Semantic Web Modeling and Programming with XDD

Chutiporn Anutariya[1], Vilas Wuwongse[1], Kiyoshi Akama[2] and Vichit Wattanapailin[1]
[1] *Computer Science & Information Management Program,*
School of Advanced Technologies, Asian Institute of Technology
Pathumtani 12120, Thailand

[2] *Center for Information and Multimedia Studies,*
Hokkaido University, Sapporo 060, Japan

Abstract. *XML Declarative Description* (*XDD*) is a unified modeling language with well-defined declarative semantics. It employs XML as its bare syntax and enhances XML expressive power by provision of mechanisms for succinct and uniform expression of Semantic Web contents, rules, conditional relationships, integrity constraints and ontological axioms. Semantic Web applications, offering certain Web services and comprising the three basic modeling components: application data, application rules and logic, and users' queries and service requests, are represented in XDD language as *XDD descriptions*. By integration of XDD language, *Equivalent Transformation* computational paradigm and XML syntax, *XML Equivalent Transformation* (*XET*)—a declarative programming language for computation of XDD descriptions in *Equivalent Transformation* computational paradigm—is developed. By means of XDD and XET languages, a new declarative approach to the development and the execution of Semantic Web applications is constructed.

Keywords. Semantic Web, Semantic Web applications, Semantic Web services, XML Declarative Description, XML Equivalent Transformation.

1 Introduction

The *Semantic Web* [7] is a vision of the next-generation Web which enables Web applications to automatically collect Web contents from diverse sources, integrate and process information, and interoperate with other applications in order to execute sophisticated tasks for humans. For the current Web to evolve from a global repository of information primarily designed for human consumption into the Semantic Web, tremendous effort has been devoted to definition and development of various supporting standards and technologies. Prominent markup languages with an aim to define a syntax convention for descriptions of the semantics of Web contents in a standardized interoperable manner include *XML*, *RDF*, *RDF Schema*, *OIL* [6,8,12] and *DAML+OIL* [13]. Moreover, for Web applications to effectively communicate and interoperate in the heterogeneous environment, a standard *Agent Communication Language* (*ACL*) [15] becomes a necessity. Two major current ACLs are *Knowledge Query and Manipulation*

www.manaraa.com

Language (*KQML*) [9] and *Foundation for Intelligent Physical Agents ACL* (*FIPA-ACL*) [10,15].

With an emphasis on the modeling and the development of Semantic Web applications offering certain Web services, there arises a need for a tool which is capable of modeling their three major components: *application data*, *application rules and logic*, and *queries and requests*. *XML Declarative Description* (*XDD*) [5,17]—a unified, XML-based Semantic Web modeling language with well-defined semantics and a support for general inference mechanisms—aims to fulfill such a requirement. XDD does not only allow direct representation and manipulation of machine-comprehensible Web contents (such as documents, data, metadata and ontologies, encoded in XML, RDF, OIL or DAML+OIL syntax), but also provides simple, yet expressive means for modeling their conditional relationships, integrity constraints and ontological axioms as well as Semantic Web applications. XDD serves the three important roles: *content language*, *application-rule language* and *query or service-request language*, in modeling such three main components of Semantic Web applications.

Based on XDD language, a declarative programming language, i.e., *XML Equivalent Transformation* (*XET*) is constructed. Given an application's model specification, represented in terms of an XDD description, an XET program capable of executing and handling the application's queries as well as service requests can be obtained directly.

Thus, the developed technologies—XDD and XET languages—present a new paradigm for modeling and programming Semantic Web applications. By integration with existing Web and agent technologies, XDD and XET also allow both syntactic and semantic interoperability among Web applications, and hence enable the development of intelligent services as well as automated software agents.

Section 2 formalizes an extended XDD language with set-of-reference functions, Section 3 presents an XDD approach to modeling Semantic Web resources and applications, Section 4 describes XET programming language and outlines an approach to its employment in Web application development, Section 5 demonstrates a prototype system which adopts the developed technologies, Section 6 reviews current related works, and Section 7 draws conclusions.

2 XML Declarative Description

XDD [5,17] is a language the *words* and *sentences* of which are *XML expressions* and *XML clauses*, respectively. XML expressions are used to express explicit and implicit as well as simple and complex facts, while XML clauses are employed to represent ontology, implicit and conditional relationships, constraints and axioms. First, the data structure of XML expressions and their sets, characterized by an *XML Specialization System*, will be given and then followed by the syntax and semantics of XML clauses.

2.1 XML Specialization System

XML expressions have a similar form to XML elements except that they can carry variables for representation of implicit information and for enhancement of their expressive power. Every component of an XML expression—the expression itself, its tag name, attribute names and values, pairs of attributes and values, contents, sub-expressions as well

www.manaraa.com

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	`$I`	Parts of XML expressions
Z-variables: Set-variables	`$Z`	Sets of XML expressions

as some partial structures—can contain variables. XML expressions without variables are called *ground XML expressions* or simply *XML elements*, those with variables *non-ground XML expressions*. Table 1 defines all types of variables and their usages.

An *XML expression* takes formally one of the following forms:

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

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

The domain of XML expressions and their sets can be defined as follows:

- $\mathcal{A}_X$: the set of all XML expressions,
- $\mathcal{G}_X$: the subset of $\mathcal{A}_X$ which comprises all ground XML expressions in $\mathcal{A}_X$,
- $\mathcal{A} = \mathcal{A}_X \cup 2^{(\mathcal{A}_X \cup V_Z)}$: the set of all XML expressions in $\mathcal{A}_X$ and sets of XML expressions and *Z*-variables in $2^{(\mathcal{A}_X \cup V_Z)}$, and
- $\mathcal{G} = \mathcal{G}_X \cup 2^{\mathcal{G}}$: the set of all ground XML expressions in $\mathcal{G}_X$, and sets of ground XML expressions in $2^{\mathcal{G}_X}$.

Note that elements of the sets $\mathcal{A}$ and $\mathcal{G}$ may be at times referred to as *objects* and *ground objects*, respectively, and when it is clear from the context, a singleton $\{X\}$ where $X \in V_Z$ is a *Z*-variable, will be written simply as X.

Instantiation of those various types of variables 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, (`$N:tag1`, `$N:tag2`), (`$N:tag2`, `Name`) and (`$E:e`, (`$E:e1`, `$E:e2`)) are basic specializations which rename the *N*-variable `$N:tag1` to `$N:tag2`, instantiate the *N*-variable `$N:tag2` into the tagname `Name`, and expand the *E*-variable `$E:e` into the sequence of the *E*-variables `$E:e1` and `$E:e2`, respectively. There are four types of basic specializations:

www.manaraa.com

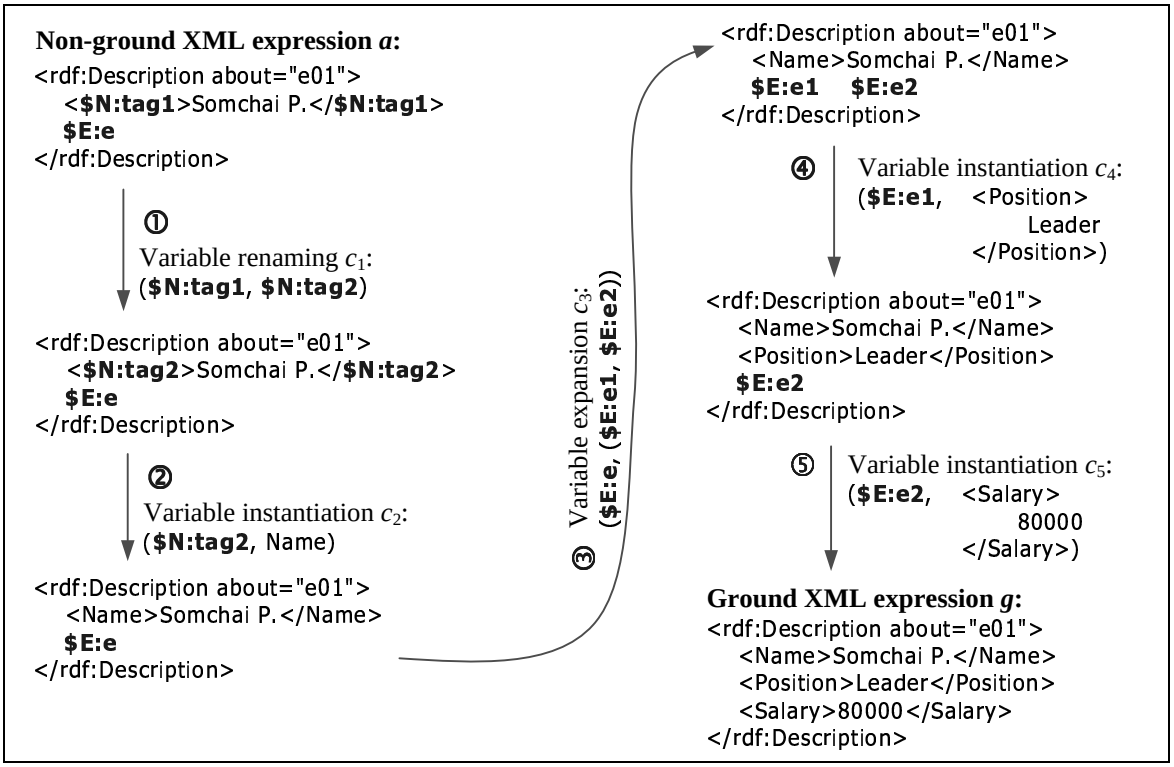


Figure 1: Successive applications of basic specializations c_1, …, c_5 to a non-ground XML expression a in $\mathcal{A}_X$ by the operator μ, yielding a ground XML expression g in $\mathcal{G}_X$.

1. Rename variables.
2. Expand a *P*- or an *E*-variable into a sequence of variables of their respective types.
3. Remove *P*-, *E*- or *I*-variables.
4. Instantiate variables to XML expressions or components of XML expressions which correspond to the types of the variables, i.e., instantiate:
   - *N*-variables to element types or attribute names,
   - *S*-variables to strings,
   - *E*-variables to XML expressions in $\mathcal{A}_X$,
   - *I*-variables to XML expressions which contains their sub-elements at an arbitrary depth, or
   - *Z*-variables to sets of XML expressions and *Z*-variables.

The *data structure* of XML expressions and sets of XML expressions are characterized by a mathematical abstraction, called **XML Specialization System**, which will be defined in terms of *XML specialization generation system* $\Delta = \langle \mathcal{A}, \mathcal{G}, \mathcal{C}, \nu \rangle$, where

- $\mathcal{C}$ is the set of all *basic specializations*, and
- ν is a mapping from $\mathcal{C}$ to *partial_map*($\mathcal{A}$) (i.e., the set of all partial mappings on $\mathcal{A}$), called the *basic specialization operator*; it determines, for each basic specialization c in $\mathcal{C}$, the change of objects in $\mathcal{A}$ caused by c.

www.manaraa.com

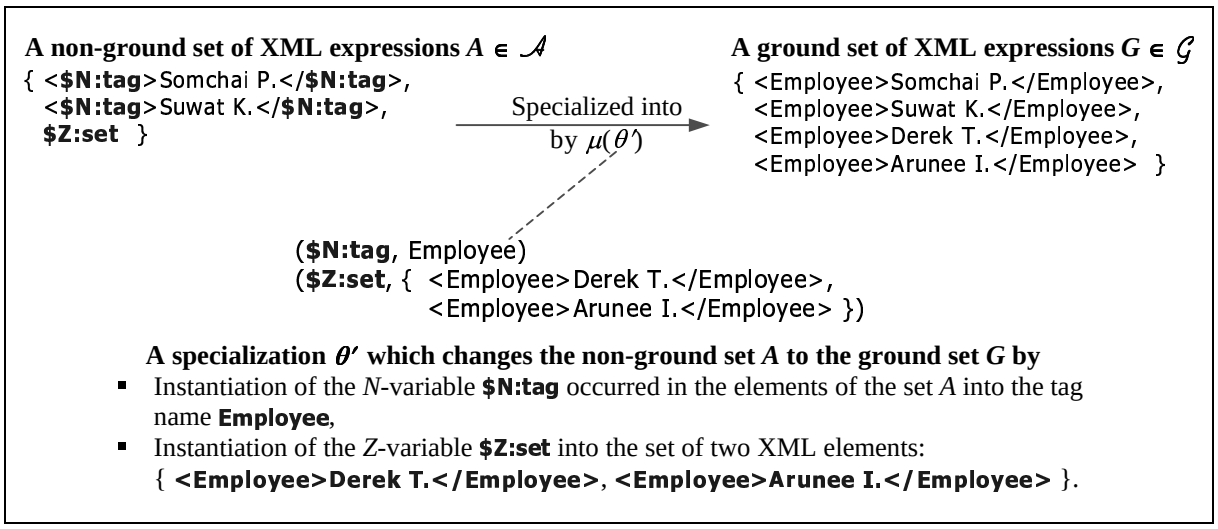


Figure 2: Specialization of a non-ground set of XML expressions in $\mathcal{A}$ into a ground set of XML expressions in $\mathcal{G}$ by the operator μ using a specialization θ' in $\mathcal{S}$.

Figure 1 illustrates examples of a non-ground XML expression a in $\mathcal{A}$, basic specializations $c_1, \ldots, c_5$ in $\mathcal{C}$ and their successive applications to a by the operator ν in order to obtain a ground XML expression g in $\mathcal{G}$.

Denote a sequence of zero or more basic specializations in $\mathcal{C}$ by a *specialization.*

Based on the XML specialization generation system $\Delta = \langle \mathcal{A}, \mathcal{G}, \mathcal{C}, \nu \rangle$, the **XML Specialization System** is $\Gamma = \langle \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu \rangle$, where

- $\mathcal{S} = \mathcal{C}^*$ is the set of all *specializations*, and
- $\mu : \mathcal{S} \rightarrow partial\_map(\mathcal{A})$ is the *specialization operator* which determines, for each specialization s in $\mathcal{S}$, the change of each object a in $\mathcal{A}$ caused by s such that:
  - $\mu(\lambda)(a) = a$, where λ denotes the null sequence,
  - $\mu(c \cdot s)(a) = \mu(s)(\nu(c)(a))$, where $c \in \mathcal{C}$ and $s \in \mathcal{S}$.

Intuitively, the operator μ is defined in terms of the operator ν such that for each $a \in \mathcal{A}$ and $s = (c_1 \ldots c_n) \in \mathcal{S}$, $\mu(s)(a)$ is obtained by successive applications of $\nu(c_1), \ldots, \nu(c_n)$ to a. Note that, when μ is clear from the context, for $\theta \in \mathcal{S}$, $\mu(\theta)(a)$ will be written simply as $a\theta$.

With reference to Figure 1, let a specialization θ in $\mathcal{S}$ denote the sequence of the basic specializations c_1, c_2, c_3, c_4 and c_5; by the definition of μ, $g = \mu(\theta)(a) = a\theta$. Similarly, Figure 2 shows examples of a non-ground set of XML expressions A in $\mathcal{A}$, a specialization θ' in $\mathcal{S}$ and its application to A by the operator μ, in order to obtain a ground set of XML expressions G in $\mathcal{G}$, i.e., $G = \mu(\theta')(A) = A\theta'$.

2.2 XDD: Syntax

The definitions of *XML declarative descriptions with references* and its related concepts are given next in terms of the XML specialization system Γ.

www.manaraa.com

Table 2: Definitions of the concepts *constraints*, *references* and *XML clauses* on Γ.

Concept	Being in Ground Form IFF	Application of a Specialization $\theta \in \mathcal{S}$ Yielding
A constraint: $q(a_1, \dots, a_n)$ where $n > 0$, $q \in K$ and $a_i \in \mathcal{A}$	$a_i \in \mathcal{G}$ for $1 \le i \le n$	$q(a_1, \dots, a_n)\theta = q(a_1\theta, \dots, a_n\theta)$
A reference: $r = \langle a, f, P \rangle$ where - $a \in \mathcal{A}$, - $f \in F$ and - P is an XML declarative description which will be called the *referred description* of r	$a \in \mathcal{G}$	$r\theta = \langle a, f, P \rangle\theta = \langle a\theta, f\theta, P \rangle$
An XML clause: $H \leftarrow B_1, B_2, ..., B_n$ where - $n \ge 0$, - H is an XML expression in $\mathcal{A}_X$, - B_i is an XML expression in $\mathcal{A}_X$, a *constraint* or a *reference* on Γ, and - the order of B_i is immaterial.	Comprising only ground objects, ground constraints and ground references	$H\theta \leftarrow B_1\theta, B_2\theta, ..., B_n\theta$

Let K be a set of *constraint predicates* and F the set of all mappings: $2^{\mathcal{G}} \rightarrow 2^{\mathcal{G}}$, the elements of which are called *reference functions*. An *XML declarative description* on Γ, simply called an *XDD description*, is a (possibly infinite) set of *XML clauses* on Γ. Table 2 defines concepts of *constraints*, *references* and *XML clauses* on Γ.

The notion of constraints introduced here is useful for defining restrictions on objects in $\mathcal{A}$, i.e., both on XML expressions in $\mathcal{A}_X$ and on sets of XML expressions in $2^{(\mathcal{A}_X \cup V_Z)}$. Given a *ground constraint* $q(g_1, \dots, g_n)$, $g_i \in \mathcal{G}$, its truth or falsity is assumed to be predetermined. Denote the set of all true ground constraints by *Tcon*. For instance:

- Define GT(a_1, a_2) as a constraint which will be true iff a_1 and a_2 are XML elements of the forms <Num>v_1</Num> and <Num>v_2</Num>, respectively, where v_1, v_2 are numbers and $v_1 > v_2$. Obviously, a constraint GT(<Num>10</Num>, <Num>5</Num>) is a true ground constraint in *Tcon*.
- Define a constraint Count(G, g) which will be true, iff G is a set of XML elements and g the XML element <Result>v</Result>, where v denotes G's cardinality.

The concept of references defined here together with an appropriate definition of a *set-of-reference function* in F will be employed to describe complex queries/operations on sets of XML expressions such as set construction, set manipulation and aggregate functions, e.g., min, max and count in SQL. Given $a, x \in \mathcal{A}_X$, let $f_{x,a} \in F$ denote a *set-of-reference function* and be defined as follows: For each $G \subset \mathcal{G}_X$,

$$f_{x,a}(G) = \{\{x\theta \in \mathcal{G}_X \mid \theta \in \mathcal{S},\ a\theta \in G\}\}. \quad (1)$$

In other words, for each subset G of $\mathcal{G}_X$, $f_{x,a}(G)$ is a singleton set, the element of which is a set of ground XML expressions of the form $x\theta$, for any specialization $\theta \in \mathcal{S}$, which makes $a\theta$ become a ground XML expression in G. Intuitively, a and x are used to define the condition for constructing a set and to determine the elements comprising that set,

www.manaraa.com

respectively, i.e., $x\theta \in f_{x,a}(G)$ iff $a\theta \in G$. The objects a and x will be referred to as *filter* and *constructor objects*, respectively. Given a specialization θ in $\mathcal{S}$, application of θ to $f_{x,a}$ yields $f_{x,a}\theta = f_{x\theta, a\theta}$. For example, assuming that G is the set

```
{  <Employee division="IT">Somchai</Employee>,
   <Employee division="HR">Malee</Employee>,
   <Employee division="IT">Suwat</Employee>    },
```

then

$f_{\text{<ITstaff name=\$S:n/>, <Employee division="IT">\$S:n</Employee>}}(G)$

= { {<ITstaff name="Somchai"/>, <ITstaff name="Suwat"/> } }.

In other words, such a set-of function filters the XML elements of the set G with the pattern

<Employee division="IT">$S:n</Employee> — the filter object

and then constructs the resulting set of XML elements using the pattern

<ITstaff name=$S:n/> — the constructor object

Note that the effect of the binding of the variable $S:n in the filter object will also cascade to the constructor object.

Based on the definition of the set-of function, a reference $r = \langle S, f_{x,a}, P\rangle$, for $x, a \in \mathcal{A}_X$ and $S \in 2^{(\mathcal{A}_X \cup V)}$, is called a *set-of reference*.

Given an XML clause $C = (H \leftarrow B_1, B_2, ..., B_n)$, H is called the *head* and $(B_1, B_2, ..., B_n)$ the *body* of C, denoted by *head*(C) and *body*(C), respectively. The sets of all XML expressions, constraints and references in the body of C are denoted by *object*(C), *con*(C) and *ref*(C), respectively. Thus, *body*(C) = *object*(C) ∪ *con*(C) ∪ *ref*(C). If $n = 0$, such a clause is called a *unit clause*, if $n > 0$, a *non-unit clause*. When it is clear from the context, a unit clause ($H \leftarrow$) is written simply as H, i.e., the left-arrow symbol is omitted. Therefore, every XML element can be considered as a ground XML unit clause, and moreover every XML document can be modeled as an XDD description comprising solely ground XML unit clauses.

The *heights* of an XML clause C and of an XDD description P, denoted by *hgt*(C) and *hgt*(P), are defined as follows:

- If *ref*(C) = ∅ (C contains no reference), then *hgt*(C) = 0;
  Otherwise *hgt*(C) is the maximum height of all the referred descriptions contained in its body plus one.
- *hgt*(P) is the maximum height of all the clauses in P.

2.3 XDD: Declarative Semantics

Given an XDD description P on Γ, its declarative semantics, denoted by $\mathcal{M}(P)$, is defined inductively as follows:

1. Given the meaning $\mathcal{M}(Q)$ of an XDD description Q with the height m, a reference $r = \langle g, f, Q\rangle$ is a true reference, iff $g \in f(\mathcal{M}(Q))$. For any $m \geq 0$, define *Tref*(m) as the set of all *true references*, the heights of the referred descriptions of which are smaller than or equal to m, i.e.:

www.manaraa.com

$$Tref(m) = \{\langle g, f, Q\rangle \mid \ g \in \mathcal{G},\ \ f \in \mathcal{F},\ \ hgt(Q) \le m,\ \ g \in f(\mathcal{M}(Q))\ \} \tag{2}$$

2. The meaning $\mathcal{M}(P)$ of the description P is a set of XML elements defined by:

$$\mathcal{M}(P) = \bigcup_{n=1}^{\infty} [T_P]^n(\varnothing) \tag{3}$$

where - $\varnothing$ is the empty set,
- $T_P{}^1(\varnothing) = T_P(\varnothing)$ and $[T_P]^n(\varnothing) = T_P([T_P]^{n-1}(\varnothing))$ for each $n > 1$, and
- the mapping T_P: $2^{\mathcal{G}} \to 2^{\mathcal{G}}$ is:

For each $G \subset \mathcal{G}$, $g \in T_P(G)$ iff there exist a clause $C \in P$ and a specialization $\theta \in \mathcal{S}$ such that $C\theta$ is a ground clause, with head g and all objects, constraints and references in its body belong to G, $Tcon$ and $Tref(n)$, for some $n < hgt(P)$, respectively, i.e.:

$$\begin{aligned} T_P(G) = \{ head(C\theta) \mid\ & C \in P,\ \ \theta \in \mathcal{S},\ \ C\theta \text{ is a ground clause,} \\ & object(C\theta) \subset G,\ \ con(C\theta) \subset Tcon, \\ & ref(C\theta) \subset Tref(n),\ \ n < hgt(P)\ \} \end{aligned} \tag{4}$$

Intuitively, the meaning of a description P, i.e., $\mathcal{M}(P)$, is a set of all XML elements, which are directly described by and derivable from the unit and the non-unit clauses in P, respectively, i.e.:

- Given a unit clause ($H \leftarrow$) in P, for $\theta \in \mathcal{S}$:

  $H\theta \in \mathcal{M}(P)$ if $H\theta$ is an XML element.

- Given a non-unit clause ($H \leftarrow B_1, ..., B_i, B_{i+1}, ..., B_j, B_{j+1}, \ldots, B_n$) in P, assuming without loss of generality that $B_1, ..., B_i$ are XML expressions, $B_{i+1}, ..., B_j$ are constraints, and $B_{j+1}, ..., B_n$ are references, for $\theta \in \mathcal{S}$:

  $H\theta \in \mathcal{M}(P)$ if - $H\theta$ is an XML element,
  - $B_1\theta, ..., B_i\theta \in \mathcal{M}(P)$,
  - $B_{i+1}\theta, ..., B_j\theta$ are true constraints, and
  - $B_{j+1}\theta, ..., B_n\theta$ are true references.

Based on the formalized concepts of XDD language, Figure 3 demonstrates two XDD descriptions denoted by Q and P, and then determines their semantics, which is sets of XML elements denoting certain objects and their relationships in a real-world domain.

3 Modeling Semantic Web Resources and Applications

XDD language allows collections of Semantic Web resources, such as documents, data, metadata and ontologies, encoded in XML, RDF, OIL or DAML+OIL syntax, to be represented in terms of XDD descriptions. In the descriptions, explicit information items are directly expressed as ground XML unit clauses, while rules, conditional relationships, integrity constraints and ontological axioms are formalized as XML non-unit clauses. The descriptions' semantics, which can be directly determined under the language itself, is defined as sets of XML elements—surrogates of objects and their relationships in a real-world domain.

www.manaraa.com

{ C_1: <Employee division="IT">Somchai</Employee> ←
C_2: <Employee division="HR">Malinee</Employee> ←
C_3: <Employee division="IT">Suwat</Employee> ← }

(a) XML Declarative Description *Q*.

- $hgt(C_1) = hgt(C_2) = hgt(C_3) = hgt(Q) = 0$
- $\mathcal{M}(Q)$ = { <Employee division="IT">Somchai</Employee>
  <Employee division="HR">Malee</Employee>
  <Employee division="IT">Suwat</Employee> }

(b) The height and the meaning of *Q*.

{ C_4: <ITStaffNumber>$S:num</ITStaffNumber>
← ⟨$Z:set, $f_{\text{<ITstaff name=\$S:n/>, <Employee division="IT">\$S:n</Employee>}}$, Q⟩,
Count($Z:set, <Result>$S:num</Result>). }

(c) XML Declarative Description *P*.

- $hgt(C_4) = hgt(Q) + 1 = 1$, and $hgt(P) = 1$.
- $f_{\text{<ITstaff name=\$S:n/>, <Employee division="IT">\$S:n</Employee>}}(\mathcal{M}(Q)) =$
  { { <ITstaff name="Somchai"/>, <ITstaff name="Suwat"/> } }
- ⟨$Z:set, $f_{\text{<ITstaff name=\$S:n/>, <Employee division="IT">\$S:n</Employee>}}$, Q⟩
  is a true reference, iff $Z:set is specialized into the set
  { { <ITstaff name="Somchai"/>, <ITstaff name="Suwat"/> } }
- The constraint Count is a true constraint, iff $S:num is specialized into the number 2.
- $\mathcal{M}(P)$ = {<ITStaffNumber>2</ITStaffNumber>}.

(d) The height and the meaning of *P*.

Figure 3:XDD descriptions *Q* and *P* and their declarative semantics.

Besides its employment to model various kinds of Semantic Web resources, XDD can also be applied to describe certain operations such as queries, document transformation and processing rules as well as program specifications. Figure 4 illustrates examples of Semantic Web resource modeling and a query formulation and shows that although information about senior employees and their bonuses is not explicit, it can be deductively inferred based on the clause R_3; this representation offers a more compact form and provides an easier way of information modification than explicit enumeration of such information. With this simple, yet expressive modeling mechanism, XDD can readily be applied to model Semantic Web applications.

A Semantic Web application, offering certain Semantic Web services, comprises three main components: *application data*, *application rules or logic*, and *users' queries or requests for services*. For instance: In a Semantic Web search engine, offering an information-gathering service,

- its application data: a catalog or descriptions of Semantic Web contents,
- its application rules: domain-model ontologies and axioms, and
- its requests: user queries describing their informational needs.

www.manaraa.com

In a business-2-business (B2B) commerce application, its three components are

- a catalog of available products and services,
- business rules and policies such as price discounting and refund rules, and
- queries and business transactions such as a request for a quotation and an order placement.

XDD language provides means for modeling Semantic Web applications in that it enables direct representation of:

- application data (facts), encoded in XML, RDF, OIL or DAML+OIL syntax, in terms of XML ground unit clauses,
- application rules or logic in terms of XML non-unit clauses—the heads and bodies of the clauses describe the consequences and antecedents of the rules, respectively—and
- users' queries or service requests in terms of XML non-unit clauses—the heads of the clauses describe the structure of the query results or the service responses and the bodies specify the queries' selection conditions or the service requests and their constraints.

Thus, XDD language has the three vital roles:

- *content language*,
- *application-rule language*, and
- *query* or *service-request language*.

See Figure 4 for an example of each role. Basically, each query/request will be executed on a specified collection of application data and rules and will return as its answer a set of XML elements, derivable from such a collection and satisfying all of its conditions. More precisely, given a set of application data and rules, modeled as an XDD description P, and a query/request, formulated as an XML clause Q: $(H \leftarrow B_1, B_2, ..., B_n)$, the response to Q is the set

$$\{H\theta \mid H\theta \in \mathcal{M}(P \cup \{Q\}), \theta \in \mathcal{S}\}.$$

By employment of *Equivalent Transformation* (*ET*) computational paradigm [2,3], which is based on semantics-preserving transformations (*equivalent transformations*) of declarative descriptions, the *computation* of an answer/response to such a query/request Q is carried out by successive transformations of the XDD description $P \cup \{Q\}$ into a simpler but equivalent description, from which the answer can be obtained readily and directly. In brief, $P \cup \{Q\}$ will be successively transformed until it becomes the description

$$P \cup \{Q_1, \ldots, Q_n\},$$

where $n \geq 0$ and the Q_i are ground XML unit clauses.

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

XET, a declarative programming language for computation of XDD descriptions in ET paradigm, will be presented next.

www.manaraa.com

E_1:
```
<rdf:Description about="e01">
    <rdf:type resource="Employee"/>
    <e:Name>Somchai P.</e:Name>
    <e:Salary>80000</e:Salary>
</rdf:Description>
```

E_2:
```
<rdf:Description about="e02">
    <rdf:type resource="Employee"/>
    <e:Boss resource="e01"/>
    <e:Name>Sawat K.</e:Name>
    <e:Salary>50000</e:Salary>
</rdf:Description>
```

E_3:
```
<rdf:Description about="e03">
    <rdf:type resource="Employee"/>
    <e:Boss resource="e02"/>
    <e:Name>Derek T.</e:Name>
    <e:Salary>40000</e:Salary>
</rdf:Description>
```

E_4:
```
<rdf:Description about="e04">
    <rdf:type resource="Employee"/>
    <e:Boss resource="e02"/>
    <e:Name>Arunee I.</e:Name>
    <e:Salary>30000</e:Salary>
</rdf:Description>
```

(a) Modeling of application data – descriptions of employee objects, based on RDF infrastructure.

R_1:
```
<EmpRelation boss=$S:X subordinate=$S:Y level=1/>
    ← <rdf:Description about=$S:Y>
          <rdf:type resource="Employee"/>
          <e:Boss resource=$S:X/>
          $E:emp
      </rdf:Description>.
```
% If Y is described as a resource of the type Employee and its Boss property is another resource X, then one can derive that X is the first-leveled boss of Y.

R_2:
```
<EmpRelation boss=$S:X subordinate=$S:Z level=$S:n1/>
    ← <EmpRelation boss=$S:X subordinate=$S:Y level=$S:n/>,
      <rdf:Description about=$S:Z>
          <rdf:type resource="Employee"/>
          <e:Boss resource=$S:Y/>  $E:emp
      </rdf:Description>,
      Add( <Num>$S:n</Num>, <Addendum>1</Addendum>,
          <Result>$S:n1</Result>).
```
% If X is the nth-leveled boss of Y and Y is referred to as a direct boss of an Employee Z, then one can imply that X is the $(n+1)$th-leveled boss of Z.

R_3:
```
<rdf:Description about=$S:X>
    <rdf:type resource="SeniorEmployee"/>
    <e:Bonus>$S:bonus</e:Bonus>
    <e:Subordinate>
        <rdf:Bag>$Z:set</rdf:Bag>
    </e:Subordinate>
    <e:Salary>$S:sal</e:Salary>  $E:emp
</rdf:Description>
  ← <rdf:Description about=$S:X>
        <rdf:type resource="Employee"/>
        <e:Salary>$S:sal</e:Salary>  $E:emp
    </rdf:Description>,
    ⟨$Z:set, f<rdf:Li resource=$S:Y/>,<EmpRelation boss=$S:X subordinate=$S:Y level=$S:any/>,
     {E1,..., E4, R1, R2}⟩,
    Count($Z:set, <Result>$S:num</Result>),
    GT(<Num>$S:num</Num>, <Num>3</Num>),
    Mul(<Num>$S:sal</Num>, <Multiplier>2</Multiplier>,
        <Result>$S:bonus</Result>).
```
% For an employee X who has more than 3 subordinates (of any level), then X is considered to be a Senior-Employee and will receive a double-salary bonus. A list of all subordinates of X is also included in X's description.

(b) Modeling of application rules and logic – descriptions of relationships among employee objects.

Q:
```
<Answer>$S:name</Answer>
    ← <rdf:Description about=$S:X>
          <rdf:type resource="Employee"/>
          <e:Name>Somchai P.</e:Name>  $E:emp1
      </rdf:Description>,
      <EmpRelation boss=$S:X subordinate=$S:Y level=2/>,
      <rdf:Description about=$S:Y>
          <rdf:type resource="Employee"/>
          <e:Name>$S:name</e:Name>  $E:emp2
      </rdf:Description>.
```
% A query Q finds names of all Somchai P.'s second-leveled subordinates.

(c) Modeling of a query.

Figure 4: Modeling of an application.

www.manaraa.com

4 XET Programming Language

XET (*XML Equivalent Transformation*) [18] is a declarative programming language which can directly and succinctly manipulates XML data. By integration of XDD language, ET computational paradigm and XML syntax, XET possesses XDD's expressiveness and ET's computational power as well as XML's flexibility, extensibility and interchangeability. Therefore, XET naturally unifies "Documents', "Programs" and "Data", and with its computational and reasoning services, it also unifies "Document Processing (Transformation)", "Program Execution" and "Query processing". Available XML editing and validating tools can also be employed to edit and validate XET programs. The syntax of XET language, described by XML Schema, is available in [18]. XET provides useful sets of built-in operations including:

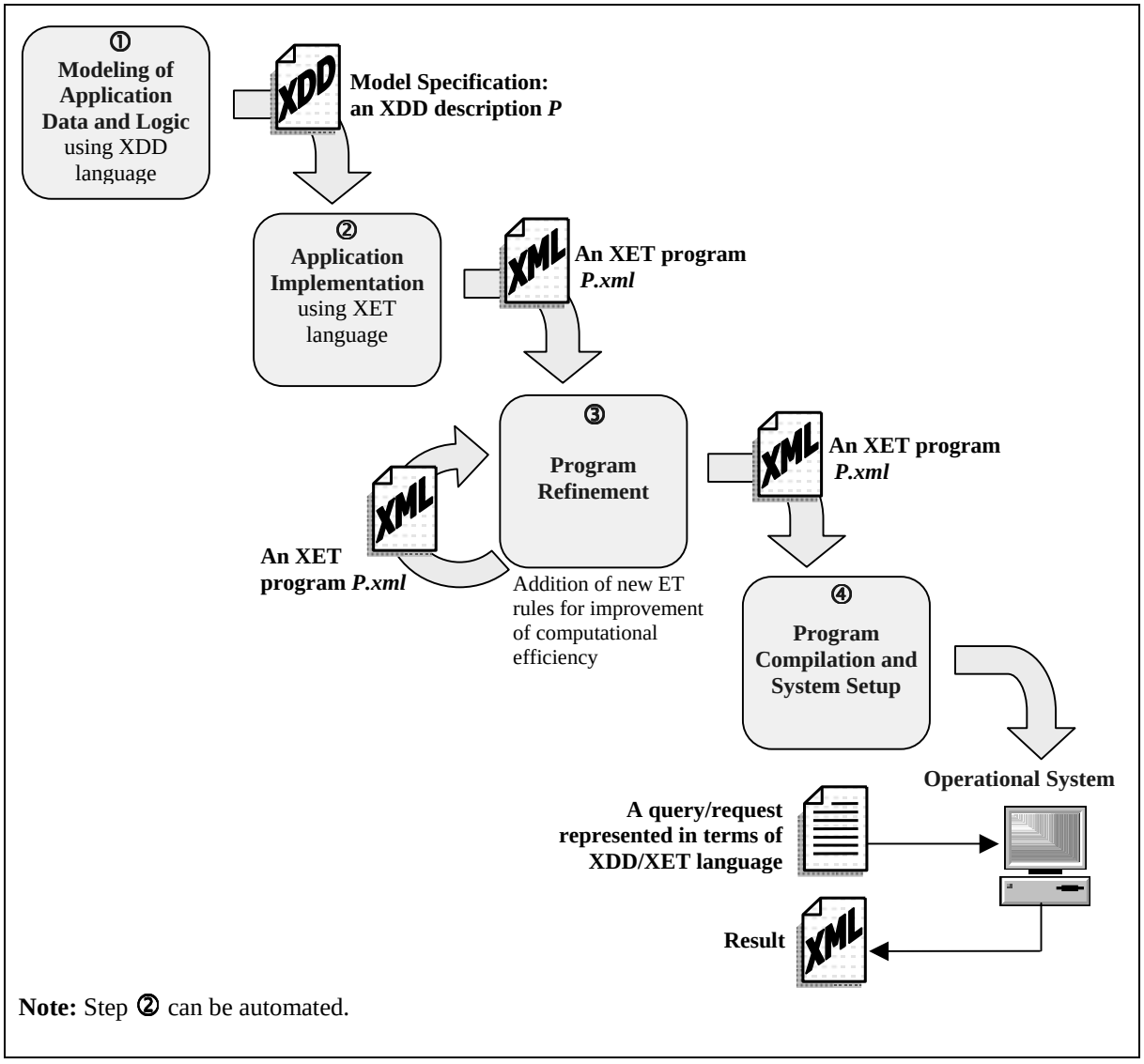


Figure 5: A declarative approach to Semantic Web application development.

www.manaraa.com

- Data type checking
- Document well-formedness and validity checking
- Arithmetic operations and relations
- String manipulation operations
- XML expression unification and matching
- XML expression operations
- Input/Output operations
- File manipulation operations
- Communication services

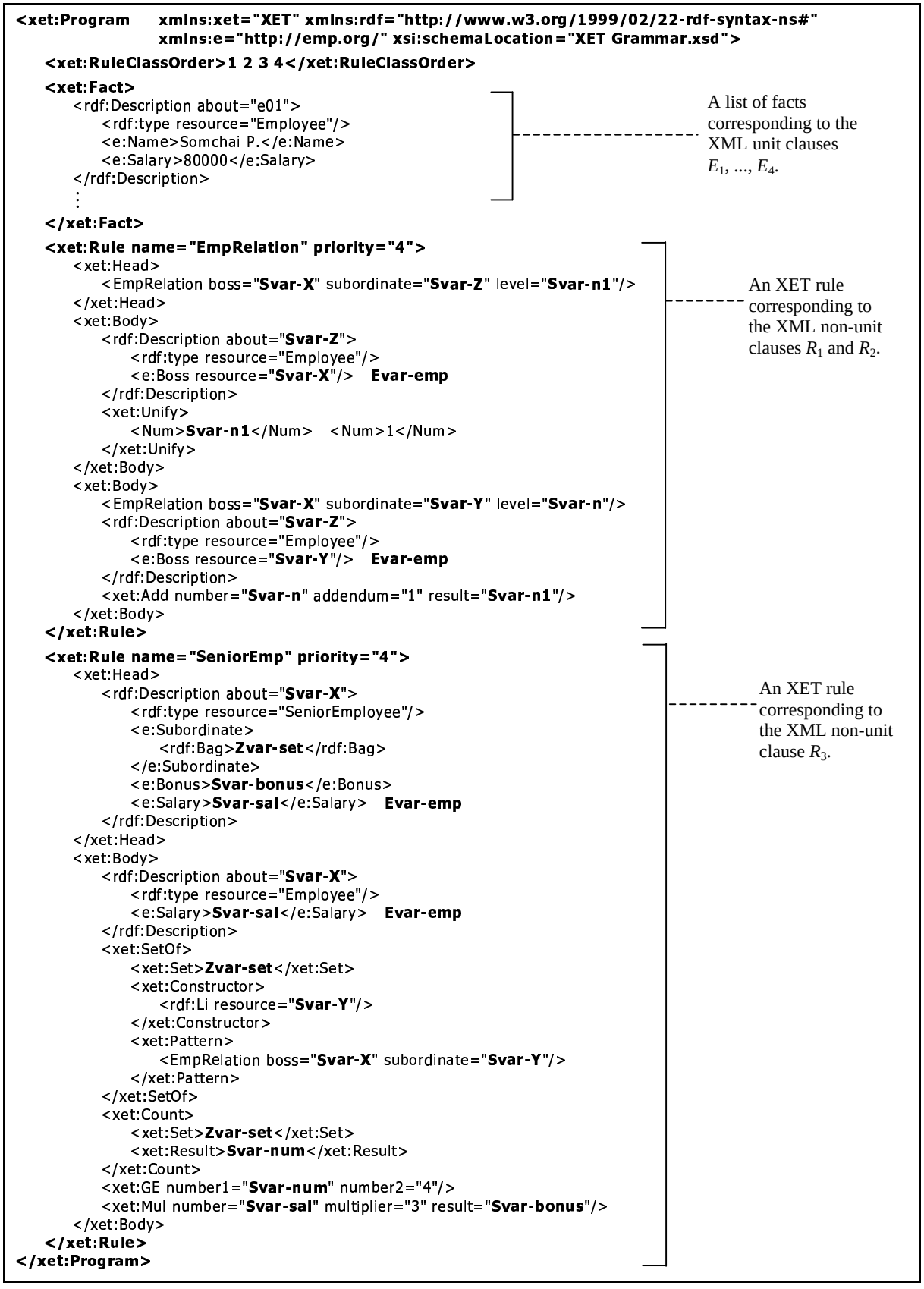
Once an XDD description which models a Semantic Web application's data and logic has been formulated, an XET program corresponding to such a description can be obtained directly. The obtained XET program can be iteratively refined, if new ET rules have been devised for improvement of computational efficiency. Finally, using *XET compiler* [18], the program is compiled and an operational system obtained. In response to a query/request submitted in terms of XDD or XET language, the system executes it and returns its result. Based on such processes, Figure 5 outlines a new declarative approach to Semantic Web application development. With reference to the XDD description and the query *Q* of Figure 4, an XET program corresponding to such a description is given by Figure 6 and the answer to the query *Q* by Figure 7.

Note that the declarative semantics of an XET program can be determined based on that of its respective XDD description.

Figure 8 depicts an example scenario of Semantic Web service execution, which starts when a user or an application *A* issues a query describing a service need together with constraints and preferences to a Semantic-Web-Service Search Engine *B*, which will then searches, from its database of Web services, for Semantic Web applications offering the demanded services with the requested properties and restrictions. Based on the returned list of applications, *A* selects an appropriate one, say Semantic Web application *C*, and sends it a query or a service-request as well as user constraints and preferences. *C* then executes such a query or request with respect to its data and application rules and logic. During its execution, *C* may forward corresponding sub-queries and/or sub-requests to other related applications based on its defined rules and logic and wait for their replies and responses. Once the execution has been finished, a reply to *A*'s query/request is returned. Note that Steps 1 and 2 can be skipped, if the user/application *A* knows at a priori which application provides the desired service. Similarly, at Steps 5.1 and 5.3 if the application *C* does not know which application it should interoperate, it may ask *B* for a list of applications offering the required services. In addition, during the execution, it is often a case that communicating parties may involve in a negotiation for modification of service conditions.

From the example scenario, one may observe that XDD and XET can serve as a tool for modeling and implementing a wide diversity of Semantic Web applications offering various kinds of services. Consider, for instance, the Semantic-Web-Service Search Engine *B*, which maintains a database of Web services described by means of Web-service metadata and provides a search facility for finding of particular services satisfying some specified criteria. Such a search engine is simply modeled as an XDD description comprising XML unit clauses, describing a collection of registered services and their properties/capabilities (in terms of Web-service metadata), and XML non-unit clauses, modeling Web-service ontologies as well as implicit relations among services in the collection. From such a description, an XET program which is capable of searching for services with desired properties and constraints can be obtained directly. Other applications *C*, *D* and *E* serving certain specific services can also be modeled and implemented in a similar manner.

www.manaraa.com




```
<xet:Program    xmlns:xet="XET" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:e="http://emp.org/" xsi:schemaLocation="XET Grammar.xsd">
  <xet:RuleClassOrder>1 2 3 4</xet:RuleClassOrder>
  <xet:Fact>
    <rdf:Description about="e01">
      <rdf:type resource="Employee"/>
      <e:Name>Somchai P.</e:Name>
      <e:Salary>80000</e:Salary>
    </rdf:Description>
    ⋮
  </xet:Fact>
  <xet:Rule name="EmpRelation" priority="4">
    <xet:Head>
      <EmpRelation boss="Svar-X" subordinate="Svar-Z" level="Svar-n1"/>
    </xet:Head>
    <xet:Body>
      <rdf:Description about="Svar-Z">
        <rdf:type resource="Employee"/>
        <e:Boss resource="Svar-X"/>   Evar-emp
      </rdf:Description>
      <xet:Unify>
        <Num>Svar-n1</Num>   <Num>1</Num>
      </xet:Unify>
    </xet:Body>
    <xet:Body>
      <EmpRelation boss="Svar-X" subordinate="Svar-Y" level="Svar-n"/>
      <rdf:Description about="Svar-Z">
        <rdf:type resource="Employee"/>
        <e:Boss resource="Svar-Y"/>   Evar-emp
      </rdf:Description>
      <xet:Add number="Svar-n" addendum="1" result="Svar-n1"/>
    </xet:Body>
  </xet:Rule>
  <xet:Rule name="SeniorEmp" priority="4">
    <xet:Head>
      <rdf:Description about="Svar-X">
        <rdf:type resource="SeniorEmployee"/>
        <e:Subordinate>
          <rdf:Bag>Zvar-set</rdf:Bag>
        </e:Subordinate>
        <e:Bonus>Svar-bonus</e:Bonus>
        <e:Salary>Svar-sal</e:Salary>   Evar-emp
      </rdf:Description>
    </xet:Head>
    <xet:Body>
      <rdf:Description about="Svar-X">
        <rdf:type resource="Employee"/>
        <e:Salary>Svar-sal</e:Salary>   Evar-emp
      </rdf:Description>
      <xet:SetOf>
        <xet:Set>Zvar-set</xet:Set>
        <xet:Constructor>
          <rdf:Li resource="Svar-Y"/>
        </xet:Constructor>
        <xet:Pattern>
          <EmpRelation boss="Svar-X" subordinate="Svar-Y"/>
        </xet:Pattern>
      </xet:SetOf>
      <xet:Count>
        <xet:Set>Zvar-set</xet:Set>
        <xet:Result>Svar-num</xet:Result>
      </xet:Count>
      <xet:GE number1="Svar-num" number2="4"/>
      <xet:Mul number="Svar-sal" multiplier="3" result="Svar-bonus"/>
    </xet:Body>
  </xet:Rule>
</xet:Program>
```

Figure 6: An XET program P.xml.

www.manaraa.com

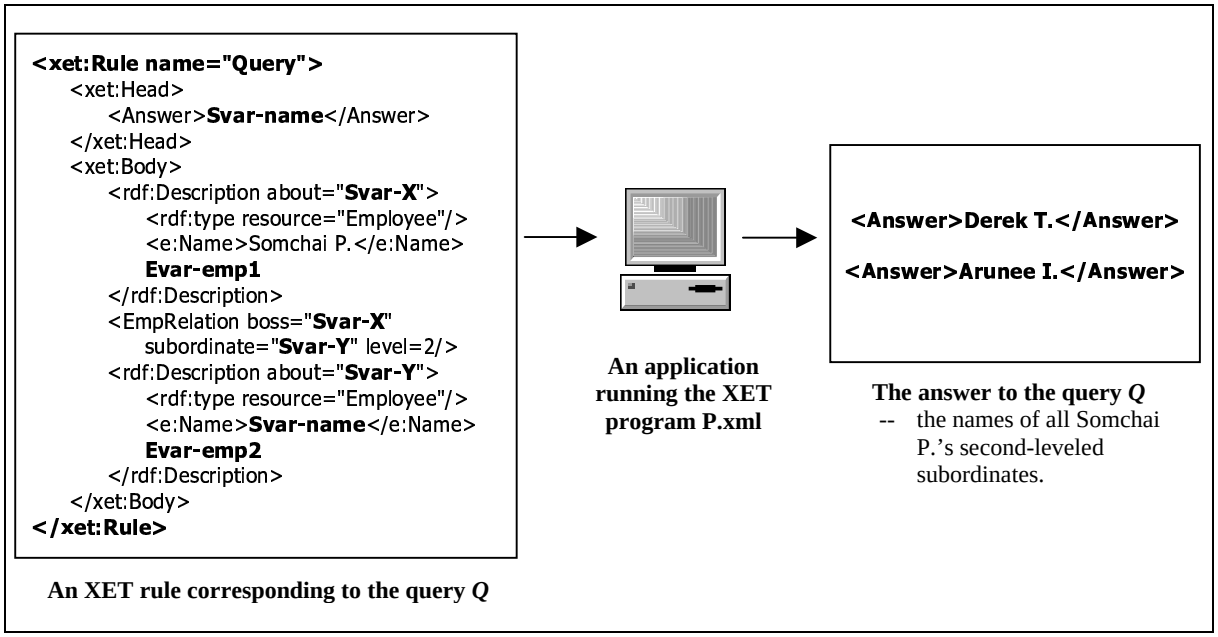


Figure 7: The answer to the query *Q*.

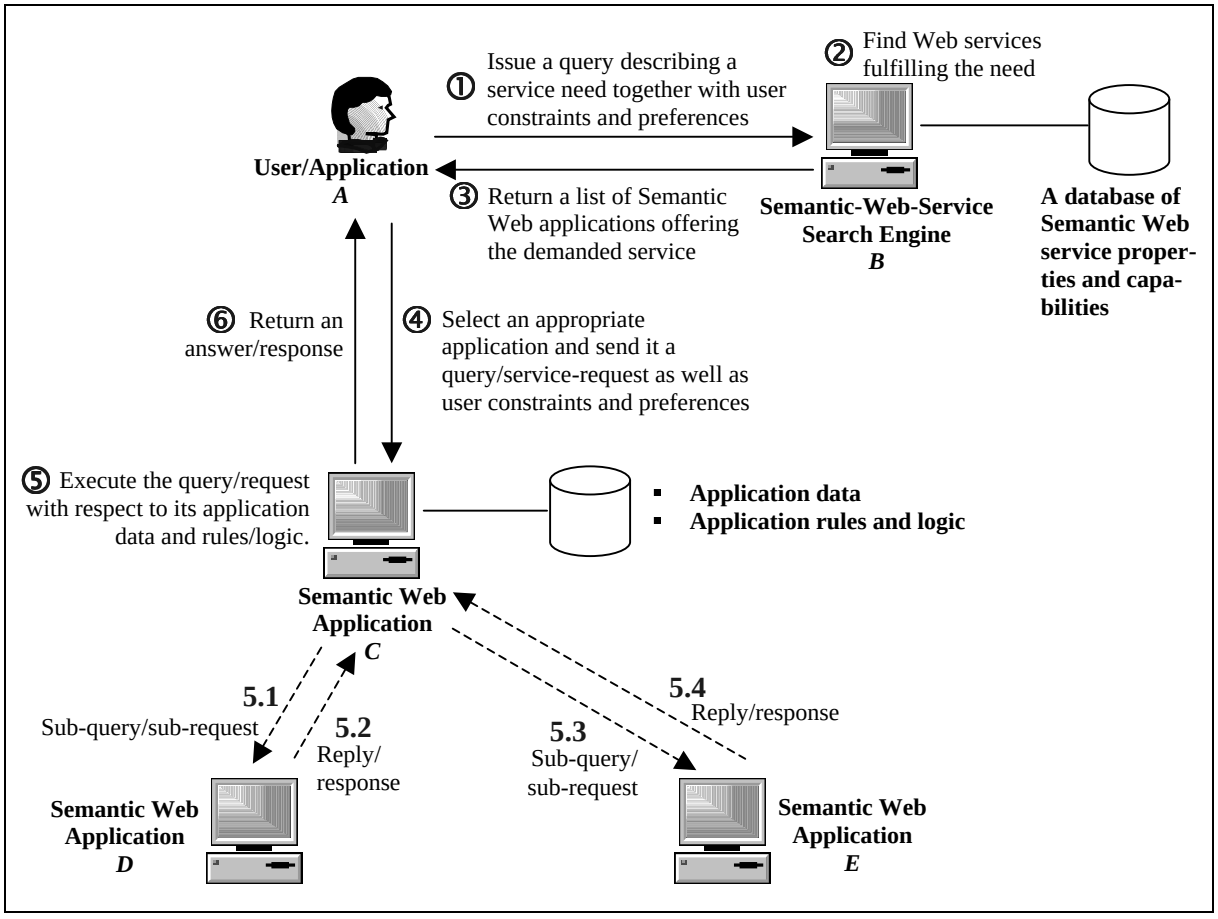


Figure 8: A scenario for Semantic Web service execution.

www.manaraa.com

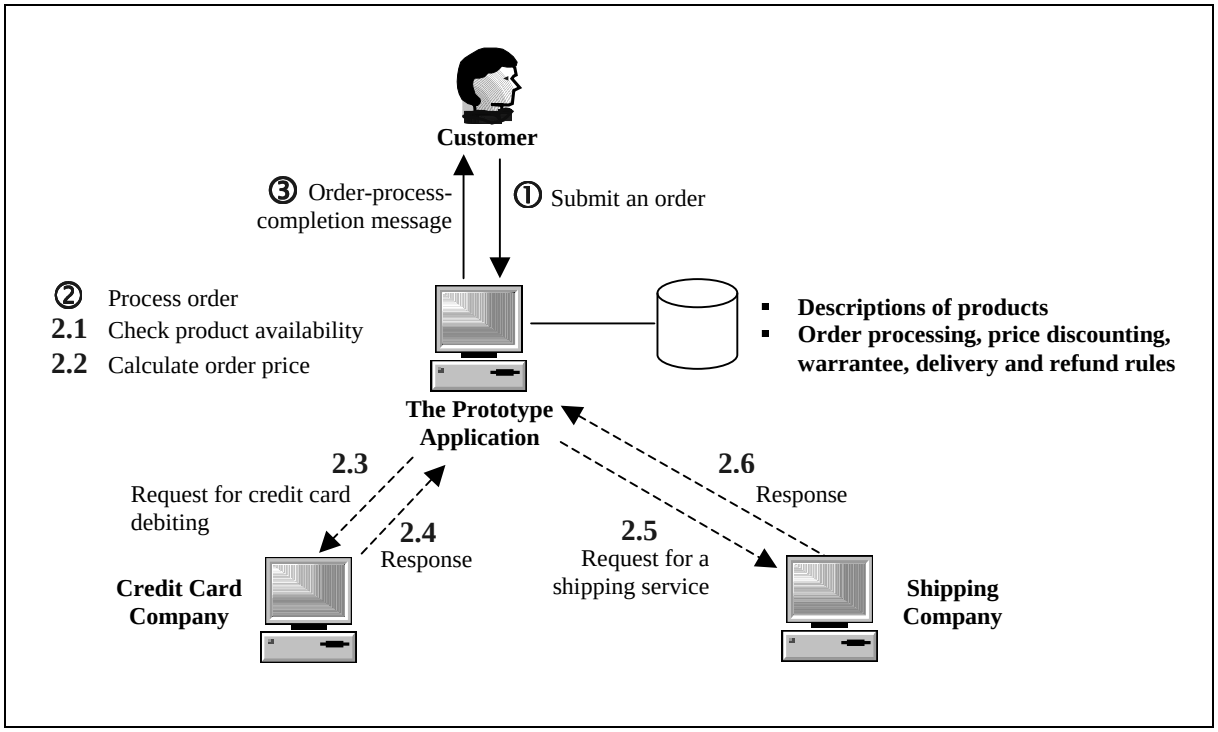


Figure 9: Prototype Semantic Web application.

5 Prototype Application

Founded on the proposed framework, a prototype Semantic Web application, which provides product-information-gathering as well as e-shopping services, has been implemented by means of XDD modeling language and XET programming language. The employed XDD and XET languages have been enhanced with the ability to handle rule conflicting problems using rule prioritization information [11]. Such an additional feature enables, for example, a formalization of a discounting policy stating that if a customer is a member of the store, a 10 percent discount is offered, and if a customer has a late-payment history, no discount is offered and that the latter has higher priority than the former [11]. Due to space limitation, declarative semantics of *prioritized XDD descriptions and XET programs* is omitted; its formal definition is available in [18].

To buy some products (Figure 9), a customer may fill out an order form and submit it on-line to the application Web site or directly send an HTTP request with appropriate parameters encoded in XML to the application URL. The application first checks its stock, and if the products are available, it will calculate the order's price, send a request to a credit card company (another Semantic Web application) for a debit of the customer's account, send a request to a shipping company and then wait for their responses. After the order process has been finished, the application notifies the customer of the completed process.

www.manaraa.com

6 Related Works

Business Rule Markup Language (*BRML*) [11] is an XML-based language for encoding of *Courteous Logic Programs* (*CLP*)—an extension of conventional logic programs by inclusion of the ability to express prioritized conflict handling. BRML is a language in the RuleML Initiative, which has been specifically designed to represent business rules and policies. However, since BRML provides merely an XML embodiment of CLPs, its expressive power is relatively limited in that its sole permitted representation is atomic formulae with simple-structure terms. Complex XML data with nesting structures cannot be directly represented in BRML. Instead they require appropriate translations into corresponding sets of atomic formulae. Figure 10, for example, shows the CLP's and BRML's representations of the XML element E_1 and the XML clause R_1 of Figure 4. Comparing XDD with BRML, one can readily observe that XDD provides a more direct and succinct modeling mechanism; While possessing sufficient expressive power to represent simple as well as complex statements and relations, its representation is still readable.

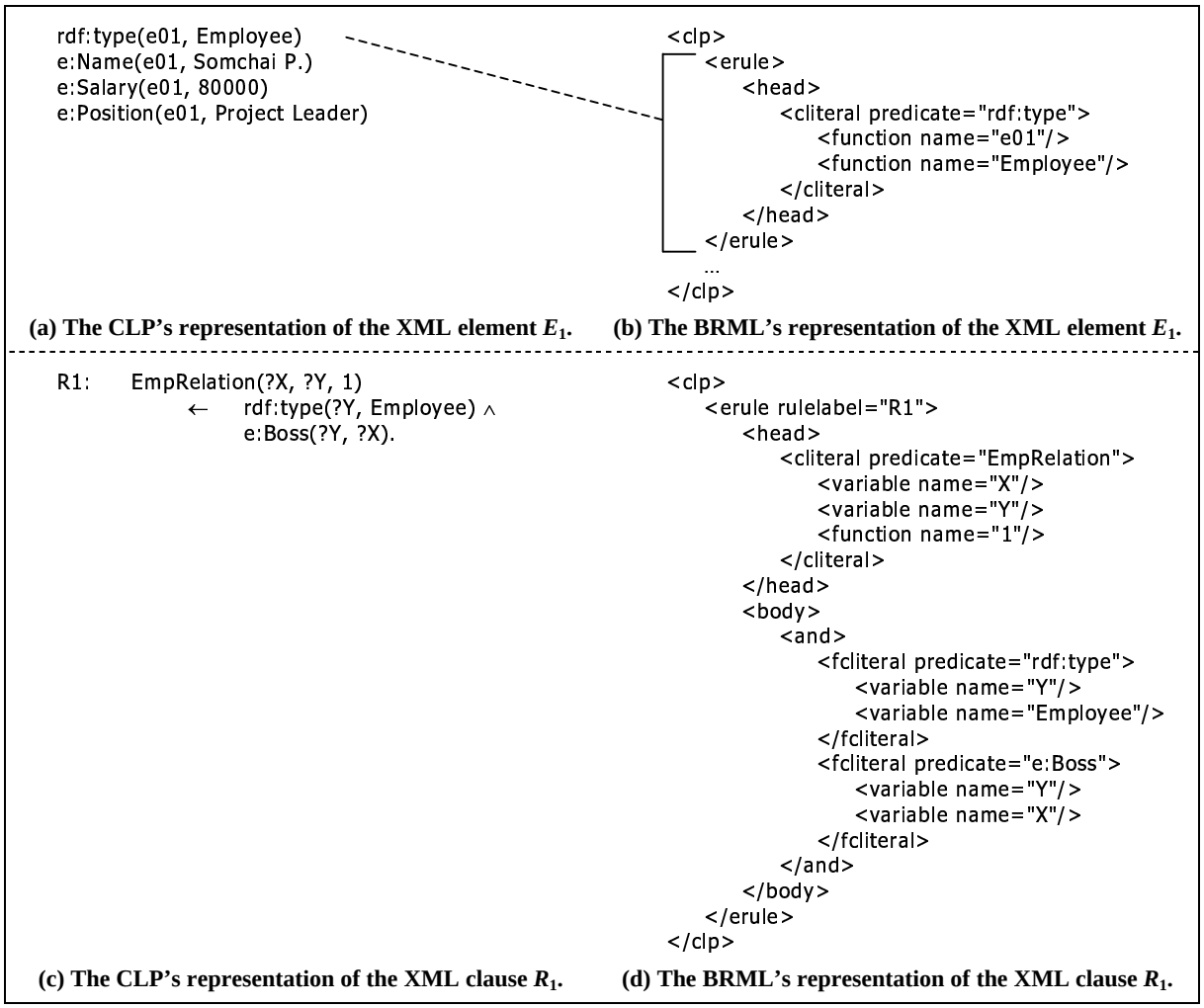


Figure 10: The CLP's and BRML's representations of the element E_1 and the clause R_1 of Figure 4.

www.manaraa.com

OIL [6,8,12] and *DAML+OIL* [13] are the two most recent, improved ontology-based semantic markup languages for Web resources which extends RDF Schema by richer sets of modeling primitives. However, their current versions still lack expressive power in that arbitrary rules and axioms cannot be described [6]. Since these languages' schemas and instances, which are encoded in RDF/XML serialization, can be directly represented in XDD as XML unit clauses, XDD can be employed to serve as their foundation, in order to help enhance their expressiveness [17]. Therefore, resources and applications modeled by these languages become immediately instances of XDD language and hence directly programmable by XET language. Note that with the awareness of the DAML+OIL's limitation in representing rules and axioms, the language is being extended with the ability to express Horn clauses and will be called *DAML-L* [16].

With reference to the overall process of Semantic Web service implementation defined in [11,16], XDD can be uniformly employed to materialize each step of the process:

- *Service Advertisement and Discovery*: A collection of service properties and capabilities described by means of metadata or *Semantic Web Service Markup Language* [16] can be directly represented as XML unit clauses, and their additional constraints and relations modeled in terms of XML non-unit clauses. Based on such declarative advertisements of Web services, discovery of a particular service having specific properties and capabilities is expressed as an XML non-unit clause, which will be evaluated on the Web service database and return as its reply a list of applications or service providers offering the requested service.
- *Negotiation*: Given particular negotiation rules and procedures for selection of Web services (e.g., response time, data accuracy and cost conditions), corresponding XML non-unit clauses can be declaratively defined. Besides definition of such rules, an appropriate employment of *Agent Communication Language* (*ACL*) [15] which allows the negotiating parties to effectively communicate and interoperate must also be considered. By a careful formulation and implementation of the two major current ACLs (i.e., *KQML* [9] and *FIPA-ACL* [10,15]) in XDD [14], XDD readily provides an effective communication in the negotiation stage, allowing every negotiating party to communicate with one another via XDD uniform interface, and hence enabling a higher level of interoperability.
- *Service Execution*: Execution of a service according to a given procedure can be represented in XDD by appropriate XML clauses. Based on such a declarative specification, an application can automatically execute the service.
- *Service Composition and Integration*: It is often a case that a service is designed as a composition or an integration of other existing services. The execution of such a composite service often requires interaction with those related services in terms of request-for-service calls and returned responses. Using XDD, one can represent service composition and integration by an XML clause, the head of which specifies the composite service and the body of which describes the composition rule as well as the service calls and the data to be exchanged with other services. Such service calls and exchanged data could be embodied in an ACL.
- *Service Customization*: In order to increase the level of share-ability, reusability and user's satisfactory of provided services, a service may be defined by a particular generic procedure which can then be customized for execution of a specific service request. Such a generic procedure is described by a set of XML clauses, and its

www.manaraa.com

customization is realized by either parameter instantiation or by addition of XML clauses into it—this latter case is equivalent to program refinement. Note that different customizations normally lead to different sequences and results of service execution.

In summary, with these supports, XDD readily provides sufficient Semantic Web modeling facilities for development of intelligent, automated Web applications requiring interoperation with other independently-developed applications.

7 Conclusions

The proposed XDD language is an expressive modeling language which allows collections of Semantic Web resources (modeled in terms of XML, RDF, OIL or DAM+OIL) to be directly represented with their semantics precisely and formally determined. In addition to such a resource modeling facility, XDD also provides a means for descriptions of Web resource manipulations, service provisions and business rules and processes. Moreover, its extension [18] by the ability to handle rule conflicting problems has enhanced its expressive power to be sufficient to capture and describe complex and conflicting rules and logic in Semantic Web applications.

Founded on XDD's expressive power and ET's computational efficiency, XET programming language and its compiler have also been developed. By means of XDD and XET languages, the paper has proposed a declarative framework for Semantic Web application development and has demonstrated that a variety of Semantic Web applications and services is simply expressible by XDD and hence programmable by XET. Moreover, the development of the prototype system based on the proposed framework has helped prove the framework's viability and potential in real applications. Integration of the proposed framework with appropriate Web and agent technologies allows intelligent as well as automated Web services, which demand syntactic and semantic interoperability, to be easily and rapidly developed. Note also that an XET program which performs particular tasks can be exchanged, shared and reused by multiple applications.

References

[1] K. Akama, Declarative Semantics of Logic Programs on Parameterized Representation Systems. Advances in Software Science and Technology **5** (1993) 45–63.

[2] K. Akama, Declarative Description with References and Equivalent Transformation of Negative References. Technical Report, Department of Information Engineering, Hokkaido University, Japan (1998).

[3] K. Akama, T. Shimitsu and E. Miyamoto, Solving Problems by Equivalent Transformation of Declarative Programs. Journal of Japanese Society of Artificial Intelligence (JSAI) **13(6)** (1998) 944–952 (in Japanese).

[4] K. Akama, C. Anutariya, V. Wuwongse and E. Nantajeewarawat, A Foundation for XML Databases: Query Formulation and Evaluation. Technical Report, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (1999)

[5] C. Anutariya, V. Wuwongse, E. Nantajeewarawat and K. Akama. Towards a Foundation for XML Document Databases. Proc. 1st Int. Conference on Electronic Commerce and Web Technologies (EC-Web 2000), London, UK. Lecture Notes in Computer Science, Springer Verlag **1875** (2000) 324–333.

[6] S. Bechhofer *et al.*, An Informational Description of Standard OIL and Instance OIL. White Paper (Nov. 2000). Available at http://www.ontoknowledge.org/oil/downl/oil-whitepaper.pdf

المنارة للاستشارات

www.manaraa.com

[7] T. Berners-Lee, Weaving the Web. Harpur, San Francisco (1999).
[8] S. Decker *et al.*, The Semantic Web: The Roles of XML and RDF, IEEE Internet Computing, (Sep./Oct. 2000) 63–74.
[9] T. Finin, Y. Labrou and J. Mayfield, KQML as an Agent Communication Language. Software Agents, AAAI/MIT Press (1997).
[10] FIPA: FIPA Specification, Version 2.0, Part 2: Agent Communication Language (1997) Available at http://www.fipa.org/spec/f8a22.zip
[11] 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. on Electronic Commerce (EC99), ACM Press (1999).
[12] F.V. Harmelen, and I. Harrocks, FAQs on OIL: The Ontology Inference Layer. IEEE Intelligent Systems **15(2)** (Nov./Dec. 2000) 69–72.
[13] J. Hendler and D. McGuinness, The DARPA Agent Markup Language. IEEE Intelligent Systems **15(2)** (Nov./Dec. 2000) 72–73.
[14] S. Jindadamrongwech, An Agent Communication Language using XML Declarative Description. Master's Thesis, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (2000).
[15] Y. Labrou, T. Finin, and Y. Peng, Agent Communication Languages: The Current Landscape. IEEE Intelligent Systems, **14(2)** (Apr./May 1999) 45–52.
[16] S.A. McIlraith, T.C. Son, and H. Zeng, Semantic Web Services. IEEE Intelligent Systems, **16(2)** (Mar./Apr. 2001) 46–53.
[17] V. Wuwongse, C. Anutariya, K. Akama and E. Nantajeewarawat: XML Declarative Description (XDD): A Language for the Semantic Web. IEEE Intelligent Systems (to appear).
[18] V. Wattanapailin, A Declarative Programming Language with XML. Master's Thesis, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (2000).

www.manaraa.com