An Intelligent Web Service System

S. Suwanapong[1], C. Anutariya[2], and V. Wuwongse[1]

[1] *Computer Science & Information Management Program, School of Advanced Technologies, Asian Institute of Technology, Pathumthani 12120, Thailand*
suwanee@beasys.co.th, vw@cs.ait.ac.th

[2] *Department of Telematics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway*
Chutiporn.Anutariya@item.ntnu.no

Abstract :   The Semantic Web extends today's Web technology by enabling machine-human as well as machine-machine interactions with well-defined semantics of Web resources and services. Automated interoperation among machines demands a tool which can manipulate semantic information, enhances understanding and reasoning, and leads to machine-machine collaboration. This *Intelligent Web Service (IWS) System* presents a declarative approach to the construction of Semantic Web applications by means of a unified modeling language *XML Declarative Description (XDD)* and an XML-based declarative programming language *XML Equivalent Transformation (XET)*. With XDD's expressiveness and inference capabilities as well as XET's computational, document- and query-processing mechanisms, IWS introduces a uniform representation of ontology axioms, ontology definitions and instances as well as application constraints and rules in machine-processible form. It is also employed to create a prototype of B2B travel businesses as a demonstration of the framework's practicality and potential usage on the Semantic Web.

Key words :   Semantic Web, Intelligent Web Services, XML Declarative Description, XML Equivalent Transformation, DAML+OIL, DAML-S, SOAP, ACL.

1.   INTRODUCTION

The *Semantic Web* [4] is an extension of today's Web technologies which makes Web resources accessible by their semantic contents rather than merely by keywords and their syntactic forms. Due to its well-established mechanisms for expressing machine-interpretable information, information and *Web services*—Web resources offering certain services—previously available for human consumption, can be created in a well-defined, structured format from which machines can comprehend, process and interoperate in an open, distributed computing environment.

Communication among heterogeneous, independently developed Web services and software agents demands a well-defined mechanism for semantic description of services and their properties, allowing them to share a similar understanding, when processing the same set of information. To achieve this goal, several ontology markup languages such as *RDF* [15], *RDF Schema* (*RDFS*) [6] and *DAML+OIL* [12] have been developed for the creation of particular domain-specific ontologies and the instantiation of these ontologies in the description of specific Web resources. The advent of these languages also lead to means for reasoning and operation of automated tasks as well as sharing and exchange of information among *ontology-enabled Web services*, or so-called *Intelligent Web Services* (*IWSs*).

There exist a number of industrial standards for description of software components, service interfaces, content data and business processes such as *UDDI* [18] and *WSDL* [7]. However, they primarily enable automatic Web service discovery and invocation, while lacking sufficient mechanisms for fully support automatic Web service interoperation, composition and execution monitoring [9].

A current approach towards the realization of IWSs includes development of a *DAML*-family markup language for Web services (*DAML-S*) [16]—a Web service ontology with a set of primitive classes and properties for the description of service properties and capabilities in an unambiguous, machine-comprehensible manner. The upper ontology class *Service* is defined at the top of *service taxonomy* and has three essential relationships to the classes:

- *ServiceProfile*: specification of service capabilities—useful for automatic service discovery,

- *ServiceModel*: description of how a service performs its operations,
- *ServiceGrounding*: description of technical details on how to access a service.

Since DAML-S does not readily supply ability for the description of application constraints and rules, its employment to develop IWS applications demands also an integration of a particular logic programming language.

A new declarative approach for IWS modeling and development is obtained by means of *XML Declarative Description* (*XDD*) [20]—a unified XML-based knowledge representation with well-defined semantics and expressive means for direct and uniform representation of ontology axioms, definitions and instances as well as application constraints and rules. Due to its generality, flexibility and expressiveness, the developed approach can directly represent those standard XML-based markup languages for domain-model ontologies and Web service descriptions, e.g., RDF(S), DAML+OIL, UDDI, WSDL and DAML-S. Moreover, it can enhance the languages' expressive power by allowing definition of arbitrary rules, constraints and axioms. Using *XML Equivalent Transformation* (*XET*) [3]—a declarative programming language for XDD—the approach is readily equipped with reasoning capabilities, computational and query-processing mechanisms. In order to provide communication means among IWSs, it employs *SOAP* [5] to encode *ACL* [14] messages, which could be either messages in *KQML* [11] or *FIPA-ACL* [10].

Section 2 introduces XDD and XET languages, Section 3 identifies basic requirements of IWSs, Section 4 develops an XDD approach to modeling IWSs, Section 5 presents an IWS system architecture, Section 6 demonstrates a prototype IWS system, and Section 7 concludes.

2. INTRODUCTION TO XDD AND XET

2.1 XML Declarative Description

XDD [20]—an XML-based knowledge representation, developed by employment of *Declarative Description* (*DD*) theory [1]—extends

ordinary XML elements by incorporation of variables for an enhancement of expressiveness and representation of implicit information into so-called *XML expressions*. Ordinary XML elements—XML expressions without variable—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.

An *XDD description* is a set of *XML clauses* of the form:

$$H \leftarrow B_1, \dots, B_n,$$

where $n \geq 0$, *H* is an XML expression, and B_i is an XML expression, an *XML constraint* or a *set-aggregation*. The XML expression *H* is called the *head*, the set $\{B_1, \dots, B_n\}$ the *body* of the clause. When $n = 0$, i.e., the body is empty, such a clause is referred to as an *XML unit clause*. When clear from the context, a unit clause ($H \leftarrow$) is represented simply by *H*; hence, an XML element or document can be mapped directly onto a *ground XML unit clause*.

An *XML constraint*—useful for defining a restriction on XML expressions, their components or their sets—is a formula of the form $q(a_1, \dots a_n)$, where $n > 0$, *q* is a *constraint predicate* and a_i is an XML expression or a set of XML expressions. The satisfaction (truth or falsity) of a *ground constraint* is predetermined.

A *set-aggregation*—a concept employed in the description of complex queries/operations on sets of XML expressions—is an expression of the form:

```
<xdd:SetOf>
    <xdd:Set>S</xdd:Set>
    <xdd:Constructor>C</xdd:Constructor>
    <xdd:Pattern>P</xdd:Pattern>
</xdd:SetOf>,
```

where *S* denotes a set of XML elements, *C* and *P* are XML expressions which specify that for each XML element, matching the pattern represented by *P*, an XML element represented by *C* will be constructed and included in the set *S*.

Intuitively, given an XDD description *D*, its meaning is the set of all XML elements which are directly described by or are derivable from the unit and non-unit clauses in *D*. Further explanations of XML expressions, clauses and XDD descriptions will be given from a practical point of view by means of examples. Paper [20] gives theoretical details of the theory.

Figure 1 illustrates an XML clause example for modelling a discount rule of an airline company: "*For any person younger than 12 years, his/her fare is at half the standard fare*". The clause can be read as follows: The Person-expression in the clause's body encodes information about a Person, whose URI and age are represented by the *S*-variables $S:personId and $S:age, respectively. The TicketInfo-expression represents information about a ticket owned by such a person such as ticketNo, flight and standardFare. The constraint LT ensures that the age of the person is less than 12, and the constraint Div computes the ticket's childFare which is half of the standardFare. The head of the clause then specifies that a Ticket-expression, which encodes information about ticketNo, ticketOwner, ticketFlight and the computed ticketFare, will be derived.

With the following properties and capabilities, it is readily seen that XDD is a suitable candidate for modeling Semantic Web resources and IWSs:

– direct representation of all XML-based application markup languages including ontologies and Semantic Web resources encoded in certain ontology markup languages, such as RDF(S) and DAML+OIL,
– sufficient expressive power to represent ontology axioms, application constraints and rules in terms of XML clauses,
– well-defined semantics.

```
<Ticket rdf:about=$S:ticketNo>
  <ticketNo>$S:ticketNo</ticketNo>  <ticketOwner rdf:resource=$S:personId/>
  <ticketFlight rdf:resource=$S:flightId/>
  <ticketFare>$S:childFare</ticketFare>
</Ticket>

←  <Person rdf:about=$S:personId>
      <age>$S:age</age>  $E:PersonProperties
   </Person>,

   <TicketInfo>
     <ticketNo>$S:ticketNo</ticketNo>  <ticketOwner rdf:resource=$S:personId/>
     <ticketFlight rdf:resource=$S:flightId/>
     <standardFare>$S:standardFare</standardFare>
   </TicketInfo>,

   LT( <num1>$S:age</num1>, <num2>12</num2>),

   Div( <num>$S:standardFare</num>, <divisor>2</divisor>,
       <result>$S:childFare</result> ).
```

Figure 1. An XML non-unit clause example.

2.2   XML Equivalent Transformation

Employment of *Equivalent Transformation* (*ET*) [2] paradigm leads to computation with XDD by successive, equivalent (semantics-preserving) transformations of a given XDD description into a desired one. Thus, during a computation, the meaning of XDD descriptions are always maintained and the correctness of a solution to a query/problem is guaranteed.

XML Equivalent Transformation (*XET*) [3]—a declarative, higher-order programming language which integrates XDD modeling language and ET computation paradigm—can directly represent both XML unit and non-unit clauses—regarded as an application's data and program, respectively—and also facilitates means for reasoning with and processing of XML documents and queries. It enables declarative programming in a non-deterministic style with a rule-prioritization mechanism; hence it is flexible and efficient for implementation of various types of applications including IWSs.

3. IWS BASIC REQUIREMENTS

The key elements in IWS materialization and a review of existing approaches to such requirements will be discussed.

3.1 Ontology Modeling

Ontologies play an important role in providing an ability to model, represent and exchange formal conceptualization as well as information of particular domains in a precise, machine-understandable form. Standard, predefined domain-specific ontologies can be shared and reused. However, two communicating IWSs— either in the same or different domains—may employ different ontologies, hence an *ontology-mapping* ability is required.

Recently developed ontology markup languages, such as RDF(S), and DAML+OIL, only provide facilities for describing concept- and property-hierarchies, some particular axioms and constraints, and ontology instances. However, they still lack support for representing arbitrary axioms, rules and constraints—an essential feature required by many applications. For example, consider the rule: "*The value of a property ticketFare for a Child is half of that for an Adult.*"; it is simple and common in airline businesses, but yet inexpressible by such languages.

3.2 IWS Description

To enable an IWS to automatically locate and utilize services offered by other IWSs, a mechanism for machine-comprehensible description of available IWSs in terms of their properties, functionalities, constraints and interfaces such as inputs and outputs must be established.

Recent industrial service description languages, such as UDDI, which defines interface of a public service registry, and WSDL, which describes protocol-independent functional description of services, can be used for service discovery, execution, and composition. However, without a support for IWS semantic description, human intervention in a verification of service discovery and composition of complicated services is often required, whence their mechanisms are insufficient for fully automated IWS tasks.

To overcome this, DAML-S [16] is developed with means for automatic service advertisement, discovery, invocation, composition and monitoring. Founded on DAML+OIL, DAML-S is equipped with facilities for describing service capabilities, properties, pre-/post-conditions, and input/output specifications.

3.3   IWS Communication

An IWS communication language assumes a vital role in various IWS service operations, which often require cooperation and interaction among related IWSs. The language should allow transportation of ontologies and data, encoded in XML syntax, over Web standard protocols, e.g., HTTP. However, such a language does not yet exist.

Since an IWS can be regarded as an autonomous, intelligent software agent offering services on the Web, employment of the ACL [14] as communication medium not only allows various IWSs to represent and exchange data, their intentions, believes and ontologies, but also enables more complex interoperations such as service negotiation by applying existing agent technologies. However, ACL is specifically designed to be transported over the Web. Extension of SOAP [5]—a non-proprietary XML-based distributed system protocol, defining a messaging standard for data binding and encoding mechanisms over HTTP—appears to meet this need by ACL capabilities.

```
<daml:Class rdf:ID="Person">
  <rdfs:label>person</rdfs:label>
</daml:Class>
<daml:ObjectProperty rdf:ID="hasChild">
  <rdfs:domain rdf:resource="#Person"/>  <rdfs:range rdf:resource="#Person"/>
</daml:ObjectProperty>
<daml:UnambiguousProperty rdf:ID="hasAirlineMembership">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#AirlineMembership"/>
</daml:UnambiguousProperty>
<daml:UniqueProperty rdf:ID="age">
  <rdfs:domain rdf:resource="#Person"/>
</daml:UniqueProperty>
<daml:ObjectProperty rdf:ID="hasParent">
  <rdfs:domain rdf:resource="#Person"/>  <rdfs:range rdf:resource="#Person"/>
  <daml:inverseOf rdf:resource="#hasChild"/>
</daml:ObjectProperty>
```

(a) An ontology—definitions of the class Person and its properties.

```
<Person rdf:about="Jack">
  <age>52</age>  <hasChild rdf:resource= "#John"/>
  <hasAirlineMembership/>
</Person>
<Person rdf:about="John">
  <age>29</age>  <hasChild rdf:resource="#Jill"/>
  <hasAirlineMembership rdf:resource="#tg9000"/>
</Person>
<Person rdf:about="Jill">
  <age>7</age>  <hasAirlineMembership/>
</Person>
<Person rdf:about="Joe">
  <age>23</age>  <hasAirlineMembership/>
</Person>
<AirlineMembership rdf:about="tg9000"/>
  <memberNo>tg9000</memberNo>
  <issuer rdf:resource="#TG"/>
</AirlineMembership>
```

Jack(52)
hasChild

Airline
Membership
No. tg9000

Joe(23)   John(29)
hasChild

Jill(7)

(b) Ontology instances—a group of persons.
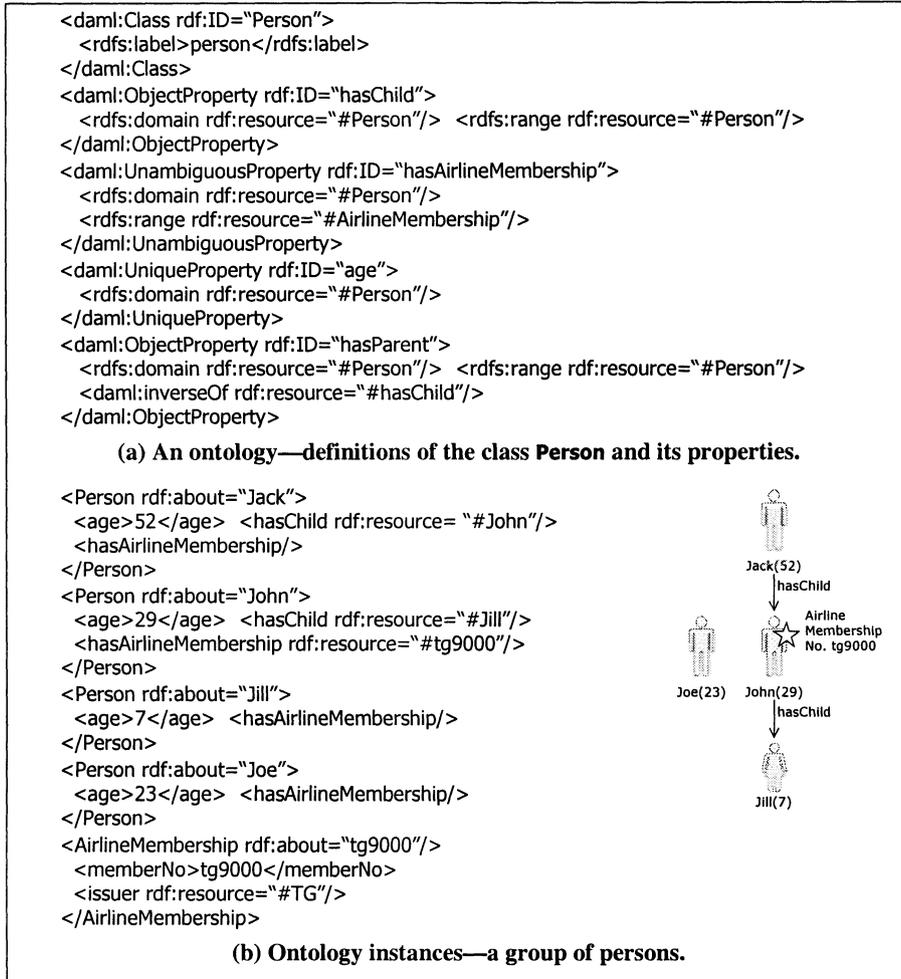
Figure 2. XDD description P_1, representing a domain-model ontology and its instances.

4.   XDD APPROACH TO IWS MODELING

4.1   Domain Ontologies and Contents

A description of domain-specific ontologies and their instances encoded in an ontology language, such as DAML+OIL, becomes immediately an XDD description comprising solely ground XML unit clauses. Consider, for example, the given DAML+OIL ontology and its instances of Figure 2, to be referred to as an XDD description P_1. Moreover, XML non-unit clauses can be employed to define the axiomatic semantics of each ontology modeling primitive which includes a certain notion of implication.

As an illustration, the meaning of DAML+OIL's inverseOf is modeled by the clause of Figure 3; let P_2 denotes an XDD description comprising that clause. Thus, although the properties hasParent of the Persons John and Jill are not explicitly given by P_1, knowing that such a property is an inverseOf hasChild property, one can implicitly such information through the clause of P_2. Figure 4*Figure4* gives the derived information.

Besides sole employment of the predefined modeling constructs, XDD goes beyond the expressiveness of a particular ontology language by yielding facilities for modeling arbitrary rules, axioms, constraints and queries in terms of XML non-unit clauses. Examples of such clauses are given in Section 6.

```
<$N:classB  rdf:about=$S:resourceY>
  $E:instance1Elmt
  <$S:propertyR rdf:resource=$S:resourceX/>
</$N:classB>

      ←        <daml:ObjectProperty rdf:ID=$S:propertyR>
                    <daml:inverseOf rdf:resource=$S:propertyP/>
                    $E:inversePropertyElmt
                </daml:ObjectProperty>,

                <$N:classA  rdf:ID=$S:resourceX>
                    <$S:propertyP rdf:resource=$S:resourceY/>
                    $E:XProperties
                </$N:classA>,

                <$N:classB  rdf:ID=$S:resourceY>
                    $E:YProperties
                </$N:classB>.
```

// If a property R is described as an inverse of a property P, then for any resource X
// the value of a property P of which is a resource Y, one can infer that
// Y also has a property R the value of which is the resource X.

Figure 3. XDD description P_2, modelling the axiomatic semantic of the construct inverseOf.

```
<Person rdf:about="John">
  <age>29</age>
  <hasChild rdf:resource="#Jill"/>
  <hasAirlineMembership rdf:resource="#tg9000"/>
  <hasParent rdf:resource="#Jack"/>
</Person>

<Person rdf:about="Jill">
  <age>7</age>
  <hasAirlineMembership/>
  <hasParent rdf:resource="#John"/>
</Person>
```


Figure 4. Information derived from XDD descriptions P_1 and P_2 of *Figures 2* and *3*.

4.2 IWS Description and Operations

4.2.1 Service Advertisement and Discovery

Before an IWS can offer a service to others, its properties, capabilities, and interfaces need be advertised in service registries which can be

publicly accessed by other IWSs. These service descriptions, represented by certain ServiceProfile in DAML-S or businessEntity and businessService in UDDI, are directly modeled as XML unit clauses, while their additional relationships such as pre-conditions, post-conditions and constraints are expressed by appropriate XML clauses. A search for a particular IWS is expressed as an XML clause; its head specifies the pattern of the returned result and its body describes properties and capabilities of the desired IWS. Such a clause will be evaluated on service registries and will return as its reply a list of qualified IWSs.

According to the following requirements for a description language to be used in service advertisement and discovery outlined in [17]:

a) high degree of flexibility and expressiveness,

b) ability to express semi-structured data,

c) type and subsumption support,

d) constraint support,

it is readily seen that XDD meets all the required features, that is,

a) It is very flexible and expressive, because it can represent all XML-based languages in terms of XML unit clauses and can additionally model rules and axioms in terms of XML non-unit clauses;

b) By appropriate use of variables, XDD can model and query XML data, although their DTDs or schemas are unknown [19], and hence can represent and handle semi-structured data.

c) By means of XML clauses, XDD can model the axiomatic semantics of RDFS's Class, subClassOf and subPropertyOf constructs, as elaborated in [20]. A search for an IWS selling computer devices, for example, would return also those selling printers and scanners.

d) Section 2 has shown that XDD already incorporates the concept of XML constraints for formulation of various types of conditions.

4.2.2   Service Composition

XDD enables service reusability by allowing an IWS to be an aggregation or integration of existing IWSs. A definition of a service composition can be represented by an XML clause with its head

specifying the composition result and its body the composition rules and constraints.

4.2.3 Service Execution

To implement a particular IWS, one can employ not only XET but also a procedural language such as Java. Hence, possible types of service execution, supported by the proposed approach, include:

Information-Providing Service Execution—execution of a request for information, residing in a service provider's information base. Such a service is simply represented by an XML clause with the head describing the query result and the body describing the selection condition. Various kinds of variables can also be used to represent implicit information as well as query parameters. Service invocation is done by parameter instantiation.

Parameterized Service Execution—execution of a request which accepts certain parameters for a service provider to perform some particular task. XML clauses for parameterized service will represent the target-task including its results at the head, and the sub-tasks or sub-goals in the body. Each sub-task can be broken down into more detailed tasks represented by other XML clauses.

Programmable Service Execution—allowing an IWS requester to attach with an execution request a declarative program or sequence of instructions (i.e., program specification in XML clauses) to be executed by other IWSs. These declarative programs, representing service instructions, pre-conditions or post-conditions, are self-descriptive and can be executed without reference to the originator.

4.3 IWS Communication

To establish a support for IWS communication over the Web and agent communication standards, the developed approach employs SOAP to encode *XML-based ACL messages* [13] (either FIPA's ACL or KQML), the contents of which could be, for example, some particular domain-specific ontologies, their instances or service requests. An extended *SOAP Envelope* example which encodes a service request with certain *service input*, sent from an IWS requester to an IWS provider, is given in Section 6. After completion of the

requested service execution, the *service output* is returned. In case of an execution error, a *SOAP Fault* message is returned instead.
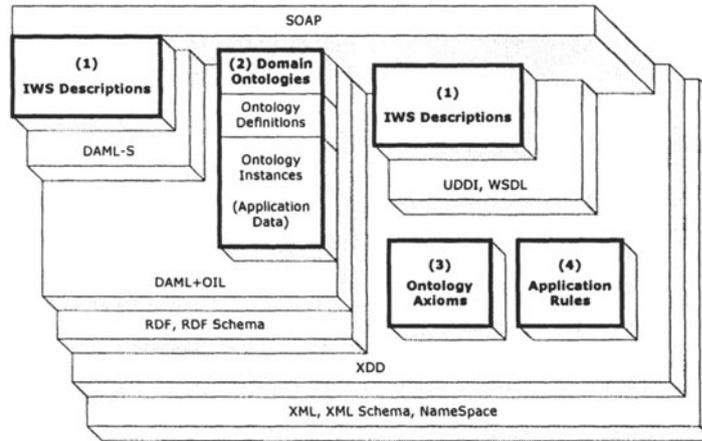

Figure 5. IWS modelling language layer.

Table 1. IWS modeling components.

IWS Modeling Components	Modeled by	Description
1. IWS Descriptions	XDD using DAML-S, UDDI or WSDL	— Service registry information for IWS advertisement and discovery, — Declarative interfaces for IWS service execution.
2. Domain ontologies	XDD using DAML+OIL	Modeling of a domain application in terms of ontology definitions and instances.
3. Ontology axioms	XDD	Definitions of the axiomatic semantics of each ontology modeling primitive.
4. Application rules	XDD	Modeling of business rules & logic, axioms, constraints and queries.
** Encoded in ACL messages, these components can be exchanged among IWSs via SOAP protocol. Moreover, by adopting available ACL technologies, more complex interoperations are enabled.		

To summarize, Figure 5 depicts the role of XDD as a foundation for IWS modeling by employment of open, most recent, prominent technologies—DAML+OIL, DAML-S, UDDI, WSDL and SOAP—as means for dealing with the three IWS requirements. In the figure, each

number in the parentheses indicates a certain IWS modeling component described by *Table 1*. Note that since the proposed framework is general and not restricted to particular languages, besides using the suggested ones, other technologies can also be employed.

5. IWS SYSTEM ARCHITECTURE

Figure 6 depicts the developed architecture which comprises the components:
1. *Client Browser*—a Web browser, which receives user requirements, submits them to an IWS requester and displays IWS requester's results. It can communicate with the IWS requester using an ordinary HTTP request/response scheme.
2. *IWS Requester*—a network application which asks an IWS provider to execute a function based on its needs. By doing this, the IWS requester may provide a user interface (possibly in HTML forms for human-users), submit a provider-lookup query to an IWS broker, and request a service from an IWS provider to complete the user's requirements. IWS requesters communicate with other IWSs by means of SOAP over HTTP.
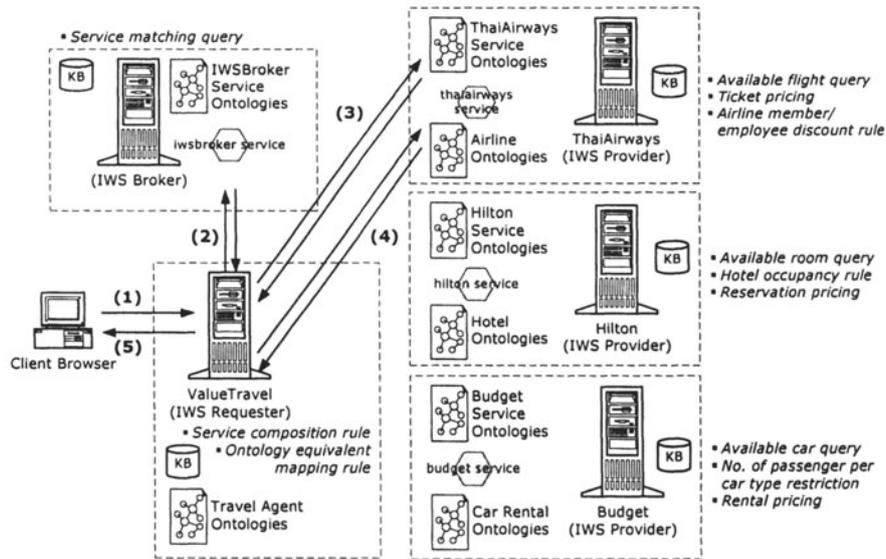
Figure 6. Prototype system scenario.

3. *IWS Provider*—a network application which offers services to IWS requesters. An IWS provider needs to publish its ability to an IWS broker before its service becomes available to other IWS requesters.

4. *IWS Broker*—a network application which performs service-registry lookup for IWS requesters as well as registers and unregisters services provided by IWS providers. Obviously, an IWS broker is a special kind of IWS providers offering well-known services to other IWS components.

6.   PROTOTYPE APPLICATION

A prototype IWS system for B2B travel businesses, comprising an IWS requester (a travel agent), an IWS broker and IWS providers (airline, hotel, and car rental service providers), will be demonstrated. Figure 6 illustrates a possible scenario of service advertisement, discovery, composition, and execution. In the scenario:

```
<daml:Class rdf:ID="ProviderServices">
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="http://kr.cs.ait.ac.th:8010/ontology/Airline.daml#Ticket"/>
    <daml:Class rdf:about="http://kr.cs.ait.ac.th:8010/ontology/Hotel.daml#Reservation"/>
    <daml:Class rdf:about="http://kr.cs.ait.ac.th:8010/ontology/CarRental.daml#Rent"/>
  </daml:unionOf>
</daml:Class>

     ←  <TravelPackage rdf:about=$S:pid>
            <travelingFrom rdf:resource="#bangkok"/>
            <travelingTo rdf:resource="#chiang_mai"/>
            $E:packageElmt
        </TravelPackage>.

// Travel package destination implies a service composition combination: To travel from
// "#bangkok" to "#chiang_mai", the IWS requester needs an Airline's Ticket, Hotel's
// Reservation, and CarRental's Rent.
-------------------------------------------------------------------------------------------
<daml:Class rdf:ID="ProviderServices">
  <daml:unionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="http://kr.cs.ait.ac.th:8010/ontology/CarRental.daml#Rent"/>
  </daml:unionOf>
</daml:Class>

     ←  <TravelPackage rdf:about=$S:pid>
            <travelingFrom rdf:resource=$S:place/>
            <travelingTo rdf:resource=$S:place/>
            $E:packageElmt
        </TravelPackage>.

// A trip in the same area needs only a CarRental's Rent.
```

Figure 7. Service-composition rules.

(1) A group of users—John and Jill—who want to buy a travel package from Bangkok to Chiang Mai, submit their information and traveling requirements to the IWS requester ValueTravel via HTML form. ValueTravel then obtains a list of required services from its *service-composition rules* given by Figure 7, and will discover 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 IWS broker to find a list of potential IWS providers as well as their service-ontology URIs.

(3) 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, and 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 (Figure 8). In response to such a message, ThaiAirways queries its flight database and generates TicketInfo-elements specifying available flight information; Figure 9-a illustrates a sample TicketInfo-element created for Jill. The XML clauses C1 and C2 of Figure 9-b, respectively, model the following pricing and discounting rules of ThaiAirways:

- Child fares for children below 12 years of age are at half-price of the standard adult fare;
- A member passenger and his immediate family will get a 5% discount.

These rules are examples of business rules and logic inexpressible by DAML+OIL and DAML-S. Employing the ontology Figure 2-*a* to model passenger information, ThaiAirways will derive that John is Jill's parent, and because Jill is younger than 12 and her parent has airline membership, it will then yield that Jill is qualified for a child fare with 5% discount (Figure 9-c).

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

In order to complete the user's request, ValueTravel then repeats Steps 2 to 4 on other service providers for hotel reservation and car rental.

(5) Finally, the service composition result—a travel itinerary—is returned to the client browser in a human-readable form (Figure 10).

Although this scenario demonstrates an IWS requester's functionality in serving a human, it can alternatively act as a service intermediary to serve other IWS requesters. In order to enable machine-machine communication in this case, the input and output interface between IWS requester parties will be based on IWS communication language. The prototype system shows various service combinations described by appropriate XML clauses. The example *service-composition rules* of Figure 7, created from different business logic constraints, such as travel sources and destinations, also demonstrate XDD capability to

construct *dynamic ontology definitions* from a collection of assorted classes.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>  (1)
<GetFlightDetails xmlns="urn:thaiairways" xmlns:acl="http://kr.cs.ait.ac.th:8010/acl/acl">
 <acl:aclMessages>  (2)
  <kqmlPerformative>
   <performativeName>ask-if</performativeName>
   <parameters>
   <sender>ValueTravel</sender>
   <receiver>urn:thaiairways</receiver>
   <in-reply-to>callService</in-reply-to>
   <reply-with>GetFlightDetails</reply-with>
   <language>SOAP</language>
   <ontology>http://kr.cs.ait.ac.th:8010/thaiairways/ThaiAirways.daml</ontology>  (4)
   <content>  (5)
    <GetFlightDetails xmlns="http://kr.cs.ait.ac.th:8010/thaiairways/ThaiAirways.daml"
                      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    ...
    <Person rdf:about= "John">
     <age>29</age>  <hasChild rdf:resource="#Jill"/>
     <hasAirlineMembership rdf:resource="#tg9000"/>
    </Person>
    <Person rdf:about="Jill">
     <age>7</age>  <hasAirlineMembership/>
     <hasParent rdf:resource="#John"/>
    </Person>

    ...
    </GetFlightDetails>
   </content>
   </parameters>
  </kqmlPerformative>
 </acl:aclMessages>
</GetFlightDetails>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

//  (1) SOAP body encoding GetFlightDetails service request
//  (2) List of ACL parameters
//  (3) Reference to provider's ontology definition
//  (4) Ontology instances conforming to provider's service input description
//  which contains the parameters: passengerGroup, departingFrom arrivingTo,
//  departingDate, departingTime, ticketClass, ticketType
```

Figure 8. An IWS-SOAP message: GetFlightDetails service request.

```
<TicketInfo>
    <ticketNo>476120</ticketNo>  <ticketOwner rdf:resource="#Jill"/>
    <ticketFlight rdf:resource="#TG101"/>  <standardFare>2000</standardFare>
</TicketInfo>
```

(a) Application data: selected ticket's information for the passenger Jill.

```
C1: <Ticket rdf:about=$S:ticketNo>
        <ticketNo>$S:ticketNo</ticketNo>  <ticketOwner rdf:resource=$S:personId/>
        <ticketFlight rdf:resource=$S:flightId/>  <ticketFare>$S:childFare</ticketFare>
        <ticketDiscount>$S:discount</ticketDiscount>
    </Ticket>

    ←  <TicketInfo>
            <ticketNo>$S:ticketNo</ticketNo>  <ticketOwner rdf:resource=$S:personId/>
            <ticketFlight rdf:resource=$S:flightId/>
            <standardFare>$S:standardFare</standardFare></TicketInfo>,

        <PersonInfo>
            <id>$S:personId</id>  <age>$S:age</age>  <percent>$S:percent</percent>
        </PersonInfo>,

        LT(<num1>$S:age</num1>, <num2>12</num2> ),
        Div(<num>$S:standardFare</num>, <divisor>2</divisor>,
            <result>$S:childFare</result> ),
        Mul(<num>$S:childFare</num>, <multiplier>$S:percent</multiplier>,
            <result>$S:r1</result> ),
        Div(<num>$S:r1</num>, <divisor>100</divisor>, <result>$S:r2</result> ),
        Sub(<num>$S:childFare</num>, <substracter>$S:r2</substracter>,
            <result>$S:discount</result>).
```

// C1 defines main program for ticket price calculation for child fare, which is half-price
// of standard fare. The program will return an ontology instance of Ticket class.

```
C2: <PersonInfo>
        <id>$S:personId</id><age>$S:age</age><percent>5</percent>
    </PersonInfo>

    ←  <Person rdf:about=$S:personId>
            <age>$S:age</age>  <hasAirlineMembership/>
            <hasParent rdf:resource=$S:memberId/>  $E:personElmt</Person>,

        <Person rdf:about=$S:memberId>
            <hasAirlineMembership rdf:resource=$S:memberNo/>  $E:memberElmt
        </Person>.
```

// Clause C2 models discount percentage for a person who has no membership, but is
// eligible for immediate family discount (his/her parent has membership).

(b) Application rules and query: C1 ticket price calculation, C2 membership discount.

```
<Ticket rdf:about=476120>
    <ticketNo>476120</ticketNo>  <ticketOwner rdf:resource="#Jill"/>
    <ticketFlight rdf:resource="#TG101"/>  <ticketFare>1000</ticketFare>
    <ticketDiscount>950</ticketDiscount>  </Ticket>
```

(c) Result from ticket price calculation.

Figure 9. Modelling of application data and rules of ThaiAirways service provider.

Figure 10. HTML output.

In the prototype, the ontologies defined in the Cyc upper ontology knowledge base [8] are simplified and extended in each area-specific domain for each IWS broker, requester, and provider. The prototype demonstrates ontology utilization in:

a) *shared ontologies*—for common terms, e.g., Person, age, Membership,

b) *exchanged ontologies*—for area-specific terms, e.g., Ticket, Flight, and TravelPackage.

The translation from one ontology into another is indispensable for every content communication. For instance, consider ValueTravel's ontology of Figure 11, simply represented as XML unit clauses. Its

intended meaning, which can be modeled by appropriate XML non-unit clauses, yields:

a) The value of the property ticketClass of each instance of the class ValueAirTicket, declared as a subclass of Airline's Ticket, must be economy,

b) airTicketFare is the same property as Airline's ticketFare,

c) airTicketDiscount is the same property as Airline's ticketDiscount.

In the prototype, an XET program for ontology mapping has been developed. Hence, in Step (4) of the given scenario of figure6, after ValueTravel receives a return message about Ticket information from ThaiAirways, it will translate from ThaiAirways' terms into its local ontology as shown in Figure 12.

```
(a) <daml:Class rdf:ID="ValueAirTicket">
     <rdfs:subClassOf
        rdf:resource="http://kr.cs.ait.ac.th:8010/ontology/Airline.daml#Ticket"/>
     <rdfs:subClassOf>
     <daml:Restriction>
      <daml:onProperty
        rdf:resource="http://kr.cs.ait.ac.th:8010/ontology/Airline.daml#ticketClass"/>
      <daml:hasValue
        rdf:resource="http://kr.cs.ait.ac.th:8010/ontology/Airline.daml#economy"/>
     </daml:Restriction>
     </rdfs:subClassOf>
     </daml:Class>

(b) <daml:UniqueProperty rdf:ID="airTicketFare">
     <daml:samePropertyAs
        rdf:resource="http://kr.cs.ait.ac.th:8010/ontology/Airline.daml#ticketFare"/>
     <rdfs:domain rdf:resource="#ValueAirTicket"/>
     </daml:UniqueProperty>

(c) <daml:UniqueProperty rdf:ID="airTicketDiscount">
     <daml:samePropertyAs
        rdf:resource="http://kr.cs.ait.ac.th:8010/ontology/Airline.daml#ticketDiscount"/>
     <rdfs:domain rdf:resource="#ValueAirTicket"/>
     </daml:UniqueProperty>
```

Figure 11. Ontology mapping.

```
<Ticket rdf:about="24670">
   <ticketNo>24670</ticketNo>  <ticketOwner rdf:resource="#Jill"/>
   <ticketFlight rdf:resource="#TG101"/>  <ticketFare>1000</ticketFare>
   <ticketDiscount>950</ticketDiscount>
</Ticket>
// Ticket information in ThaiAirways' ontology.

<ValueAirTicket rdf:about="24670">
   <ticketNo>24670</ticketNo>  <ticketOwner rdf:resource="#Jill"/>
   <ticketFlight rdf:resource="#TG101"/>  <airTicketFare>1000</airTicketFare>
   <airTicketDiscount>950</airTicketDiscount>
</ValueAirTicket>
// Ticket information after translation into ValueTravel's ontology.
```

Figure 12. Ontology mapping result.

7. CONCLUSIONS

By means of XDD modeling language and XET programming language, a declarative approach to IWS systems has been developed. It unifies the representations of IWS descriptions, application ontologies and data as well as application rules, logic and constraints. That is, in addition to direct representation of RDF(S), DAML-family markup languages as well as those industrial standards for Web service descriptions (e.g., UDDI and WSDL) in terms of XML unit clauses, the approach also allows specifications of ontology axioms, rules and constraints in terms of XML non-unit clauses. XDD's expressiveness and well-defined declarative semantics together with XET's computational and query-processing capabilities lead to an IWS system with:

- inference mechanisms,
- self-described executable declarative programs for service pre- and post-conditions,
- query processing for IWS service discovery, service composition and ontology translation.

Employment of SOAP as a communication medium to transport machine-interpretable ACL messages enables semantic communication, sharing and exchange of ontologies, and thus leads to automated collaboration among IWSs. Finally, the implemented prototype system with the capabilities for service discovery,

composition and execution demonstrates the proposed framework's effectiveness and indicates its potential as a solid approach to IWS systems.

REFERENCES

1. Akama, K. (1993). Declarative Semantics of Logic Programs on Parameterized Representation Systems, Advances in Software Science and Technology, 5, 45–63.
2. Akama, K., Shimitsu, T. and Miyamoto, E. (1998). Solving Problems by Equivalent Transformation of Declarative Programs, J. Japanese Society of Artificial Intelligence, 13(6), 944–952 (in Japanese).
3. Anutariya, C., Wuwongse, V., Akama, K. and Wattanapailin, V. (2001). Semantic Web Modeling and Programming with XDD, Proc. 1st Semantic Web Working Symposium CA, 161–180.
4. Berners-Lee, T., Fischetti, M. and Dertouzos, T. M. (1999). Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor, Harpur.
5. Box, D. et al. (2000). Simple Object Access Protocol (SOAP) 1.1, W3C Note [http://www.w3.org/TR/SOAP/].
6. Brickley, D. and Guha, R. V. (2000). RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft [http://www.w3.org/TR/rdf-schema/].
7. Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1 [http://www.w3.org/TR/wsdl].
8. Cyc. (2001). The Upper Cyc Ontology. [http://opencyc.sourceforge.net/daml/cyc.daml].
9. DARPA (2001). Relationship to Existing Industry Web Services Efforts, DAML-S 0.5 [http://www.daml.org/services/daml-s/2001/05/survey-f-release.pdf].
10. FIPA (2001). FIPA ACL Message Structure Specification, Foundation for Intelligent Physical Agents.
11. Finin, T., Labrou, Y. and Mayfield, J. (1997). KQML as an Agent Communication Language. Software Agents, AAAI/MIT Press.
12. Hendler, J. and McGuinness, D. (2000). The DARPA Agent Markup Language. IEEE Intelligent Systems 15(2), 72–73.
13. Jindadamrongwech, S. (2000). An Agent Communication Language using XML Declarative Description, Master's Thesis, Computer Science and Information Management Program, Asian Institute of Technology, Thailand.
14. Labrou, Y. Finin, T. and Peng, Y. (1999). Agent Communication Languages: The Current Landscape, IEEE Intelligent Systems, 14(2), 45–52.
15. Lassila, O. and Swick, R. R. (1999). Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, 1999 [http://www.w3.org/TR/REC-rdf-syntax/]
16. McIlraith, S. A. Son, T. C. and Zeng, H. (2001). Semantic Web Services, IEEE Intelligent Systems, 16(2), 46–53.

17. Trastour, D., Bartolini, C. and Gonzalez-Castillo, J. (2001). A Semantic Web Approach to Service Description for Matchmaking of Services, Proc. 1st Semantic Web Working Symposium, CA.

18. UDDI: The UDDI Technical White Paper, September 2000. [http://www.uddi.org]

19. Wuwongse, V., Akama, K., Anutariya, C. and Nantajeewarawat, E. (2001). A Data Model for XML Databases, Proc. 1st Int'l Conf. Web Intelligence, LNAI 2198, 237–246.

20. Wuwongse, V., Anutariya, C., Akama, K. and Nantajeewarawat, E. (2001). XML Declarative Description (XDD): A Language for the Semantic Web, IEEE Intelligent Systems, 16(3), 54–65.