XML COMMUNICATING AGENTS IN THE RULE BASED DISTRIBUTED EVENT SYSTEM (RUBDES)

Ozgur Koray SAHINGOZ
Air Force Academy
Computer Engineering Department
Yesilyurt, Istanbul, TURKEY
sahingoz@hho.edu.tr

Nadia ERDOGAN
Istanbul Technical University
Electrical-Electronics Faculty
Computer Engineering Department, Ayazaga
80626, Istanbul, TURKEY
erdogan@cs.itu.edu.tr

ABSTRACT

Efficient agent based systems require flexible Agent Communication Languages, such as FIPA ACL, to define the exchange of structured and unstructured information between agent components of the system. The problem of encapsulating semantically rich data, which are to be exchanged between users, applications or agents, can be tackled by XML (Extensible Markup Language). XML is proving to be the backbone of open, platform-neutral data solutions. Therefore, we investigate how agent technologies and Agent Communication Languages can be integrated with XML. This paper discusses relevant technology issues related to the integration task. A rule based distributed event system scenario is outlined to demonstrate the technologies and their integration.

1. INTRODUCTION

With the use of mobile or large-scale systems, the need for asynchronous, loosely coupled and point to multipoint communication pattern arises. Event models are application independent infrastructures that satisfy communication requirements of such systems. Event-based communication generally implements what is commonly known as the publish/subscribe protocol. As shown in Figure 1, an event supplier (publisher) asynchronously communicates event data to a group of event consumers (subscribers), ideally without knowledge of their number and location.

We have developed a rule based distributed event system (RUBDES) [1, 2], which allows the use of composite events in publish/subscribe computational model. In this system, an event is represented as an object and a rule is represented as an expression or a function that is evaluated or executed depending on the occurrence of events.
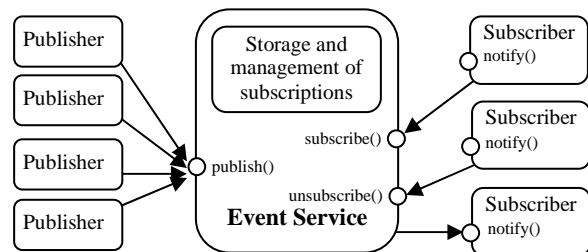

Figure 1. Publish/Subscribe model

We use agents, which are based on Java RMI technology, in our event system. These agents can operate on any platform capable of supporting a Java Virtual Machine and communicating with TCP/IP. They are implemented by using open source, standards-based software including Java, Java RMI, and World Wide Web Consortium (W3C) XML DOM.

Jennings [3] defines agent as an encapsulated computer system that is situated in some environment, and that is capable of flexible and autonomous action in that environment in order to meet its design objectives. Wooldridge[4] says, an agent should have the following properties; autonomy, reactivity, pro-activeness and social ability. In social ability, agents interact with other agents (and possibly humans) via some kind of agent-communication language [5], and typically have the ability to engage in social activities (such as cooperative problem solving or negotiation) in order to achieve their goals. Therefore we want to make conversation between agents via XML encoded FIPA ACL. Because of XML's features to describe both meta-data and data., software agents can easily interpret XML-based messages. XML provides a rich syntax for creating transactions that allow software agents to interact with each other in a platform-independent way. For these reasons, we decided to use XML as the underlying language for agent communication in our work.

The rest of this paper is organized as follows. In the next section, we present a definition and classification of agent communication languages with references to related work. Section 3 introduces the general structure of the RUBDES. Section 4 focuses on event types and how Rule Definition Language (RDL) is used. Section 5 presents an evaluation of the performance of the system. Our conclusions and plans for future work are presented in Section 6.

2. AGENT COMMUNICATION LANGUAGE (ACL)

As the demand for more powerful, efficient and versatile agents grows, so too does the pressure on developers. After all, there is only so much that any one agent can do! If you make your agent perform too many tasks, then the complexity of development and maintenance increases. Just like any other class of application, the more demands we put on our software, the more work must be put in to achieve that functionality.

Agents are generally designed with a specific purpose in mind. They do one or perhaps several tasks very well, but often are not designed as a jack-of-all-trades. If agents must perform more tasks, we can either increase their complexity (which increases the development effort), or we can make them work cooperatively. For cooperation between agents to succeed, effective communication is required. It can be viewed that a collection of agents that work together cooperatively as a small society and for any society to function coherently we need a common language and communication medium.

This language and communication medium is critical for co-operation between agents. A prerequisite to the agent communication is that all the participating agents should be able to understand the communication contents. This means that the agents should use the same language and ontology.

Agent Communication Languages (ACL) have been a cornerstone for the development of systems of communicating agents, and simultaneously they have been the subject of intensive standardization efforts. A persistent theme throughout agents' conceptual evolution has been their ability to interact (communicate) with one another and thus be able to tackle collectively problems that no single agent can, individually. Agent Communication Languages are intended to be above the layer of mechanisms of Agent Middleware (physical protocol, encoding schema and content language) as shown in Figure 2.

If we look at the evolution of ACL, first research can be seen as Knowledge Sharing Effort (KSE) [6, 7]. KSE was initiated as a research effort circa 1990 with encouragement and relatively modest funding from U.S. government agencies Its goal was to develop techniques,

methodologies and software tools for knowledge sharing and knowledge reuse between knowledge based systems, at design, implementation or execution time. Agents, especially intelligent agents, are an important kind of such knowledge-based systems. The central concept of the KSE was that knowledge sharing requires communication, which in turn, requires a common language; the KSE focused on defining that common language.
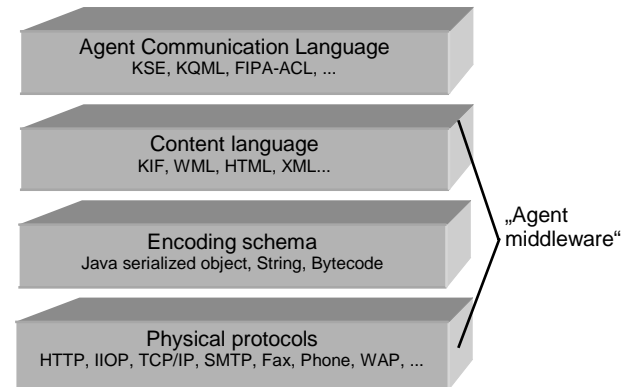

Figure 2 . Levels of Agent Communication

Knowledge Query and Manipulation Language (KQML) [8] is a high-level, message-oriented communication language and protocol for information exchange independent of content syntax and applicable ontology. Thus, KQML is independent of the transport mechanism (TCP/IP, SMTP, IIOP, or another), independent of the content language (KIF, SQL, STEP, Prolog or another), and independent of the ontology assumed by the content..

Foundation for Intelligent Physical Agents (FIPA) [9] is a nonprofit association whose purpose is to promote the success of emerging agent-based applications and services. FIPA's goal is to make available specifications that maximize inter-operability across agent-based systems. FIPA operates through the open international collaboration of member organizations, which are companies and universities active in the .field. European and Far Eastern technology companies have been among the earliest and most active participants, including Alcatel, British Telecom, France Telecom, Deutsche Telecom, Hitatchi, NEC, NHK, NTT, Nortel, Siemens, and Telia.

FIPA ACL is the language developed by the FIPA, the first organized effort focusing on developing standards in the broader area of agents. FIPA ACL is the centerpiece of the FIPA effort. The emergence of FIPA ACL was touted as an attempt for a cleaner purer ACL with well-defined semantics. FIPA's agent communication language draws on speech act theory: messages are actions or communicative acts, as they are intended to perform some action by virtue of being sent. The FIPA ACL specification consists of a set of message types and the
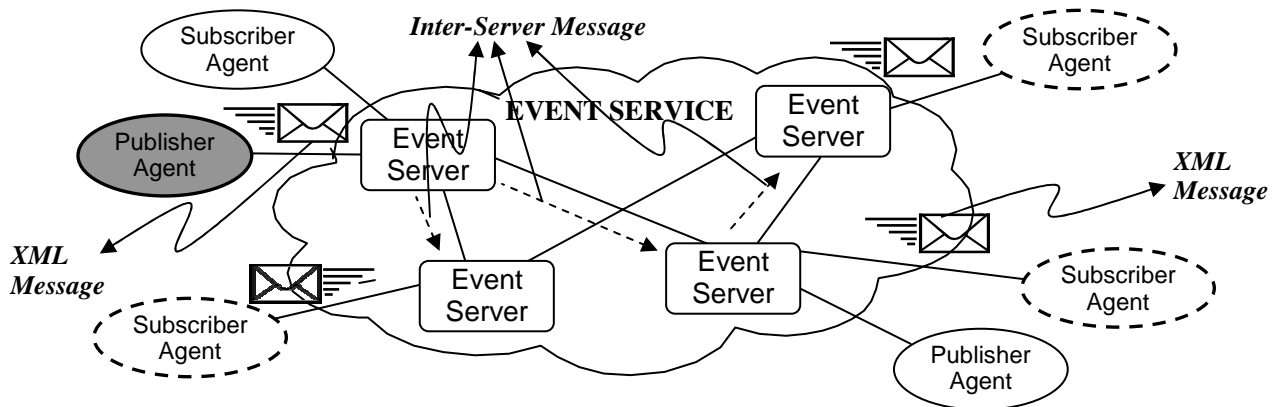
Figure 3. General Architecture of RUBDES

description of their pragmatics that is, the effects on the mental attitudes of the sender and receiver agents.

As mentioned above most popular ACLs are KQML and FIPA ACL. Both have similar syntax helps that a developer will not have to alter the code about messages. We select FIPA ACL as an our agent communication language because it is more powerful with composing new primitives and it is used more than 56 members from 17 countries worldwide.

3. SYSTEM DESIGN

RUBDES is an event-based publish/subscribe system that uses rules for subscribing to an event service. Many of the event systems use predefined events. RUBDES implements a content-based subscription mechanism, similar to that proposed by Carzaniga [10], which enables handling of application-defined events.

RUBDES, being implemented in Java, makes use of Java RMI facility extensively to access remote objects. To create a uniform structure, the components of the system are designed to be accessed over well-defined interfaces and, naturally, they are expected to implement the methods included in those interfaces. Figure 3 depicts the general architecture of RUBDES. The system consists of three main components: Subscribers, Publishers and Event Servers.

SUBSCRIBER: Subscribers of events determine what types of information they are interested in and describe them in a rule form usable by the Event Service. A subscriber has to know the address of the Event Server to which it should register. Subscribers have to implement the Subscriber interface, which consists of a single method, "notify", as depicted in Figure 4. An event server issues a remote call to the notify method of the subscriber to deliver an event. The subscriber is expected to process the event in the context of this method.

```
import java.rmi.*;

public interface SubscriberInterface extends Remote
{
  public void notify(XMLMessage[] data) throws RemoteException;
}
```

Figure 4. Interface of the Subscriber Agent

PUBLISHER: Publishers of information decide on what events are observable, how to name or describe those events, how to actually observe the event, and then how to represent the event as a discrete entity. A publisher process is required to implement a particular interface which is shown in Figure 5. The Event Server's address has to be known by the publisher so that it can issue a remote call to its "publish" method.

```
import java.rmi.*;

public interface PublisherInterface extends Remote
{
public Download_class get_class(XMLMessage s1)
        throws RemoteException;
}
```

Figure 5. Interface of the Publisher Agent

EVENT SERVER: The main function of the Event Server is to dispatch incoming event notifications from publishers to (possibly multiple) subscribers. The event server implements the EventServer interface, which consists of the following four methods, as depicted in Figure 6:

- *subscribe:* A client (a subscriber or an event server) registers interest in a particular event by invoking the subscribe method of the event server. It supplies its RMI contact address and a rule that describes the events it is interested in as parameters to the call.
- *unsubscribe:* A client can cancel its registration by calling the unsubscribe method, supplying parameters needed to identify the subscription previously made.

- *publish:* A dispatcher (a publisher or an event server) calls the publish method to announce an event.
- *serverreceive:* This method is used in communication between Event Servers. Because Event Servers are static entities there is no need to use XML between their communication.

```
import java.rmi.*;

public interface EventServerInterface extends Remote
{
public void publish(XMLMessage pubs) throws RemoteException;

public void subscribe(XMLMessage subs) throws RemoteException;

public void unsubscribe(XMLMessage unsub) throws RemoteException;

public void serverreceive(Server_Data s) throws RemoteException;
}
```

Figure 6. Interface of the Event Server

By using these interface objects, an application that generates data does not need to know anything about an application that will accept and use the data. The generating application only needs to know about the properties of the interface object. As it can be seen easily the message transfers between agents if performed via XML coded FIPA ACL messages, except inter server messages.

3.1 Serializing FIPA ACL with Extensible Markup Language

An event message can be formed in a FIPA ACL message as shown in Figure 7.

```
(inform
        : sender          agent1@ozgur.hho.edu.tr
        : receiver        server@erdogan@itu.edu.tr
        : content         (heat 27)
        : in-reply-to     round-04
        : reply-with      event04
        : language        sl
        : ontology        hpl-event
    )
```

Figure 7. FIPA ACL sample

A sender can encode easily a FIPA ACL message in a String and send it to a receiver by using StringTokenizer and RMI facilities of Java. Ig You want to send this ACL message to the receiver as an object then you can compose an object as shown in Figure 8 and then send this object to the receiver via RMI facilities. For sending an ACLMessage object, that is a serializable object, you set the instance variables of it and send it by calling remote method of the receiver with this object as a parameter.

The receiver must have the ACLMessage interface or necesary classes to interpret or use this message. We want to develop a more scalable event system and thereore we use easily understandable and interpretable message by XML.

```
ACLMessage incoming;
ServiceAgent me;

// suppose that agent me has received an incoming event about
// heat value from the thermometer. Here is an example of how to
// formulate reply using a FIPA like platform:

ACLMessage reply = new ACLMessage("inform");
reply.setDest(msg.getSource());
reply.setSource(me.getName());
reply.setContent("true");
reply.setReplyTo(msg.getReplyWith());
reply.setProtocol("fipa-inform");
reply.setOntology(Constants.ONTOLOGY);
reply.setLanguage(Constants.LANGUAGE);
me.send(reply);
```

Figure 8. Java Program Code Sample

EXtensible Markup Language (XML) [11] is a simplified meta-language, derived from SGML, emerging as the standard for self-describing data exchange in Internet applications. XML was developed by the World-Wide Web Consortium in 1997 and is being implemented rapidly by such major platform vendors as IBM, Microsoft, Netscape, and Sun Microsystems. XML's power derives from its extensibility and ubiquity. Anyone can invent new tags for particular subject areas, defining what they mean in document type definitions (DTDs). Content-oriented tagging enables a computer to understand the meaning of data, including, say, whether a number represents a price, a date, or a quantity.

Using XML for the representation of data would be a good basis for retrieving data by the agents and also for the provider of it: An agent can easily extract information from XML as it includes the concept of an explicit definition of the data structure. Therefore, no additional transformation before extraction of information is required.

Encoding ACL messages in XML offers some advantages that we believe are potentially quite significant.

- The XML-encoding is easier to develop parsers for than the Lisp-like encoding.
- The XML markup provides parsing information more directly. One can use the off-the-shelf tools for parsing XML, instead of writing customized parsers to parse the ACL messages.
- A change or an enhancement of the ACL syntax does not have to result to a re-writing of the parser. As long as such changes are reflected in the ACL DTD, the XML parser will still be able to handle the XML-encoded ACL message.

- In short, a significant advantage is that the process of developing or maintaining a parser is much simplified.
- More generally, XML-ifying makes ACL more WWW-friendly, which facilitates Software Engineering of agents.

```
<?xml encoding="UTF-8"?>

<!ENTITY % communicative-acts "accept-
  proposal | agree | cancel | cfp |
  confirm | disconfirm | failure | inform
  |inform-if | inform-ref | not-
  understood | propose | query-if |
  query-ref | refuse | reject-proposal |
  request | request-when | request-
  whenever | subscribe | unsubscribe ">

<!ELEMENT message (messagetype,
  messageparameter* )  >
<!ELEMENT fipa-message (%communicative-
  acts;)>
<!ELEMENT messageparameter ( sender |
  receiver | content | reply-with |
  reply-by | in-reply-to | envelope |
  language | ontology | protocol |
  conversation-id)>

<!ELEMENT sender (agentname)>

<!ELEMENT receiver (#PCDATA)>

<!ELEMENT content (#PCDATA)>

<!ELEMENT reply-with (#PCDATA)>

<!ELEMENT reply-by   (#PCDATA)>

<!ELEMENT in-reply-to (#PCDATA)>

<!ELEMENT language (#PCDATA)>

<!ELEMENT ontology (#PCDATA)>

<!ELEMENT protocol (#PCDATA)>

<!ELEMENT conversation-id (#PCDATA)>

<!ELEMENT agentname (#PCDATA)>
```

Figure 8. FIPA ACL's Document Type Definition (DTD)

Anyone can invent new tags for particular subject areas, defining what they mean in document type definitions (DTDs). Therefore for using a general communication language we develop a "fipa.dtd", as shown in Figure 8, compatible with FIPA ACL Message Representation [12]

There are three main primitives (publish, subscribe and unsubscribe) in an event based system. In our

"fipa.dtd" we use (inform, subscribe and unsubscribe) communicative acts respectively. A publisher creates an XML message of the FIPA ACL sample (defined above) by constructing a message as shown in Figure 9.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE fipa SYSTEM "fipa.dtd">
<message>
<fipa-message>inform</fipa-message>
<messageparameter>
      <sender>
          agent1@ozgur.hho.edu.tr
      </sender>
</messageparameter>
<messageparameter>
      <receiver>
          server@erdogan@itu.edu.tr
      </receiver>
</messageparameter>
<messageparameter>
      <content>
          (heat 27)
      </content>
</messageparameter>
<messageparameter>
      <in-reply-to>
          round-04
      </in-reply-to>
</messageparameter>
<messageparameter>
      <reply-with>
          event04
      </reply-with>
</messageparameter>
<messageparameter>
      <language>
          sl
      </language>
</messageparameter>
<messageparameter>
      <ontology>
          hpl-event
      </ontology>
</messageparameter>
</message>
```

Figure 9. A sample of XML encoded FIPA ACL message

The receiver (Event Server or Subscriber Agent) receives this XML message, decodes and use in its internal operations.

3.2 Rule Definition Language (RDL)

A rule is an expression or function that is evaluated or executed depending on the occurrence of events. We have developed a language, Rule Definition Language (RDL), to state rules to aid the specification of a single or a pattern of events in distributed systems. The grammar of the language is presented in Figure 10, in BNF notation, with highlighted keywords.

```
<Rule_def> ::=   <Rule>| <Rule> where <Condition>

<Rule>      ::=   rule identifier
                  onEvent <Events>
                  getData <Attributes>

<Events> ::=   class/interface_type identifier |
                class/interface_type identifier, <Events>

<Condition>::=Condition <Boolean_Operator> Condition
             | (Condition) |! Condition
             | <Exp> <Relation_Operator> <Exp>
             | true |  false

<Attributes>::=  event_attribute identifier |
                 event_attribute identifier,  <Attributes>

<Exp>     ::=   ( <Exp> ) | identifier
              |  <Exp> Arith_Operator <Exp>

<Arith_Operator>   ::=   + | - | * | /

<Relation_Operator>::=   > | < | >= | <= | == |!=

<Boolean_Operator>::=     and | or
```

Figure 10. The grammar of RDL in BNF notation

Various programming examples of rules are given in RUBCES [1], which is a centralized (with a single Event Server) version of RUBDES. A subscription rule can be created by the Graphical User Interface (GUI) at the subscriber site. This GUI can be specific to subscription event type and specific types of rules can be produced. Otherwise, a rule can be written manually in a text area component of a general GUI. Of course, this rule must conform to the RDL's grammar.

A rule definition is composed of four parts, each introduced by the keywords `rule, onEvent, getData` and `where,` respectively, as shown in Figure 11. The first part sets a unique identifier for the rule, the second part specifies the type of the target event, the third part specifies the specific information data about the event that subscriber wants to be notified with, and the last part describes the conditions on which a filtered or a composite event should be caught.

In RUBDES, it is possible to define rules for three different event types: `simple events`, events with filtering and composite events. Simple events, shown in Fig. 10.a, are used when subscribers are interested in only one event type. An event-based system may include a multiple number of publishers. Thus, the number of events propagated in an event-based system may be quite large. However, a particular consumer is usually interested in only a subset of the events propagated in the system. `Event filters` are a means to control the propagation of events. Filters enable a particular consumer to subscribe to the exact range of events it is interested in receiving. An event that is delivered uses network bandwidth and CPU processing power on the consumer side. It is therefore desirable to prevent the delivery of unwanted events. RUBDES allows for event filtering as shown in Figure 10.b.

Clients may require to be notified on events from multiple sources and may want to detect a specific pattern of event occurrences from these different publishers. Such a combination of event occurrences, where a client is interested in a sequence of event occurrences but not in any of the events alone, is called an `event composition`. Intuitively, while a filter selects one event notification at a time, a pattern can select several notifications that together match an algebraic combination of filters. An advanced feature of RUBDES is that it allows subscribers to specify composite events, as shown in Figure 10.c.

4. CONCLUSION

In this paper, we investigate how agent technologies and Agent Communication Languages can be integrated with XML in a rule based event system. Our event system consists of agents, which are implemented using open source, standards-based software including Java, Java RMI, and W3C XML DOM. I have also used IBM's de facto standard XML parser utility for Java, XML for Java (XML4J), since it is very well known. (Several other parsers are also available, such as those from Microsoft, Oracle, and Sun.)

Although by wrapping data in XML, the total quantity of data can grow by orders of magnitude

```
rule rule_1              rule rule_2                rule rule_3
onEvent HeatEvent h1     onEvent HeatEvent h1       onEvent Temperature t1,
getData h1.value         getData h1.value                   Humidity h1
                         where (h1.value > 25 and   getData h1.value,t1.value
                                h1.value < 37)       where   (t1.value < 27 and
                                                                  h1.value < 70)


    a. Simple Event            b. Filtered Event          c. Composite Event
```

Figure 11. Sample Rules in Rule Definition Language (RDL)

estimated at two to 10 times the original quantity of data, depending on the amount of data and the amount of XML information with which the data is tagged, the scalability of the system is increased and entrance of different application is enabled.

As future work, we plan to apply the system in different application domains and focus on new design decisions to improve its performance and scalability. We want to add features for mobile subscribers that can connect from different locations to different Event Servers and test the entire system in a large scale platform.

REFERENCES

[1] O. K. Sahingoz, N. Erdogan, "RUBCES: Rule Based Composite Event System", accepted for publication in XII. Turkish Artificial Intelligence and Neural Network Symposium (TAINN), Turkey, July (2003)

[2] O. K. Sahingoz, N. Erdogan, "RUBDES: Rule Based Distaributed Event System", accepted for publication in Springer-Verlag LNCS, ISCIS XVIII - Eighteenth International Symposium on Computer and Information Sciences, Turkey, Nov. (2003)

[3] N. R. Jennings (1999) "Agent-Oriented Software Engineering" Proc. 12th Int Conf on Industrial and Engineering Applications of AI, Cairo, Egypt, 4-10. (Invited paper) [Also appearing in Proc. 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-99), Valencia, Spain 1-7

[4] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. The Knowledge Engineering Review, 10(2):115–152, 1995.

[5] M. R. Genesereth and S. P. Ketchpel. Software agents. Communications of the ACM, 37(7):48–53, July 1994.

[6] R Neches, R Fikes, T Finin, T Gruber, R Patil, T Senatir and W R Swartout. "Enabling Technology for Knowledge Sharing". AI Magazine, 12(3), pp 36-56, Fall 1991.

[7] Ramesh S. Patil, Richard E. Fikes, Peter F. Patel-Schneider, Don McKay, Tim Finin, Thomas Gruber and Robert Neches. "The DARPA Knowledge Sharing Effort: Progress Report", in Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, ed. B. Nabel, C.Rich, and W. Swartout, Cambridge, MA. Oct 25-29, 1992.

[8] T. Finin, R. Fritzson, D. McKay, R. McEntire "KQML as an Agent Communication Language ", 3rd International Conference on Information and Knowledge Management (CIKM94), ACM Press, December 1994

[9] Foundation for Intelligent Physical Agents. http://www.fipa.org.

[10] A. Carzaniga, "Architectures for an Event Notification Service Scalable to Wide-area Networks", PhD Thesis, Politecnico di Milano, Italy, December (1998).

[11] Extensible Markup Language (XML). http://www.w3.org/XML

[12] FIPA ACL Message Representation in XML Specification. http://www.fipa.org/specs/fipa00071/XC0 -0071C .html