February 26, 2001
RT0271
Computer Science 19 pages

Research Report

An XML Schema for Agent Interaction Protocols

Yuhichi Nakamura and Gaku Yamamoto

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

IBM Research Report, RT0271, 1998.

An XML Schema for Agent Interaction Protocols

Yuhichi Nakamura and Gaku Yamamoto
IBM Research, Tokyo Research Laboratory
1623-14, Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan
e-mail: nakamury@jp.ibm.com, yamamto@jp.ibm.com

Abstract

Multi-agent systems can be viewed as agent execution environments that host independently developed agents, enabling them to interact with each other. In accordance with this hypothesis, we seek a framework for representing agent interaction protocols, and propose an XML schema. Although existing XML schema activities focus on only strong data formats, our schema also takes account of messages so that agents can communicate with each other by expressing intentions such as requests and responses. In addition, our schema incorporates with a concept of sessions so as to define the order of messages, taking account of finite state machine formalism of Knowledge Query Manipulation Language (KQML). Accordingly, an interaction protocol defined by our schema can be combined with KQML in a natural way.

1. Introduction

Software agents can be viewed as programs, each of which is situated within and part of an environment, senses changes of that environment, and acts on it so as to affect what it will sense in the future [1]. Such an environment can contain more than one agent, and is therefore called a *multi-agent system*. Inherently, agents are self-serving in the sense that they behave in pursuit of their own agendas, and can be developed independently by different programmers who do not have a common goal. Consequently, to enable interactions between unfamiliar agents, we need an *interaction protocol,* which is a set of rules for communications. How to represent such a protocol is a central issue in the agent research area.

Basically, an interaction protocol defines data formats for information exchange, and such data formats are often called *metadata*. The importance of metadata has been more widely recognized recently because of the exponential increase in the number of

World Wide Web (WWW) servers on the Internet. XML (eXtensible Markup Language) [2] is considered as a standard for information exchange, and provides a means for defining metadata. Since we can expect that information on various sources, including WWW servers, will be provided in XML, it is valuable to study how to represent interaction protocols on the basis of XML.

In this paper, we propose an XML schema for representing agent interaction protocols. With XML Document Type Definition (DTD) syntax, it seems possible to define interaction protocols. However, it is difficult, because there are no data types such as Integer, Long and Date, and no inheritance hierarchy. Consequently, some XML-based schemas are being designed: Resource Description Framework (RDF) [3], XML-Data [4], and Document Content Definition (DCD) [5]. Following this direction, we designed yet another XML schema specialized for agent interactions.

Although existing XML schema activities focus on only strong data formats, our XML schema takes account of messages so that agents can communicate with each other by expressing intentions such as requests and responses. In addition, our schema incorporates a concept of sessions so as to define an order of messages, taking account of finite-state machine formalism of Knowledge Query Manipulation Language (KQML) [6]. Accordingly, an interaction protocol defined by our schema can be combined with KQML in a natural way.

The idea described here is derived from our experience in the development of a real application. We have developed an electronic marketplace framework, called e-Marketplace, which has a system construct for defining interaction protocols [7]. A travel information service called TabiCan [8] has been developed on top of the framework, and we are running it as a business. Although interaction protocols are currently represented in a proprietary format, they are all translated into our new XML schema representation.

In Section 2, we give an example of a multi-agent system, to show the key entities for our XML schema. Section 3 overviews the XML schema, and describes representation constructs with examples. Section 4 describes the implementation of e-Marketplace, focusing on the use of interaction protocols. Section 5 compares the e-Marketplace framework with other agent systems in terms of protocols, and discusses the relation between our XML Schema and KQML. Finally, Section 6 describes the current status of our work, and outlines our future plans.

2. A Typical Application Scenario

Here, to illustrate the kind of multi-agent system we are addressing, we describe an electronic marketplace application, and identify key entities that serve as a basis for considering interaction protocols. The scenario described here is a revised version of TabiCan [8], which is a commercial travel service that we developed. Although the application incorporates mobile agents, mobility is not crucial to the discussion in this paper. Rather we intend to describe a situation in which unfamiliar agents interact with each other.

Figure 1 shows an overview of an electronic marketplace for travel products such as package tours*, and car rental services. A consumer agent issues a query about package tours, specifying conditions such as "The destination is Honolulu, the airline should be Japan Airlines (JAL), and the price should be equal to or less than $600." The query is then delivered to merchant agents, each of which returns product information based on its own selling policy. For example, Shop A returns an exactly matched product, Shop B returns a product that does not meet the consumer's requirement but is cheaper, and Shop C returns information on car rental services. In addition, there is a bulletin board for specially priced products. Assume that a consumer agent has posted his requirement to the bulletin board. As soon as a shop agent offers a product that meets the consumer's requirement, the consumer agent will receive a notice from the bulletin board. One possible activity of a consumer agent is to send e-mail to its owner to notify him or her that a product has become available.

---

*A package tour includes hotel stays and air travels. Products of this kind are reasonably priced, and very common in Japan.

Multiple marketplaces can be linked by means of a special agent, namely, an advertising agent (see Figure 1). The advertising agent is created in one marketplace and dispatched to another marketplace. When a consumer agent issues a query, the advertising agent introduces its home marketplace to the consumer agent, showing where it is located (e.g., in the form of a URL). Then, the consumer agent may go to that marketplace, issue a query, and obtain further information on products.

The scenario here suggests that there are two kinds of key entities: agents and market resources. Consumer, merchant, and advertising agents are obviously examples of one kind of key entity, namely, *agents*. On the other hand, a bulletin board is a different kind of entity from ordinary agents. It stays in a particular marketplace and provides

agents with a service: we call entities of this type *market resources*. Our schema for interaction protocols is designed on the basis of interactions between these two types of entities.

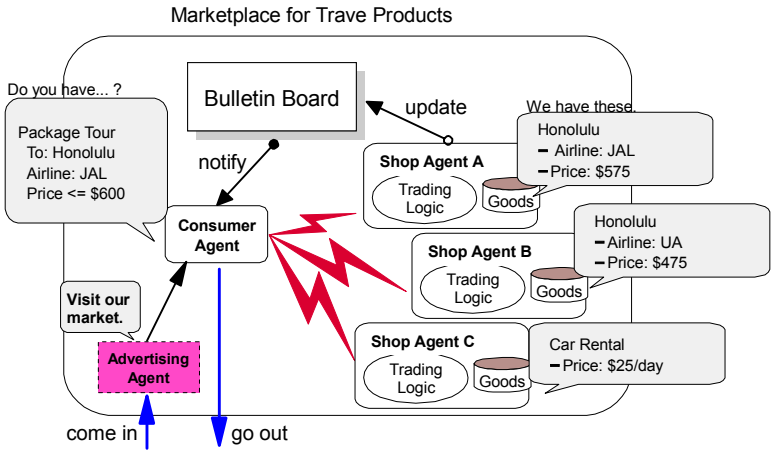

Figure 1. An Electronic Marketplace for Travel Products

3. Representation Schema

3.1 XML Schema Issues

Since we expect that information sources, including WWW servers, will support XML in the near future, we will start our discussion by considering how to represent interaction protocols on the basis of XML. With XML DTDs, it seems possible to define an agent interaction protocol, but it is not enough in the sense that there are no data types such as Integer, Long and Date, and no inheritance hierarchy. Therefore, there are some working groups and working drafts considering "schema" issues. However, these activities focus on only stronger *data* validation. For example, Resource Description Framework (RDF) [3] focuses on how to represent semantic networks; XML-Data [4] takes account of data types such as Integer, Long, and Date; Document Content Description (DCD) [5] is a simplification of RDF that takes account of the data types of XML-Data.

We clarify what is missing in currently proposed schemas, taking our travel example. For interaction between consumer and merchant agents, we obviously need to define data a format for package tours. For example, package tours should be characterized by properties such as "point of departure," "destination," "departure date," and "price." With XML-Data, we can define this kind of metadata, even specifying the data types of properties, such as String, Integer and Date. However, agents cannot interact solely on

the basis of product information. Rather, it is important to notice that agents communicate with each other by expressing *intentions* such as requests, responses, and advertisements, in addition to product information. For example, a consumer agent "requests" package tours, a merchant "recommends" a consumer collection of package tours, and an advertising agent "advertises" its home marketplace. Since such intentions are not considered in existing schemas, we introduce *messages* to represent intentions explicitly, and let message parameters contain data such as product information and addresses of marketplaces. Furthermore, when agents exchange messages, the order of messages is also important. Our representation schema also takes account of *sessions* that define the sequence of messages.

3.2 Overview of the Schema

In Section 2, we identified two key entities for multi-agent systems: *agents* and *market resources*. Therefore, when designing a representation schema for agent interactions, we classify interactions into two kinds: agent-agent interactions and agent-resource interactions. Note that resource-resource interactions obviously do not exist. Here, we give an overview of the schema, identifying constructs used for representing interaction protocols.

Regarding agent-agent interactions, we pay particular attention to the distinction between *data* and *service*, and identify the layers for protocol representation, namely, the item, message, and session layers (Figure 2). In the item layer, we are concerned only with data. The representation construct here is called an *item*, which is a unit of information, and can be considered as an abstraction of "product."

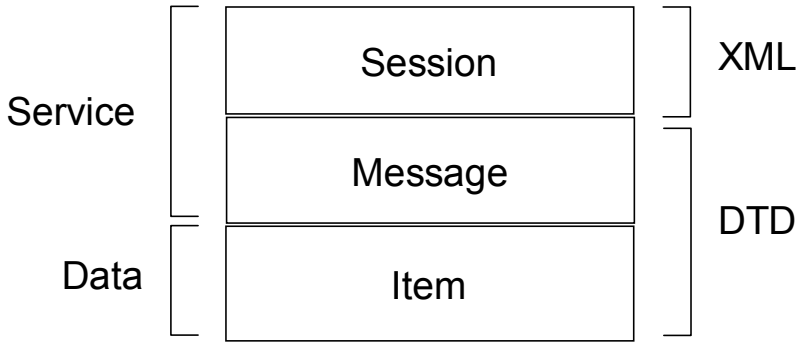

Figure 2. Layers for Representing Interaction Protocols

Representing services is not as straightforward as representing data. A service can be viewed as a job that is requested by an agent, and is done for the requester by another

agent. This indicates that there exist a service requester and a service provider, and that one communicates its intention to another. *Messages* are introduced to represent such intentions regarding services. For example, we can define a message for a request to search for items, and another message for information passing to return items.

Obviously, the request message and the information-passing message depend on each other in the sense that the recipient of the first message has to send the second message to the request sender. Such a relationship between or among messages is called a *session*, and should be explicitly represented. We can restate that a session defines a rule for exchanging messages for a particular service (see Figure 2).

From the representation perspective, it is better to represent items and messages in XML, because they are entities exchanged between agents. Therefore, metadata for items and messages should be defined in DTD, or in a representation that can be converted into DTD as in XML-Data. As will be explained later, we take the second approach; that is, we provide a means for describing metadata by using XML itself. On the other hand, sessions are used for identifying how interactions proceed within an agent system. Since information on sessions is not exchanged between agents, we represent sessions in XML, but do not convert them into DTD.

Regarding agent-resource interaction, we assume that agents and market resources exchange information on items as in agent-agent interaction. Although information exchange is performed via messages, we do not take the session layer into consideration. We consider resources as parts of an environment, and an agent acts on them so as to affect what it will sense in the future. Although there might be some relationship between an action and sensed data, its causality is not sufficiently clear to be represented. Therefore, we do not represent sessions for agent-resource interaction, but only take account of the item and message layers.

We have identified three types of messages for agent-resource interaction: *update* messages, *look-up* messages and *events*. An update message is sent from an agent to a resource when the agent wants to update the contents of the resource. Messaging is performed asynchronously, and therefore the actual update will take place later. A look-up message is sent from an agent to a resource when the agent wants to look up the contents of the resource. Messaging is performed synchronously, so that the agent can obtain the result simultaneously. An event is sent from a resource to an agent when the contents of the resource are changed. A typical situation is that in which a resource

receives an update message from an agent, and the resource then sends events to other agents to notify them of a change in the resource.

3.3 Examples

Here, we describe the actual representation of agent interaction protocols, taking the travel information example described in Section 2. In Section 3.2, we described two categories for interaction: agent-agent interaction and agent-resource interaction. In this section, we first describe representation of item, message and session respectively for agent-agent interaction. We then overviews messages for agent-resource interactions, because their representation is similar to that of messages for agent-agent interactions.

Item

Metadata for items is defined with a name and a set of properties. As in a style such as XML-Data, data types and inheritance of properties are introduced. The predefined data types are String, Integer, Date, and so on. In addition to these basic types, Collection and Enumeration are also supported. Metadata for items is like *class* in Java, and inheritance of properties is also provided. Note that the most abstract item is called *Thing*, which is like Object in Java. As an example, in Figure 3 we show an item definition for package tour, namely, "Tour."

```
  <ItemDef name="Tour" super="TravelGoods">
      <PropertyDef name="ProductName" valueType="String" />
      <PropertyDef name="TourID" valueType="String" />
.....
      <PropertyDef name="Destinations" valueType="Collection">
          <ValueType name="String" />
      </PropertyDef>
.....
      <PropertyDef name="LeavingDate" valueType="Date" />
      <PropertyDef name="Price" valueType="Integer" />
.....
      <PropertyDef name="Status" valueType="Enumeration">
          <EnumerationDef valueType="String">
              <StringDef value="E" />
              <StringDef value="C" />
              <StringDef value="F" />
          </EnumerationDef>
      </PropertyDef>
....
  </ ItemDef>
```

Figure 3. Item Definition

In Figure 3, the first line indicates the beginning of an item definition named "Tour," specifying TravelGoods as its superclass. The property definitions as follows:. The

ProductName and TourID properties are defined as String, and the Destinations property is defined as Collection of Strings, because some package tours have multiple destination cities. The Status property is defined as Enumeration of String, and its property value should be one of three values: E (empty), C (almost full), or F (full).

In connection with item definition, we provide *queries* for specifying constraints on items. A query is represented by a class name of items and a condition body. For example, a query about Tour can be described as follows:

```
<query target="Tour">
    <condition>
        <and>
          <Equal>
              <left> destination </left>
              <right> 'HNL' </right>
          <Equal>
         <LessThanOrEqual>
            <left> price </left>
            <right> 600 </right>
         </LessThanOrEqual>
        </and>
    </condition>
</query>
```

This is a query on tours, whose condition is "The destination is Honolulu (HNL), and the price should be equal to or less than $600." In the description, “Equal” and “GreaterThanOrEqual” are both operators, and left and right indicate their parameters. “destination” and “price” are properties of the “Tour” class, and ‘HNL’ and 600 are values. A query is embedded as a parameter in a message, which is explained below.

Message

Metadata for messages is defined with a name and a set of parameters. Messages for agent-agent interactions are called *utterances*, in order to distinguish them from messages for agent-resource interaction. Figure 4 shows examples of utterance definitions used in TabiCan. The first utterance, “RequestTravelGoods,” is used when a consumer agent issues a requirement to a merchant agent. The first element specifies the name of the utterance, and parameter definitions follow. The first parameter, “Requirements,” is defined as Query, and the query on Tour described in the previous paragraph is substituted to this parameter. The second parameter indicates how may items the sender — in this case a consumer agent — wants for the query.

The second utterance, “ProvideExactGoods,” is used when a merchant agent returns exactly matched items to the consumer agent that issued the first message. Its

parameter, “ExactGoods,” is defined as Collection of Things. Since Thing is the most abstract category, this parameter can contain items of any classes. The third utterance, “RecommendGoods,” is used when a merchant agent recommends items that do not match the consumer’s request, but meet the selling policy of the merchant.

The last utterance definition, “AdvertiseMarket,” is used when an advertising agent in Figure 1 gives its home address to a consumer agent. The data type for its parameter is defined as URL, because the advertising agent gives its home address as a URL.

```
<UtteranceDef name="RequestTravelGoods">
    <ParameterDef name="Requirements" valueType="Query" />
    <ParameterDef name="MaxSize" valueType="Integer" />
  </UtteranceDef>
  <UtteranceDef name="ProvideExactGoods">
    <ParameterDef name="ExactGoods" valueType="Collection">
      <ValueType name="Thing" />
    </ParameterDef>
  </UtteranceDef>
  <UtteranceDef name="RecommendGoods">
    <ParameterDef name="RecommendedGoods" valueType="Collection">
      <ValueType name="Thing" />
    </ParameterDef>
  </UtteranceDef>
  <UtteranceDef name="AdvertiseMarket">
    <ParameterDef name="NewMarket" valueType="URL" />
  </UtteranceDef>
```

Figure 4. Definition of Utterances

Session

In addition to utterance definitions, we define *sessions*, each of which specifies a sequence or sequences of utterances. For example, if a consumer agent issues a query with RequestTravelGoods, a merchant agent has to return results with ProvideExactGoods. This kind of information is defined in a session. Taking account of KQML, we define sessions based on a finite-state machine (FSM) model, and represent them in XML. The relation of our schema to KQML is detailed in the Discussion section.

Figure 5 shows an FSM, which defines the order of utterances between agents. State-1 is an initial state, State-2 and State-3 are intermediate states, and State-100 is a final state. Links indicates transitions, each of which is characterized by an utterance and schematic variables indicating sender and receiver agents. In a typical interaction in TabiCan, the state transits via link1, link2, and link4. More specifically, a consumer

agent sends a request to a merchant agent with RequestTravelGoods, and the merchant first returns exactly matched items with ProvideExactGoods, then returns recommended items with RecommendGoods. On the other hand, transition via link1 and link3 defines an interaction between a consumer agent and an advertising agent as in Figure 1; that is, as soon as an advertising agent receives a RequestTravelGoods message from a consumer agent, it returns its home URL to the consumer agent with AdvertiseMarket.

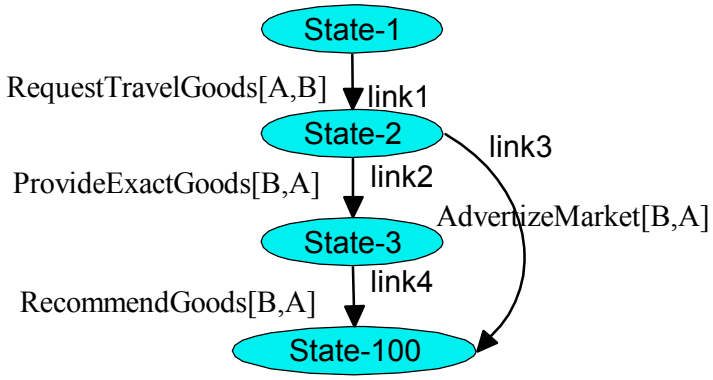

Figure 5. Finite State Machine for a Session

The FSM can be represented in XML as shown in Figure 6. The first declaration indicates that the session to be defined is called "request." Then, states are definitions with their names and types. State type indicates whether the state is initial, final, or intermediate (since this is a default value, it is not explicitly shown in the figure). Finally, transitions from link1 to link4 in Figure 5 are defined. A transition is characterized by four attributes: “name” indicates the name of a link, “utterance” indicates the name of the utterance, “prev” and “next” are the source and destination states, and “sender” and “receiver” are schematic variables indicating sender and receiver agents.

Constraints on utterance parameters can also be defined in a session. In the constraint declaration in Figure 6, the attribute "transition" indicates when the constraint should be checked. In this case, it will be checked at the transition "link2." The contents of the constraint tags constitute the actual description of a constraint. This indicates that the MaxSize value for a RequestTravelGoods message should be greater than or equal to the number of elements in ExactGoods for a ProvideExactGoods message. Note that message names are omitted in actual descriptions such as link1.MaxSize, because a message can be identified by specifying a particular transition. GreaterThanOrEqual is an operator for comparing two values, left and right indicate its parameters, and size is a function for calculating the number of elements contained in the collection.

```
<session name="request">
<state name=”State-1” type=“initial” />
<state name=”State-2” />
<state name=”State-3” />
<state name=”State-100” type=“final” />
<transition name="link1"
    utterance="RequestTravelGoods"
    prev=”State-1" next="State-2"
    sender="A" receiver="B" />
<transition name="link2"
    utterance="ProvideExactGoods"
    prev="State-2" next="State-3"
    sender="B" receiver="A" />
<transition name="link3"
    utterance="AdvertizeMarket"
    prev="State-2" next="State-100"
    sender="B" receiver="A"
    type="terminal" />
<transition name="link4"
    utterance="RecommendGoods"
    prev="State-3" next="State-100"
    sender="B" receiver="A" />
<constraint transition="link2">
    <GreaterThanOrEqual>
        <left> link1.MaxSize </left>
        <right>
            <size> link2.ExactGoods </size>
        </right>
    </GreaterThanOrEqual>
</constraint>
</session>
```

Figure 6. Representation of a Session

Agent-Resource Interaction

In addition to a schema for agent-agent interaction, we also provide a schema for representing agent-resource interaction. Figure 7 shows examples of messages. UpdateData is an update message, getCityNameByCode is a look-up message, and NotifyUpdated is an event. Its description format is almost the same as that of utterances for agent-agent interaction: The only difference is that a look-up message has a return value indicated by a "ReturnDef" element.

```
<UpdateDef name="UpdateData">
  <ParameterDef name="Item" valueType="Thing" />
</UpdateDef>
<LookUpDef name="getCityNameByCode">
  <ReturnDef valueType="String" />
  <ParameterDef name="Code" valueType="String" />
</LookUpDef>
<EventDef name="NotifyUpdated">
  <Parameter name="Item" />
</EventDef>
```

Figure 7. Message Definition for Agent-Resource Interaction

4. Implementation of e-Marketplace

We have already developed an electronic marketplace framework, called e-Marketplace, incorporating the idea of agent interaction protocols (see Figure 8). The framework was developed on top of Aglets Workbench [9], a Java-based mobile agent system developed by our team, and applications such TabiCan have been developed on top of e-Marketplace. Here, we overview the e-Marketplace framework, and describe some class methods to show how interaction protocols are used in the framework.

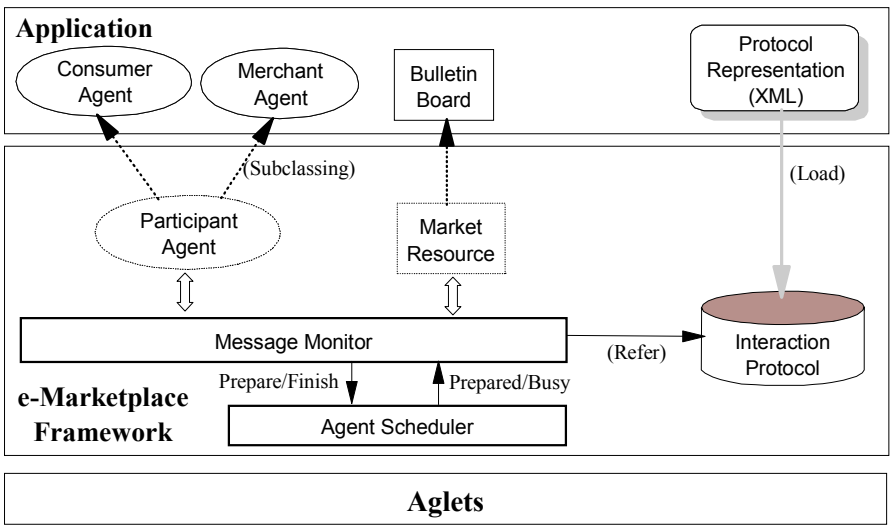

Figure 8. Architecture of e-Marketplace Framework

Let us examine the architecture in Figure 8. In the framework, abstract classes for participant agents and market resources are provided, and application developers implement concrete classes such as a consumer agent, a merchant agent, and a bulletin board, subclassing the framework classes. On the other hand, a (human) market

administrator prepares an XML file or files defining an interaction protocol, and loads it or them into the framework. While agents interact with each other via messages, the interactions should be carried out in accordance with the interaction protocol. Message Monitor monitors every message between agents to ensure that it does not violate the protocol. Since Message Monitor can identify which agents are taking part in a particular interaction, more specifically a session, it can obtain information on agent usage. Agent Scheduler receives such usage information, and activates or deactivates agents to avoid overloading the server. In summary, the interaction protocol is used for agent scheduling, and helps to avoid system overload. The overload issue is one that we encountered in the development of TabiCan, because thousands of consumer agents can be created on the TabiCan server.

The framework contains a class library for representing interaction protocols. Although a protocol described in an XML file can be loaded, internal representation is currently not in DTD, but a proprietary format. The class library also provides a means for representing messages that are communicated between agents or between an agent and a resource. Although the messages can theoretically be represented in XML, that capability is not implemented in the current framework. The current classes for representing messages are proprietary, but they are comprehensive because they reflect the structure of messages well. In the next version of the framework, we will provide three ways of representing messages: the current proprietary representation, XML which is written in strings, and Document Object Model (DOM), which is an object model for XML [2].

Next, we explain ParticipantAgent and MarketResource classes, which are central classes for application development. ParticipantAgent provides the following methods for agent-agent interaction:

`say`: Used when an agent sends a message to another.

`shout`: Used when an agent multicasts a message. Message delivery is carried out by MessageMonitor,

`advertise`: Used when an agent registers what kinds of shouted messages the agent wants to receive. MessageMonitor manages the registered information.

`handleSay`: Callback method for receiving a message. The body of method is implemented in each agent.

ParticipantAgent also provides the following methods for agent-resource interaction:

`lookUpResource`: Used when an agent looks up the contents of a resource.

`updateResource`: Used when an agent updates a resource.

`subscribe`: Used when an agent registers what kinds of event the agent wants to receive. MessageMonitor manages the registered information.

`handleResourceEvent`: Callback method for receiving an event. The body of the method is implemented in each agent.

MarketResource provides the following methods:

`handleLookUp`: Returns a result in response to a look-up message from an agent. The body of the method is implemented in each resource.

`handleUpdate`: Updates its own contents in response to an update message from an agent. The body of the method is implemented in each resource.

`sendEvent`: Used when an resource sends an event. MessageMonitor delivers the event to agents in accordance with registered information.

MessageMonitor not only checks messages between agents and market resources in accordance with the interaction protocol, but also dispatches messages to agents that have been registered. This facility is not provided by the interaction protocol itself, but is related to communication protocols. When developing this framework, we took account of KQML facilitator messages. The relation to KQML is described in the discussion session in more detail.

5. Discussion

5.1 Comparison with Other Marketplace Systems

There are various types of multi-agent systems. Obviously, distributed artificial intelligence (DAI) is an area for multi-agent systems. However, it is quite different from our system in the sense that DAI systems solve a single problem by dividing it into several small problems and solving them on different machines. Our goal here is not to compare our system with a broad range of multi-agent systems; rather, we specify a narrow range of multi-agent systems to make the discussion fruitful and concrete. Our assumption regarding multi-agent systems is as follows: “Agents that are independently developed are situated within an environment where they interact with each other.” In addition, we take account only of commercial or public systems, and therefore prototype systems and experimental systems are beyond the scope of our discussion. Since systems that meet our requirements can be found in the electronic marketplace area [10], we take some examples from this area to compare with ours.

Auction systems should be considered, because auctions have rapidly achieved enormous popularity on the Internet and because such systems can provide a function for

developing agents that interact with them. AuctionBot [11] operated by Michigan University is an online auction site, and provides developers with explicit support for developing their own agents. It can be viewed as a central entity of a multi-agent system in the sense that various agents can be developed and accessed. However, agents interact only with AuctionBot, and not interact with each other. Furthermore, in comparison with our system, AuctionBot incorporates a particular interaction protocol for performing auctions, and was developed to achieve good performance in the auction protocol.

Unlike AuctionBot, Kasbah [12, 13], which was developed and is experimented by MIT, provides an environment in which agents can directly interact with each other. It focuses on negotiation between agents; for example, a buying agent tries to reduce the price quoted by a selling agent. From the perspective of interaction protocols, it incorporates a fixed set of messages -- what-is-item, what-is-price, and accept-offer – that should be hard-coded in each agent. Furthermore, although there must be relationships between messages like state transitions in our sessions, they are not explicitly represented. As a result, even if a malicious agent conveys the given messages in the sequence, Kasbah cannot identify the agent.

In summary, there are some marketplace applications on the Internet, but to the best of our knowledge, they merely incorporate a fixed particular protocol. The protocols are not represented explicitly; in particular, sequences of messages are not represented in spite of their importance. On the other hand, we provide a schema for representing various kinds of protocols. Once a protocol is represented, our marketplace system can detect wrong behaviors of agents, and can even achieve good performance in hosting a huge number of agents (See [14]).

5.2 Relation to KQML

The finite state-machine (FSM) model in the session representation comes from KQML, and the message-handling mechanism in the e-Marketplace framework is also based on KQML facilitator messages. Here, we relate our XML schema and e-Marketplace framework to KQML more precisely.

We take a layer model proposed in a KQML paper [6], which identifies the following three layers:

1. A *transport layer* is the actual transport mechanism for transferring data from one machine to another, such as TCP or HTTP.

2. A *communication layer* is the medium through which the attitudes regarding the contents of exchange are communicated. KQML is a language that defines an attitudes such as an assertion, a request, or some form of query.
3. An *interaction layer* is a definition of contents to be exchanged, which specifies vocabulary and its semantics. Our XML schema was designed to address this layer.

KQML was designed on the basis of the speech act theory [15], which is that an utterance can be viewed as an "act", and may cause another "act" of the utterance receiver. In accordance with this idea, KQML defines a set of messages, called *performatives*, to explicitly indicate the attitude of the agent who utters. Labrou defined a detailed semantics for KQML, specifying the mental states of agents that exchange messages [6]. Apart from the semantics, he also provided a pragmatic approach to implementing KQML agents by introducing the finite-state machine (FSMs) formalism. In FSMs, details of KQML semantics are omitted, but elements for implementation are only extracted. We think that although the KQML semantics is too difficult for ordinary (non-AI) developers, the FSM formalism is acceptable for them. Therefore, we adopted this formalism to represent sessions as FSMs.

Another advantage of KQML is that it provides a rich set of facilitation messages. For example, the *recruit-all* message enables agents to multicast messages, *advertise* message enables agents to register what kind of messages they want to receive, and so on. Although facilitation messages were introduced as an extension of speech act messages, fortunately they work even with the FSM formalism. Actually, while we represent sessions as FSMs, we were able to implement MessageMonitor (see Figure 8) as a built-in facilitator that supports "recruit-all" ("shout" in our terminology) and "advertise" messages.

Although KQML provides a good basis for developing multi-agent systems, it is used in very few real (commercial) applications as far as we know. We think there are two reasons for this. First, it is difficult for ordinary (non-AI) developers to understand the KQML semantics based on the speech act theory. What is worse, some developers cannot understand even why the communication layer is necessary. We believe that the FSM formalism is more comprehensive, and solves such problems to some extent.

Second, there is no adequate means of representing interaction protocols. When KQML is discussed in the literature, KIF (Knowledge Interchange Format) [16] and Ontolingua [17] are often introduced as languages for the interaction layer. However,

since these languages are based on predicate logic, non-AI developers cannot use them, or at least never try to use them. We have a further problem regarding these languages in the development of multi-agent systems: Since we assume that agents are developed independently, knowledge sharing among agents is an important issue. Once we adopt a language with strong high expressive power, we will soon face a knowledge-sharing problem that has been struggled with in the ontology area [18]. In our approach, we did not try to solve the big problem. Rather, we extended currently accepted concepts as little as possible. More specifically, we took a metadata concept that is broadly accepted in the database area, and an Interface Definition Language (IDL) concept that is broadly accepted in the distributed objects area, and organized them into an XML schema, incorporating the FSM formalism from KQML. Obviously, even if we provide a simple representation schema, we still have a knowledge-sharing issue. However, our approach is reasonable, because the sharing issue here is just an *interface-sharing* issue that developers always encounter in ordinary system development.

6. Conclusions

In this paper, we have proposed an XML schema for representing agent interaction protocols. We have already been working on the protocol issue for more than one year, and have developed commercial applications. In actual development, we created a middleware application, called the e-Marketplace framework, and directed the development of applications on top of the framework. One crucial problem was that it was difficult for the programmers to define interaction protocols. However, once an interaction protocol was defined, they were able to develop agent programs as in an ordinary development.

How can we benefit from XML in our work? Obviously, one benefit is comprehensive representation. We can expect that developers who are familiar to XML will be able to learn our XML schema easily. The second, and more important, benefit is that metadata is being recognized as a key concept for combining various systems. So far, the importance of interaction protocols has not been recognized outside the agent community. However, there is no need for us to discuss why protocols are important, because XML has already been broadly accepted. We want to take this opportunity to propose an XML schema that will serve as a basis for work on standardization.

Currently, we are working on two research issues: the schema issue addressed in this paper, and a performance issue. As explained in Section 4, our framework implementation is a special usage of KQML messages; in other words, MessageMonitor

is viewed as a built-in facilitator. This architecture derives from our second goal which is to achieve high performance. In other words, adopting KQML whole makes it difficult to optimize performance. Furthermore, we are not sure whether all KQML messages are needed for real applications. Therefore, we are going to take other messages in KQML step by step, clarifying the real *need* for application development.

Acknowledgements

We would like to thanks our project manager, Mr. Kazuya Kosaka, and the other members of the Aglets team at IBM Tokyo Research Laboratory, who have all contributed to this work. We also thank Mr. Michael McDonald for checking the wording of this paper.

References

1. J. Bradshow (ed.): Software Agents, AAAI Press / MIT Press, Cambridge, Massachusetts, 1997.

2. Extensible Markup Language (XML). http://w3c.org/TR/1998/REC-xml-19980210

3. Resource Description Framework (RDF) Schema Specification http://www.w3.org/TR/WD-rdf-schema/

4. XML-Data. http://www.w3.org/TR/1998/NOTE-XML-data

5. Document Content Description for XML. http://www.w3.org/TR/NOTE-dcd

6. Y. Labrou: Semantics for an Agent Communication Language. Doctoral Dissertation, University of Maryland Graduate School Baltimore, 1996.

7. Y. Nakamura and G. Yamamoto: Aglets-Based e-Marketplace: Concept, Architecture and Applications. IBM Research, Tokyo Research Laboratory, Research Report, RT-0253, 1998. (http://aglets.trl.ibm.co.jp/RT0253/RT0253.html)

8. TabiCan. http://www.tabican.ne.jp/ (Japanese)

9. Aglets Workbench. http://www.trl.co.jp/aglets/

10. R. Guttman, A. Moukas, and P. Maes: Agent-Mediated Electronic Commerce: A Survey. To appear, Knowledge Engineering Review Journal, June 1998.

11. P. Wurman, M. P. Wellman and W. Walsh: The Michigan Internet AuctionBot: A

Configurable Auction Sever for Human and Software Agents, in Proc. of the Second International Conference on Autonomous Agents (Agent ’98), Minneapolis, MN, USA, 1998.

12. A. Chavez and P. Maes: Kasbah: An Agent Marketplace for Buying and Selling Goods. Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM '96), pp.75-90. London, UK. April 1996.

13. R. Guttman, P. Maes, A. Chavez and D. Dreilinger: Results from a Multi-Agent Electronic Marketplace Experiment. Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM '97), pp.75-90. London, UK. April 1997.

14. G. Yamamoto and Y. Nakamura: Architecture and Performance Evaluation of a Massive Multi-Agent System. IBM Research, Tokyo Research Laboratory, Research Report, RT-0272, 1998.

15. P. Cohen and H. Levesque: Intention is choice with commitment. Artificial Intelligence, Vol. 42, pp. 213-261, 1990.

16. KIF Knowledge Interchange Format. http://www.csee.umbc.edu/agents/kse/kif/

17. Ontolingua. http://www.cs.umbc.edu/agents/kse/ontology

18. Knowledge Sharing Effort. http://www.cs.umbc.edu/agents/kse.shtml