IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP5046

Using XML for Simple Hierarchical Communication between Agents

Paul Darbyshire
School of Information Systems, Victoria University, P.O. Box 14428, Melbourne City MC, Victoria 8001, Australia, Phone: +61 3 9588-4393, email: Paul.Darbyshire@vu.edu.au

ABSTRACT

The importance of the role of communication between the agents has also been highlighted by many researchers, particularly for multi-agent systems and for distributed communicating agents. But the form of agent communication often remains a mystery. Agent communication languages and standards are being developed, however the dust hasn't really settled yet, and many implementations use ad-hoc techniques. XML provides an excellent form for agent communication by facilitating the construction of hierarchical message structures. Many of the practicalities for implementation of message recognition can be overcome by the utilization of existing libraries for XML parsing. A message system using an XML-type syntax is more extensible and adaptable for use in a changing environment. This paper discusses the use of XML for the construction of agent-based messages, and presents a simple approach for the deconstruction of messages by receiving agents.

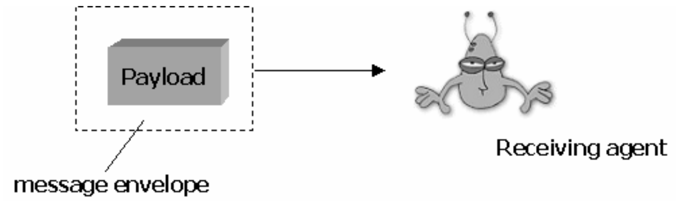
INTRODUCTION

The agent-based paradigm promises to be the next evolutionary step in software design, especially for distributed applications. However, the success or otherwise of these agent-based systems will largely rely on the inter-agent communication systems utilized. The whole approach of the paradigm is small persistent software units working together to solve a problem and fundamental to cooperation between agents is the ability to communicate effectively. In many of the descriptions of agent-based applications, the communication is implied but not detailed directly. Those papers dealing extensively with the communication aspects concentrate on the semantic structure of the messages. But the question remains, what of the structure of the actual message itself?

There are a number of standards describing message structure for communication between agents, for example, KQML, ACL and more recently FIPA ACL. While these standards are well advanced, the specifications stop short at defining a structure for the actual message payload. The message payload is that part of the message which is actually delivered to the receiving agent for subsequent action (depicted in Figure 1). The structure of the payload is important for a number of reasons: the receiving agent must de-construct the message-payload to derive meaning, hence there are practical considerations from the programming perspective; the complexity of the message payload will dictate to some degree the flexibility of the agents in relation to changes in the payload structure; a hierarchically structured payload will allow for extensibility of the messaging system without requiring changes to existing receiving agents.

Using an XML syntactical structure we can send the message payload in an XML hierarchical format, which then affords us a number of practical advantages, including easier message deconstruction and extensibility of the messaging system. This paper discusses some of the practical aspects of message payload deconstruction and demonstrates some of the advantages of structuring the payloads using XML. In the following sections, some background information is given on agents, and agent communication languages, followed by the construction of agent messages. The practical application of XML for structuring these messages, and the subsequent use of the XML Document Object Model for the deconstruction is then detailed. Finally, details of further research and some conclusions are presented.

Figure 1 message payload



BACKGROUND

Agent technology emerged from the field of AI research, so the term 'Intelligent Agent' is often used. However, agents need not be intelligent, and in fact most tasks do not warrant the use of 'smart agents' (Nwana, 1996). Other adjectives often used with agents are, interface, autonomous, mobile, Internet, information and reactive. The term 'agent' can be thought of as an umbrella term under which many software applications may fall, but is in danger of becoming a noise term due to over use (Wooldridge & Jennings, 1995). Many agents are currently characterized by descriptive terms that accompany them, for example intelligent, smart, autonomous etc…

What makes agents different from standard software is the characteristics that agents must possess in order to be classified as agents. Nwana (Nwana, 1996) classifies agents according to primary attributes which agents should exhibit, such as cooperation, learning and autonomy. Indeed, by their very nature, cooperation is one of the primary characteristics which an agent must possess. Genesereth (Genesereth & Fikes, 1992; Labrou, Finin, & Peng, 1999), actually equates 'agency', with the ability to cooperate and exchange data. But while this may be a bit extreme, the nature of agents, being small autonomous software units for specific tasks, means they must cooperate with other agents to perform larger tasks. It is the practical form of this cooperation which has created a landscape of Agent Communication Languages (ACL's).

ACL's had their root in the Knowledge Sharing Effort (KSE) initiated by DARPA (Neches et al., 1991). The core concept of the KSE was that knowledge sharing required communication which in turn meant that a common language was required. The KSE focused on defining a language and proposed the Knowledge Interchange Format, based on a predicate calculus. At this time agents weren't considered when designing the language, but obviously the concepts were directly translatable to agents. Prior to this, each project would implement their own form of ACL (Singh, 1998).

The Knowledge Query Language Management (KQML) project was the first significant inter-project ACL (Singh, 1998) by the KSE in the late 1980's. The KQML language consists of 3 layers: the message layer; the communication layer; the content layer (DARPA, 1993; Labrou et al., 1999). The content layer provides for the actual message content, or the payload to be delivered to the receiving agent. KQML can carry payloads in any representation language, including strings and binary format, but every KQML implementation ignores the content layer (Labrou et al., 1999), and leaves the payload format up the implementing application.

An agent communication language simply called ACL was a variant on KQML, and actually specified or assumed KIF as the payload language. However, the latest emerging standards for agent communication are from the Foundation for Intelligent Physical Agents (FIPA), with FIPA ACL (FIPA, 2002). FIPA ACL does provide a comprehensive message specification language, and also provides the specification in an XML format (FIPA, 2003). This XML specification is in the form of a Document Type Definition, but again, like KQML, stops short in any specification for the message payload.

In May 2000, the Internet Engineering Task Force (ITEF) defined a Simple Commerce Messaging Protocol (SCMP) as an agent language for electronic commerce applications using the Internet (Arnold & Eaton, 2000). While this document does give an example of a message payload using an XML structured message, it was clearly stated that, "*The SCMP protocol doesn't specify payload definitions or how trading partners are expected to process the payload, beyond basic functions related to processing SCMP headers*". The objective was to allow trading partner's flexibility in implementing a standard commerce message format or some other non-standard payload format.

Pragmatically it's difficult to provide a specification for message payload. There are many applications, both agent-based and traditional that may need to exchange all manner of data. However, the vast majority of communication between agents will take the form of simple messages that could be exchanged using a simple format. The XML specifications (Quin, 2003) provide for such a format. XML is almost universally becoming a standard for data exchange between applications and the Application Programming Interfaces (API's) for processing XML documents are well advanced. In particular, Java provides standard classes for dealing with XML documents, and these could readily be used by programmers to provide a practical and extensible message payload format between agents.

TYPICAL MESSAGE PAYLOAD CONSTRUCTION

Given that the message payload format is usually left to the agent developers, it will depend heavily on the application and may include the transmission of binary data. However most applications including e-business applications can normally transmit message in the form of a simple string. The complexity of the string depends on the data being transmitted, for example if each data item is no more than a single word or number, then the items within the string can be simply separated by spaces, eg:

" SKU 167843T1 SIZE 12 STORE 8"

If a data item contains more than one sequence then the string will be delimited by a special character, such as a comma or a colon, eg:

"NAME,Paul James,CREDIT LIMIT,5000,ADDRESS,11 City Road"

However, more often than not, the data items appear without any preceding identifiers such as

"167843T1 12 8" or "Paul James,5000,11 City Road"

In such a situation, both the sending and receiving agents must be intimately aware of the structure of the message payload. During the deconstruction of the message, the receiving agent must parse the string into its various tokens, and assume the tokens are in the correct order. Continual error checking on the tokens as the message is parsed is the only way to check against an invalid message. The received tokens are checked against the agent's beliefs of the structure and makeup of the message, and any deviation from this is marked as an error. This leaves little room for extensibility of the message format without altering the beliefs of the receiving agents.

If we wish to add extra components to the message for some agents, we could append this to the end of the message. Depending on the message parsing implemented in all receiving agents, this may or may not require modification. In some cases, it may not be prudent to append the data to the message, but rather embed it within the message, thus changing the structure. This would require all agents receiving such messages to be aware of the new structure and deconstruct the messages accordingly.

The emergence of XML as dominant standard for data transfer provides us with an opportunity to utilize standardized XML API's for processing message payloads when structured using XML. This in turn will provide us standardized routines for deconstructing the message, and a message format that is essentially extensible in nature.

USING XML FOR MESSAGE PAYLOAD CONSTRUCTION

XML is good at representing information that is extensible and hierarchical in nature. In most cases the messages in agent-based systems, including web-based eBusiness applications can be represented in a hierarchical structure. In the example given previously, we can represent the customer information in an XML format as shown in Listing 1.

Listing 1 XML example message

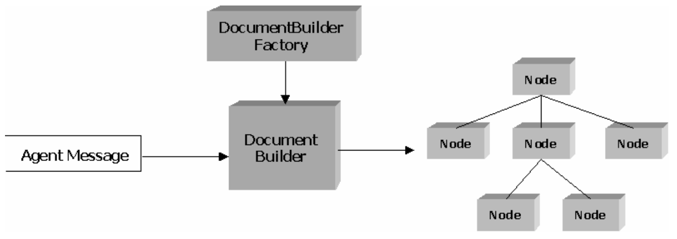
```
<CustomerRequest>
    <CustomerName>Paul James</CustomerName>
    <CreditLimit>5000</CreditLimit>
    <Address>
        <Street>11 City Road</Street>
    </Address>
</CustomerRequest>
```

The advantage of using XML for message structuring lies in the use of the XML Document Object Model (DOM), for retrieving the data in the message deconstruction. The Document Object Model is an Application Programming Interface for valid HTML and well-formed XML documents and is the foundation of XML. XML documents have a hierarchy of informational units called nodes and the DOM is a way of describing those nodes and the relationships between them. For instance, when processing an XML document, the document is read through a parser that analyses the structure of the document, and from there a representation of the document can be constructed in memory. As an XML document is hierarchical in nature beginning with the root element, the representation of the document in memory is also hierarchical, represented as a tree structure. Once we have a representation in memory of the XML document, it can be manipulated under program control. Although we use the term document here, the XML can be in the form of string passed to an agent as the payload of a message.

In a survey of 58 commercial and academic agent construction tools, the Java language is used in 31 of these tools (Odell, 2003). Java is becoming the language of choice for the constructing of agents due to the close association between the Web, Java and agent development. Agent technology is an offshoot from AI research, but its rise in popularity has coincided with that of the Web, as the Web offers an almost perfect environment for agent development. Java development is also closely related to Web development and Java includes many Applications Programming Interfaces for network programming and Web interfacing. Another set API's included with Java are those for parsing XML documents and interfacing to the DOM as specified by the Document Object Model Level 3 Core from the W3C (Le Hors et al., 2003).

Using the DocumentBuilderFactory and DocumentBuilder classes we can very simply parse an agent message payload which is constructed

Figure 2 Parsing a message into a DOM



using well-formed XML and produce a DOM internal structure in a few lines. This is depicted in Figure 2. The actual Java code snippet to achieve this is shown in Listing 2. As can be seen, from a practical perspective, the code to parse an XML message and build the DOM is quite small. If multi-agent systems that utilize communication to achieve cooperation are to be commonplace, then we need to be able to make use of such standards to cheaply and efficiently implement all communication aspects.

Listing 2 Parsing an XML message

```
public void process(String msg) {
      DocumentBuilderFactoryfactory= DocumentBuilderFactory.newInstance();
      try {
             DocumentBuilder builder = factory.newDocumentBuilder();
             ByteArrayInputStream is = new ByteArrayInputStream(msg.getBytes());
             Document doc = builder.parse(is);
                    :
```

By utilizing the existing XML application programming interfaces in Java, the coding effort is minimal and results in a very practical structure for deconstructing the message. As indicated in Figure 2, the DOM is a hierarchical tree structure beginning with the root node of the XML message (the opening tag of the message). In the XML message shown in Listing 1, the opening tag would be <*CustomerRequest*>. An abstraction of the resultant DOM obtained by parsing the message in Listing 1 is shown in Figure 3. The DOM in reality is slightly more complex with the separation of the XML tag information and actual element content into separate sub-nodes, but Figure 3 closely represents the structure of the DOM.

With the document object model created, the task of deconstructing has already been partially completed, a complete understanding of the message is then simply a matter of traversing the structure looking for the required information. The Java application programming interfaces include methods to traverse and update the document object model again simplifying the coding effort required by the programmer.

Extensibility of Messages

With the document object model representing the parsed message, to retrieve elements of the message, the code can '*drill down*' the DOM looking for the elements it expects. Thus while the agent still needs a knowledge the elements of the message it expects to find, the order and placements of these elements in the message payload is no longer a primary concern. The document object model is what the agent will interrogate to derive meaning from the message.

Figure 3 DOM structure for XML example

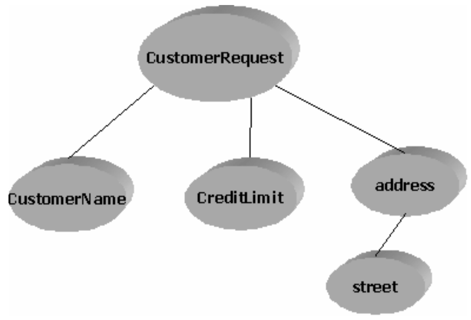
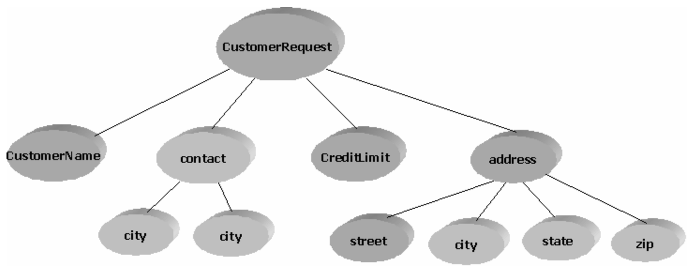


Figure 4 Modified payload document object model



The message payload can be substantially modified without affecting the receiving agents. For example, we may need to modify the content of the message in Listing 1 to add further *address* information and unrelated *contact* information for a newly developed agent which requires this extra information. By using XML syntax, Listing 1 could be modified which would result in the parsed document object model shown in Figure 4. Such an expanded document model would not affect existing agents, as by *drilling down* from the root of the DOM, the information they require is still there in the same format. Yet a new agent will also find the extra information added to the message. The placement of the tags and element data within the message payload is of no consequence to the receiving agents provided the XML message is well-formed.

Such a system is far more extensible than a message constructed using a string. In such a case, expansion of the message is error prone and dependent on the parsing methods used by these agents.

Validation

Another aspect of agent messaging is the validation of the structure of the message payload itself. Each token in a string based payload is validated against the type of data that is expected at that particular point in the token sequence. Thus the code of the parser is highly structured towards the expected sequence. By using XML for the message payload, as we have seen, the order of the XML tags in the input message is no longer important. The parsing routines implemented by the *documentBuilder* class ensure the XML is well formed, otherwise the document object model would not be constructed and an error would result. Thus the placement of the code in Listing 2 within a Java *try … catch* statement.

If the XML is well formed and the document object model is built, there is still no guarantee that the DOM contains all the required tags for the receiving agent. In this case the document can be validated by the agent as it *drills down* the DOM looking for the nodes and data it requires, much as an agent deconstructing a string payload might do. However, XML provides a unique method for automatic validation with the use of Document Type Definitions (DTD's) or via an XML Schema. Built into the Java API's for processing the XML message payload, is the ability to apply a DTD to the message to automatically validate the payload. Listing 3 contains the DTD required to validate the XML message payload of Listing 1.

Listing 3 DTD for CustomerRequest message

```
<!ELEMENT  CustomerRequest (CustomerName, CreditLimit, Address)>
<!ELEMENT Address(Street)
<!ELEMENT  CustomerName  (#PCDATA)>
<!ELEMENT  CreditLimit  (#PCDATA)>
<!ELEMENT  Street  (#PCDATA)>
```

The DTD can be included in either the XML message payload itself, or externally in a document supplied to the Java API at parsing. There is very little change in the Java code in Listing 2 to perform DTD validation. With this validation in place, once the message payload is parsed then the document object model not only represents well-formed XML, but also XML conforming to the DTD. As a consequence, the agent can be sure all required tags and elements are present. This then relieves the programmer from further error checking code. If constructed appropriately, then earlier DTD's need not be incompatible with later DTD's representing an expanded form of the message (as in Figure 4). Thus extensibility of the message systems remains unaffected.

CONCLUSIONS

If multi-agent systems are to become widely accepted as a paradigm for large-scale applications, or for networks of cooperating applications over the Internet, then concrete practical methods of implementation will be essential. In particular, the communication issue needs to be addressed. As communication is an essential component for cooperating agents, programmers need to be able to implement a simple, extensible form of communication. The current standards are quite complex, with little attention being given to actual message payload. Many of the papers dealing with communication are developing logic based languages for intelligent agents, yet the vast majority of agents will not be intelligent and will only need to deal with simple communication.

XML provides us with a means to specify a message payload using a simple hierarchical format. With current research and development pushing XML to be the standard for data transfer on the Web, the development of API's for XML parsing and recognition are well advanced. Utilization of these API's provides the agent programmer with a practical and simple method to implement message payload construction and deconstruction with minimal effort. This also provides us with a way to structure the messages in a format which is flexible and extensible, allowing for future expansion.

REFERENCES

Arnold, T., & Eaton, J. (2000). *Simple Commerce Messaging Protocol (SCMP) Version 1 Message Specification*. IETF. Retrieved 1/9/2003, 2003, from the World Wide Web: http://www.globecom.net/ietf/draft/draft-arnold-scmp-06.html

DARPA. (1993). *Knowledge Sharing Initiative. Specification of the KQML agent-communication language.*: DARPA Knowledge Sharing Initiative, External Interfaces Working Group.

FIPA. (2002). *ACL Message Structure Specification* [Web Page]. Foundation for Intelligent Physical Agents. Retrieved 20 Aug, 2003, from the World Wide Web: http://www.fipa.org/specs/fipa00061/

FIPA. (2003). *ACL Message Representation in XML Specification* [Web Page]. Foundation for Intelligent Physical Agents. Retrieved 20 Aug, 2003, from the World Wide Web: http://www.fipa.org/specs/fipa00071/

Genesereth, M., & Fikes, e. a. (1992). *Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report*: Computer Science Department, Stanford University.

Labrou, Y., Finin, T., & Peng, Y. (1999). The Current Landscape of Agent Communication Languages. *IEEE Intelligent Systems, 14*(2).

Le Hors, A., Le Hégaret, P., Wood, L., Nicol, G., Robie, J., & Byrne, S. (2003, 9/6/2003). *Document Object Model (DOM) Level 3 Core Specification*. W3C. Retrieved 1/9/2003, 2003, from the World Wide Web: http://www.w3.org/TR/2003/WD-DOM-Level-3-Core-20030609/

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout, W. (1991). Enabling Technology for Knowledge Sharing. *AI Magazine, 12*(3), 36-56.

Nwana, H. (1996). Software Agents: An Overview. *Knowledge Engineering Review, 11*(3).

Odell, J. (2003). *Agent Construction Tools*. Retrieved 12/9/2003, 2003, from the World Wide Web: http://www.paichai.ac.kr/~habin/research/agent-dev-tool.htm

Quin, L. (2003). *XML Core Working Group Public Page*. W3C. Retrieved 25/6/2003, 2003, from the World Wide Web: http://www.w3.org/XML/Core/

Singh, M. P. (1998). Agent Communication Languages: Rethinking the Principles. *IEEE Computer, 31*(12), 40-47.

Wooldridge, M., & Jennings, N. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review, 10*(2, June 1995).