XML Declarative Description: A Language for the Semantic Web

Vilas Wuwongse and Chutiporn Anutariya, *Asian Institute of Technology*
Kiyoshi Akama, *Hokkaido University*
Ekawit Nantajeewarawat, *Sirindhorn International Institute of Technology*

Although XML and RDF are widely used languages for Web applications, they have insufficient mechanisms to fulfill the requirements of a Semantic Web. The authors show how XML Declarative Description expands the capabilities of XML and RDF to meet these requirements.

Today's Web serves primarily as a global space to present information for human consumption. However, it lacks a mechanism for providing information that machines can comprehend or process, allowing them to communicate and interoperate. So, software engineers must work hard to develop intelligent services and automated software agents because they must first agree on the data's syntax and semantics before hard-coding them into their applications. In addition, changes to syntax and semantics necessitate expensive application modifications.

We need a Semantic Web[1] with a well-established mechanism to express information that is machine-interpretable and allows syntactic and semantic interoperability among Web applications. Although XML (eXtensible Markup Language) and RDF (Resource Description Framework) offer foundations for, respectively, syntactic and semantic interoperability, their mechanisms cannot accomplish this goal. XML by itself will let the same semantic unit be expressed in more than one syntactic structure. XML, RDF, and RDF Schema combinations might solve this multiple-structures problem, but they would still lack expressive power. For example, axioms, conditions, and constraints could not be specified.[2]

OIL[3] (Ontology Inference Layer) and DAML+OIL[4] (DARPA Agent Markup Language + OIL) are two recent improved *frame-based* languages. They extend RDF Schema by richer sets of modeling primitives for representation of Boolean expressions, axioms, and property restrictions. The semantics of these languages involves mapping such extended RDF statements on corresponding representations in a particular logical theory—for example, on first-order logic sentences—followed by corresponding determination of their semantics. OIL's semantics relies on a translation into the description logic $\mathcal{SHIQ}$,[5] whereas DAML+OIL's is based on KIF (Knowledge Interchange Format)[6]—a language designed for knowledge interchange and based on first-order predicate logic. Their main inference services are class consistency and subsumption checking with other reasoning services, for example, query subsumption and query answering over classes and instances, reformulated in terms of subsumption checking.

All these extensions require additional formalisms for XML and RDF to define their semantics or specify axioms and constraints. We need an XML-based language with a single formalism that meets all the requirements of the Semantic Web. Its intended semantics should be precisely and formally definable under its single formalism, and it should support general inference mechanisms. *XML Declarative Description*[7] aims to fill this need.

An informal introduction

XDD[7] is a language that enables representation of

1094-7167/01/$10.00 © 2001 IEEE

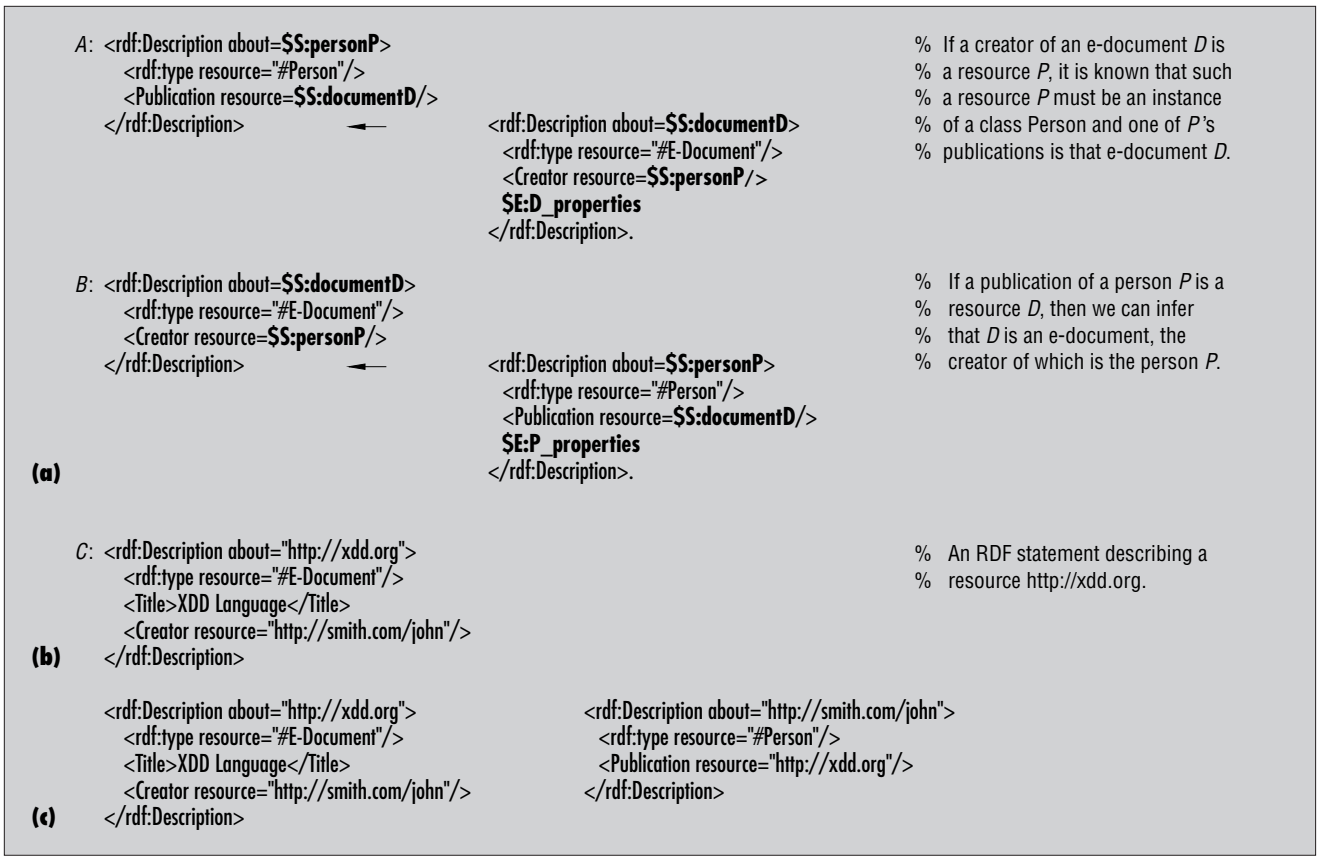

Figure 1. An XDD description example and its semantics: (a) XML clauses *A* and *B* model an inverse-relation axiom; (b) XML clause *C* models an RDF statement; (c) the semantics of an XDD description, which comprises the clauses *A*, *B*, and *C*, contains two RDF statements.

a Web resource's semantics. It employs XML as its bare syntax and enhances XML expressive power by employing Declarative Description theory.[8] A description in XDD is a set of ordinary XML elements, extended XML elements with variables, and the XML elements' relationships in terms of XML clauses. An ordinary XML element denotes a semantic unit and is a surrogate of an information item in the real application domain. An extended XML element represents implicit information or a set of semantic units. Clauses express rules, conditional relationships, integrity constraints, and ontological axioms. We define the precise and formal semantics of an XDD description as a set of ordinary XML elements, without employing other formalisms.

Important axioms that are missing in XML and RDF but expressible in XDD include *symmetry*, *composition-of*, and *inverse* relations. As an example of the inverse-relation axiom, consider the XML clauses *A* and *B* in Figure 1a. They model the `Creator` and `Publication` properties' inverses assumed by some particular domain. Figure 1b then gives an example of representing an RDF statement *C*, "A creator of a document entitled 'XDD language' is John." The semantics of an XDD description, which comprises the clauses *A*, *B,* and *C*, will also contain an RDF statement, "A publication of John is 'XDD language'" (Figure 1c), hence allowing inverse inference of such implicit information. Obviously, this axiom cannot be represented in RDF.

XDD can directly represent all XML-based application markup languages. It also can simply represent XML applications that provide common conventions of semantics, syntax, and structures for certain specific domains. In addition to RDF, these domains include the following:

- MathML (Mathematical Markup Language);
- XMI (XML Metadata Interchange Format—the Object Management Group's recommended technology for representing Unified Modeling Language and Meta Object Facility diagrams in XML); and
- WML (Wireless Markup Language).

Encoded in XDD, these languages can have their intended semantics formally defined. XDD removes the boundary and brings about the convergence of these languages' syntax and semantics. This readily enables interoperability of independently developed Web applications.

XDD applications include

- *e-commerce:* flexible communication, interaction, and collaboration schemes for negotiation agents;
- *resource discovery* and *digital libraries:* an intelligent search engine that understands and catalogs Web contents and incorporates knowledge into the search to improve precision and recall; and
- *software engineering:* automatic configuration of a new software component based on existing ones and certain specific configuration rules.[9]

Figure 2 depicts XDD's role in modeling the Semantic Web. The *Unicode layer* merely views exchanged data as a stream of Unicode characters. Next, the *XML layer* creates an XML document from the stream. Typically, the obtained document can be merely a document, an XML Schema, an RDF document, an RDF Schema, or a combination of these, because they are similarly encoded in XML syntax. The document's semantics is formally determined in the *XDD layer*, yielding a set of data objects that are encoded in

Specialization Systems

Let's look at the definition of specialization systems, defined in the *Declarative Description* theory.[1]

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

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

Intuitively, Conditions 1 to 3 mean

1. For all specializations s_1 and s_2, there exists a specialization s such that the corresponding partial mapping of s is the composition of the two mappings corresponding to s_1 and s_2,
2. There exists a specialization that does not change any objects (*identity specialization*), and
3. Ground objects are objects.

When μ is clear from the context, for $\theta \in S$, we write $\mu(\theta)(a)$ simply as $a\theta$. If b exists such that $a\theta = b$, θ is said to be applicable to a, and a is specialized to b by θ.

Reference

1. K. Akama, "Declarative Semantics of Logic Programs on Parameterized Representation Systems," *Advances in Software Science and Technology*, Iwanami Shoten, Publishers and Acadamic Press, Tokyo, vol. 5, 1993, pp. 45–63.

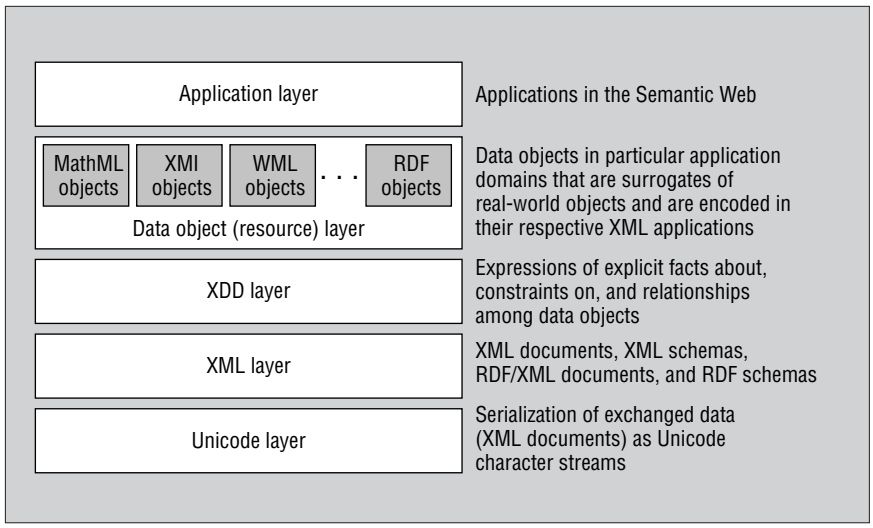

Figure 2. XDD's role in the Semantic Web architecture.

their respective XML applications in the *data object layer*. These objects can be explicitly described by the document or derived from the relationships, rules, or axioms in the document. In the *application layer*, such objects are surrogates of real-world objects in a particular application domain.

A formal introduction

XDD's words and sentences are *XML expressions* and *XML clauses*, respectively. XML expressions represent explicit and implicit as well as simple and complex facts. XML clauses represent ontology, implicit and conditional relationships, constraints, and axioms.

XML expressions and XML elements have a similar form. However, XML expressions can carry variables to represent implicit information and to enhance XML elements' expressive power. Every component of an XML expression—the expression itself, its tag name, attribute names and values, attribute–value pairs, contents, subexpressions, and some partial structures—can contain variables. XML expressions without variables are called *ground XML expressions* or simply *XML elements*. Those with variables are called *nonground XML expressions*. Table 1 defines all variable types and their usages.

Variable instantiation is defined by *basic specializations*, each of which has the form (v, w) where v specifies the name of the variable to be specialized and w the specializing value. For example, `($N:name1, $N:name2)`, `($S:url, "http://smith.com")` and `($E:properties, ($E:p1, $E:p2))` are basic specializations that rename the N-variable `$N:name1` as `$N:name2`, instantiate the S-variable `$S:url` into the string `"http://smith.com/"`, and expand the E-variable `$E:properties` into the sequence of the E-variables `$E:p1` and `$E:p2`. Basic specializations come in four types:

- rename variables,
- expand a P- or an E-variable into a sequence of variables of their respective types,

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)	`$P`	Sequences of zero or more attribute-value pairs
E-variables (XML expression variables)	`$E`	Sequences of zero or more XML expressions
I-variables: (Intermediate expression variables)	`$I`	Parts of XML expressions

- remove P-, E-, or I-variables, or
- instantiate variables to XML expressions or components of XML expressions that correspond to the variables' types.

A sequence of basic specializations is a *specialization*. The data structure of XML expressions is characterized by a mathematical abstraction $\Gamma_X = \langle \mathcal{A}_X, \mathcal{G}_X, S_X, \mu_X \rangle$, called the *XML Specialization System* (see the related sidebar), where

- $\mathcal{A}_X$ is the set of all XML expressions,
- $\mathcal{G}_X$ is the subset of $\mathcal{A}_X$ that comprises all ground XML expressions in $\mathcal{A}_X$,
- S_X is the set of all specializations that reflect the data structure of the XML expressions in $\mathcal{A}_X$, and
- μ_X is the *specialization operator*, which determines for each specialization s in S_X the change of each XML expression in $\mathcal{A}_X$ caused by s.

Figure 3a illustrates an example of a nonground XML expression a in $\mathcal{A}_X$. Figure 3b illustrates a specialization θ in S_X, and the application of θ to a by the operator μ_X to obtain a ground XML expression g in $\mathcal{G}_X$ (Figure 3c). That is $g = \mu_X(\theta)(a)$ or, by postfix notation, $g = a\theta$. Specialization θ changes the nonground expression a to the ground expression g by

- instantiation of the N-variable `$N:title` into the tag name `dc:Title`,
- instantiation of the S-variable `$S:url` into the string `"http://smith.com"`,
- expansion of the E-variable `$E:properties` to the sequence of the E-variables `$E:p1` and `$E:p2`,
- instantiation of the E-variable `$E:p1` into the XML expression `<dc:Creator resource="http://smith.com/john"/>`, and
- instantiation of the E-variable `$E:p2` into the XML expression `<dc:Language>English</dc:Language>`.

Constraints are useful for defining restrictions on XML expressions or components of XML expressions. A constraint is a formula $q(a_1, \ldots, a_n)$, where $n > 0$, q is a constraint predicate, and a_i is an XML expression. The application of $\theta \in S_X$ to a constraint $q(a_1, \ldots, a_n)$ yields $q(a_1\theta, \ldots, a_n\theta)$. A *ground constraint*, which takes the form $q(g_1, \ldots, g_n)$, $g_i \in \mathcal{G}_X$, has a predetermined truth or falsity. For instance, given two XML expressions a_1 and a_2, define $\mathtt{GT}(a_1, a_2)$ as a constraint that will be true if and only if a_1 and a_2 are XML elements of the forms `<Num>`v_1`</Num>` and `<Num>`v_2`</Num>`, respectively, where v_1 and v_2 are numbers and $v_1 > v_2$.

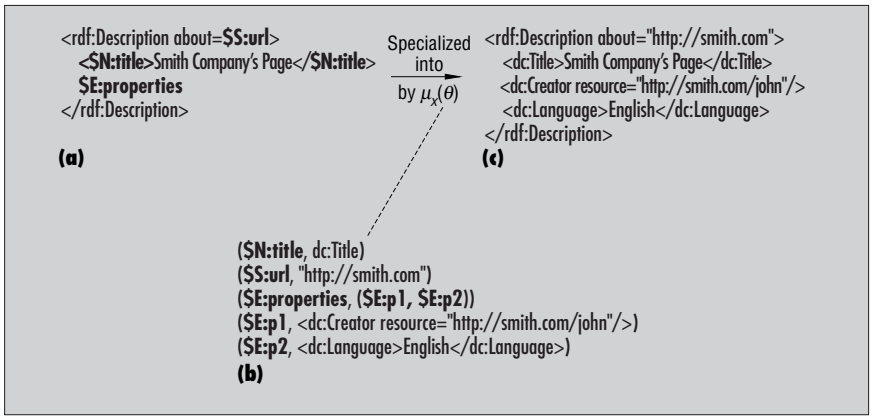

Figure 3. Specialization of a nonground XML expression into a ground XML expression: (a) nonground XML expression *a*; (b) specialization θ; (c) ground XML expression $g = a\theta$.

Based on the XML specialization system Γ_X and the concept of constraints, an *XML declarative description* on Γ_X, simply called an XDD description, is a set of XML clauses. Each clause has the form

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

where

- $n \geq 0$,
- H is an XML expression in $\mathcal{A}_X$,
- B_i is an XML expression in $\mathcal{A}_X$ or a constraint on Γ_X, and
- the order of the B_i is immaterial.

H is the *head* and $(B_1, B_2, \ldots, B_n)$ is the *body* of the clause.

For a *unit clause*, $n = 0$; for a *non-unit clause*, $n > 0$. Normally, we use $(H \leftarrow)$ to indicate a unit clause. However, when the context clearly implies a unit clause, we will write it simply as H. With this representation, every XML document becomes immediately an XDD description without a non-unit clause.

The meaning of a given XDD description P, denoted by $\mathcal{M}(P)$ (see the "Semantics of Declarative Description" sidebar), is the set of all XML elements that are directly described by and are derivable from the unit

The Semantics of Declarative Description

Here we formally define a given declarative description's semantics on a particular specialization system.

Let $\Gamma = \langle \mathcal{A}, \mathcal{G}, S, \mu \rangle$ be a specialization system and C be a clause $(H \leftarrow B_1, B_2, \ldots, B_n)$ 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. For a specialization $\theta \in S$, application of θ to the clause C is $C\theta = (H\theta \leftarrow B_1\theta, B_2\theta, \ldots, B_n\theta)$. A clause C is a *ground clause* if and only if C comprises only ground objects and ground constraints.

Let *Tcon* denote the set of all true ground constraints and P be a declarative description on Γ.

Associated with P is the mapping T_P on $2^{\mathcal{G}}$, which we define as this:

For each $X \subset \mathcal{G}$, a ground object g is contained in $T_P(X)$ if and only if a clause $C \in P$ and a specialization $\theta \in S$ exist such that $C\theta$ is a ground clause with the head g, and all the objects and constraints in the body of $C\theta$ belong to X and *Tcon*, respectively; that is, $T_P(X) = \{head(C\theta) \mid C \in P, \theta \in S, C\theta \text{ is a ground clause}, object(C\theta) \subset X, con(C\theta) \subset Tcon\}$.

Based on T_P, the meaning of P, denoted by $\mathcal{M}(P)$, is defined as

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

where $\varnothing$ 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 $n > 1$.

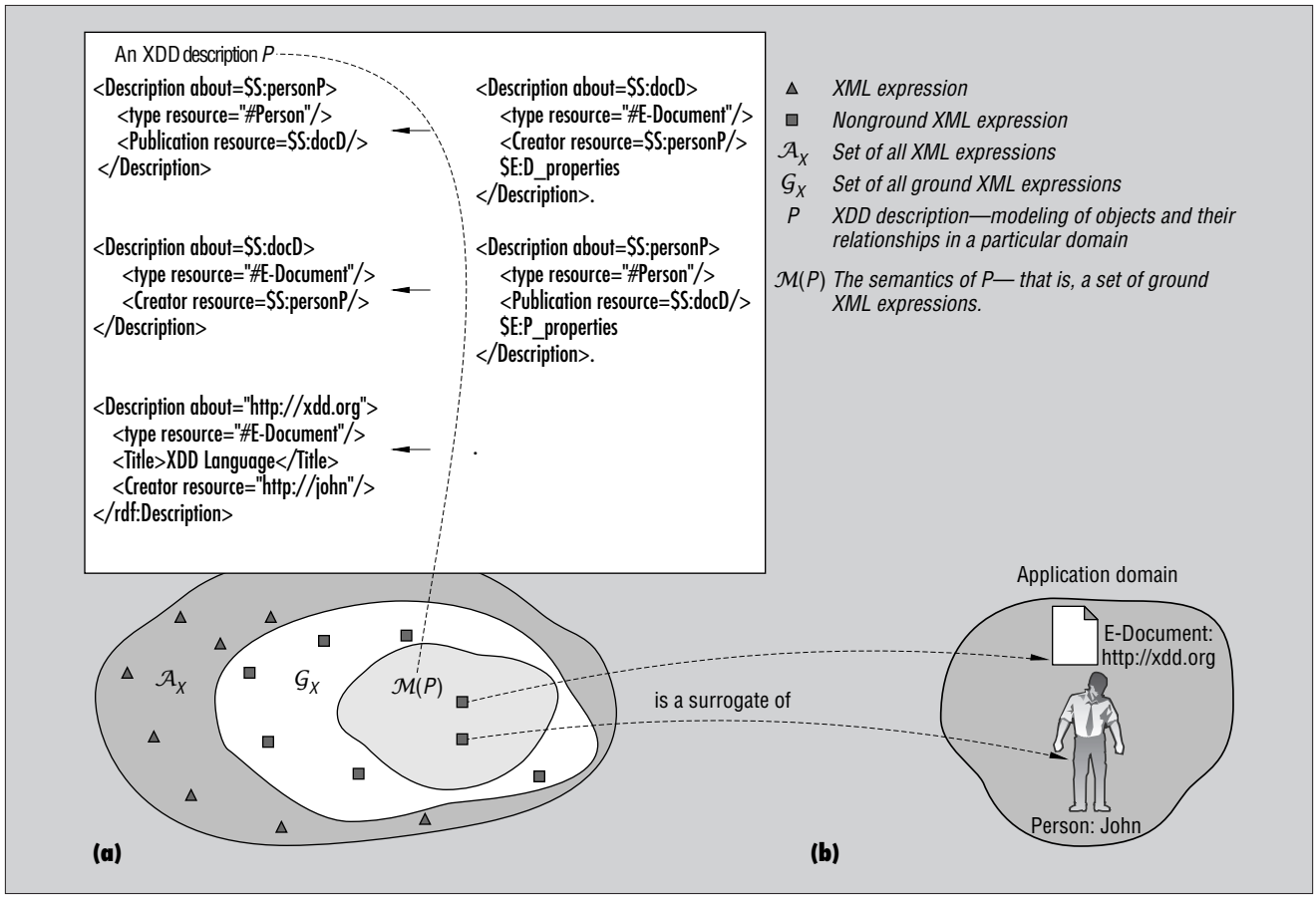

Figure 4. The relationships between XDD and a real-world domain.

and the non-unit clauses in P, respectively—that is,

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

Figure 4 illustrates an XDD approach to modeling objects (Figure 4a) and the relationships between XDD language and a real-world domain (Figure 4b).

XDD and the Semantic Web

Interoperation among various applications in the Semantic Web demands common representation and interpretation of exchanged data. Each application domain requires an appropriate definition of standard document syntax together with an agreement about, or a common understanding of, the employed ontology.[10] In general, an ontology is a specification of concepts, their hierarchical relationships and axioms in a particular application domain. Objects in the domain are instances of one or more concepts. Each concept also contains a set of properties.

From this point of view, the Semantic Web's components are constraints, ontologies, and contents, all of which can be modeled and manipulated by XDD:

- Constraints or restrictions on the information exchange format can be represented and imposed by a corresponding set of XML non-unit clauses.
- Concepts and their properties in an ontology are described as XML unit clauses. Their hierarchical relationships and ontological axioms, such as symmetry and inverse, are modeled as XML non-unit clauses. As an example of modeling concept hierarchies, consider the expression that the concept "Web-Page" is a specialization of the concept "E-Document," which can be represented as a clause, assuming that concepts are described as tag names in XML (see Figure 5).
- Contents, describing certain objects and their relationships, are also modeled as XML unit and non-unit clauses, respectively.

Table 2 describes the Semantic Web com-

```
...                                          ...
<E-Document> SE:X </E-Document> ◄— <Web-Page> SE:X </Web-Page> % This clause specifies that
...                                          ...                % every Web page object is
                                                                % an e-document object.
```

Figure 5. An example of modeling a concept hierarchy.

ponents and XDD's role in modeling each component. On the basis of this modeling technique, a Web resource, modeled as the XDD description P, will uniquely convey its meaning, represented as a set of ground XML expressions in terms of $\mathcal{M}(P)$.

Modeling the Semantic Web

We now show how to employ XDD to model the Semantic Web. This example assumes that RDF syntax, RDF Schemas, and RDF statements describe the constraints, ontologies, and contents, respectively. Moreover, we show how to apply XDD to axiom modeling—an important notion missing from the RDF framework. We also give the semantics.[11]

Constraint modeling: RDF syntax

For simplicity, consider the partial RDF serialization syntax in Figure 6a, which we can represent simply as the XDD description P_1 in Figure 6b. Based on P_1, we formulate the clause V to determine whether the RDF statement of Figure 1b conforms to the given RDF Syntax or not (see Figure 6c). If the statement is valid, $\mathcal{M}(P_1 \cup \{V\})$ will include the XML expression `<xdd:ValidDescription about="http://xdd.org"/>`.

We can similarly define XML clauses, which can parse and validate complete RDF serializations as well as abbreviated syntax.

Ontology modeling: RDF Schemas and axioms

RDF Schema is a language that provides a simple ontology definition facility. Two essential constructs, `subClassOf` and `subPropertyOf`, let you specify hierarchical relationships among a set of classes and properties, respectively. The constructs `range` and `domain` let you impose constraints on a property's value and on the types (classes) of objects to which a property can be applied, respectively.

Figure 7 shows an RDF Schema graph and its corresponding RDF Schema document, which defines a simple ontology for describing electronic resources. That is,

- `Person`, `E-Document`, `E-Article`, and `Web-Page` are `rdfs:Class`,
- `Person` and `E-Document` are subclasses of `rdfs:Resource`,
- `E-Article` and `Web-Page` are subclasses of `E-Document` (Figure 7a),
- `Publication`, `Name`, `Title`, `Creator`, `Author`, and `Illustrator` are `rdf:Property`, and
- `Author` and `Illustrator` are subproperties of `Creator` (Figure 7b).

Because RDF Schema documents are XML documents, they correspond directly to XDD descriptions containing only unit clauses. So, the example document in Figure 7c directly becomes the XDD description P_2. P_2 comprises 10 XML unit clauses, the heads of which are the RDF statements in the example document.

To model the meanings of `subClassOf` and `subPropertyOf`, which are transitive and include some notion of implication, we formulated the XML non-unit clauses C_6 through C_9 in Figure 8. These clauses constitute XDD description P_3; that is, $P_3 = \{C_6, C_7, C_8, C_9\}$. The semantics of $P_2 \cup P_3$ explicitly yields the following implicit information and relations:

- `E-Article` and `Web-Page` are subclasses of `rdfs:Resource` (through the `subClassOf` transitivity property).
- Every instance of `E-Article` or `Web-Page` is also an instance of the class `E-Document`.
- Every instance of `Person`, `E-Document`, `E-Article`, or `Web-Page` is also an instance of `rdfs:Resource`.
- Every resource having an `Author` or an `Illustrator` property also has a `Creator` property with a value similar to that of `Author` or `Illustrator`.

Additional relations among classes or properties, as well as the meanings of `rdfs:range` and `rdfs:domain` constraints, are also expressible in XDD. Moreover, by facilitating specifications of rules, ontological axioms, and conditions and constraints on classes and properties, XDD overrides RDF Schema, which permits only a simple ontological modeling mechanism. As we mentioned before, Figure 1a shows the modeling of the inverse of the `Creator`–`Publication` axiom. XDD description P_4 denotes the set of clauses A and B in the figure; that is, $P_4 = \{A, B\}$. Clause A describes that the inverse of `Creator` is the `Publication` property; clause B represents the reverse. Other kinds of axioms can be modeled similarly.

Table 2. Modeling the Semantic Web.

Semantic Web component		Expressed as
Constraints on the information exchange format		XML non-unit clauses
Ontologies		
Concept and property descriptions		XML unit clauses
Hierarchies of concepts and properties		XML non-unit clauses
Axioms		XML non-unit clauses
Contents		
Objects		XML unit clauses
Relationships among objects		XML non-unit clauses
A resource of the Semantic Web (contents + ontology + constraints)	Modeled as ⇒	An XDD description P on Γ_X comprising XML unit clauses + XML non-unit clauses
The semantics of the resource	Is	$\mathcal{M}(P)$

Content modeling: RDF statements

RDF statements, which model given contents of the Semantic Web and are encoded in XML syntax, are directly mapped onto XML elements or XML unit clauses in XDD. Figure 9 shows four RDF statements that describe certain Web resources; we call these statements XDD description P_5.

The semantics

The union of the XDD descriptions P_1 through P_5 becomes a simple example of the Semantic Web modeling—that is,

$P = P_1$	Constraints on the data-exchange format (RDF syntax).
$\cup P_2$	Concept and relation descriptions (RDF Schema).
$\cup P_3$	Concept and relation hierarchies (modeling the meanings of the RDF Schema constructs `subClassOf` and `subPropertyOf`).
$\cup P_4$	Axioms (inverses of the `Creator` and `Publication` properties).
$\cup P_5$	Resources of the Semantic Web (RDF statements).

```
rdf-statement  ::= <rdf:Description about=string> propertyElt* </rdf:Description>
    propertElt  ::= <propName> string </propName> |
                    <propName resource=string>
         string ::= (any XML text)
    propName ::= (any valid property name)
```

(a)

```
C1: <xdd:DescriptionCheck>
        <rdf:Description about=$S:uri>
          $E:propertyElt
        </rdf:Description>
    </xdd:DescriptionCheck>  ←  <xdd:PropertyCheck>
                                    $E:propertyElt
                                </xdd:PropertyCheck>.

C2: <xdd:DescriptionCheck>
        <rdf:Description ID=$S:uri>
          $E:propertyElt
        </rdf:Description>
    </xdd:DescriptionCheck>  ←  <xdd:PropertyCheck>
                                    $E:propertyElt
                                </xdd:PropertyCheck>.

C3: <xdd:PropertyCheck>
        <$N:propName>
          $S:value
        </$N:propName>
        $E:propertyElt
    </xdd:PropertyCheck>     ←  <xdd:PropertyCheck>
                                    $E:propertyElt
                                </xdd:PropertyCheck>.

C4: <xdd:PropertyCheck>
        <$N:propName resource=$S:uri/>
        $E:propertyElt
    </xdd:PropertyCheck>     ←  <xdd:PropertyCheck>
                                    $E:propertyElt
                                </xdd:PropertyCheck>.

C5: <xdd:PropertyCheck>
    </xdd:PropertyCheck>     ←  .
```

```
% This clause together with the clause C2 specifies that
% every rdf:Description element must contain exactly
% one about or ID attribute and may contain zero or
% more property elements. The xdd:PropertyCheck
% element contained in the body restricts that such
% property elements will be further restricted by the
% clauses whose heads are xdd:PropertyCheck—that is,
% C3, C4, and C5.
```

```
% This clause and the clause C4 impose restrictions on
% the syntax of every property element, contained in
% rdf:Description. That is,
% • Its tag name could be any valid property name
%   (represented by the name variable $N:propName),
% • If its content is any textual string (represented by
%   the String variable $S:value), it must contain no
%   attribute-value pair; and
% • If its content is empty, it must contain exactly
%   one resource attribute.
```

```
% This clause together with the clauses C3 and C4 restricts
%  that rdf:Description may contain zero or more
% property elements.
```

(b)

```
V: <xdd:ValidDescription about=" http://xdd.org"/>
                ←  <xdd:DescriptionCheck>
                     <rdf:Description about="http://xdd.org">
                       <rdf:type resource="#E-Document"/>
                       <Title>XDD Language</Title>
                       <Creator resource="http://smith.com/john"/>
                     </rdf:Description>
                   </xdd:DescriptionCheck>.
```

```
% This clause specifies that if the RDF
% statement contained in its body
% conforms to the syntax described by
% the clauses C1 through C5, the XML
% expression specified in its head will
% be obtained.
```

(c)

Figure 6. Constraint modeling: (a) a partial RDF serialization syntax; (b) its representation as the XDD description $P_1 = \{C_1, \ldots, C_5\}$; (c) an XML clause for checking the conformance of the RDF statement of Figure 1b.

The semantics of P, $\mathcal{M}(P)$ includes this information, which is implicit in the Semantic Web's content:

- All RDF statements in P_2 (see Figure 7c) and P_5 (see Figure 9) conform to the RDF syntax.
- The resources `"http://smith.com"` and `"http://xdd.org"` are instances of `E-Document`.
- The resources `"http://smith.com"`, `"http://xdd.org"`, `"http://smith.com/john"`, and `"http://smith.com/joe"` are instances of `rdfs:Resource`.

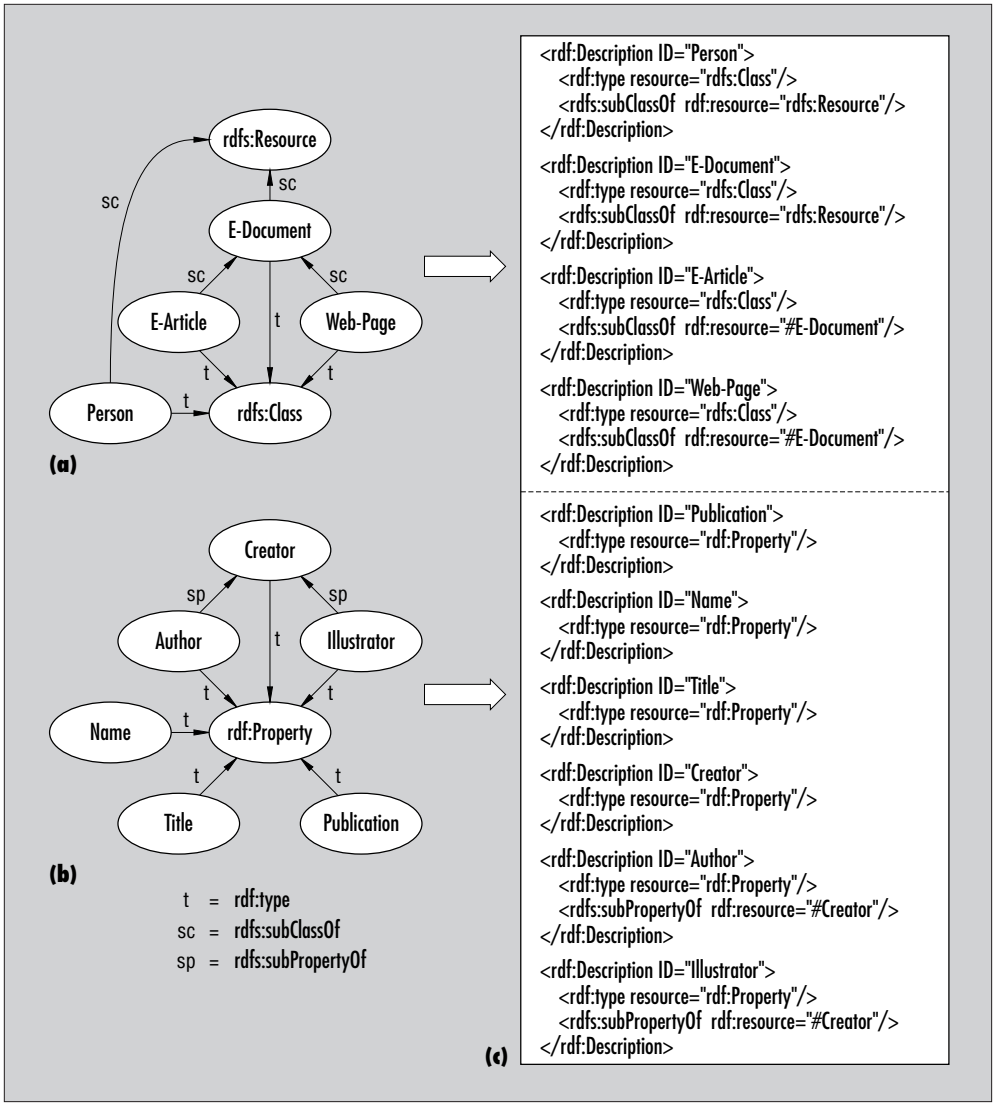

Figure 7. An RDF Schema graph and its corresponding RDF-Schema document: (a) a class model; (b) a property model; (c) an RDF Schema document denoted by XDD description P_2.

- The person referred to by `"http://smith.com/john"` is a `Creator` of `"http://smith.com"` and `"http://xdd.org"`.
- The person referred to by `"http://smith.com/joe"` is a `Creator` of `"http://xdd.org"`.
- The resources `"http://smith.com"` and `"http://xdd.org"` are `Publications` of the person referred to by `"http://smith.com/john"`.
- The resource `"http://xdd.org"` is a `Publication` of the person referred to by `"http://smith.com/joe"`.

So, a search of all `E-Document` resources of which John Smith is a `Creator`, for instance, will return also the resources `"http://smith.com"` and `"http://xdd.org"`, although such resources have not been declared as `E-Document` and John Smith has not been declared explicitly as their `Creator` but only as `Author` and `Illustrator`. This implicit information is uncovered through the predefined hierarchical relationships among `E-Document`, `E-Article`, and `Web-Page` and among `Creator`, `Author`, and `Illustrator`.

Requirements of a Semantic Web language

Tim Berners-Lee pointed out that a good language for the Semantic Web should have

- compact syntax;
- well-defined semantics;
- sufficient expressive power to represent human knowledge;
- an efficient, powerful, and understandable reasoning mechanism; and
- the potential for building large knowledge bases.[12]

However, the third and fourth properties conflict. In addition, these five properties focus merely on the information representation and computation aspects; information presentation is missing. To emphasize that the Semantic Web should be a means for not only machine-to-machine communication but also machine-to-human communication, we should add a sixth property: it includes a presentation or rendering scheme.

Because XDD concentrates on the information representation aspect with an attempt to provide a concise, expressive language with precise, well-defined semantics, it has all but the fourth and sixth properties. These limitations can be resolved in two ways.

The first solution is to provide efficient computation. We employed ET (Equivalent Transformation),[13] a new computational paradigm, to allow efficient manipulation of and reasoning with XDD. (An equivalent transformation is a transformation that preserves the equivalence of the transformed descriptions—a semantic-preserving transformation.) We carried out computation through ET by successive transformation of a given XDD description R_1 into R_2, R_3, …, until we obtained a desirable XDD description R_n. During the transformation, we must preserve each XDD description's semantics—that is, $\mathcal{M}(R_1) = \mathcal{M}(R_2) = \mathcal{M}(R_3) = ... = \mathcal{M}(R_n)$. To guarantee the computation's correctness, we applied only semantic-preserving transformations or equivalent transformations at every step. The unfolding transformation, a widely used program transformation in conventional logic programming, is a kind of ET. We can also devise other kinds of ET, especially to improve computation efficiency. ET thus provides a more flexible, efficient computational framework.

The second solution is to provide various flexible presentation forms. To employ available XML applications and products—XML editors, parsers, and rendering tools—we

```
C6: <rdf:Description about=$S:resourceR>                                  % If a class A is a subclass of
      <rdf:type resource=$S:classB/>                                      % another class B, every
      $E:R_properties                                                     % resource that is an instance of
    </rdf:Description>      ←—    <rdf:Description ID=$S:classA>        % class A will also be an instance
                                      <rdf:type resource="rdfs:Class"/>   % of B.
                                      <rdfs:subClassOf rdf:resource=$S:classB/>
                                      $E:A_properties
                                  </rdf:Description>,
                                  <rdf:Description about=$S:resourceR>
                                      <rdf:type resource=$S:classA/>
                                      $E:R_properties
                                  </rdf:Description>.

C7: <rdf:Description ID=$S:classA>                                        % SubClassOf transitivity property:
      <rdf:type resource="rdfs:Class"/>                                   % If A is a subclass of B and B is a
      <rdfs:subClassOf rdf:resource=$S:classC/>                           % subclass of some broader class C,
      $E:A_properties                                                     % this implies that A is also
    </rdf:Description>      ←—    <rdf:Description ID=$S:classA>        % a subclass of C.
                                      <rdf:type resource="rdfs:Class"/>
                                      <rdfs:subClassOf rdf:resource=$S:classB/>
                                      $E:A_properties
                                  </rdf:Description>,
                                  <rdf:Description ID=$S:classB>
                                      <rdf:type resource="rdfs:Class"/>
                                      <rdfs:subClassOf rdf:resource=$S:classC/>
                                      $E:B_properties
                                  </rdf:Description>.

C8: <rdf:Description about=$S:resourceR>                                  % If a property P1 is a subproperty
      <$S:propertyP2 resource=$S:valueV/>                                 % of another more general property
      $E:R_properties                                                     % P2, and if a resource R has a P1
    </rdf:Description>      ←—    <rdf:Description ID=$S:propertyP1>     % property with a value V, it is
                                      <rdf:type resource="rdf:Property"/> % implied that such a resource R has
                                      <rdfs:subPropertyOf rdf:resource=$S:propertyP2/> % also a property P2 with the value V.
                                      $E:P1_properties
                                  </rdf:Description>,
                                  <rdf:Description about=$S:resourceR>
                                      <$S:propertyP1 resource=$S:valueV/>
                                      $E:R_properties
                                  </rdf:Description>.

C9: <rdf:Description ID=$S:propertyP1>                                    % SubPropertyOf transitivity property:
      <rdf:type resource="rdf:Property"/>                                 % If P1 is a subproperty of P2 and
      <rdfs:subPropertyOf rdf:resource=$S:propertyP3/>                    % P2 is a subproperty of P3,
      $E:P1_properties                                                    % P1 is also a subproperty of P3.
    </rdf:Description>      ←—    <rdf:Description ID=$S:propertyP1>
                                      <rdf:type resource="rdf:Property"/>
                                      <rdfs:subPropertyOf rdf:resource=$S:propertyP2/>
                                      $E:P1_properties
                                  </rdf:Description>,
                                  <rdf:Description ID=$S:propertyP2>
                                      <rdf:type resource="rdf:Property"/>
                                      <rdfs:subPropertyOf rdf:resource=$S:propertyP3/>
                                      $E:P2_properties
                                  </rdf:Description>.
```

Figure 8. The XDD description $P_3 = \{C_6, \ldots, C_9\}$ models `subClassOf` and `subPropertyOf`.

should translate XDD descriptions into conventional XML documents. For example, by introducing XML syntax for the encoding of XML non-unit clauses, the clause C_1 in Figure 6b becomes the XML example shown in Figure 10. So, we can represent any given XDD description equivalently as a corresponding XML document, which can then be transmitted, exchanged, and handled sim-

```
<rdf:Description about="http://smith.com">
  <rdf:type resource="http://schema.org/appl#Web-Page"/>
  <Title>Smith Company's Page</Title>
  <Author resource="http://smith.com/john"/>
 </rdf:Description>

<rdf:Description about="http://xdd.org">
  <rdf:type resource="http://schema.org/appl#E-Article"/>
  <Title>XDD Language</Title>
  <Author resource="http://smith.com/john"/>
  <Illustrator resource="http://smith.com/joe"/>
 </rdf:Description>

<rdf:Description about="http://smith.com/john">
  <rdf:type resource="http://schema.org/appl#Person"/>
  <Name>John Smith</Name>
</rdf:Description>

<rdf:Description about="http://smith.com/joe">
  <rdf:type resource="http://schema.org/appl#Person"/>
  <Name>Joe Smith</Name>
</rdf:Description>
```

Figure 9. XDD description P_5 consists of four RDF statements that describe certain Web resources.

ilarly to ordinary XML documents. Moreover, we can use XSL (eXtensible Stylesheet Language) or XSLT (XSL Transformations) to flexibly display an XDD description in various forms. We can also define an RDF Schema for encoding XDD descriptions.

Instead of developing a single language that satisfies the three important but conflicting aspects of a language for the Semantic Web, we propose a new approach that separately manipulates each aspect by employing XDD, ET, and XSLT (see Figure 11). (See the "Related Works" sidebar.)

On the basis of the XDD and ET paradigms, we have implemented the *XDD System*—a Web-based XML processor available at http://kr.cs.ait.ac.th/xdd. Preliminary tests on several XML and RDF applications, including software configuration management, human resource management, and agent-based systems, reveal its feasibility and potential in real applications. We are conducting a more thorough evaluation with a large collection of XML documents. ■

Acknowledgments

The Thailand Research Fund partially funded this work. We'd like to thank the anonymous reviewers for their insightful comments that helped improve this article.

References

1. T. Berners-Lee, *Weaving the Web*, Harper, San Francisco, 1999.
2. S. Staab et al., "An Extensible Approach for Modeling Ontologies in RDF(S)," *Proc. 1st Workshop Semantic Web at the 4th European Conf. Digital Library* (ECDL 2000), Lisbon, Portugal, Sept. 2000, http://www.ics.forth.gr/proj/isst/SemWeb/proceedings/session2-1/paper.pdf (current 2 July 2001).
3. F.V. Harmelen, and I. Harrocks, "FAQs on OIL: The Ontology Inference Layer," *IEEE Intelligent Systems*, vol. 15, no. 6, Nov./Dec. 2000, pp. 69–72.
4. J. Hendler and D.L. McGuinness, "The DARPA Agent Markup Language," *IEEE Intelligent Systems*, vol. 15, no. 6, Jan./Feb. 2000, pp. 72–73.
5. I. Horrocks, U. Sattler, and S. Tobies, "Practical Reasoning for Expressive Description Logics," *Proc. 6th Int'l Conf. Logic for Programming and Automated Reasoning* (LPAR 99), *Lecture Notes in Computer Science*, no. 1705, Springer Verlag, Heidelberg, 1999, pp. 161–180.
6. M.R. Genesereth, "Knowledge Interchange Format," http://logic.stanford.edu/kif (current 19 June 2001).
7. C. Anutariya et al., "Towards a Foundation for XML Document Databases," *Proc. 1st Int'l Conf. Electronic Commerce and Web Technologies* (EC-Web 2000), *Lecture Notes in Computer Science*, no. 1875, Springer-Verlag, Heidelberg, 2000, pp. 324–333.


```
<xdd:Clause>
  <xdd:Head>
    <xdd:DescriptionCheck>
      <rdf:Description about=$S:uri>$E:propertyElt </rdf:Description>
    </xdd:DescriptionCheck>
  </xdd:Head>
  <xdd:Body>
    <xdd:PropertyCheck>$E:propertyElt</xdd:PropertyCheck>
  </xdd:Body>
</xdd:Clause>.
```

Figure 10. The XML representation of the non-unit clause C_1 in Figure 6b.

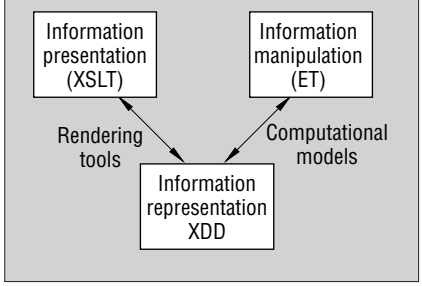

Figure 11. The three aspects of a language for the Semantic Web and their relationship.

Related Work

XML Declarative Description is not a logic-programming language, although its clauses and descriptions are similar to Datalog clauses and Datalog programs, respectively. Unlike Datalog,[1] XDD has been formally defined without such complicated concepts as *interpretation* and *model* (see the "Specialization Systems" and "The Semantics of Declarative Description" sidebars). Moreover, it has a higher-order syntax because it allows complex, nesting structured objects. For example, XML (eXtensible Markup Language) expressions can be directly represented and manipulated without decomposition or translation into sets of flat structured objects.

The various XML-based rule markup languages in the RuleML Initiative[2] are a class of representation schemes that we can use for the Semantic Web. Examples include BRML[3] (Business Rules Markup Language) and RFML[4] (Relational-Functional Markup Language), which merely encode CLP (Courteous Logic Programs) and Relfun-style declarative programming and knowledge representation in XML syntax, respectively. The defined language's semantics relies on translation into corresponding sets of rules in its original framework. Although such languages can be employed for modeling the Semantic Web, providing syntactic and semantic interchangeability and interoperability among Web applications appears to be unnatural and difficult.

OIL (Ontology Inference Layer) is an ontology-based language that extends RDF (Resource Description Framework) Schema with more expressive modeling primitives, including ontology metadata, class, and slot (binary relation) definitions. The DAML+OIL (DARPA Agent Markup Language + OIL) ontology markup language has been defined on the basis of RDF, RDF Schema, and OIL. OIL and DAML+OIL share important characteristics and are equivalent in their expressive powers. However, their current versions do not support a mechanism for a description of arbitrary rules and axioms.[5] Such a mechanism is an essential feature in many application domains, because it enables definition of additional relationships among classes and relations other than the generalization–specialization relationship.

Such a limitation would seem to demand major extensions and refinement of the languages. Alternatively, we can employ XDD to serve as their foundation, which not only helps enhance their expressiveness but also lets their intended meanings be determined directly. Their modeling primitives—for example, `subClassOf`, `subPropertyOf`, `inverseOf`, and `TransitiveProperty`—can be modeled by appropriate XML non-unit clauses. Their schemas and instances, which are encoded in RDF or XML serialization, immediately become XML unit clauses. In addition, definitions of arbitrary rules and axioms become possible. For example, the assertion "If a person *P* has authored e-document *D* in language *L*, that person must be able to speak that language," which is inexpressible in OIL or DAML+OIL, can be represented in XDD as a clause (see Figure A).

With the support of XDD—a well-established, generic tool for the Semantic Web—those languages can have their semantics formally defined together with full-fledged ontological modeling and reasoning services.

References

1. M. Liu, "Deductive Database Languages: Problems and Solutions," *ACM Computing Surveys*, vol. 31, no. 1, Mar. 1999, pp. 27–62.
2. H. Boley, "Rule Markup Language (RuleML)," 2001, www.dfki.uni-kl.de/ruleml (current 19 June 2001).
3. B.N. Grosof, Y. Labrou, and H.Y. Chan, "A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML," *Proc. 1st ACM Conf. Electronic Commerce* (EC 99), ACM Press, New York, 1999, pp. 68–77.
4. H. Boley, "RFML: Relational-Functional Markup Language," 2000, www.relfun.org/rfml (current 19 June 2001).
5. S. Bechhofer et al., "An Informal Description of Standard OIL and Instance OIL," Nov. 2000, www.ontoknowledge.org/oil/downl/oil-whitepaper.pdf (current 19 June 2001).


```
<rdf:Description about=$S:personP>
   <rdf:type resource="#Person"/>
   <CanSpeak>$S:languageL</CanSpeak>
 </rdf:Description>                    <rdf:Description about=$S:documentD>
                                          <rdf:type resource="#E-Document"/>
                                          <Author resource=$S:personP/>
                                          <Language>$S:languageL</Language>
                                          $E:D_properties
                                       </rdf:Description>.
```

```
% If a person P has authored an
% e-document D in a language L,
% we can infer that such a person
% P must be able to speak that
% language.
```

Figure A. An example of modeling an axiom.

8. K. Akama, "Declarative Semantics of Logic Programs on Parameterized Representation Systems," *Advances in Software Science and Technology,* Iwanami Shoten, Publishers and Academic Press, Tokyo, vol. 5, 1993, pp. 45–63.
9. S. Kitcharoensakkul and V. Wuwongse, "Unified Versioning Using Resource Description Framework," to be published in *Annals of Software Eng.*, Kluwer Academic Publishers, Dordrecht, The Netherlands, vol. 11, 2001.
10. M. Uschold and M. Grüninger, "Ontologies: Principles, Methods and Applications," *Knowledge Eng. Rev.*, vol. 11, no. 2, June 1996, pp. 93–136.
11. C. Anutariya et al., "Towards Computation

with RDF Elements," *Proc. Int'l Symp. Digital Library 1999* (ISDL 99), Univ. of Library and Information Science, Tsukuba, Japan, 1999, pp. 112–119.

12. T. Berners-Lee, "The Semantic Web as a Language of Logic," Sept. 2000, www.w3.org/DesignIssues/Logic.html (current 19 June 2001).

13. K. Akama, "ET Computational Paradigm," 2001, http://kr.cs.ait.ac.th/et (current 19 June 2001).

The Authors

Vilas Wuwongse is a professor in the Computer Science and Information Management Program at the Asian Institute of Technology, Thailand. He is also an adviser to the Revenue Department of the Thai Ministry of Finance and the Bank of Thailand, where he leads teams to develop large-scale Web-based software systems and data warehouses. His research interests include knowledge representation, data modeling, and the Semantic Web. He serves as editor of the *Journal of Natural Language Processing* and the *Computer Processing of Oriental Languages*. He has a BEng, MEng, and DEng from the Tokyo Institute of Technology. He is a member of the IEEE Computer Society, ACM, Information Processing Society of Japan, and Japanese Society for Artificial Intelligence. Contact him at AIT/CSIM, PO Box 4, Klong Luang, Pathumthani 12120, Thailand; vw@cs.ait.ac.th.

Chutiporn Anutariya is a doctoral student in the Computer Science and Information Management Program at the Asian Institute of Technology. Her research interests include the Semantic Web, knowledge representation, and software engineering. She is a member of the Knowledge Representation Laboratory. She has a BS in statistics from Chulalongkorn University, Thailand, and an MS in computer science from the Asian Institute of Technology, Thailand. Contact her at AIT/CSIM, PO Box 4, Klong Luang, Pathumthani 12120, Thailand; ca@cs.ait.ac.th.

Kiyoshi Akama is a professor at the Center for Information and Multimedia Studies, Hokkaido University. His research interests include artificial intelligence, knowledge processing, and programming languages. He has a BEng, MEng, and DEng from the Tokyo Institute of Technology. He is a member of the Japanese Society for Artificial Intelligence, Japan Society for Software Science and Technology, and the Information Processing Society of Japan. Contact him at the Center for Information and Multimedia Studies, Hokkaido Univ., Kita 11, Nishi 5, Kita-ku, Sapporo, 060-0811, Japan; akama@cims.hokudai.ac.jp.

Ekawit Nantajeewarawat is an assistant professor of information technology at Sirindhorn International Institute of Technology, Thailand. His research interests include deductive object-oriented systems, knowledge representation, and object-oriented software engineering. He has a BEng in computer engineering from Chulalongkorn University, Thailand, and an MEng and DEng in computer science from the Asian Institute of Technology, Thailand. Contact him at Information Technology Program, Sirindhorn Int'l Inst. of Technology, PO Box 22,Thammasat Rangsit Post Office, Pathumthani 12121,Thailand; ekawit@siit.tu.ac.th.