An Equivalent-Transformation-Based XML Rule Language

Chutiporn Anutariya[1], Vilas Wuwongse[2], and Vichit Wattanapailin[2]

[1] Department of Telematics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway
Chutiporn.Anutariya@item.ntnu.no
[2] Computer Science & Information Management Program, Asian Institute of Technology, Pathumtani 12120, Thailand
vw@cs.ait.ac.th

Abstract. This paper proposes *XML Equivalent Transformation* (*XET*) as an XML-based rule langauge for the Web, which seamlessly integrates human-readable documents and computer-interpretable programs by considering XML documents and *XML expressions*—an extension of ordinary XML elements with variables—as its first class programming entities. With XET, arbitrary XML documents, representing application data, information or knowledge on the Web possibly encoded in certain XML applications, become immediately a program's input data. Manipulation and computation of such an input document (data) is performed by semantically-equivalently transforming the document successively until a desirable one is obtained. The input document could be, for example, an XML database query, and thus the output or the desirable document is a set of XML elements yielding the answer to the query. The paper presents the syntax and the computation mechanism of XET and also demonstrates its application to e-business systems.

1 Introduction

Due to their expressive power, flexibility, simplicity and ease of understanding, rules are an important information representation in computer science. They appear as execution statements in programming languages, as constraint descriptions in databases and as knowledge representations in knowledge-based systems. They have dual interpretations: procedural and declarative ones. Moreover, a rule is normally complete within itself and does not depend on other rules which lead to an incremental and modular manner for development of a rule-based system. These desirable properties of rules have recently yielded increasingly active attempts to apply rules to represent and manipulate data, information and knowledge on the Web. Since Web data will be exchanged and probably also stored by means of XML, most of the attempts have been focusing on the design and development of XML-based rule languages.

There exist several XML-based rule languages which have been developed for various purposes and applications, e.g., *XSLT* [7] for XML document transformation, *XQuery* [4] for XML query formulation and *XRML* [12] for knowledge-based

systems. The Web site [5] provides a source of and link to these languages. In order to cope with these different languages and to enable their interchange, a shared *Rule Markup Language* (*RuleML*) is being designed [5]. Apart from XSLT, most if not all of these rule languages are mere XML encoding of their related original language formalisms. For example, the following rule [5] in RuleML:

```
<imp>
   <_head>
      <atom>
         <_opr>
            <rel>own </rel>
         </_opr>
         <var>person </var>
         <var>object </var>
      </atom>
   </_head>
   <_body>
      <!– explicit 'and' –>
      <and>
         <atom>
            <_opr>
               <rel>buy </rel>
            </_opr>
            <var>person </var>
            <var>merchant </var>
            <var>object </var>
         </atom>
         <atom>
            <_opr>
               <rel>keep </rel>
            </_opr>
            <var>person </var>
            <var>object </var>
         </atom>
      </and>
   </_body>
</imp>
```

could be viewed as an XML encoding of the following Horn (Prolog) clause

$$own(Person, Object) \leftarrow buy(Person, Merchant, Object), keep(Person, Object).$$

(Recall that Prolog denotes variables by words starting with an upper case letter.) This kind of encoding is quite different from that used by normal XML documents. It is rare to see any business document containing tag names such as *atom*, *rel*, and *var* unless its schema is specially designed to work with RuleML. Hence, there arises a mismatch in encoding between the rules and the documents

to be processed. Extra tasks of schema as well as data conversions have to be performed to eliminate mismatches. Moreover, these rule languages lack theoretical or semantic foundations, unless they are translated back into their original language formalisms aggravating formulation of a mechanism to validate the correctness of any of their rule statements.

An XML rule language—*XML Equivalent Transformation* (*XET*) [3]—is presented which is founded on the *Equivalent Transformation* (*ET*) paradigm [2]—a new, flexible and efficient computational framework. Similar to XSLT, an XET program—a set of XET rules—is an XML document. It receives as an input a description, e.g., a business report, a process description or an instruction document, possibly encoded in a standard XML application such as XBRL [9]. It then processes—semantically-equivalently transforms—the input until a desirable report, process description or action instruction is obtained. In other words, an XET program can directly manipulate an XML document without any need of schema or data conversion. Semantically-equivalent transformation is a transformation of an XML document while its meaning is preserved. The meaning or semantics of an XML document is a set of XML elements each of which denotes a real domain object or relationship. Each rule in an XET program must possess this semantics-preservation property, whence its correctness can be verified while its development and application are self-dependent.

Section 2 recalls the ET paradigm as well as *XML Declarative Description* (*XDD*) [19, 20]—a language for the Semantic Web possibly employed to describe XET's target Web data, Section 3 presents XET, Section 4 demonstrates an e-business application of XET, and Section 5 concludes the paper.

2 ET and XDD

2.1 ET

The *ET* paradigm is a new computational model which solves a given problem, described in an appropriate language, by simplifying it through repetitive application of (semantically-)equivalent transformation rules. Let P_1 be a description of the original problem and $\mathcal{M}(P_1)$ its meaning. The meaning of a description is a set of concrete statements each of which is a surrogate of a real, tangible or intangible object or relationship in the domain of interest. The paradigm applies ET rules in order to successively transform P_1 into P_2, P_3, etc., while maintaining the conditions $\mathcal{M}(P_1) = \mathcal{M}(P_2) = \mathcal{M}(P_3) = \ldots$, until the description P_n, the meaning of which contains desirable statements or solutions, is obtained. An example of ET rules is the unfolding transformation which is a fundamental computational mechanism in logic programming. Prolog, the most well-known logic programming language, employs Horn logic to describe problems, defines the meaning of a description as a set of ground unit clauses, and materializes the unfolding transformation in terms of SLD resolution—its only single computational mechanism.

The general form of ET rules is

$$Head \rightarrow Body_1;\ Body_2;\ \ldots;\ Body_n.$$

where the $Head$ consists of $Object$, $\{Condition\}$ and each $Body_i$ comprises $\{Execution_i\}$, $ObjectList_i$. In other words, an ET rule is of the form:

$$\begin{array}{rcl} Object, \{Condition\} & \rightarrow & \{Execution_1\}, ObjectList_1; \\ & \rightarrow & \{Execution_2\}, ObjectList_2; \\ & & \vdots \\ & \rightarrow & \{Execution_n\}, ObjectList_n. \end{array}$$

and reads: if the object of the head matches with the target object and the condition is satisfied, then the n bodies fire simultaneously, i.e., each execution is carried out and each object list replaces the target object.

An object in ET rules models a real object or relationship in an application domain. Depending on the complexity and characteristics of the structure of real objects and relationships, an appropriate representation language should be employed to model them. As Web data encoded in XML are going to be dealt with, a flexible and expressive language for the representation of XML data is desirable. Next subsection presents such a language.

2.2 XDD

XML Declarative Description (*XDD*) [19, 20] is an XML-based modeling language, which extends ordinary, well-formed XML elements by incorporation of variables for an enhancement of expressive power and representation of implicit information into so called *XML expressions*. Ordinary XML elements—variable-free XML expressions—are called *ground XML expressions*. Every component of an XML expression can contain variables, e.g., its expression or a sequence of sub-expressions (*E-variables*), tag names or attribute names (*N-variables*), strings or literal contents (*S-variables*), pairs of attributes and values (*P-variables*) and some partial structures (*I-variables*). Every variable is prefixed by '$\$T$:', where T denotes its type; for example, $S:value and $E:expression are *S*- and *E*-variables, which can be specialized into a string or a sequence of XML expressions, respectively.

Basically, objects and their simple relationships are explicitly represented as ground XML expressions, while a more complex and possibly implicit one is modeled by an *XML clause* C, which is an expression of the form:

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

where $n \geq 0$, H is an XML expression and B_i an XML expression or an *XML constraint*, the satisfaction of which is predetermined and independent of any XML clauses. H is called the *head* and $\{B_1, \ldots, B_n\}$ the *body* of C. When its body is the empty set, C will be referred to as an *XML unit clause* and the symbol $\leftarrow$ is often omitted; hence, an XML element or document can be mapped directly onto a ground XML unit clause.

An *XML declarative description* or simply an *XDD description* is a set of XML clauses. Intuitively, given an XDD description D, its meaning is the set of

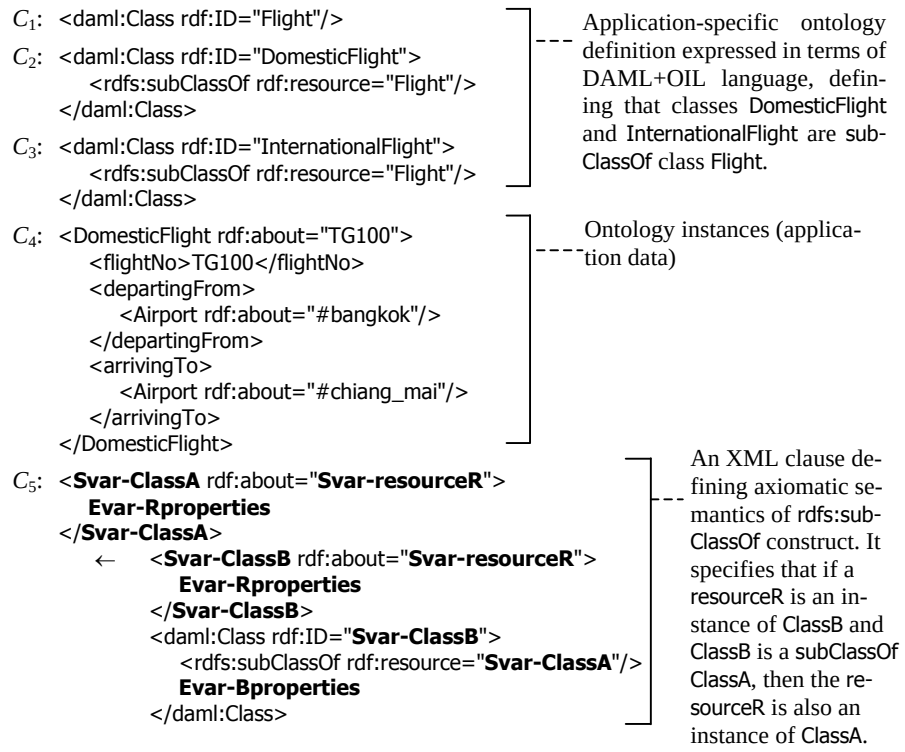

Fig. 1. An example of an XDD description D, comprising ontology definitions, instances and a non-unit clause defining the axiomatic semantics of rdfs:subClassOf construct.

all XML elements which are directly described by and are derivable from the unit and non-unit clauses in D, respectively. Papers [19, 20] give theoretical details of the theory including *XML specialization system*—a mathematical abstraction which reflects the data structure and specialization/instantiation behavior of XML expressions—and the formal semantics of XDD descriptions.

Due to its generality and expressiveness, XDD can be employed to model various types of applications such as constrained XML databases and their queries [19], the Semantic Web [20], UML diagrams [14] and e-government knowledge management [17]. Fig. 1 gives a sample XDD description which models a simple ontology application. It comprises four unit clauses $C_1 - C_4$, representing ontology definitions and instances and encoded in *RDF* [11], *RDF Schema* (*RDFS*) [6] and *DAML+OIL* [10] languages, and a non-unit clause C_5, representing the axiomatic semantics of the ontology modeling primitive rdfs:subClassOf. The meaning of the description yields not only those elements explicitly described by the four unit clauses, but also a derived one based on the defined non-unit clause.

3 XET

XET (*XML Equivalent Transformation*) [3] is an XML-based, declarative, higher-order programming language which seamlessly integrates XML syntax and the ET computational paradigm. Founded on the XDD theory, XET can directly and succinctly manipulate XML documents without a necessity for data conversion. An *XET program* comprises a set of *XET rules* and XML elements/ documents—regarded as the program's data or *facts*. Each XET rule has a similar structure to an ET rule given in Section 2.1 except that every component of an XET rule could be an arbitrary XML expression—a modeling of a real-world object in a domain of interest.

Fig. 2 depicts the structure and syntax of an XET program. For the complete grammar of XET language including all available built-in operations, expressed in terms of an XML Schema, the reader is referred to [15].

xet:Fact is used for specification of an application's data, which could be, for example, a product catalog, a business report (financial statement), a process description or an instruction document, possibly encoded in a standard XML application, e.g., XBRL. Moreover, ontologies—modeling of application-specific concepts, their properties and hierarchies—expressed by ontology markup languages such as RDF(S) and DAML+OIL, can be immediately represented in xet:Fact. Fig. 3 gives a simple XET program which directly encodes an application-specific ontology and its instances expressed in DAML+OIL as part of the program's data.

XET rules are of two types: *derivation* and *reaction*. The former are used to specify relationships/axioms among elements in the program or used to define a transformation behavior of certain XML elements, and the latter are employed to define rules which will be invoked when predefined events occur. Thus, business rules, logic, policies, axioms and queries as well as program transformation rules are expressed as derivation rules, while event-based computation rules are materialized by reaction rules.

With reference to Fig. 2, an xet:Rule comprises the following components:

- name;
- priority: for handling rule conflicting problems, i.e., a rule with higher priority will override the lower ones;
- Head: containing a *HeadElement* which specifies an XML expression pattern to be matched and transformed, or defines an event to be monitored for firing the rule. In the former case, the rule becomes a derivation rule, and in the latter a reaction rule;
- Condition (optional): encoding a list of *CondElements* that must be satisfied for execution of the rule. These *CondElements* could be built-in or user-defined predicates;
- Body: consisting of zero or more *BodyElements*, each of which is one of the following
  - an XML element to be matched with the head of other rule in the program,

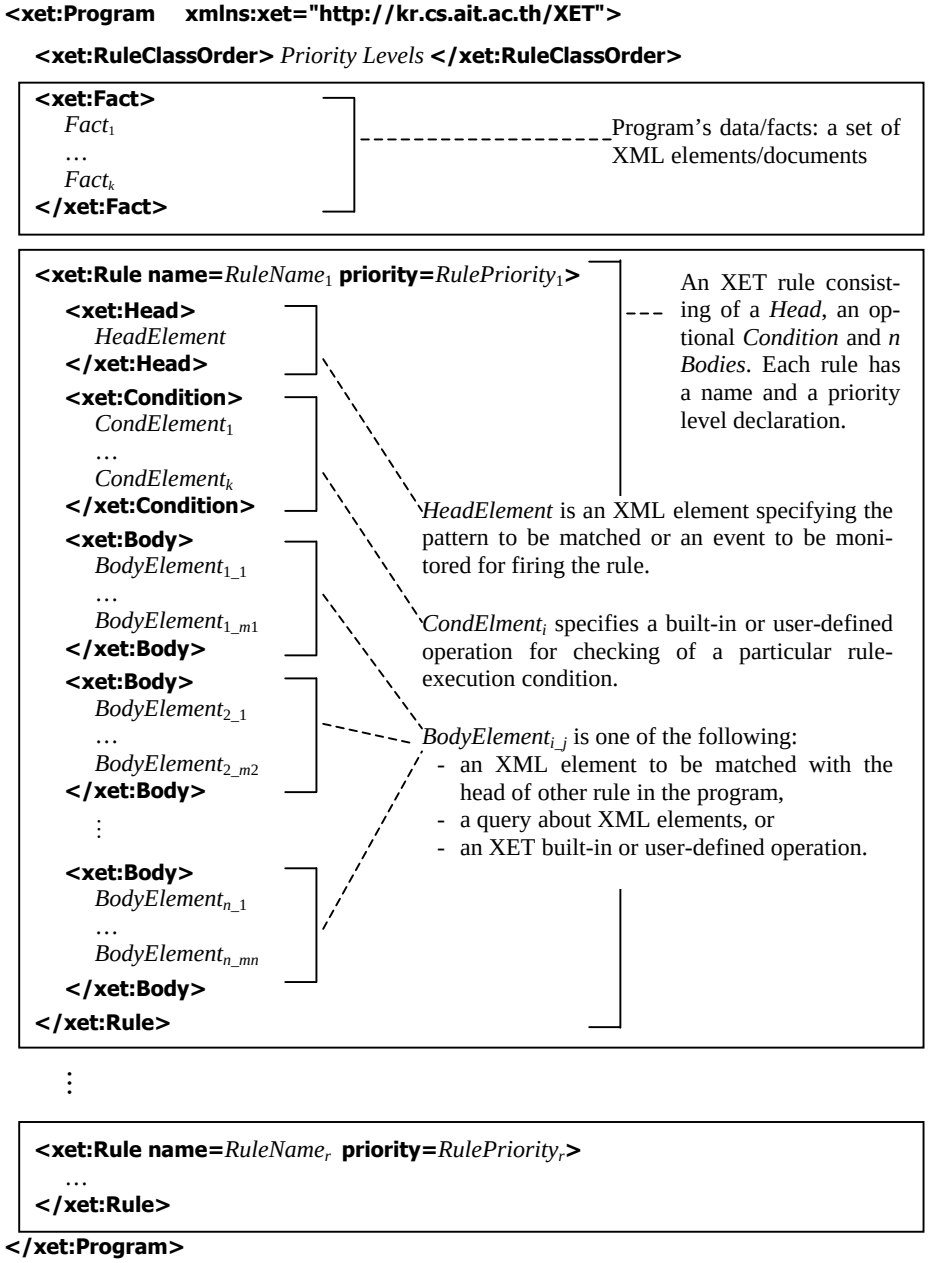

Fig. 2. XET program structure.

```
<xet:Program  xmlns:xet="http://kr.cs.ait.ac.th/XET">

  <xet:RuleClassOrder> 1 2 3 4</xet:RuleClassOrder>

  <xet:Fact>
    <daml:Class rdf:ID="Flight"/>
    <daml:Class rdf:ID="DomesticFlight">
      <rdfs:subClassOf rdf:resource="Flight"/>
    </daml:Class>
    <daml:Class rdf:ID="InternationalFlight">
      <rdfs:subClassOf rdf:resource="Flight"/>
    </daml:Class>

    <DomesticFlight rdf:about="TG100">
      <flightNo>TG100</flightNo>
      <departingFrom>
        <Airport rdf:about="#bangkok"/>
      </departingFrom>
      <arrivingTo>
        <Airport rdf:about="#chiang_mai"/>
      </arrivingTo>
    </DomesticFlight>
  </xet:Fact>

  <xet:Rule name="SubClassOf" priority="4">
    <xet:Head>
      <Svar-ClassA rdf:about="Svar-resourceR">
        Evar-Rproperties
      </Svar-ClassA>
    </xet:Head>
    <xet:Body>
      <Svar-ClassB rdf:about="Svar-resourceR">
        Evar-Rproperties
      </Svar-ClassB>
      <daml:Class rdf:ID="Svar-ClassB">
        <rdfs:subClassOf rdf:resource="Svar-ClassA"/>
        Evar-Bproperties
      </daml:Class>
    </xet:Body>
  </xet:Rule>

</xet:Program>
```

Application-specific ontology definition expressed in terms of DAML+OIL language, defining that classes DomesticFlight and InternationalFlight are subClassOf class Flight.

Ontology instances (application data)

An XET rule defining axiomatic semantics of rdfs:subClassOf construct. It specifies that if a resourceR is an instance of ClassB and ClassB is a subClassOf ClassA, then the resourceR is also an instance of ClassA.

Fig. 3. An example of an XET program, corresponding to the XDD description D of Fig. 1 and denoted by P.xml, comprising ontology definitions, instances and a rule defining the axiomatic semantics of rdfs:subClassOf construct.

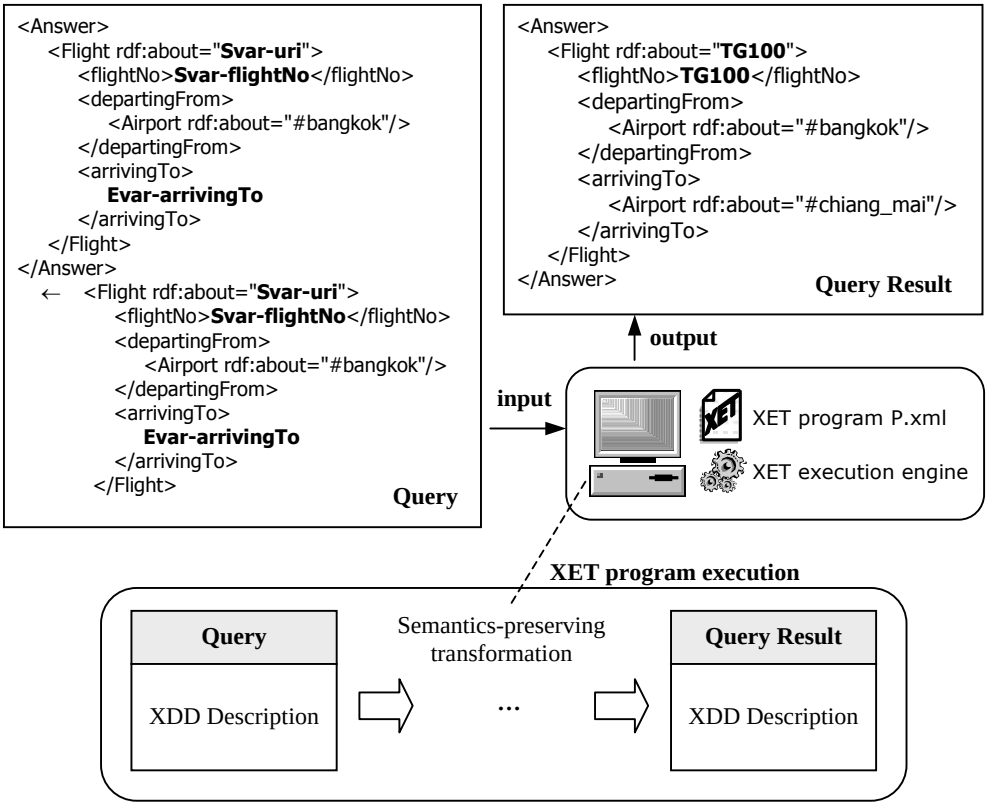

Fig. 4. An example of a query and its execution by an XET program.

- a query about XML elements in the program, or
- an XET built-in or user-defined function.

The sample XET rule of Fig. 3 shows how to construct rules for defining the axiomatic semantics of rdfs:subClassof construct. Other ontology modeling constructs, such as rdfs:subPropertyOf, daml:inverseOf and daml:TransitiveProperty, can also be defined as corresponding XET rules in the same manner; this is part of the work accomplished by [16]. Note that all variables in XET rules are prefixed with their variable-type specifications; for instance, an S-variable named uri is represented in a program as Svar-uri.

Based on the defined facts and rules of the given program P.xml of Fig. 3, a query about all flights originating from Bangkok airport will also return those domestic and international flights flying from Bangkok, although they are not explicitly defined as instances of the class Flight (cf. Fig. 4).

Fig. 4 also depicts computation/execution of an XET program. A given problem—e.g., a database query, a business transaction request or a process execution command—is executed by successively applying semantics-preserving

transformation rules to an XDD description, describing such a problem, until a desirable XDD description yielding its answer is obtained. In general, given an XDD description describing a particular problem, a set of XET rules for implementing such a problem can be easily derived; thus, XDD descriptions can be viewed as XET program specifications. Additional XET rules for improvement of computational efficiency can also be devised based on certain specific characteristics and properties of application data. Besides manually implementing XET rules, their automatic generation based on a given XDD description is envisaged. Note also that if only semantic-preserving transformation is applied in each transformation step, the correctness of the computation is always guaranteed.

It is readily seen that other XML-based rule languages can be considered to be special cases of XET language with strict rule syntax and rigid computation mechanism. In addition, since XET programs are also XML documents, any off-the-shelf tools can readily be used to edit, parse and validate the grammatical correctness of the programs.

4 E-business Application

An XET approach to e-business application development will be presented and a simple application of B2C and B2B travel businesses demonstrated.

4.1 E-business Application Components

This subsection discusses the key components of an e-business application and shows how each component can be programmed in XET.

1. *E-business service descriptions*
   In order to enable an e-business application to automatically locate and utilize a service offered by another application, a mechanism for machine-comprehensible description of available services in terms of their properties, functionalities, constraints and interfaces such as inputs and outputs must be established. Recent efforts on development of such a mechanism include *Universal Discovery Description and Integration* (*UDDI*) [18], *Web Services Description Language* (*WSDL*) [8] and *DAML-family markup language for service description* (*DAML-S*) [13]. Since they are XML-based languages, descriptions of services expressed in these language become immediately data or facts of an XET program.
2. *Ontology definitions and ontology instances*
   Ontologies play an important role in providing an ability to model, represent and exchange formal conceptualization as well as information of particular application domains in a precise, machine-understandable form. Hence, they can provide a means for semantic interoperability among independently-developed e-business applications. Ontology definitions—description of concept- and property-hierarchies, some particular axioms and constraints (e.g.,

Table 1. Components of an e-business application.

E-business Application Components	Programmed by
• E-business service descriptions	XET facts encoding DAML-S, UDDI or WSDL instances
• Ontology definitions and ontology instances	XET facts encoding RDF(S) and DAML+OIL elements
• Ontology axiomatic semantics	XET rules
• Application rules and axioms	XET rules

inverseOf, domain and range)—of an application domain as well as ontology instances (application data), encoded in recently developed ontology markup languages, such as RDF(S) and DAML+OIL, can also be mapped directly onto elements of xet:Fact as demonstrated by Fig. 3.

3. *Ontology axiomatic semantics*
   A definition of the axiomatic semantics of each ontology modeling primitive can be implemented by an appropriate XET rule. Since these modeling primitives often include certain notions of implication, defining their axiomatic semantics in terms of XET rules will allow derivation of and reasoning with XML elements in the application. Fig. 3 shows an example of such a rule which corresponds to the meaning of rdfs:subClassOf.
4. *Application rules and axioms*
   Apart from the ability to directly encode application-specific ontologies and instances, XET also yields facilities for implementation of business rules, processes, axioms and constraints, such as discounting rules and return policies, by means of XET rules. Although these business rules are essential elements in e-business applications, they are inexpressible by those available ontology modeling languages. Thus, sole employment of ontology programming languages is insufficient to e-business application materialization.

Table 1 summarizes these basic components of an e-business application and discusses how they can be programmed in XET.

4.2 A Framework for E-business Applications

In order to provide a framework for development of e-business applications, the following facilities have been incorporated into *XET program execution engine*:

1. *Ontology execution engine*: a predefined/precompiled XET program defining axiomatic semantics of each RDF(S)'s and DAML+OIL's ontology modeling primitives;
2. *Communication support and Web service execution engine*: a predefined/precompiled XET program which can generate an appropriate XML-based request/response message encoded in a SOAP envelope and send it to other e-business applications according to their defined service descriptions and service operation interfaces expressed in DAML-S;

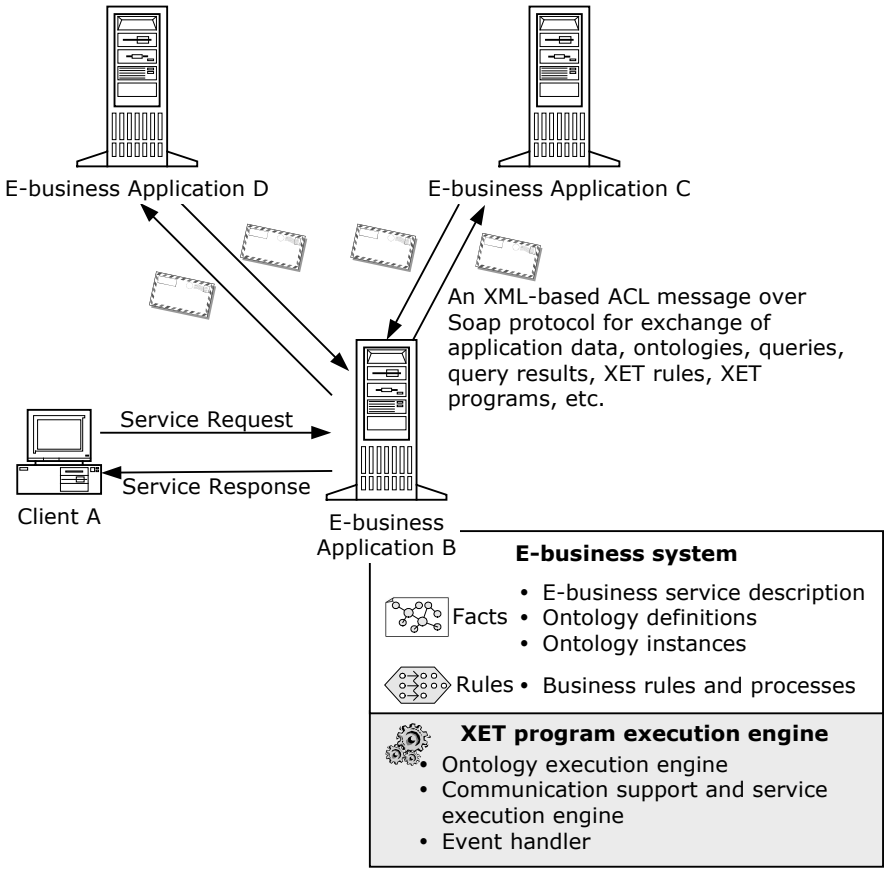

Fig. 5. A framework for e-business applications.

3. *Event handler*: an integrated set of DOM and SAX Java API.

Based on the developed XET engine for e-business applications, an operational e-business system can be implemented as an XET program consisting of facts and rules describing business service descriptions, ontologies and business rules as outlined in the previous subsection. Fig. 5 illustrates the proposed framework.

4.3 A Prototype System

By means of the proposed framework, this subsection demonstrates a prototype system for B2C and B2B travel business applications, which comprises a B2C service provider (a travel agent), B2B service providers (an airline, hotel, and car rental businesses) and an e-business broker. Each of these applications are

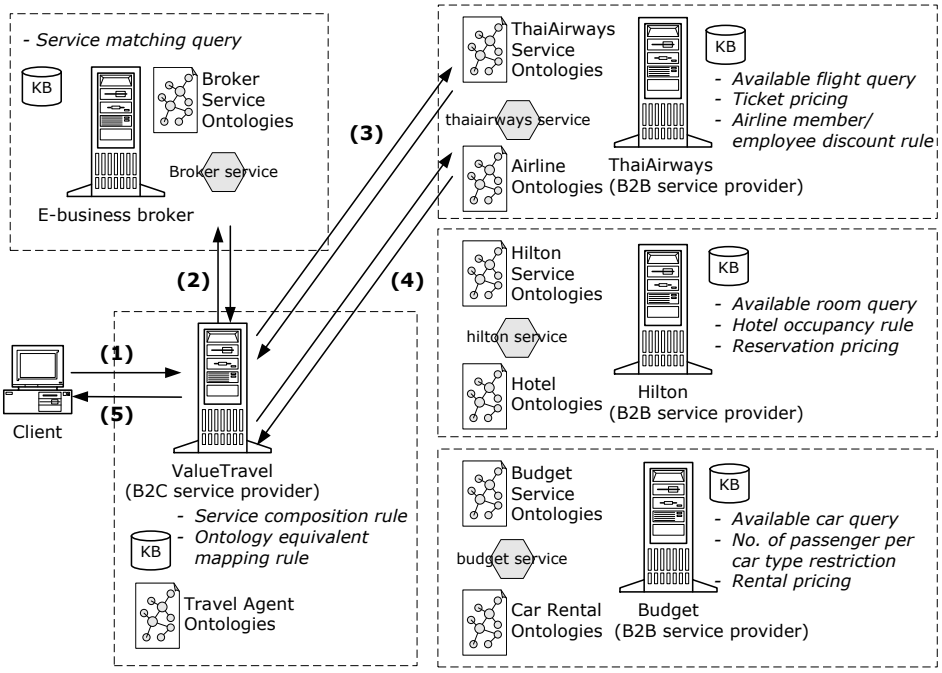

Fig. 6. A prototype system.

implemented independently by an XET program. The prototype also presents possible employment of DAML+OIL for representation of domain-specific ontologies and instances and DAML-S for e-business service descriptions; these are encoded as facts of XET programs. Application constraints, rules and logic, inexpressible by DAML+OIL and DAML-S, are implemented in the prototype by appropriate XET rules.

Fig.6 illustrates a possible scenario of service advertisement, discovery, composition, and execution, which starts when a user submits a request for service to a travel agent via a Web browser. In the scenario:

1. A user, who wants to buy a travel package, submit their personal information and traveling requirements to the B2C service provider ValueTravel via HTML form. Based on the user's requirements, ValueTravel then obtains a list of required services from its service-composition rules. In this scenario, assume that airline-ticket, hotel-reservation and car-rental services are demanded.
2. From the obtained list of required services, ValueTravel invokes a registry-lookup service provided by the e-business broker to find a list of potential B2B service providers.
3. In order to buy an airline ticket, let ValueTravel select ThaiAirways from the given list of providers, qualifying the specified travel origin/destination

constraint. It then obtains a ThaiAirways's service ontology; thus, it will know that, in order to acquire an airline ticket, it has to complete two processes: GetFlightDetails and ConfirmFlightBooking.

4. Based on ThaiAirways's service ontology specifying declarative interfaces and conditions, ValueTravel sends a SOAP envelope encoding a request message with passenger and travel information to ThaiAirways for execution of the GetFlightDetails process. In response to such a message, ThaiAirways queries its flight database and calculates ticket prices according to its predefined set of business rules implemented as XET rules, such as:
   - Child fares for children below 12 years of age are at half-price of the standard adult fare;
   - A member passenger and his/her immediate family will get a 5% discount.

   These rules are examples of business rules and logic which cannot be expressed by DAML+OIL and DAML-S.

   After completion of its execution, ThaiAirways returns a list of Ticket-elements with ticket fare and discount information to ValueTravel, which will then automatically select tickets that best match the users' constraints and send a confirmation message back to the ThaiAirways' ConfirmFlightBooking process.

   In order to complete the user's request, ValueTravel then repeats Steps 2–4 on other B2B service providers for hotel reservation and car rental.
5. Finally, the service composition result, i.e., a travel itinerary, is returned to the client browser in a human-readable form.

5 Conclusions

Founded on an expressive XML modeling language—XDD—and a flexible and efficient computational model—ET—this paper has developed an equivalent-transformation-based XML rule language, namely XET, which allows direct, succinct and efficient computation of and reasoning with both explicit and implicit XML elements without a necessity for data conversion. By employment of XET, a framework for ontology-enabled e-business applications with automated business-process capabilities has also been proposed, and a prototype system for B2C and B2B business applications implemented with the capabilities for service advertisement, discovery, composition, execution and interoperation. The system, available at http://kr.cs.ait.ac.th, also helps demonstrate the framework's viability and indicate its potential as a solid approach to e-business applications.

References

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

2. Akama, K., Shimitsu, T., Miyamoto, E.: Solving Problems by Equivalent Transformation of Declarative Programs. J. Japanese Society of Artificial Intelligence, Vol. 13 No.6, pp. 944–952 (1998) (in Japanese)
3. Anutariya, C., Wuwongse, V., Akama, K. and Wattanapailin, V.: Semantic Web Modeling and Programming with XDD. Proc. 1st Semantic Web Working Symposium 2001 (SWWS'01), CA, pp. 161–180 (2001)
4. Boag, S., Chamberlin, D., Clark, J., Fernandez, M., Florescu, D., Robie, J., Siméon, J., Stefanescu, M.: XQuery 1.0: An XML Query Language, W3C Working Draft (April 2002) http://www.w3.org/TR/xquery/
5. Boley, H.: Rule Markup Language (RuleML) (2002) http://www.dfki.uni-kl.de/ruleml
6. Brickley, D. and Guha, R. V.: Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation (March 2000) http://www.w3.org/TR/rdf-schema/
7. Clark, J.: XSL Transformations (XSLT) Version 1.0. W3C Recommendation, (November 1999) http://www.w3.org/TR/xslt
8. Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S.: Web Services Description Language (WSDL) 1.1 (March 2001) http://www.w3.org/TR/2001/NOTE-wsdl-20010315
9. Hampton, L. and vun Kannon, D.: Extensible Business Reporting Language (XBRL) 2.0 Specification (2001) http://www.xbrl.org/TR/2001/XBRL-2001-12-14.htm
10. Hendler, J. and McGuinness, D.L.: The DARPA Agent Markup Language. IEEE Intelligent Systems, Vol. 15, No. 6, pp. 72–73 (2000)
11. Lassila, O. and Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, (February 1999) http://www.w3.org/TR/REC-rdf-syntax.
12. Lee, J.K and Sohn, M.M.: Extensible Rule Markup Language – Toward the Intelligent Web Platform. Communications of the ACM (2002) (to appear)
13. McIlraith, S. A. Son, T. C. and Zeng, H.: Semantic Web Services. IEEE Intelligent Systems, Vol. 16, No. 2, pp. 46–53 (March/April 2001)
14. Nantajeewarawat, E., Wuwongse, V., Anutariya, C., Akama, K. and Thiemjarus, S.: Towards Reasoning with UML Diagrams Based-on XML Declarative Description Theory. Proc. Intl. Conf. Intelligent Technologies (InTech'2000), Bangkok, Thailand (2000)
15. Wattanapailin, V.: A Declarative Programming Language with XML. Master's Thesis, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (2000)
16. Suwanapong, S.: Intelligent Web Services. Master's Thesis, Computer Science and Information Management Program, Asian Institute of Technology, Thailand (2001)
17. Teswanich, W., Anutariya, C. and Wuwongse, V.: Unified Representation for E-Government Knowledge Management. Proc. 3rd Workshop on Knowledge Management in Electronic Government (KMGov2002), Copenhagen, Denmark (to appear).
18. UDDI: The UDDI Technical White Paper (September 2000) http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf
19. Wuwongse, V., Akama, K., Anutariya, C. and Nantajeewarawat, E.: A Data Model for XML Databases. Proc. 2001 Intl Conf. Web Intelligence (WI-01), Maebashi, Japan, LNAI 2198, pp. 237–246 (2001)
20. Wuwongse, W., Anutariya, C., Akama, K., Nantajeewarawat, E.: XML Declarative Description (XDD): A Language for the Semantic Web. IEEE Intelligent Systems, Vol. 16, No. 3, pp. 54–65 (May/June 2001)