72–73, Nov./Dec. 2000.
11. S. Kitcharoensakkul and V. Wuwongse. Unified Versioning using Resource Description Framework. *Annals of Software Engineering, Special Volume on Software Management*, 11, 2001.
12. O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. *W3C Recommendation*, February 1999. http://www.w3.org/TR/REC-rdf-syntax/
13. M. Marchiori and J. Saarela. Query + Metadata + Logic = Metalog. *Proc. Query Languages Workshop* (*QL'1998*), MA, 1998. http://www.w3.org/TandS/QL/QL98/pp/metalog.html
14. S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2): 46-53, Mar./Apr. 2001.
15. L. Miller. Inkling: RDF Query using SquishQL. *Home Page*. http://swordfish.rdfweb.org/rdfquery/

Appendix: Declarative Description Theory

This section recalls certain fundamental definitions of *Declarative Description* (*DD*) theory [1,2]—

https://doi.org/10.23106/dcmi.952106423

an axiomatic theory inspired by the concept of conventional logic programs with an attempt to cover a wider variety of data domains. The data structure of a given data domain is characterized by a mathematical abstraction, called a *specialization system*. Despite its simplicity, the specialization system provides a sufficient structure for the definition of *declarative descriptions* and their *meanings*. Thus, by appropriate construction of a specialization system for a given data domain, a framework for the representation and computation of data in that domain can be directly obtained. Correspondingly, in Section 2, DD theory is employed to develop the theory of RDF declarative descriptions.

A.1 Specialization systems

Definition 1 (Specialization System) Let $\mathcal{A}$, $\mathcal{G}$ and $\mathcal{S}$ be sets of *objects*, *ground objects*, and *specializations*, respectively, and μ be a mapping from $\mathcal{S}$ to *partial_map*($\mathcal{A}$) (i.e., the set of all partial mappings on $\mathcal{A}$). The quadruple $\langle\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu\rangle$ is a specialization system under the conditions:

1. $\forall s_1, s_2 \in \mathcal{S}, \exists s \in \mathcal{S} : \mu(s) = \mu(s_1) \circ \mu(s_2)$,
2. $\exists s \in \mathcal{S}, \forall a \in \mathcal{A} : \mu(s)(a) = a$,
3. $\mathcal{G} \subset \mathcal{A}$,

where $\mu(s_1) \circ \mu(s_2)$ is the composite mapping of the partial mappings $\mu(s_1)$ and $\mu(s_2)$. The set $\mathcal{G}$ is called the *interpretation domain*. □

In the sequel, let $\Gamma = \langle\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu\rangle$ be a specialization system. When μ is clear from the context, for $\theta \in \mathcal{S}$, $\mu(\theta)(a)$ will be written simply as $a\theta$. If there exists b such that $a\theta = b$, then θ is said to be applicable to a, and a is specialized into b by θ.

A.2 Declarative descriptions and their declarative semantics

A declarative description on Γ and other related concepts can now be defined.

Let a set $\mathcal{K}$ comprise constraint predicates. A *constraint* on Γ is a formula $q(a_1, \ldots, a_n)$, where q is a constraint predicate in $\mathcal{K}$ and a_i an object in $\mathcal{A}$. Given a ground constraint $q(g_1, \ldots, g_n)$, $g_i \in \mathcal{G}$, its truth and falsity are predetermined. Denote the set of all true ground constraints by *Tcon*. A specialization θ is applicable to a constraint $q(a_1, \ldots, a_n)$ if θ is applicable to $a_1, \ldots, a_n$. The result of $q(a_1, \ldots, a_n)\theta$ is the constraint $q(a_1\theta, \ldots, a_n\theta)$; and $q(a_1, \ldots, a_n)$ is said to be specialized into $q(a_1\theta, \ldots, a_n\theta)$ by θ.

Definition 2 (Declarative Description) A *clause* on Γ is a formula of the form:

$$H \leftarrow B_1, B_2, \ldots, B_n$$

where $n \geq 0$, H is an object in $\mathcal{A}$ and B_i an object in $\mathcal{A}$ or a constraint on Γ. H is called the *head* and $(B_1, B_2, \ldots, B_n)$ the *body* of the clause. A *declarative description* or simply a *description* on Γ is a (possibly infinite) set of clauses on Γ. □

The head of C will be denoted by *head*(C) and the set of all objects and constraints in the body of C by *object*(C) and *con*(C), respectively. Let *body*(C) = *object*(C) ∪ *con*(C). A clause C' is an *instance* of C iff there is a specialization $\theta \in \mathcal{S}$ such that θ is applicable to $H, B_1, B_2, \ldots, B_n$ and $C' = C\theta = (H\theta \leftarrow B_1\theta, B_2\theta, \ldots, B_n\theta)$. A clause C is a *ground clause* iff C comprises only ground objects and ground constraints.

Let P be a declarative description on Γ. Associated with P is the mapping T_P on $2^{\mathcal{G}}$ defined by: For each $X \subset \mathcal{G}$, a ground object g is contained in $T_P(X)$ iff there exist a clause $C \in P$ and a specialization $\theta \in \mathcal{S}$ such that $C\theta$ is a ground clause the head of which is g and all the objects and constraints in the body of which belong to X and *Tcon*, respectively, i.e.:

$$T_P(X) = \{head(C\theta) \mid C \in P,\ \theta \in \mathcal{S},\ C\theta \text{ is a ground clause},\ object(C\theta) \subset X,\ con(C\theta) \subset Tcon\}$$

Based on T_P, the meaning of P can now be defined.

Definition 3 (Semantics of a Declarative Description) Let P be a declarative description on Γ. The meaning of P, denoted by $\mathcal{M}(P)$, is defined by

$$\mathcal{M}(P) = \bigcup_{n=1}^{\infty} [T_P]^n(\varnothing)$$

where ∅ is the empty set, and $[T_P]^1(\varnothing) = T_P(\varnothing)$ and $[T_P]^n(\varnothing) = T_P([T_P]^{n-1}(\varnothing))$ for each $n > 1$. □

A.3 Equivalent Transformations

Equivalent Transformation (*ET*) [2,3] is a new computational model based on semantics-preserving transformations (*equivalent transformations*) of declarative descriptions. Computation by means of ET is carried out by successive transformation of a given description P_1 into P_2, P_3, … until a desirable description P_n is obtained; in the transformation process, the semantics of each description must be preserved, i.e., $\mathcal{M}(P_1) = \mathcal{M}(P_2) = \mathcal{M}(P_3) = \ldots = \mathcal{M}(P_n)$.

In order to guarantee the correctness of the computation, only equivalent transformations are applied at every step. The unfolding transformation, a widely-used program transformation in conventional logic programming, is a kind of equivalent transformation. Other kinds of equivalent transformation can also be devised, especially for improvement of computation efficiency. Thus, ET provides a more flexible, efficient computational framework.

https://doi.org/10.23106/dcmi.952106423