Using XML To Help Isolate Software Systems and Agents From Change Due to Communications

Paul Darbyshire
Victoria University, Australia

Abstract

Development and research into distributed and agent based systems has grown enormously over the last few years, and the number of practical applications for such systems has grown along with it as the technology and infrastructure improves to accommodate such systems. As with all systems, evolution and change is inevitable, but with the growth of distributed systems and the Service Oriented Architecture, we have another dimension of change we need to consider; that of communication. The importance of the role of communication between these systems has been highlighted by many researchers, particularly for multi-agent systems and for distributed communicating agents. But the form of such communication often remains a mystery. Communication aspects are often dependent on other factors within an architectural framework, particularly the data. In order to reduce unnecessary changes to the communication aspects of a system, we need to insulate the communication as much as possible from consequential change effected by architectural other framework elements. A message system using an XML-type syntax is more extensible and adaptable for use in a changing environment. It helps to isolate the communication from the structure and content of the message, thereby reducing consequential change. 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.

Keywords

Agent, Communication, XML, Java, ACL, Architecture Framework

Introduction

The topic of software communication has become important in the wider business context, not just in computer science circles. Given the current trend towards agent based software, SOA architecture, distributed systems and Enterprise Application Integration in the general IS application domain, communication aspects of software systems will become a major component. If little thought is given to the structure of the protocols and messages used, follow-on maintenance costs are likely to be incurred, and likely to be expensive. Such follow-on maintenance can be caused by development efforts in other subsystems, or from further developments in external systems that communicate with the software, and affect the structure of the messages to be sent and received.. We need a way of isolating the communication component from changes in other systems or subsystems to reduce the costs of maintenance.

Distributed systems, web services and software agents represent the first steps in new software development paradigms. The Service Oriented Architecture (SOA) supports the requirements of business processes by linking together loosely coupled software services (Channabasavaiah, Holley and Tuggle. 2003). The SOA does not restrict itself to

Web Services alone, though these have achieved much attention lately. The SOA is an architecture that is not restricted to any one technology, and agent based systems fit comfortably within the SOA domain. While the success of Web Services have shown the IT community that SOA works, 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. Fundamental to cooperation between agents and agent systems 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 11**). Most of the standards specify the structure of the envelope, such the standards specifying the SOAP envelope used by many Web Services, but leave the payload structure to the application. However, 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.

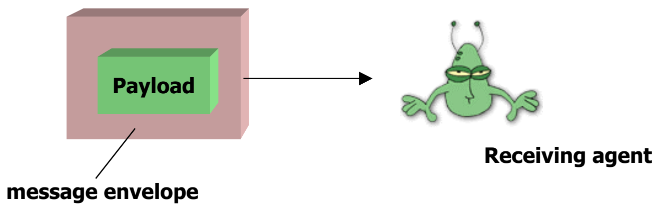

Figure 11 message payload

The payload is that component which is most vulnerable to change when there is a corresponding change in the data model or the business process model of the systems underlying architectural framework. If the structure of the payload is too closely aligned with either of these, then chances are that a change in either will effect a consequential change in the communication. One of the benefits of agent based systems and the SOA architecture is the promise of ease of component integration and integration with legacy systems. A need for a communication change in one system could trigger a cascade requiring communication change in others. We need a way of limiting this change and separating the data and business process models from the structure of the message payload. Using an XML syntactical structure we can send the message payload in an XML hierarchical format which then affords us a number or 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. The remainder of this paper is organized as follows: some background information is given on agents, and agent communication languages, followed by a discussion on 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.

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.

It is interesting that John McCarthy proposed a formal Common Business Communication Language (CBCL) in his paper 'The Common Business Communication language', written in 1975 (McCarthy 1982), but not actually published until 1982. In this paper, McCarthy proposed a language based on LISP which forecasted much of what XML was later to become. A language that was somewhat less verbose than XML, but one which was extendible so that as software improved, the messages could be extended. McCarthy was clear that it was important to keep the language incrementally extendable so more detail could be added to messages at a later time. It is this type of language that we need to embrace for agent communication, one that is extendable, but won't require extensive modifications to the underlying communication software when the message is extended. Unfortunately, the features of a communication language alone won't achieve this, but a suitable communication language coupled with a platform independent standardized application programming interfaces will facilitate this.

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.

Dimensions of Change?

The architectural framework underlying any computer systems always has a number of elements or dimensions where change can be effected. A well known architectural framework is the Zachman Framework initially presented in its early form in 1987 (Zachman 1987). This framework has since been refined and is presented in textual format in Table 1 below.

	What (Data)	**How (Function)**	**Where (Location)**	**Who (People)**	**When (Time)**	**Why (Motivation)**
Scope {contextual} **Planner**	List of things important to the business	List of processes that the business performs	List of locations in which the business operates	List of organizations important to the business	List of events/cycles important for the business	List of business goals/strategies
Enterprise Model {conceptual} **Business Owner**	e.g., Semantic Model	e.g., Business Process Model	e.g., Business Logistics System	e.g., Workflow Model	e.g., Master Schedule	e.g., Business Plan
System Model {logical} **Designer**	e.g., Logical Data Model	e.g., Application Architecture	e.g., Distributed System Architecture	e.g., Human Interface Architecture	e.g., Process Structure	e.g., Business Rule Model
Technology Model {physical} **Implementer**	e.g., Physical Data Model	e.g., System Design	e.g., Technology Architecture	e.g.,Presentation Architecture	e.g., Control Structure	e.g., Rule Design
Detailed Representation {out of context} **Subcontractor**	e.g., Data Definition	e.g., Program	e.g., Network Architecture	e.g., Security Architecture	e.g., Timing Definition	e.g., Rule Definition
Functioning System	e.g., Data	e.g., Function	e.g., Network	e.g., Organization	e.g., Schedule	e.g., Strategy

Table 1: Textual representation of the Zachman Framework (Zachman 1987)

The Zachman Framework defines six (6) aspects of systems architecture: date, function, location, people, time and motivation. Each of these aspects becomes a dimension of possible change during the systems incremental change regime or evolution. We need to add another dimension to this; that of communication. As systems become distributed, or are developed using an SOA utilizing Web services or agents, communication takes on a greater role. Unfortunately, all these dimensions of change are not orthogonal, that is, each is not independent of the others when changes are implemented. Some areas in software engineering and systems architecture research are concerned with reducing the follow-on affects of change in some dimensions, due to change in another. For instance, implementing a Model View Controller design in database applications helps separate the business processes from the underlying data manipulation, thus allowing changes in one to not effect changes in the other.

Changes in the Data and Functionality dimensions in the Zachman framework are most likely to effect follow-on changes in communication. Changing the data to be delivered via a communication message may require communication protocols to be changed, and the corresponding sender and receiver to do something differently. Similarly, a change in functionality may require different message structure and consequently changes in the communication aspect. Thus we need to design or select a communication language which will aid in reducing communication based changes due to other dimensional changes.

In the following sections, the practical aspects of using XML for message payloads are discussed and it is demonstrated that utilizing XML for message payload can help reduce these follow-on changes.

Typical Message Payload Construction

Given that the message payload format is usually left to the agent developers, then what form does it usually take? The form will depend heavily on the application and may include the transmission of binary data, but 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, and hence requiring modifications to the agent’s message parsing components.

In the case where we wish to add extra data to the message for some agents, we could append this to the end of the current message. Depending on the message parsing implemented in agents which are to receive the message but not process the additional data, 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. Of course, in some messaging systems, the structure of the message itself may be included in the payload, but this will require more complex coding and comes with its own problems.

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 the XML format as shown in Listing 1.

```
<CustomerRequest>
```

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

Listing 1 XML example message

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 was 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).

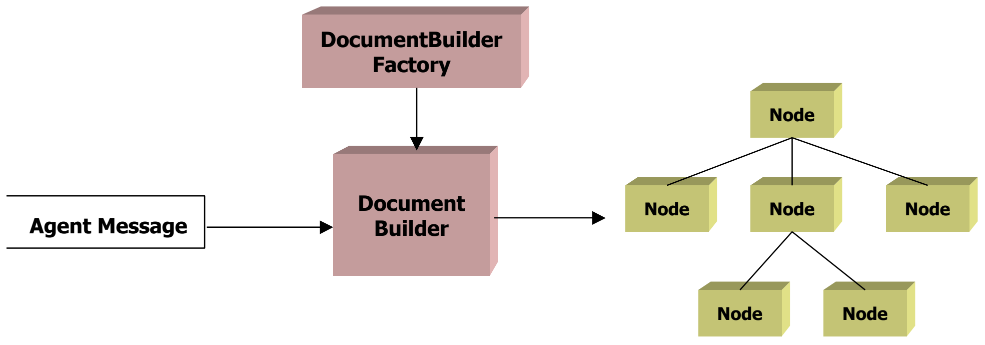

Figure 2 Parsing a message into a DOM

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

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

Listing 2 Parsing an XML message

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. Of course 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.

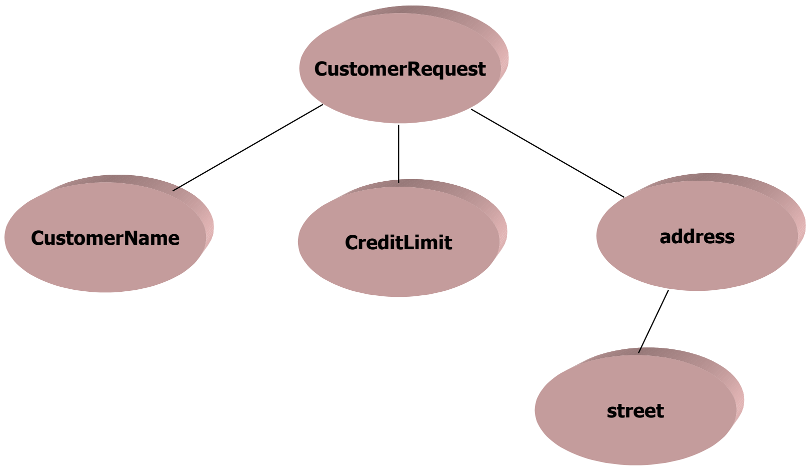

Figure 3 DOM structure for XML example

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

Messages constructed by an agent and subsequently deconstructed by another in this fashion are far more extensible than those expressed as simple strings. 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.

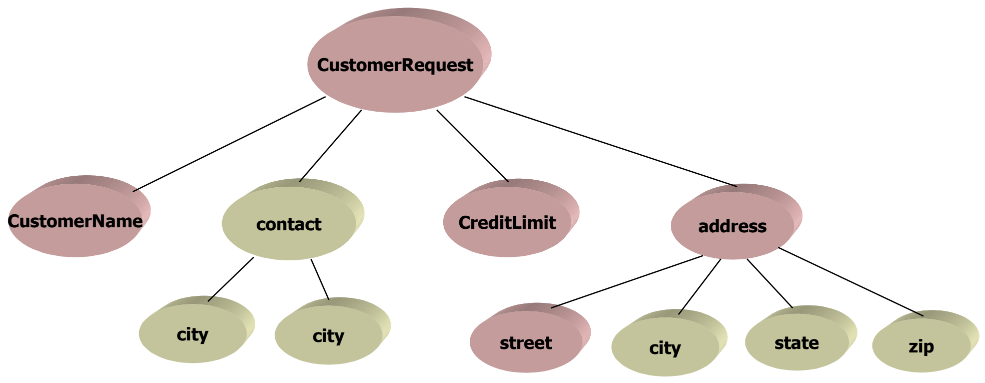

Figure 4 Modified payload document object model

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 1. 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 the 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.

Essentially, given the technique outlined, the structure of a message can be changed significantly from the original, as the XML application programming interfaces allow a message to be "drilled down into" when looking for elements within a hierarchy. Thus, no matter how the message structure is modified, as long as the original message structure is embedded somewhere in the expanded message due to the modifications, the receiving agents can correctly decode and interpret the message. This is depicted in Figure 5 below.

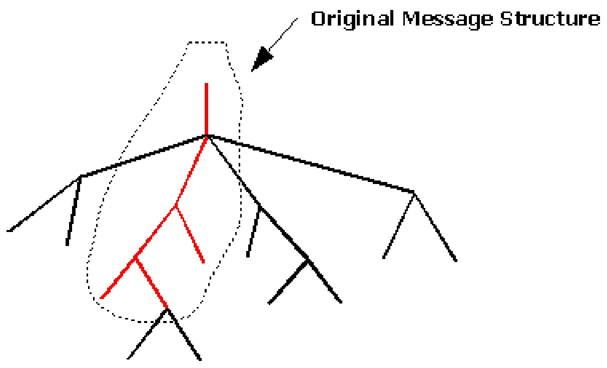

Figure 5 Original message structure embedded in expanded message

Such a system is far more extensible than a message constructed using a string as outlined in a previous section. In such a case, expansion of the message is error prone depending on the parsing methods used by these agents. More than likely, all receiving agents would need updating.

Validation

Another aspect of agent messaging is the validation of the structure of the message payload itself. This was mentioned briefly before, but in a string based message, the structure of the message is continually checked for errors as the string is parsed and tokens are extracted. Each token 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.

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

Listing 3 DTD for CustomerRequest message

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.

Further Work

Further investigation into two consequences of using XML for agent message payloads is warranted. By using XML as the payload specification the size of the message increases with the inclusion of the metadata (or XML tags). While this will increase traffic over any network that agents communicate through, for most agent applications this should not be significant. With increasing bandwidth and computing power, only time critical agents may be affected, but the vast majority of agents operate in a delayed asynchronous environment. However, further studies will need to try and quantify this.

The other main aspect is that of security. With the inclusion of metadata in the payload, context is given to data within the payload itself. How much of a concern this is to the majority of agent systems remains to be seen, but encryption may be needed. Many agent systems may already include encryption of the message payload. If this is done by the messaging system without the intervention of the sending and receiving agents then these agents can remain out of the encryption loop. Otherwise, the agents may need to become part of the loop.

Concluding Remarks

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

Channabasavaiah, C., Holley, K., Tuggle, E. (2003), Migrating to a Service Oriented Architecture, IBM Developerworks, http://www-128.ibm.com/developerworks/library/ws-migratesoa/, accessed 15/10/2006

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/

McCarthy, J. (1982), Common Business Communication Language, in Albert Endres and Jurgen Reetz, editors, Textverarbeitung und Burosysteme, R. Oldenbourg Verlag, Minich and Vienna, 1982, accessable from http://www-formal.stanford.edu/jmc/cbcl2.pdf

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).

Zachman, J. (1987), A Framework for Information Systems Architecture, IBM Systems Journal, Vol 26 No. 3, 1987, http://www.zifa.com/framework.pdf