Towards Computation with RDF Elements

Chutiporn Anutariya[1], Vilas Wuwongse[1], Ekawit Nantajeewarawat[2] and Kiyoshi Akama[3]

ca@cs.ait.ac.th, vw@cs.ait.ac.th, ekawit@siit.tu.ac.th, akama@complex.eng.hokudai.ac.jp

[1]Computer Science & Information Management Program
Asian Institute of Technology
Pathumtani 12120, Thailand

[2]Department of Information Technology
Sirindhorn International Institute of Technology,
Thammasat University
Pathumtani 12120, Thailand

[3]Department of Information Engineering
Faculty of Engineering
Hokkaido University
Sapporo 060, Japan

Abstract

RDF is a World Wide Web Consortium recommendation with emphasis on facilities to exchange *human-readable* as well as *machine-understandable* information on the Web. However, computation with RDF data is very limited because the RDF itself provides users with only an expressive knowledge/information representation which has neither computational mechanism nor query processing capability. Thus, it is currently very difficult to manipulate and query RDF data in response to users' information need.

By employment of the *Declarative Program* (*DP*) theory, this paper proposes a theoretical framework for the representation of as well as computation and reasoning with RDF data, and presents an approach to the formulation and processing of RDF queries. Since the framework is developed with generality and expressibility in mind, it allows RDF data to be represented directly in their XML encoded forms without any necessity to translate them into other form. Moreover, the framework facilitates the use of rules to define relationships between RDF elements, hence allows the derivation of implicit information in RDF elements. Consequently, in addition to those simple, basic queries which are simply based on text/pattern matching, users can issue the queries about this derived information. Various examples of RDF inference rules as well as queries are also presented in the paper.

Keywords: Declarative Program, RDF element, RDF program, RDF query processing

1 Introduction

Resource Description Framework (*RDF*) is a unified, conceptual framework for encoding, exchange and reuse of *structured metadata* with a mechanism for the distribution of both *human-readable* and *machine-understandable* Web information [11]. By means of RDF data, users may not only represent and describe contents of Web resources in a more *machine-processable* manner, but may also be able to more accurately formulate specific queries yielding highly relevant and precise answers. However, computation with RDF data is very limited because the present formulation of RDF data, encoded in *XML* (*eXtensible Markup Language*) [4] syntax (to be referred to as *RDF elements*), does not address an inference or reasoning mechanism. Other means of computation which go beyond simple text/pattern matching are not readily devised. The *RDF Schema* (*RDFS*) Specification [5] includes features that need some basic inference facilities. For example, it contains transitive constructs such as *subClassOf* and *subPropertyOf* which include the notion of implication [7, 9]. *subPropertyOf* construct is used to specify that one property is a specialization of another. For example, if Property Editor is a subproperty of Property Contributor, and if an editor of a resource, say *X*, is John Smith, then it is implied that John Smith is also a contributor of the resource *X*. *subClassOf* construct is used to indicate subclass/superclass relation between classes. If, for instance, ElectronicMail is a subclass of ElectronicDocument which is also a subclass of ElectronicResource, then ElectronicMail is also implicitly a subclass of ElectronicResource; hence, resources that are instances of Class ElectronicMail will also be instances of Class ElectronicResource. Therefore, a search for resources in a general category should also examine resources in more specific categories.

In addition, to allow effective processing of RDF data, the knowledge/information representation provided by the RDF data model solely is insufficient; there arises an essential need for a standard query language for RDF, which enables users as well as agents to query and gather RDF data that precisely meet their requirements. Recently, there have been certain attempts to develop syntax and specifications for RDF query languages [7, 8, 12, 13]. Many of them, such as *RDF Query* [12], are

expressed in terms of XML syntax, using their predefined RDF Schemas or XML DTDs. Moreover, since RDF is defined using XML syntax, certain proposals apply a query language for XML, such as *XQL* [15] or *XML-QL* [8], as a query language for RDF. These proposals, however, aim at constructing user-friendly languages rather than providing effective computational mechanisms to process a query and obtain its results. *Metalog* [13] and *F-logic* [7] approaches, on the other hand, view *RDF statements* – *triples* of subjects, predicates and objects – as binary predicates in logic programming and as F-logic formulas, respectively, in order to provide the ability to reason with RDF statements. The processing of RDF queries are then performed on these corresponding translations instead of direct operation on RDF statements or on their XML encoded forms.

This paper attempts to develop a theoretical framework for the representation of RDF data and their relationships, which allows RDF data to be expressed directly in terms of XML syntax without any need for translation into other form. In addition, the framework provides a foundation for the development of computational mechanisms to effectively manipulate and reason about RDF elements, which will be useful for information retrieval, query processing, transformation, assembly and filtering.

The proposed approach employs *Declarative Program* (DP) theory [1, 2] – a generalization of the conventional logic program theory, in which terms are generalized into any *data objects* and substitutions into *specializations*. The theory has been developed with generality and applicability to data structures of any domains, each of which is characterized by a mathematical structure, called a *specialization system*. Employing the DP theory, this paper formulates an appropriate specialization system for RDF elements, and then develops a theory for *RDF Programs*. An RDF program comprises *facts* and *rules* which together describe resources and their relationships as well as some other derivable information. By means of the RDF program theory, this paper also develops an approach to the formulation and processing of RDF queries, which not only facilitates the selective retrieval and construction of RDF data, but also supports the reasoning capability. This would hence enable one to query about implicit information contained in RDF data. Furthermore, since RDF queries represented in other query languages can be easily translated into the proposed query formulation, one can employ XQL or XML-QL, for example, as an interface language and then apply the proposed approach to effectively execute the issued queries. Thus, the proposed approach can be utilized as the foundation for the processing of RDF queries expressed in other query languages.

Section 2 recalls fundamental definitions of the DP theory, Section 3 develops a specialization system for RDF elements and RDF Programs, Section 4 models Web resources, Section 5 presents RDF query formulation by means of RDF definite clauses and Section 6 draws conclusions.

2 Declarative Program Theory

A *specialization system* is an abstract structure derived from the generalization of *substitutions* in the conventional logic program theory, and defined in terms of certain simple axioms. Formally, a specialization system is a quadruple $\Gamma = <\mathcal{A}, \mathcal{G}, \mathcal{S}, \mu>$ of three sets $\mathcal{A}$, $\mathcal{G}$, $\mathcal{S}$ and a mapping μ from $\mathcal{S}$ to *partial_map*($\mathcal{A}$), i.e., the set of all partial mappings on $\mathcal{A}$, such that:

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*, the elements of $\mathcal{A}$, $\mathcal{G}$ and $\mathcal{S}$ are called *objects*, *ground objects*, and *specializations*, respectively. 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 $a\theta = b$, θ is said to be applicable to a, and a is specialized to b by θ. Given $a \in \mathcal{A}$, denote the set of all ground atoms that are specialized from a by *rep*(a), i.e., for $g \in \mathcal{G}$, $g \in$ *rep*(a) iff there exists a specialization θ in $\mathcal{S}$ such that $a\theta = g$.

A *definite clause*, or simply called a *clause*, on Γ is a formula of the form $H \leftarrow B_1, B_2, ..., B_n$, where H, $B_1, B_2, ..., B_n$ are *objects* in $\mathcal{A}$, often referred to as *atoms*. H is called the *head* and $(B_1, B_2, ..., B_n)$ the *body* of the clause. Let C be a definite clause $(H \leftarrow B_1, B_2, ..., B_n)$. The head of C will be denoted by *head*(C) and the set of all atoms in the body of C by *body*(C). A clause C is called a *unit clause* if *body* (C) is empty. A clause C' is an instance of C iff there is a specialization $\theta \in \mathcal{S}$ such that θ is applicable to $H, B_1, B_2, ..., B_n$ and $C' = C\theta = (H\theta \leftarrow B_1\theta, B_2\theta, ..., B_n\theta)$. A clause C is a ground clause iff all atoms in C are ground atoms in $\mathcal{G}$.

A *Declarative Program* on Γ is a (possibly infinite) set of definite clauses on $\mathcal{A}$. Let P be a

Declarative Program on Γ. The declarative semantics of P, $M(P)$, is a set of ground atoms defined by

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

where $\varnothing$ is the empty set, $[T_P]^n(\varnothing) = T_P([T_P]^{n-1}(\varnothing))$ and the mapping T_P: $2^{\mathcal{G}} \rightarrow 2^{\mathcal{G}}$ is defined as follows: for each $I \subseteq \mathcal{G}$, $g \in T_P(I)$ 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 atoms in the body of which belong to I.

3 Specialization System for RDF Elements and RDF Programs

RDF data model instances encoded in XML syntax (called *RDF elements*) are defined either in *serialization* or *abbreviated syntax*. The serialization syntax expresses the full capabilities of the RDF data model, while the abbreviated syntax includes additional constructs in order to provide a more compact form for representation of a subset of the data model. This paper considers only the serialization syntax.

By convention, RDF elements are *ground*, i.e., they contain no variable. However, in order to express implicit information contained in RDF elements, the RDF elements introduced in this paper can contain variables. The formal definition of RDF elements (with variables) will be given later in this section.

Next a *specialization system for RDF elements*, $\Gamma_R = <\mathcal{A}_R, \mathcal{G}_R, \mathcal{S}_R, \mu_R>$ will be constructed. Assume that Δ_R is an alphabet comprising the symbols in the following sets:

1. Σ : a set of *characters*,
2. T : a set of *tag names*,
3. B : a set of *attribute names* (or *qualifiers*),
4. *TVAR* : a set of *tag-name-variables* (or *T-variables*),
5. *BVAR* : a set of *attribute-name-variables* (or *B-variables*),
6. *SVAR* : a set of *string-variables* (or *S-variables*),
7. *PVAR* : a set of *attribute-value-pair-variables* (or *P-variables*),
8. *EVAR* : a set of *element-variables* (or *E-variables*), and
9. *IVAR* : a set of *intermediate-element-variables* (or *I-variables*).

T-, *B*-, *S*-, *P*-, *E*- and *I*-variables introduced here are useful for the expression of implicit information contained in RDF elements. Intuitively, a *T*-variable will be instantiated to a tag name in T, a *B*-variable an attribute name in B and an *S*-variable a string in Σ^*, while a *P*- and an *E*-variable will be specialized to attribute-value pairs and RDF elements, respectively. The use of *I*-variables will be explained later. In order to distinguish between elements of the above sets, assume that:

1. Every element in *TVAR* begins with "`$T:`", *BVAR* with "`$B:`", *SVAR* with "`$S:`", *PVAR* with "`$P:`", *EVAR* with "`$E:`" and *IVAR* with "`$I:`".
2. No element in T and B begins with "`$T:`" and "`$B:`", respectively, and '`$`' $\notin \Sigma$.

An RDF element on Δ_R is defined recursively as follows:

1. $evar \in EVAR$
2. $<tag\ pvar_1 \ldots pvar_k\ attr_1{=}val_1 \ldots attr_m{=}val_m/>$
3. $<tag\ pvar_1 \ldots pvar_k\ attr_1{=}val_1 \ldots attr_m{=}val_m>\ val_{m+1}\ </tag>$
4. $<tag\ pvar_1 \ldots pvar_k\ attr_1{=}val_1 \ldots attr_m{=}val_m>\ e_1 \ldots e_n\ </tag>$
5. $<ivar>\ val_1\ </ivar>$
6. $<ivar>\ e_1 \ldots e_n\ </ivar>$

where
- $l, m, n \geq 0$,
- $tag \in T \cup TVAR$,
- $pvar_i \in PVAR$,
- $attr_i \in B \cup BVAR$,
- $val_i \in \Sigma^* \cup SVAR$,
- $ivar \in IVAR$, and
- the e_i are RDF elements on Δ_R.

In general, the order of the $pvar_i$ (*P*-variables), the order of the pairs $attr_i{=}\ val_i$ (pairs of attribute name and value) and the order of the e_i (RDF elements) are immaterial. Only the order of those e_i nested in `rdf:Seq` elements is considered to be significant.

Let $\mathcal{A}_R$ be the set of all RDF elements on Δ_R and $\mathcal{G}_R$ the subset of $\mathcal{A}_R$ that consists of all ground (variable-free) RDF elements in $\mathcal{A}_R$.

Let `$X:P-Holder` be a special, reserved variable which is not included in Δ_R. An *intermediate element* on Δ_R has a similar form as an RDF element on Δ_R except that it always has exactly one occurrence of the variable `$X:P-Holder` nested inside, while an RDF element does not contain any occurrence of `$X:P-Holder`. Then, let $\mathcal{I}_R$ be the set of all intermediate elements on Δ_R.

Let $\mathcal{D}_R$ be $(TVAR \times TVAR) \cup (BVAR \times BVAR) \cup (SVAR \times SVAR) \cup (PVAR \times PVAR) \cup (EVAR \times EVAR) \cup (IVAR \times IVAR) \cup (PVAR \times (PVAR \times PVAR)) \cup (EVAR \times (EVAR \times EVAR)) \cup (EVAR \cup PVAR \times \{\varepsilon\}) \cup (TVAR \times T) \cup (BVAR \times B) \cup (SVAR \times \Sigma^*) \cup (PVAR \times (BVAR \times SVAR)) \cup (EVAR \times \mathcal{A}_R) \cup (IVAR \times \mathcal{I}_R)$. Elements of $\mathcal{D}_R$ are called *restrictions*. Let $a \in \mathcal{A}_R$. The specialization mapping ν_R: $\mathcal{D}_R \rightarrow partial\_map(\mathcal{A}_R)$ is defined as follows:

1. *Variable Renaming*

   When $d = (var_1, var_2) \in (TVAR \times TVAR) \cup (BVAR \times BVAR) \cup (SVAR \times SVAR) \cup (PVAR \times PVAR) \cup (EVAR \times EVAR) \cup (IVAR \times IVAR)$,

   $\nu_R(d)(a)$ is obtained from a by replacing all occurrences of var_1 in a by var_2.

2. *Variable Expansion*

   When $d = (var, (var_1, var_2)) \in (PVAR \times (PVAR \times PVAR)) \cup (EVAR \times (EVAR \times EVAR))$,

   $\nu_R(d)(a)$ is obtained from a by replacing all occurrences of var in a by the sequence var_1 var_2.

3. *Variable Removal*

   When $d = (var, \varepsilon) \in (EVAR \cup PVAR \times \{\varepsilon\})$, where ε denotes the null symbol,

   $\nu_R(d)(a)$ is obtained from a by removing all occurrences of var in a.

4. *Variable Instantiation*

   When $d = (var, value) \in (TVAR \times T) \cup (BVAR \times B) \cup (SVAR \times \Sigma^*) \cup (PVAR \times (BVAR \times SVAR)) \cup (EVAR \times \mathcal{A}_R)$,

   $\nu_R(d)(a)$ is obtained from a by replacing all occurrences of var in a by $value$.

   When $d = (ivar, e) \in IVAR \times \mathcal{I}_R$,

   $\nu_R(d)(a)$ is obtained from a by replacing each *ivar* element nested in a by the intermediate element e and replacing the variable `$X:P-Holder` in e by the content of that occurrence of *ivar* element.

From the definitions of $\mathcal{D}_R$ and the specialization mapping ν, which is used to determine the application of each restriction $d \in \mathcal{D}_R$ to an atom $a \in \mathcal{A}_R$, there are four types of restrictions:

1. the restrictions that rename variables,
2. the restrictions that expand *P*- or *E*-variables to sequences of variables of their respective types,
3. the restrictions that remove *P*- or *E*-variables, and
4. the restrictions that instantiate variables to some values which correspond to the types of the variables, i.e., a *T*-variable can be instantiated into only a tag name, a *B*-variable an attribute name, an *S*-variable a string, a *P*-variable an attribute-value pair, an *E*-variable an RDF element and an *I*-variable an intermediate element.

Next, let $\mathcal{S}_R = \mathcal{D}_R^*$, i.e., the set of all sequences of restrictions. Based on ν_R, the mapping μ_R: $\mathcal{S}_R \rightarrow partial\_map(\mathcal{A}_R)$ is defined by:

$\mu_R(\lambda)(a) = a$, where λ denotes the null sequence,

$\mu_R(d \cdot s)(a) = \mu_R(s)(\nu_R(d)(a))$, where $d \in \mathcal{D}_R$, $s \in \mathcal{S}_R$ and $a \in \mathcal{A}_R$.

That is, for each $a \in \mathcal{A}_R$ and $s \in \mathcal{S}_R$, where s is a sequence $d_1\ d_2\ \ldots\ d_n$, $n \geq 0$, $d_i \in \mathcal{D}_R$ and $i = 1, \ldots, n$, $\mu_R(s)(a)$ is obtained by successive applications of d_1, d_2, ..., and d_n to a. Also note that $\mu_R(s)(a)$ is defined only when all restrictions in s are successively applicable to a.

In the sequel, let $\Gamma_R = <\mathcal{A}_R, \mathcal{G}_R, \mathcal{S}_R, \mu_R>$. From the definitions of the sets $\mathcal{A}_R$, $\mathcal{G}_R$, $\mathcal{S}_R$ and the mapping μ_R, it is readily seen that Γ_R is a specialization system. The definitions of *RDF definite clauses*, *RDF Programs* and the *declarative semantics* of an RDF program are therefore obtained directly from the DP theory (cf Section 2).

4 Modeling of Web Resources

A Web resource described by RDF data can be represented directly as a ground RDF element in $\mathcal{G}_R$. A class of Web resources containing certain similar properties can also be represented as an RDF element with variables; these variables are used to

represent unknown or dissimilar properties among those resources in the class. For example, to represent a set of Web pages written by John Smith, one can simply construct an RDF element containing Creator property the value of which is John Smith, where other properties that vary from one another, such as Title and Language, are expressed implicitly through the use of variables. In addition to this simple representation, the proposed approach facilitates the use of rules to define (possibly complex) relationships between RDF elements. A rule is simply written as an RDF definite clause. A collection of Web resources, probably contains resources of different types each with different properties, can be specified by an RDF program P which is the union of a set of unit clauses – called facts, representing selected Web resources, and a set of non-unit clauses, representing rules. These rules are employed to derive information that is implicit in RDF elements. The declarative meaning of P, which is obtained by applying the rules to the set of atomic facts, then yield all the directly represented information together with all the derived information of the resources in the specified collection. Thus, by means of the RDF and the proposed approach, today's Web – the vast unstructured mass of information – can be transformed into a rich knowledge base rather than being merely a gigantic information repository.

Example 1 Let P be a program on Γ_R, comprising the following seven clauses:

```
C1: <rdf:Description
      about="http://www.w3.org/TR/rdf-intro">
       <dc:Title>
          Introduction to RDF data
       </dc:Title>
       <dc:Creator resource
         ="http://www.nokia.com/staffID/85740"/>
       <dc:Publisher>
          The World Wide Web Consortium
       </dc:Publisher>
       <dc:Type>Note</dc:Type>
       <dc:Language>en</dc:Language>
    </rdf:Description>      ←.

C2: <rdf:Description
      about="http://www.w3.org/TR/rdf-syntax">
       <dc:Title>The RDF Model and Syntax
       </dc:Title>
       <dc:Creator resource
         ="http://www.nokia.com/staffID/85740"/>
       <dc:Creator resource
         ="http://www.w3.org/staffID/12345"/>
       <dc:Publisher>The World Wide Web Consortium
       </dc:Publisher>
       <dc:Type>Technical Report</dc:Type>
       <dc:Language>en</dc:Language>
    </rdf:Description>      ←.

C3: <rdf:Description
      about="http://www.nokia.com/staffID/85740">
       <v:Name>Ora Lassila</v:Name>
       <v:Email>lassila@research.nokia.com
       </v:Email>
       <v:Affiliation>Nokia Research Center
       </v:Affiliation>
    </rdf:Description>      ←.

C4: <rdf:Description
      about="http://www.w3.org/staffID/12345">
       <v:Name>Ralph R. Swick</v:Name>
       <v:Email>swick@w3.org</v:Email>
       <v:Affiliation>
          The World Wide Web Consortium
       </v:Affiliation>
    </rdf:Description>      ←.

C5: <$I:AnyElement>W3C</$I:AnyElement>
        ←   <$I:AnyElement>
               The World Wide Web Consortium
            </$I:AnyElement>.

C6: <rdf:Description about=$S:RefResource>
       <CanSpeak>$S:Lang</CanSpeak>
       $E:PersonProperties
    </rdf:Description>
        ←   <rdf:Description
             about=$S:RefResource>
               $E:PersonProperties
            </rdf:Description>,
            <rdf:Description about=$S:URI>
               <dc:Creator
                resource=$S:RefResource/>
               <dc:Language>$S:Lang
               </dc:Language>
               $E:DocProperties
            </rdf:Description>.

C7: <$I:AnyElement>
       <$T:RefTag>
          <rdf:Description about=$S:URI>
             $E:PropertyElements
          </rdf:Description>
       </$T:RefTag>
    </$I:AnyElement>
        ←   <$I:AnyElement>
               <$T:RefTag rdf:resource=$S:URI/>
            </$I:AnyElement>,
            <rdf:Description about=$S:URI>
               $E:PropertyElements
            </rdf:Description>.
```

This program specifies a collection of Web resources, which contains four resources described by means of *Dublin Core* [16] and *vCard* [10] schemas, using the namespaces `dc` and `v`, respectively. The unit clauses (facts) C_1, C_2, C_3 and C_4 represent RDF elements describing those resources in the collection. The clause C_5 gives the abbreviated name of the World Wide Web Consortium; clause C_6 adds the property `CanSpeak` to the descriptions of persons, basing on the knowledge that if a person has written a document in a particular language, he/she must be able to speak that language; and, clause C_7 combines the content of a referring resource with the description of its referenced resource, i.e., the entire `Description` element of the referenced resource will become a nested element of that referring resource. The following nested `Description`, for instance, is derivable from the program P by specializing the two body atoms of the clause C_7 into the facts C_1

and C_3 (more formally, this nested `Description` is contained in $[T_P]^1(\varnothing) \subseteq M(P)$).

```
<rdf:Description
 about="http://www.w3.org/TR/rdf-intro">
   <dc:Title>Introduction to RDF data</dc:Title>
   <dc:Creator>
      <rdf:Description
      about="http://www.nokia.com/staffID/85740">
         <v:Name>Ora Lassila</v:Name>
         <v:Email>
            lassila@research.nokia.com
         </v:Email>
         <v:Affiliation>Nokia Research Center
         </v:Affiliation>
      </rdf:Description>
   </dc:Creator>
   <dc:Publisher>The World Wide Web Consortium
   </dc:Publisher>
   <dc:Type>Note</dc:Type>
   <dc:Lang>en</dc:Lang>
</rdf:Description>
```

The program P given in Example 1 will be used in the rest of this paper.

5 Query Formulation

An approach to the formulation of RDF query by means of RDF definite clauses will be presented. Here, a query is represented by a collection of one or more *rules*, each of which is simply written as a definite clause C, where *head*(C) describes the structure of the resulting RDF elements and *body*(C) describes selection criteria required by the query. Each query will be executed on some specified collection of Web resources, defined by means of an RDF program, and will return as its answer a set of RDF elements which describe those resources satisfying all the conditions of the query. Besides simply returning RDF elements which describe qualifying resources, it is possible to create a new element with the desirable structure from those selected elements. This can be achieved by an appropriate definition of the head atom. Moreover, since each query executes on a set of RDF elements and returns also a set of RDF elements, it is possible to issue a query against the result of another query, i.e., nested queries can be readily formulated.

With respect to the specified Web resource collection represented by an RDF program, the answer to a query is obtained by successive *equivalent transformation* [2] of the formulated query rules until they have become ground unit-clauses [3]. However, the detailed steps of how to execute a query and obtain its result set are beyond the scope of this paper.

The following subsections explain the formulation of all the basic query operations – *projection*, *selection* and *transformation* – that a particular query language for RDF should provide; these basic operations are suggested by [12, 14].

5.1 Projection

A query, which returns a set of resource descriptions with only selected properties, can be expressed as a single rule. The body of the rule consists of a single atom representing resources with complete descriptions. The head atom of the rule then extracts the required properties and constructs the projection results.

Example 2 (Projection) A query q_1 which returns only `title` and `publisher` of all resources having information about their titles and publishers in the collection is formulated as:

R_1 :
```
<answer>
  <rdf:Description>
    <dc:Title>$S:Title</dc:Title>
    <dc:Publisher>
      $S:Publisher
    </dc:Publisher>
  </rdf:Description>
</answer>
```
←
```
<rdf:Description about=$S:URI>
  <dc:Title>$S:Title</dc:Title>
  <dc:Publisher>
    $S:Publisher
  </dc:Publisher>
  $E:PropertyElements
</rdf:Description> .
```

The body atom of R_1 represents all the resources in the collection which have at least `title` and `publisher` properties. For each of these resources, the contents of the two properties are bound to the *S*-variables `$S:Title` and `$S:Publisher`, respectively, and if the resource has some other properties, they will be represented by the *E*-variable `$E:PropertyElements`. The head atom of R_1 specifies that the resulting resource descriptions contain only `title` and `publisher` properties.

5.2 Selection

Selection of resources which satisfy given conditions can be expressed by one or more rules depending on the relationship between each selection condition, i.e.,

- Selection conditions formed by *AND* will be expressed as a single rule.
- Selection conditions formed by *OR* are similar to selection using each condition separately and then combination of their results; therefore, the *OR* of n conditions can be expressed by n rules.

Each rule can contain one or more body atoms. These body atoms are used to represent the RDF elements to be selected. *Joining* of two or more body atoms (in the same rule) is achieved by an appropriate use of variables or values to specify the joined data.

Example 3 (Expressing the *AND* of two selection conditions) Let q_2 represent a query which returns the descriptions of those resources the author's names and resource types of which are Ora Lassila *AND* Technical Report, respectively. This query q_2 can be formulated as a single rule:

```
R2: <answer>
      <rdf:Description about=$S:URI>
         <dc:Creator resource=$S:RefResource/>
         <dc:Type>Technical Report</dc:Type>
         $E:PropertyElements
      </rdf:Description>
    </answer>
  ← <rdf:Description about=$S:RefResource>
      <v:name>Ora Lassila</v:name>
      $E:PropertyElementsOfOra
    </rdf:Description> ,
    <rdf:Description about=$S:URI>
      <dc:Creator resource=$S:RefResource/>
      <dc:Type>Technical Report</dc:Type>
      $E:PropertyElements
    </rdf:Description>.
```

Notice that the value of `Creator` property for each document-typed resource in the collection is a reference to another resource. Thus, to select all Technical Reports written by Ora Lassila, the rule R_2 is formulated with two atoms in the body. The first body atom represents the individual whose name is Ora Lassila and refers to the identifier of this individual by the *S*-variable `$S:RefResource`. The second body atom then represents all Technical Reports in the collection which are written by the individual referred to by `$S:RefResource`, i.e., the one whose name is Ora Lassila. The head of R_2 defines the `answer` elements each of which contains the description of each selected resource.

Alternatively, the query q_2 can be equivalently expressed by:

```
R2': <answer>
       <rdf:Description about=$S:URI>
          <dc:Creator>
             <rdf:Description $P:1>
                <v:Name>Ora Lassila</v:name>
                $E:PropertyElementsOfOra
             </rdf:Description>
          </dc:Creator>
          <dc:Type>Technical Report</dc:Type>
          $E:PropertyElements
       </rdf:Description>
     </answer>
   ← <rdf:Description about=$S:URI>
       <dc:Creator>
          <rdf:Description $P:1>
             <v:Name>Ora Lassila</v:name>
             $E:PropertyElementsOfOra
          </rdf:Description>
       </dc:Creator>
       <dc:Type>Technical Report</dc:Type>
       $E:PropertyElements
     </rdf:Description>.
```

By means of the rule C_7 (cf. Example 1), which integrates the description of the referenced resource into the description of each referring resource, users have the ability to directly specify queries' selection criteria using nested RDF elements. Hence, instead of having two body atoms representing the selection criteria as in R_2, R_2' contains only one body atom which explicitly specifies that `Name` property of any `Creator` elements nested within `Descriptions` of Technical Reports must be Ora Lassila.

Example 4 (Joining elements by values) The query q_3, which returns those resources written by the one who has also written the resource located at `http://www.w3.org/TR/rec-xml/`, can be formulated as:

```
R3: <answer>
      <rdf:Description about=$S:URI>
         <dc:Creator resource=$S:CreatorURI/>
         $E:PropertyElements2
      </rdf:Description>
    </answer>
  ← <rdf:Description
      about="http://www.w3.org/TR/rec-xml/">
        <dc:Creator resource=$S:CreatorURI/>
        $E:PropertyElements1
    </rdf:Description> ,
    <rdf:Description about=$S:URI>
      <dc:Creator resource=$S:CreatorURI/>
      $E:PropertyElements2
    </rdf:Description>.
```

Here, the rule R_3 is formulated with two atoms in the body, which will be denoted by b_{31} and b_{32}, respectively. The atom b_{31} represents the resource located at `http://www.w3.org/TR/rec-xml/` the creator of which is identified by the *S*-variable `$S:CreatorURI`. The atom b_{32} then represents all resources in the collection the creators of which are also the creator of the resource represented by b_{31}. The occurrences of `$S:CreatorURI` in both b_{31} and b_{32} clearly specify that (at least) one of the creators of the resource represented by b_{31} must also be one of the creators of the resources represented by b_{32}.

5.3 Transformation

The transformation of RDF elements conforming to one schema into another is expressed by a rule with single body atom, where its head and its body atom, respectively, represent the resulting elements and the original elements. This query operation is very useful especially for interchange of RDF elements. In the simplest case, the transformation involves merely renaming particular property names into others without restructuring the original elements. For example, one can simply issue a query which selects all Technical Reports in the collection and renames `Creator` property of those resources to `Author`. A more complicated example involves creation of new RDF elements which group multiple elements according to the specified properties. For instance, a user may wish to group together the

resources with the same author, i.e., for each author, an *RDF container* which comprises a *set* of resources written by that particular author would be created. Other examples of very complex queries are those that perform aggregation functions, such as *count()*, *min()* and *max()*, on sets of RDF elements. Formulation and processing of complex transformation queries which concern set construction, set manipulation and aggregation functions are parts of the theory being developed.

6 Conclusions

Encountering the problems of manipulating and querying RDF data, this paper has proposed a theoretical framework that formally extends the RDF data model with the computation and query processing capabilities. Based on the DP theory, which is general and applicable to data structures of any particular domains, the framework allows RDF elements – the representations of RDF data encoded in XML syntax – to be directly operable. In addition, the framework facilitates the use of rules to define relationships between RDF elements, hence allows the expression of not only simple queries that are based on text/pattern matching, but also those that question about implicit information in RDF elements. As one can see, the primary advantages of the DP approach over others are twofold: (i) it encourages the ability to effectively compute and reason with RDF data, and (ii) its expressive power allows RDF data to be represented directly in their XML encoded forms without any necessity and overheads to translate them into other form, which would help provide a more insight into computation with structured information in RDF elements.

Future research works include the development of query optimization techniques as well as the implementation of a prototype system which will help demonstrate and evaluate the effectiveness of the proposed framework. Moreover, in order to enforce integrity constraints on RDF data, the framework can be extended to manipulate and handle XML namespaces, RDF schemas [5] and *path and type constraints* [6].

References

1. Akama, K. Declarative Semantics of Logic Programs on Parameterized Representation Systems. *Advances in Software Science and Technology, Vol. 5, pp. 45-63, 1993.*
2. Akama, K., Shimitzu, T. and Miyamoto, E. Solving Problems by Equivalent Transformation of Declarative Programs (in Japanese). *Journal of the Japanese Society of Artificial Intelligence* (to appear).
3. Anutariya, C., Wuwongse, V., Nantajeewarawat, E. and Akama, K. Resource Description Framework (RDF) Programs. *Technical Report, Computer Science and Information Management Program, Asian Institute of Technology, 1998.*
4. Bray, T., Paoli, J. and Sperberg-McQueen, C.M. Extensible Markup Language (XML) 1.0. *http://www.w3.org/TR/REC-xml, 1998.*
5. Brickley, D., Guha, R.V. and Layman, A. Resource Description Framework (RDF) Schema Specification. *http://www.w3.org/TR/WD-rdf-schema-19981030/, 1998.*
6. Buneman, P., Fan, W. and Weinstein, S. Interaction between Path and Type Constraints. *Technical Report, Department of Computer Science and Information Science, University of Pennsylvania, 1998.*
7. Decker, S., Brickley, D., Saarela, J. and Angele, J. A Query and Inference Service for RDF. *http://www.purl.org/net/rdf/papers/QL98-queryservice-19981118, 1998.*
8. Deutsch, A., Fernandez, M., Florescu, D., Levy, A. and Suciu, D. XML-QL: A Query Language for XML. *http://www.w3.org/TR/NOTE-xml-ql-19980819.html, 1998.*
9. Guha, R.V., Lssila, O., Miller, E. and Brickley, D. Enabling Inferencing. *http://www.w3.org/TandS/QL/QL98/pp/enabling.html, 1998.*
10. Internet Mail Consortium. vCard: the Electronic Business Card. *http://www.imc.org/pdi/vcardwhite.html, 1998.*
11. Lassila, O. and Swick, R.R. Resource Description Framework (RDF) Model and Syntax Specification. *http://www.w3.org/TR/WD-rdf-syntax-19981008, 1998.*
12. Malhotra, A. and Sundaresan, N. RDF Query Specification. *http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html, 1998.*
13. Marchiori, M. and Saarela, J. Query + Metadata + Logic = Metalog. *http://www.w3.org/TandS/QL/QL98/pp/metalog.html, 1998.*
14. Quass, D. Ten Features Necessary for an XML Query Language. *http://www.w3.org/TandS/QL/QL98/pp/quass.html, 1998.*
15. Robie, J., Lapp, J. and Schach, D. XML Query Language (XQL). *http://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998.*
16. Dublin Core Metadata Initiative. Dublin Core Metadata Element Set: Reference Description. *http://purl.org/DC/about/element_set.htm, 1997.*