Towards A Language for Metadata Schemas for Interoperability

Vilas Wuwongse[1, 2] and Masatoshi Yoshikawa[2]
[1]Asian Institute of Technology, Pathumthani, Thailand
[2]Nagoya University, Nagoya, Japan
Email: vw@cs.ait.ac.th, yosikawa@itc.nagoya-u.ac.jp

Abstract: In order for metadata to be interoperable, its schema must be able to precisely describe its terms, context as well as semantic and syntactic constraints. With this precise information, other applications or machines can analyze, understand and utilize metadata. A metadata schema therefore demands an expressive language. Languages requirements are presented and a language to meet them is proposed.

Keywords: application profile; metadata; metadata schema; OWL/XDD; schema language

1 Introduction

Metadata is often defined as data about data, or, more precisely, as the information required to make data useful and widely usable. It has been recognized to be an important technology for the storage, management, discovery, access and utilization of data resources on the Internet. Various types of metadata exist such as metadata for cataloguing, usage terms and conditions, administration, content ratings and provenance. Metadata is typically expressed as a set of pairs of property type and value. A property type characterizes a feature of data that should be described as part of its metadata and is normally called a metadata term or metadata element. The meaning of a term must be precisely provided for it to be shareable and interoperable. A term can have qualifiers to enhance its given basic meaning. A property value can be a simple, atomic literal, or it can be a complex structure representing a group of related values or another data resource. There can be a constraint on the range of a property value. All this underlying information about metadata, i.e., its terms and values as well as their semantics, syntaxes and constraints are described in its schema. A metadata schema is required for metadata to be analyzed and understood by other applications.

A set of metadata terms may sometimes not be able to meet the requirements of a resource to describe all desirable and necessary aspects of its data, hence a combination of parts of different metadata sets may be required. A schema detailing such a combination is called an application profile (1). An application profile is thus a metadata schema which specifies required terms drawn from parts of one or more metadata sets, combined and optimized or constrained for a particular application. An application profile cannot define its own terms (1); it can only refine or constrain the existing ones from some metadata sets to suit the needs of its application. As a result, an application profile could become very complicated.

A metadata instance which conforms to a metadata set schema or an application profile can be expressed in many syntactical forms, e.g., HTML, XML or tables, depending on its real

implementations. The syntax of a metadata instance can be flexibly converted from one form to another while its meaning has to refer to its registered metadata set schema or application profile.

Sugimoto, et al. (2), in their proposal of a layered model of metadata schemas, state that a metadata schema defined for an application consists of three layers: Semantic Definition, Structural Constraints Definition and Implementation Dependent Syntax Definition Layers. A metadata schema is indeed required to describe information of a wide range of varieties and at different levels, hence it demands an expressive language.

Section 2 presents a detailed discussion of requirements for metadata schema languages, Section 3 proposes a language aiming to meet these requirements and Section 4 draws conclusions.

2 Requirements for Metadata Schema Languages

A language employed to specify a schema for shareable and interoperable metadata should possess the following properties or capabilities:

Being formal, precise and machine-processable

A metadata schema is the grammar for the verification of the well-formedness and validity of its metadata instances when they are created. Moreover, it is used by other application profiles for their schema definitions, and by other applications or machines for their analysis and understanding of metadata instances. Hence, it must be formally and precisely described so that, given a metadata schema, its syntactic and, if possible, semantic checkers, processors or analyzers can be readily developed.

Definition of the meaning of various types and domain-specific terms

When a large number of domains adopt the employment of metadata, various domain-specific metadata terms will be defined. Each domain has its own terms for property and value types as well as qualifiers used to enhance the meanings of term. There is the possibility that two or more different domains may use the same term with different meanings, i.e., an existence of homonym problems. These problems might be easily solved syntactically by prefixing terms with their domain name-spaces, but, to be able to distinguish their real differences, one has to refer to their detailed meaning definitions. In addition to defining terms of different types and domains, there is also a need to define terms at different levels: Metadata term set and application profile levels. At the application profile level, a term is defined by refining a term belonging to one of the existing metadata sets. A good schema language must provide necessary constructs for these complex term definitions and refinements.

Expression of term relationships

There can be relationships between terms in a metadata term set, e.g., the inverse relationships between the Dublin Core (DC) qualifiers *IsRequiredBy* and *Requires* and between the qualifiers *IsFormatOf* and *HasFormat*. In DC, if it is always the case that the content of element *Description* contains all the keywords in element *Subject*, then there is a subsumption relationship between the

two elements, i.e., *Description* subsumes *Subject*. Besides these relationships in-between the same type of terms, there can also exist relationships across different term types, e.g., between a property type and a value type. For example, in a collection of Buddhist documents, property type *Date* may be specified to have its values belonging to Buddhist calendar year type. Term relationships become more diversified and complicated when it comes to application profiles. In an application profile, terms can be drawn from a number of different metadata term sets. Some of these terms are equivalent, some are more general than others, and some may be generated by composition, union or intersection of some others.

Expression of constraints, conditions and rules

As a technique for refinement, application profiles may impose certain constraints on existing metadata terms. For example, the range of a value type may be narrowed down. In addition, an application profile may introduce conditions on how its adopted and refined terms can be used in metadata instances: Whether they are mandatory or optional, what their minimal and maximal occurrences are and whether they are repeatable. Some general rules or guidelines may need to be described so that only proper metadata instances are created. As an example, consider the Dump-Down Principle (2) which provides a guideline for the application of qualifiers. It states that the value of a term with qualifiers must be consistent with that of the term when the qualifiers are removed. This principle is considered to be important for interoperability and should be satisfied by any term created. A language describing this principle should enable easy implementation of its automatic verifier.

Specification of syntactic transformation

A sharable metadata instance of a data resource may simultaneously appear in many syntactic forms depending on user or implementer requirements. Consequently, the metadata schema of an application profile must be able to specify how its metadata instances may be transformed from their internal syntax into a required syntactic format. A good metadata schema language must also be a flexible syntactic transformation specification language.

The next section presents a language which attempts to satisfy all the above properties or capabilities.

3 OWL/XDD

OWL/XDD is a language which combines OWL (Web Ontology Language) (3) and XDD (XML Declarative Description) (4). OWL, a W3C's recommendation, is a language for describing ontologies as well as their schemas. OWL can formally and precisely specify concepts or terms and their various relationships. However, it cannot express complex constraints and rules. On the other hand, XDD is a general XML-based information representation language with well-defined semantics and the capability of expressing constraints and rules. OWL/XDD incorporates OWL into XDD by basing XDD's basic constructs, i.e., XML expressions or XML elements with variables, on OWL elements and their semantics. In other words, OWL/XDD is an extension of OWL in which its elements are allowed to have

variables and their relationships expressed as constraints and rules. Ordinary OWL elements and those with variables are together called *OWL expressions*. An ordinary OWL element without any variable is specifically called a *ground OWL expression*. Every component of an OWL expression can contain variables, e.g., its expression or a sequence of sub-expressions (*E-variables*), tag names or attribute names (*N-variables*), strings or literal contents (*S-variables*), pairs of attributes and values (*P-variables*) and some partial structures (*I-variables*). Every variable is prefixed by '$*T*:', where *T* denotes its type; for example, $S:value and $E:expression are *S*- and *E*-variables, which can be specialized into a string and a sequence of OWL expressions, respectively. As a result, OWL/XDD is an XML-based language which can formally and precisely specify terms, their semantics and syntaxes, term relationships, constraints and rules. The rules can include rules for syntax transformation, hence OWL/XDD is a language which can be used to describe metadata schemas and has a potential to meet the requirements discussed in the preceding section._A complete specification of a metadata schema in OWL/XDD is called an *OWL/XDD description*.

Formally, an *OWL/XDD description* is a set of *OWL clauses*, each of which has the form:

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

where $n \geq 0$, H is an OWL expression, and B_i is an OWL expression, or a *constraint*. The order of the B_i is immaterial. H is called the *head* and (B_1, B_2,..., 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 OWL document, containing a set of OWL elements and describing a certain metadata schema, is directly mapped onto an OWL/XDD description comprising solely ground OWL unit clauses. A constraint in a clause is expressed by a first-order-logic formula in which connectives AND, OR and NOT can be used, enabling inclusion of negative and complex constraints (3). Since there exists an execution engine for general XDD descriptions, computation or reasoning with OWL/XDD descriptions can be readily carried out.

As an example, consider the OWL clause in Fig. 1. It states that, if a property type R is an inverse of a property type P, then, for any resource X the value of a property type P of which is a resource Y, one can infer that Y also has a property type R the value of which is the resource X. Note that OWL is an extension of RDF(S), hence it also employs RDF(S) vocabulary.

Employing simple examples, the remainder of this section outlines and demonstrates how OWL/XDD can satisfy the requirements presented in Section 2.

Being formal, precise and machine-processable

As mentioned, OWL/XDD is a combination of two formal languages, both of which possess well-defined semantics. Therefore, OWL/XDD is formal and precise. Moreover, it is based on XML and an OWL/XDD description is a well-formed XML document, hence it can be processed and analyzed by machines.

```
<$N:classB  rdf:about=$S:ResourceY>
                    $E:instance1Elmt
                    <$S:propertyR rdf:resource=$S:ResourceX/>
</$N:classB>
                    ℜ          <owl:ObjectProperty rdf:ID=$S:propertyR>
                                        <owl:inverseOf rdf:resource=$S:propertyP/>
                                        $E:inversePropertyElmt
                               </owl:ObjectProperty>,
                               <$N:classA  rdf:ID=$S:resourceX>
                                        <$S:propertyP rdf:resource=$S:ResourceY/>
                                        $E:XProperties
                               </$N:classA>,
                               <$N:classB  rdf:ID=$S:ResourceY>
                                        $E:YProperties
                               </$N:classB>.
```

Fig. 1 An example of OWL clauses

```
<owl:DatatypeProperty rdf:ID="Creator">
     <rdfs:domain rdf:resource="#GeneralDataResource"/>
_   <rdfs:range rdf:resource="#string"/>
</owl:DatatypeProperty>
```

Fig. 2 A general definition of property type *Creator*

```
<owl:Class rdf:ID="MyCollection">
     <rdfs:subClassOf rdf:resource="#GeneralDataResource"/>
     <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#Creator"/>
                <owl:minCardinality rdf:datatype= "#nonNegativeInteger">1</owl:minCardinality>
                <owl:maxCardinality rdf:datatype="#nonNegativeInteger">5</owl:maxCardinality>
        </owl:Restriction>
     </rdfs:subClassOf>
</owl:Class>
```

Fig. 3 An application profile component specifying *Creator* occurrence number

Definition of the meaning of various types and domain-specific terms

OWL/XDD is an extension of an ontology language OWL; it is thus equipped with constructs necessary to define property types and their values. Examples of such constructs are *rdfs:domain*, *rdfs:range*, *owl:hasValue*, *owl:minCardinality*, *owl:maxCardinality, owl:ObjectProperty*, *owl:Datatype-Property*. The OWL element in Fig. 2 defines Creator as a data property type the domain of which is General Data Resource and the range of which is character strings. As an example of definitions at the application profile level, the element in Fig. 3 specifies that each resource in My Collection, a subclass of General Data Resource, has at least one Creator and at most five Creators.

Expression of term relationships

OWL has some simple constructs which can be used to indicate property relationships, e.g., *owl:inverseOf*, *owl:InverseFunctionalProperty* and *owl:equivalentProperty*. The OWL element in Fig. 4 defines that Painter in Painting Collection is the same as DC's *Creator*. For complex relationships, one needs to employ OWL clauses. Furthermore, Paints can be defined to be *owl:inverseOf* Painter and the clause in Fig. 1 can be used to compute and derive required implicit information.

```
<owl:DatatypeProperty rdf:ID="Painter">
      <owl:equivalentProperty rdf:resource="http://pur1.org/metadata/dublin-core#Creator"/>
      <rdfs:domain rdf:resource="#PaintingCollection"/>
      <rdfs:range rdf:resource="#string"/>
</owl:DatatypeProperty>
```

Fig. 4 An example of term relationship specification

Expression of constraints, conditions and rules

Since the body and the head of an OWL/XDD clause, respectively, can be viewed as a condition and an action, rules and conditions can readily be modeled. Moreover, a component of a clause body can be any first-order logical formula and complex constraints can thus be expressible. For simple constraints such as the one in Fig. 3, one may only employ OWL constructs.

Specification of syntactic transformation

An OWL/XDD can also be interpreted as a rewriting rule in which its body is an input and its head an output, hence it can function like a rule in XSLT and perform syntax transformation. As long as the head of a clause is a well-formed XML element, its content can be in any format, hence one metadata schema element can be instantiated into a metadata element with a number of different forms.

4 Conclusions

Metadata will certainly be an indispensable technology for efficient sharing of information resources. Its usage will spread horizontally to cover a variety of domains and applications, and vertically to include both simple metadata as well as complex, structured-value and multi-layer metadata. A language employed to specify the schemas of such metadata must be expressive and readily extensible. Such a language has been sketched. Detailed representation of schemas, application to metadata schema registries as well as generation of metadata instances form part of future work.

References

1. R. Heery and M. Patel. Application profiles: mixing and matching metadata schemas. Ariadne Issue 25, 2000
2. S. Sugimoto, et al. Developing community-oriented metadata vocabularies: Some case studies. In Proc. Digital Libraries and Knowledge Communities, pp. 128-135, 2004
3. http://www.w3.org/TR/2004/REC-owl-features-20040210/
4. C. Anutariya, V. Wuwongse and K. Akama. XML Declarative Description

with First-Order Logical Constraints. In Proc. Knowledge Grid and Grid Intelligence, 2003