My Agent Will Call Your Agent … But Will It Respond?

*Martin L. Griss*

Laboratory Scientist, HP Laboratories

Abstract

As Web based E-commerce and component software become more real, interest in software agents has increased. Agents can be viewed as specialized distributed components, offering greater flexibility than traditional components when developing certain applications. Different kinds of agents have different amounts of personality, mobility, interaction, collaboration, persistence and intelligence. This article focuses on agent communication and multi-agent systems (groups of communicating, collaborating agents). Workflow can help define how agents with different roles might interact in a flexible, yet constrained manner to accomplish a collaborative task. Agents and workflow can combine to provide a new technology for application development, providing significant benefits beyond those traditionally associated with components and scripting.

Keywords: Agents, Components, XML, Workflow, Scripting, Multi-Agent System, Ontology, Conversations

1   Introduction

There is a growing interest in distributed agent systems of various kinds. Every day, Web based E-commerce, message-based middleware and component technology are becoming more real, more compelling and more pervasive.

Figure 1- Agent-Mediated E-Commerce
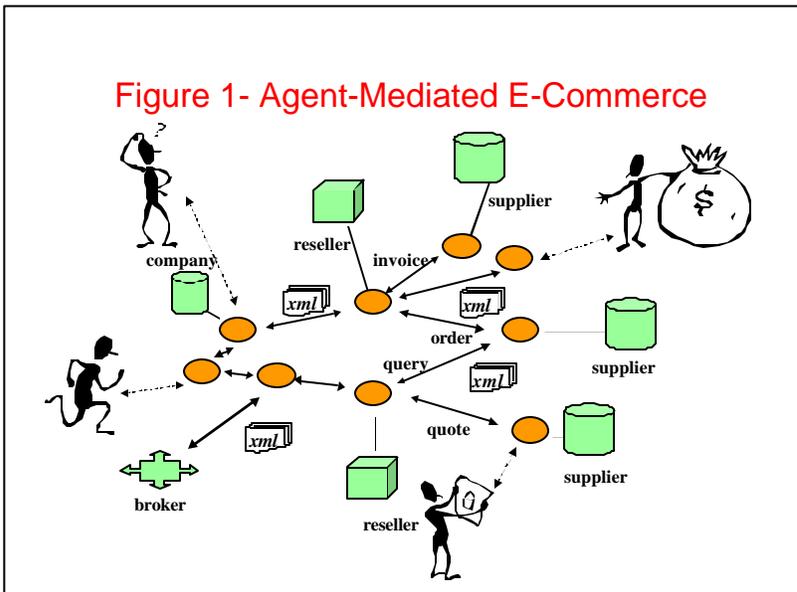
As Sharma describes in his article on building E-Commerce applications with components [Sharma99] the next generation of E-commerce applications requires much more flexibility, larger and more complex applications and many more applications, integrating processes across enterprises. He calls this Inter-Enterprise Process Engineering. Many of these application components will be written at different times, by different developers. Developers are seeking more powerful ways of quickly building flexible distributed systems and new services that will provide a more compelling user experience and new capabilities to a larger body of users. Component agent technology offers significant promise.

Agent-based e-commerce systems have great potential. Agents will dynamically discover and compose e-services and mediate interactions; XML will become the lingua-franca of agent-based E-commerce

interaction [Glushko99]. As illustrated in Figure 1, agents can serve as representatives or delegates to handle routine affairs, monitor activities, set up contracts, execute business processes, and find the best services[Chen99].

Consistent with the direction of the industry we use XML and HTTP for our agent systems. XML and HTTP are simple to use, are ubiquitous, heterogeneous and platform neutral. Robust and scalable implementations, rapidly growing tools and other support are widely available. Like Java, these have caught on quickly, and provide many services, such as robust and scalable web servers, firewall access and levels of security. Agents use XML extensively to encode exchanged messages, documents, invoices, orders, service descriptions and other information. Automated negotiation, and complex search to find and establish a connection to participant e-services will be key.

Three major classes of technologies are maturing to provide a basis for far more ubiquitous agents in E-commerce:

- Pervasive middleware (InterNet, HTTP, web-objects, HP E"Speak, Microsoft BizTalk and Sun's Jini)

- Business to business communication standards and models (XML and vocabularies defined by CommerceNet's eCo, CommerceOne's CBL, ontology.org, and RosettaNet)

- Intelligent Agents (genetic algorithms, fuzzy logic, neural nets, rule-based systems, and agent technology).

Most people have an informal sense of what agents are - intelligent pieces of software that operate somewhat autonomously to represent a user's interests. From a software development point of view, agents can be seen as a kind of distributed software component. The benefits and process of working with agents adds to those of working with components - modular, independently developed pieces of software (components) can be combined in a variety of ways to produce new application systems. Agents offer the promise of even more flexible and extensible systems.

There are many web sites, papers, books about agents [Huhns98, Jennings98, Bradshaw97].

There are many definitions of software agents, such as, "*an autonomous software component that interacts with its environment and with other agents."* Some definitions emphasize the mobility aspects, some the collaborative aspects, others the intelligence, and yet others the social aspects. For our purposes, agents can be usefully viewed as an evolution or combination of distributed objects, active objects, business objects and scriptable components, using languages such as TCL, VBScript or Perl. Software agents are active components, sometimes mobile, that expose specialized interfaces, are usually driven by goals and plans (rather than procedural code), have "business" or "domain" knowledge, operate autonomously, and quite often interact using a specialized, declarative agent communication language (ACL), such as KQML (rather than direct method invocation). Organizations such as FIPA (Foundation for Intelligent Physical Agents) are defining reference models, mechanisms and agent communication language standards (see www.fipa.org).

There are several different kinds of agent systems that are of interest. These include:

- **Personal agents** that interact directly with the user, often presenting some "personality" or "social skills," perhaps as an anthropomorphic character, with the goal of monitoring and adapting to the user's activities, learning the user's style and preferences, and automating or simplifying certain rote tasks. R&D involves animation techniques, natural language interaction, personalities, user profiles, machine learning and intelligence. For a simple example, consider Microsoft's Agents "Bob," "Wizard" and other more complex personal agents. A number of toolkits exist to help construct these.

- **Mobile agents**, sent out to collect information or perform actions at one or more remote sites, and then return with results. These are typically implemented in Java or a portable scripting language such as TCL, Perl or Python. R&D includes security, scalability and fault tolerance, developing techniques to determine what capabilities agents can access at the remote site, how to specify itineraries and interactions, and the appropriate levels of security and certification. The Java VM, sandbox-based security and class loader have provided a very nice framework for mobile agent experimentation. Examples include complex information access (better performed at the source of the data rather than

sending raw data back and forth), (Telecom) network management and service provisioning, and other "touring" agents that go from site to site to find and aggregate data pertinent to a particular query.

- **Collaborative or social agents**, communicating and interacting as members of a multi-agent system. Groups of agents represent users, organizations and services. Multiple agents engage in conversations as patterns of messages, to negotiate, exchange information, etc. R&D includes work on intelligence, conversation management, agent-communication languages, ontologies, auction systems, markets, etc. Examples include online auctions (such as ebay.com, priceline.com and amazon.com), planning, negotiation, e-commerce, logistics, supply chain, telecommunications, and system management.

Over 100 varieties of agent system have been developed, ranging from A-Z (e.g., Agent-TCL, Aglets, Concordia, Grasshopper, Jackal, JATLite, Jumping Beans, Voyager, and Zeus), focusing on different mixes of mobility, adaptability, intelligence, ACL and multi-language support.

For our work in E-commerce, we are most interested in multi-agent systems. While many agents often are used individually, things become much more interesting when a group or society of agents collaborate together to perform some task. Several agents, representing people and institutions, work to together to accomplish some business or information management goal. For example, to support the purchase of books, a group of agents might exchange a series of messages as part of a stylized conversation to find the best deal (perhaps by bidding during an auction), arranging financing, selecting a shipper, and tracking the order.

In this article we will use as a small example, a synthesis and simplification of work done by several groups in HP for service definition, composition and management for a mini-E-commerce system, such as a book buying service, a travel service or a payment service. More details on this example and additional agent references are available at www.hpl.hp.com/reuse/agents.

In the rest of the article, we first discuss agents as next-generation components and discuss some features and variants of typical agent systems. We then show how agents communicate using XML encoded messages, and finally how agent conversations can be managed and choreographed. In a subsequent article, we apply the technology to an E-services example.

2   Agents as next generation components

COM, ActiveX, CORBA, JavaBeans and Enterprise JavaBeans (EJB) define slightly different component models, but the concepts are similar enough for our purposes. Components are packaged software modules, with some number of well-defined interfaces, some distribution and composition properties, and well-defined lifecycle. For example, an ActiveX component is a COM component with many additional interfaces and events, such that any ActiveX component can be used appropriately within the ActiveX framework. Not only does the component provide some required interfaces to conform to the framework, it also makes use of many standard interfaces to access the support services, such as those provided by ActiveX containers, property bags, etc. It is these services, and the way the components conform to the framework, that makes component-based development so powerful, yet sometimes so complex.

Component-based software development involves selecting or designing an application architecture, defining core services and mechanisms that become part of the infrastructure and framework, and defining and implementing standard interfaces that enable components to interact with each other and to access the services in the infrastructure/framework.

Component interfaces can be defined at several levels of precision, to support increasingly more effective component composition and conformance checking with the framework. For example, interfaces can be simply given as a list of method signatures, essentially as is done in Java. An interface definition language (IDL) can be used with strong type checking. Interfaces can also be modeled, using the Unified Modeling Language (UML). Correct usage and relationships between the methods in a single interface can be described as a protocol using an interaction diagram or a state machine. For example, that certain methods should be called before others. Constraints and relationships between methods as using UML types, collaborations and the object constraint language (OCL).

Scripting is an increasingly popular and productive way of gluing components into complete solutions. Scripting languages are typically small, interpretive languages that have good access to the underlying component model (e.g. VB Script or VBA for COM components, Java or Java Script Java for JavaBeans, and TCL, PERL or Python). For a component to be scriptable, it typically needs to expose additional interfaces. Developers and end-users can create applications by using scripting to combine, control or modify robust components created by domain experts; this is a key strategy that underlies the great appeal of Visual Basic in the enterprise.

Agent-based software development is very similar to conventional component development, and leads to even more flexible componentized systems, with potentially less design time and implementation time, and decreased coupling between the agent components. In the same way that models, collaborations, interaction diagrams, patterns and aspect-oriented programming help build more robust and flexible component and component systems, so can the same techniques be applied to agent component and agent systems. For example, agent patterns, agent collaboration roles and aspect-oriented programming for agents.

The agent infrastructure and framework should include additional services and mechanisms to enable the agent components to have simpler, more composable interfaces. Agent-oriented programming (AOP)[Shoham93], can be thought as a new way of looking at the development of certain large complex distributed systems at a higher level of abstraction. The idea is to treat each agent as an autonomous entity, driven by a set of beliefs, desires and intentions (BDI). Just as an object-oriented developer might use role playing to simulate object responsibilities and interactions when using CRC design, an agent-oriented developer will design the system in terms of what agents "believe," their "goals," and by "telling" information or "asking" questions. Some agents may in fact be implemented using "intelligent" AI technology; others will simply exchange messages as if they are operating from beliefs and goals. This BDI approach will enable some complex systems to be developed more quickly and flexibly, and allow a more effective handling of evolution, distribution, and complexity.

Just like component-based development, there are similar considerations concerning interfaces on agent components, the structure of messages, the scripting of agent components using agent languages. Similar methods are used to define and constrain legal message sequences, known as conversations or conversation protocols.

3 Major features of different kinds of agents

A component can be usefully viewed as an agent component when some amount of properties such as mobility, intelligence, autonomy or others are present. Such properties can be shown as axes of a diagram, such as Figure 2, useful in discussing the differences between various agent systems. The greater the area enclosed on the diagram, the more "agent-like" the component or set of components, and the more appropriate an agent-oriented approach. These properties work together to make agent-based system more flexible and robust to change.

- **Adaptability -** the degree to which an agent' s behavior may be changed, by downloading new programs, sending new rules or otherwise changing the knowledge they have and customizing their behavior.

- **Autonomy-** the degree to which agents are active, execute one or more threads of control, and have the ability to pursue some goal largely independent of messages sent from other agents (as distinct from objects in which methods are only invoked by messages **).**

- **Collaboration -** the degree to which agents work cooperatively and communicate with other agents to form multi-agent systems working together on some task.

- **Intelligence -**the degree to which the agent displays knowledge and understanding, and ability to reason from goals and knowledge. Often an intelligent agent requires some AI technology, such as rules, blackboards, neural nets, genetic algorithms, fuzzy logic or knowledge bases to support this.

- **Mobility -** the ability for an agent to move from place to place, either just moving the code, and starting the agent afresh ("code mobility") or using complete serialization of code and state, allowing the agent to begin execution at one location, do some work there, and then move itself to another

location, retaining its state so it can continue work. Many applications and agent systems use only single-hop mobility, while others exploit full multi-hop itineraries.

Figure 2 - Agent Dimensions
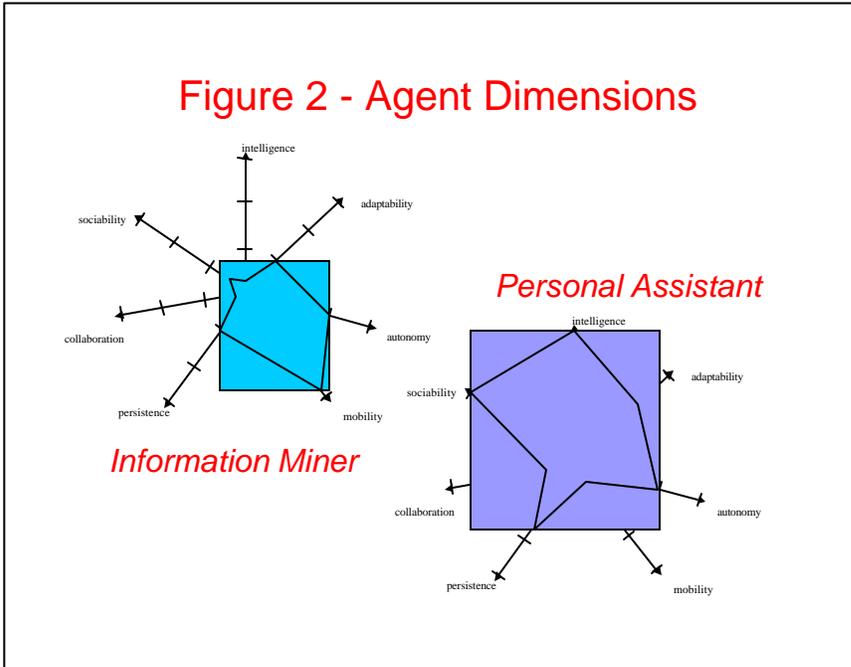
- **Persistence -** the degree to which agents retain knowledge and state over periods of time, system crashes, multiple sessions and so on.

- **Personality/Sociability -** the degree to which the agent exhibits a pleasant style (personality) appealing and appropriate to particular user and particular task. Also relates to customization, perhaps by learning.

Other properties, such as heterogeneity, security, etc. might also be useful in comparing agents systems to each other, and to component systems.

4   A typical agent system

We will use the term "agency" to refer to the location (typically a process or server) which is the local environment on each machine in which hosts agents. The agency provides basic management, security, communication, persistence, naming, and agent transport in the case of mobile agents. In addition to basic agent platform and Agent Communication Language (ACL), a typical FIPA compliant agent system, such as Zeus or Grasshopper, provides additional services in the form of specialized agents. Figure 3 summarizes an overall agent system architecture:

- Agent Management System (AMS) - controls creation, deletion, suspension, resumption, authentication and migration of agents. Provides local "white pages" service (naming and locating) of agents.

- Agent Communication Channel (ACC) - routes messages between local and remote FIPA agents, realizing messages using ACL.

- Directory Facilitator (DF) - provides "Yellow Pages" service for FIPA agents; they register some agent capabilities so an appropriate task-specific agent to handle the task can be found.

- Agent Platform (AP) - provides communication and naming infrastructure, using the Internal Platform Message Transport

The agent management system and platform include capabilities for naming, communication and conversation management. Many can be implemented by using features and services of HTTP and XML, such as XML structured documents for messages using DTD, XML Schema , URI for naming and HTTP headers for some of the context:

Naming - To identify the agents, for the purposes of sending the messages and changing their programs, rules and knowledge. Agents may have a globally unique name, perhaps based on a GUID, or they may have one or more local names based on the agency or agent group in which they are located, perhaps using a URI. While a local or regional name is quite adequate, in the

case of mobile agents, a globally unique name is needed, using a distributed, hierarchical name service (such as DSN or URN) , or the home agency could host proxies for forwarding addresses.

Communication - To provide a systematic and consistent way for agents to interact with each other, appropriately visible to the agent management system, using an agent communication language (ACL), based on structured messages, known languages and known vocabularies.

Conversation management – provides support for the monitoring and controlling of stylized message exchanges to insure that they conform to a desired protocol or pattern.

Infrastructure and Agent Services - The middle-ware, mechanisms and services (often presented as other agents or libraries) to support naming, communication, management, mobility, persistence and legacy wrappers for (non-agent) software.

Figure 3 - Agent System Reference Model
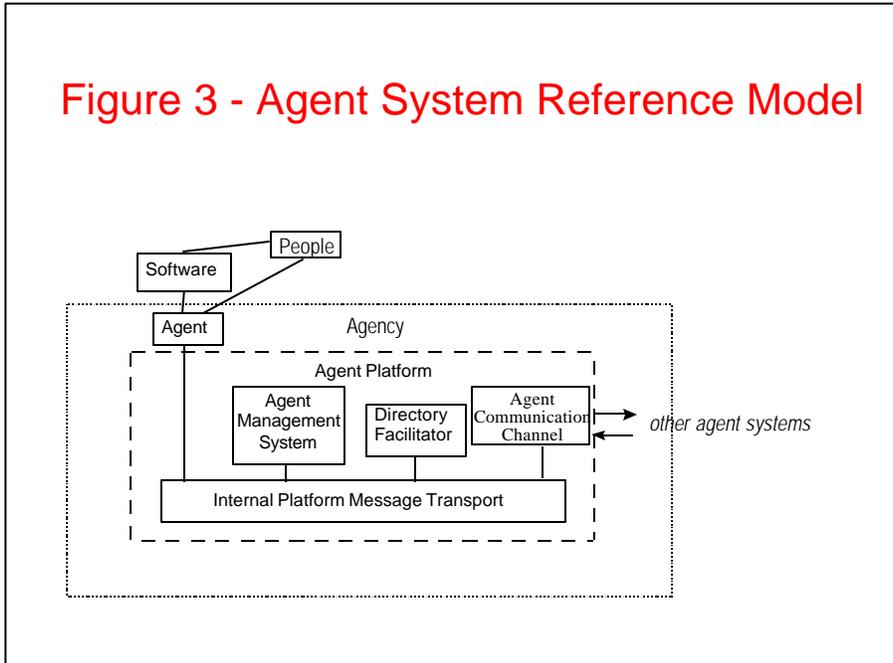
Different agent systems will provide additional services, usually in a form of distinguished agents, such as a Broker agent. For example HP's E"Speak environment, which provides many of the needed capabilities, as well as distinguished services, such as Broker (www.hp.com/e-speak).

5   Tower of Babel - The language and ontology problem

Whether we're building our system using distributed components or intelligent autonomous agents, communication proceeds by exchanging messages in some agreed format. If we are specifying conventional component interfaces using IDL, we will need to agree on the specific method name and parameter names and types, and how they are intended to be used -- that is both their syntax (method signature) and their semantics. Generally speaking there are quite a large number of methods, each with a relatively small number of fixed parameters, each of fixed, relatively simple type and structure.

As more services and components are added to the system, we will find it useful to create a framework which defines several required interfaces, each with a number of methods, and supporting services. The system will grow and change. Each time we extend the system, we will need to periodically refactor the framework, changing interfaces, and re partitioning the system. This can become an increasingly rigid system, hard to extend and change.

The agent-oriented alternative is for all components have a very similar, simple interface, but allow the messages to have much more complex structure. These messages can easily be changed and extended dynamically as the system evolves.

The simplest possible way to do this is to define a specialized XML-based language for communication in the particular problem area. For example, using a (too) simple agent-oriented book buyer's bidding language, the seller advertises an offer to all potential buyer agents:

```
<bidding version="9.2" sender="seller" receiver="all-potential-buyers">
  <offer startingprice="200" currency="USD" expire="11/16/99 5pm MST"/>
  <item code="273495" units=10>Rare books by Shakespeare</item>
</bidding>
```

My agent would then respond to bid on 3 books:

```
<bidding version="9.2" sender="martin" receiver="seller">
  <bid price="250" currency="USD" expire="11/15/99 3pm PST" />
  <item code="273495" units=3/>
</bidding>
```

This language is much too simple to be useful; it does not deal with authentication, security, anonymity, multiple bids on different bundles, and so on. More powerful, specialized languages could be defined for other kinds of interaction, leading to a plethora of similar, incompatible languages. Instead, we could define and standardize a more complex, generalized auctions on proposal language.

There is a clear benefit to using an XML-based agent language. Instead of defining many interfaces and methods, we can use a simple interface with a more complex, structured message, (with structure defined by a DTD.) However, while convenient and direct for the problem at hand, these specialized languages typically have no well-understood relationship to other similar languages. While it is much easier to define and introduce new versions and variants, these quickly lead to a large number of incompatible languages.

All of these languages have to deal with common problems and standard patterns of interaction between agents. Standardizing this "outer" structure of agent communication is the essence of a general purpose agent communication language (ACL). An ACL defines the generic structure of a conforming agent message, and provides some rules and guidelines about the range of values those various message elements can assume. Additional parameters in the message will then indicate the domain being described and the expected conversation pattern, perhaps allowing general purpose agents to be dynamically customized to support the detailed needs.

5.1   Multi-Agent Communication

The idea in an ACL is to factor the communication problem into several relatively independent parts: the message type, some addressing, and some context and the content or body of the message. This factoring enables more support from the infrastructure (the agent and message management systems.) This approach makes it much easier to dynamically extend the agents to new problem areas, yet to still have the system check conformance to expectations, and enable the infrastructure to manage messages and agents in various ways. The message type indicates how the messages should be processed individually, and as part of a larger conversation. Addressing indicates from whom and to whom the message is sent. Context enables the content of the message to be understood. The context typically includes elements to make explicit the language, vocabulary and conversation patterns.

Most agent communication languages are loosely based on Speech Act Theory (SAT)[Bach79], a linguistic theory concerning how people communicate and engage in stylized conversations, using typed messages that indicate the intent of each communicative act. These typed messages are exchanged in accord with stylized conversation patterns. Based on the type of the message sent (e.g., request or command or information), we expect a certain of response. The message type indicates the key intent of the message. In SAT, these are called "communicative acts," or "performatives," " verbs" or "actions". These distinguish commands, requests, information, and various kinds of responses, such as acknowledgement or errors. The idea is that the basic structure and purpose of a conversation between several autonomous agents can be understood, even without examining the detailed content.

There are a number of agent communication languages, with KQML [Finan97] and FIPA ACL (www.fipa.org), the most well-known,. There are many others designed for special purposes that can be considered to be "KQML-like"[Moore98]. Some ACL's are intended support the BDI model, while others make it easy to check that agents conform to organizational conventions and display acceptable behavior.

Singh [Singh98] describes the evolution of several of these and their deficiencies for systems such as E-commerce, and highlights the need for a new ACL, which makes more visible roles and commitments in a social context. This is believed to be more effective than agent models based on internal beliefs or mental models.

Different ACLs factor messages in different ways, and provide different standard parts. The intent is to make the parts more reusable, and the intent clearer and more actionable. KQML has over 30 standard performatives. FIPA ACL uses significantly fewer communicative acts (CA), by allowing some combined CAs to be built up as compositions, or embedded in the content or context, or in messages sent to the specific DF and AMS agents. Other ACL's use the basic KQML/FIPA style, but replace or extend the set for special purposes, such as contract negotiation, offers, and bids.

The content of the message can be written in different languages, for example a subset of LISP or a logic or knowledge inter-change language (such as Prolog or KIF), or a scripting language (such as TCL, Perl, Java Script or VB Script); these can be encoded in XML. Specifying the language typically means that the syntax of the content and semantics of keywords are defined, but the rest of the content will refer to terms that relate primarily to the concepts in the problem area being communicated about.

For effective communication, and maximal support from the underlying agent management system , each message has a standard structure, showing the message type, context information with addressing information, and message sequencing, and the body of the message itself expressed in the specified language and ontology.

Our simplified XML encoding, called "KXML 1.0," combines our favorite aspects of KQML, FIPA ACL and other ACLs. A sample message is illustrated in Figure 4.

```
        <message type = "REQUEST" version="kxml 1.0">
          <address sender="//hplmlg3/martin" receiver="//hpbooks/seller">
          <context protocol="english-auction" conversation-id="c12"
                       reply-with="m123" in-reply-to="m17" reply-by="10/9/99 3pm pst"/>
        <content language="Xpression" vocabulary="book-buying" >
         <expression op="offer-to-buy">
            <price currency="USD">30</price>
            <item code="27345" units="3"/>
         </expression>
        </content>
        </message>
```

Figure 4: Sample KXML message

Each part of the message has a distinct role:

- **Message type** - (e.g. *type="REQUEST"*) - also called "performatives," "communicative acts," "verbs" or "actions". Standard KQML includes performatives, such as (ADVERTISE, BROKER, ASK-IF, ASK-ONE, ASK-ALL, TELL, SORRY, ERROR), etc.).   FIPA ACL has similar verbs, such as (REQUEST, INFORM, REFUSE, FAILURE, NOT-UNDERSTOOD, etc.) These verbs can used in conjunction with specified conversation protocols to allow the system to monitor and control the progress of conversations, and confirm the compatibility with conventions and communication pragmatics.

- **Addressing** - (e.g., *sender="A12"* and *receiver="A14"*) used to identify the sender and the intended receiver(s) of the message, using some form of local, regional or global naming scheme. May also use *originator* for forwarded or brokered messages. We use URI's.

- **Message Sequencing** - (e.g., *reply-with="M12" in-reply-to="m27")* used to connect a series of messages to a specific conversation and to each other, providing "in-reply-to" and "reply-with" identification.

- **Conversation Control** - *conversation-id* used to connect a series of messages to a specific conversation. *Reply-by =date-time* sets a deadline for a timeout. *Protocol* identifies a specific

conversation type, which details an expected pattern or protocol of message exchange between two or more agents. (E.g., auction, contract-net, proposal)

```
<!ELEMENT message (address?,  context?,  content*)>
 <!ATTLIST message
        type CDATA #REQUIRED
        version CDATA #IMPLIED>
<!ELEMENT address EMPTY>
 <!ATTLIST address
        sender  CDATA #IMPLIED
        receiver  CDATA #IMPLIED
        originator CDATA #IMPLIED>
<!ELEMENT context EMPTY>
 <!ATTLIST context
        in-reply-to  CDATA #IMPLIED
        reply-by  CDATA #IMPLIED
        reply-to  CDATA #IMPLIED
        conversation-id  CDATA #IMPLIED
        protocol  CDATA #IMPLIED>
<!ELEMENT content (expression | statement | message)>
 <!ATTLIST content
        language  CDATA #IMPLIED
        vocabulary  CDATA #IMPLIED>
```

Figure 5: Simple DTD defining syntax of KXML 1.0.

- **Vocabulary** - (e.g.,*vocabulary="service-management")* - also called ontology, identifies the domain of discourse to which the message content applies, for example "Buying,", "Payment", "Banking," "Cars," "Management," etc. This defines objects and legal data types ("load"), action words ("adjust"), simple attributes and more complex attributes.

- **Language** - (e.g., *language="Xpression")* specifying the language in which the content is expressed, such as an AI language (LISP, KIF, or PROLOG) an agent language (FIPA-SL or KQML), a scripting language (TCL, Vbscript), encoded in XML.

- **Content** - a statement, expression in the chosen language using terms from the chosen ontology. Can be written in KXML to include nested messages.

Figure 5 shows a simplified DTD for KXML 1. A more complete DTD will explicitly define all of the legal message types, provide additional constraints on the attributes, and define the explicit expression and statement structure. See also http://www.hpl.hp.com/reuse/agents/kxml). The statement language is an XML encoding of LISP expressions, Prolog or KIF.

There are several other XML encoded ACL's being defined somewhat like this. Finan and Labrou have defined an XML DTD for standard KQML. Moore has defined FLBC [Moore98], which has a different approach to decomposing the speech acts, claimed to provide a better match to a more open multi-agent system. FLBC has been demonstrated to successfully and effectively encode EDI documents.

5.1.1   Ontology - What do words mean anyway

"Words mean what I want them to mean!"-Humpty-Dumpty, Alice in Wonderland

Unlike the communication problems that Alice encountered in speaking to Humpty-Dumpty, agents need to communicate using words that have agreed meanings, and which make sense in the given context.

The set of words, their relationships, and their meanings is known as an ontology or vocabulary.

To be precise, the ontology is a specification of the concepts to be used in the exchange of messages. It is not enough to know the syntax of the message, nor just the list of legal words -- the agent sending the message and the agent receiving the message (or at least the programmers creating each of the agents) must share an understanding of what the words are intended to mean. Some words can be defined formally in

terms of other words, while others have to be described in a standard "dictionary," telling the agent program(er) how they are to be used.

For example the word "charge" could be used when talking about finance, battles or electrical batteries - and would mean different things depending on the context. In banking, we might use the words "charge," "fee," "levy," "settlement," "payment," "interest" and so on. These words have particular relationships to each other and may not be automatically substituted.

An ontology is a domain specific vocabulary, used to specify the concepts in some domain. The words' meanings and intents are defined, and in addition the ontology will show key relationships between words (such as synonyms, antonyms and hyper-nyms), showing how some words may be replaced by others or are generalizations, etc. Some words can be defined formally in terms of others ("macros") but others must simply be agreed upon by people using these ontologies, by using agreed standards. There are several basic ways to come up with an ontology:

- Certain stable, well-defined problem areas have agreed-upon terminology, which is the basis of the ontology, even if not formally specified

- An international standards committee can define the single standard ontology via laborious negotiation e.g., HTTP, OSI, DMTF, UML, CYC

- Individual groups can develop a several ontologies, and then the agent system would have to name and negotiate, and perhaps translate between them

- An ontology might be dynamically grown based on statistical analysis of word usage.

In practice, a combination of these methods will be used, and so the agent language needs to be able to specify and dynamically select the ontology to use. Standard ontologies will need to be defined for various domains of interest -- such as catalogs of products, banking, credit cards, EDI, workflow, and so on. Several organizations are working to collect and standardize these ontologies or related e-commerce frameworks. For example, Ontology.org (www.ontology.org), CommerceNet (www.commercenet.com), Commerce One's Common Business Library (CBL 2.0) (www.commerceone.com), and RosettaNet's technical and business dictionaries, and partner interaction process templates (PIPs) (www.rosettanet.org). HP E"Speak provides a basic vocabulary for describing and managing vocabularies (www.hp.com/e-speak).

There are several ways in which a useful ontology may be represented

1. The simplest is a natural language "dictionary," listing terms, their meanings and intended use. Many of the ontologies described above are defined this way. Then a structure for XML interchange is defined as a DTD. XML, while useful to specify a standard structure and syntax for messages, does not in itself define the meaning of the terms.

2. Next is to start with a DTD defining the names of terms and some of their relationships (syntax and some typing), and then add comments or an external dictionary to explain the meaning, additional relationships and intent of the terms.   A DTD can define elements and attributes, some types, and some required, optional and default values for those attributes. However, the set of available data types is limited, and the ability to precisely control how many elements of any type (multiplicity) is minimal.

3. One could instead use an object oriented XML Schema (see www.w3c.org)[1] to more precisely define structured datatypes, and to obtain the benefits of element inheritance, and more precise control over how many of each type of element and attribute are permitted. More relationships between types, elements and attributes can be defined, allowing more precision in the ontology. Again, inline comments or an external dictionary is used to give the intent of the words. However, not all relationships are shown clearly or explicitly.

4. Finally, one could proceed more formally using a modeling language, such as UML or a tool such as Ontolingua, to define the elements (as classes or types), relationships between them (as inheritance or

---

[1] Or SOX, a precursor XML Schema Language, as used by CommerceOne.

associations) and provide semantics and constraints using OCL and natural language comments. In some domains, such as network and system management, a standard object model based on UML such as the DMTF CIM, with supporting documents, provides a rich and natural ontology. As another example, RosettaNet uses UML diagrams to precisely model elements in its technical dictionary.

For agent-mediated E-commerce, in addition to the standard set of words, some relationships and a standard XML representation will be given as DTD or some form of XML schema.

One of the challenges is that several different vocabularies have emerged and for the same or related domain, such as payment and banking. And more are being created all the time. In some cases, this is because several organizations have worked quickly and independently, in some cases a desire to dominate a market segment. If a single standard can not be agreed by the industry, then agents will either need to negotiate on which vocabulary to use, translate to a common vocabulary, or use some intermediary agent as translator.

Since E-commerce is such a dynamic environment, we expect that dynamic ontologies will be important. These are vocabularies in which new concepts and words are introduced during interaction. The ontology grows and evolves. Several techniques can be used. Data-mining techniques can observe the sequence of interactions and gather statistically significant words. Data-mining techniques can also cluster terms and connect them based on relationships, building a concept structure. Other machine learning techniques can also be used.

In many cases, words will be drawn from multiple vocabularies for the same expression (for example to "*charge the payment for the battery charge."* In this case, we associate a distinct XML name space with each ontology, and use techniques similar to those in the W3 resource description framework (RDF) to combine terms and relationships from several vocabularies, using prefixes such as FIN: for finance, POWER for power-management. This leads to an expression like:

```
<FIN:charge>
    <FIN:payment amount=30 currency=USD/>
    <POWER: charge>battery-4</POWER:charge>
</FIN:charge>."
```

5.2   Multi-Agent Coordination

Multi-agent systems (groups of agents working together to achieve some purpose) are critical for large-scale E-commerce, such as service provisioning, systems management, negotiation, and auctions. The group of agents can be a static or dynamic grouping. The communication can be between people and agents, or between agent and agent, or a mix. We will focus on agent-agent communication.

A key issue is how to coordinate the interactions between members of the group in order to achieve some higher-level task, such as requesting, offering and accepting a contract for some services. We call this "choreography." A group of agents might be expected to have a series of stylized message exchanges to accomplish this task, perhaps by advertising and using a brokered service (Figure 6), bidding during an auction, responding to a call for proposals, or negotiating a price.

There are several possible levels of choreography which relate to the expected message sequences. The choice largely relates to choosing or needing a relatively "loose" or more "tight" control of the allowed interactions. Below we discuss the built-in message sequence assumptions, conversation protocols which explicitly constrain message sequences (typically pair wise) to a stylized conversation, and finally even more constraining use of a workflow in which the various agents are participants with specific roles.

5.2.1   Built-in Assumptions on Legal Sequences of Agent Messages

By using a speech act based agent communication language, such as KXML, the message type (performative) automatically provides some indication of the expected message sequence, without the need to look at the detailed content of messages. This leads to certain standard and expected message sequences between agents. The system can watch for and monitor compliance with these message sequences. For example, agent A might express interest in a fact or service, by issuing an "request" to one or more agents.

One then would expect one or more messages from other agents, either an "inform" of their ability to provide this service or fact, or to indicate that they did "not understand" the message or an "error" when processing the request. The system can monitor timeouts, perhaps by using the "reply-by."

5.2.2   Conversation Protocols

If a group of agents are expected to have a series of stylized message exchanges to accomplish a task such as bidding during an auction, or responding to a call for proposals, then it makes sense to specify a standard protocol of message exchanges as a new type of conversation.

Protocols can be defined and represented in a variety of ways. The simplest is a message flow diagram, as used by FIPA. More complex protocols will be better represented using a UML sequence or interaction diagram or UML StateChart, deterministic finite state machine (DFSM), or Colored Petri Net (CPN).

Figure 6 uses UML diagrams to show the use of the "broker" performative, for the use of a weather service agent WS. First the weather service agent, WS, will advertise to broker B that it can respond to certain "request"s, by sending the message "advertise(request(x))" to B. Then some user agent, UA, will then issue a "broker (request(x))," which will then obtain the answer from WS, via an "request(x)," which then responds with "inform(answer)". B then relays this to UA with its own "inform(answer)". As shown, this can be expressed as a sequence or interaction diagram. The sequence of messages is shown in Figure 7.

Figure 6 - Message Sequencing
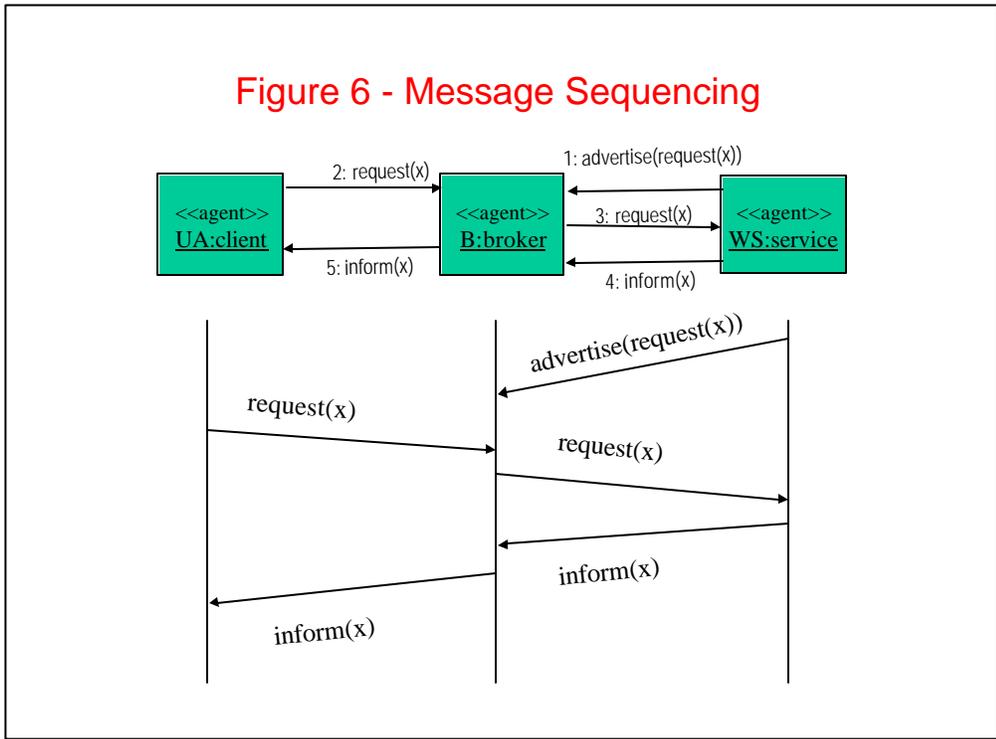
When more complex conversations or coordination is needed, a simple protocol may not be adequate. For example, when part of the interaction is to prescribe the selection of a particular resource, within a specified time, else select an alternative and seek approval of a manager. Then a complex workflow structure can be used, either provided to a single coordinating agent that assigns tasks and monitors progress for each member of the group, or the piece of the workflow structure corresponding to the role an agent is to play (say as "buyer" or "seller") can be assigned to that agent, perhaps as a protocol or state machine. Finally, given non-malicious agents, they perhaps can have access to the entire workflow but only be allowed to execute steps corresponding to that designated role.

```
<message type="ADVERTISE> <address sender ="WS" receiver="B">
  <content language="KXML">
    <message type="REQUEST">
      <content language="Xpression" vocabulary="weather">
        <expression op="weather"><param name="x"/></expression>
      </content>
    </message>
  </content></message>
<message type="REQUEST> <address sender ="UA" receiver="B"/>
      <content language="Xpression" ontology="weather">
        <expression op="weather">San Francisco</expression>
      </content>
</message>
<message type="REQUEST> <address sender ="B" receiver="WS"/>
      <content language="Xpression" vocabulary="weather">
        <expression op="weather">San Francisco</expression>
      </content>
</message>
<message type="INFORM> <address sender ="WS " receiver="B"/>
      <content language="Xpression" vocabulary="weather">
        <expression op="weather">
          <expression op="and">Cold Windy</expression>
        </expression>
      </content>
</message>
<message type="INFORM> <address sender ="B" receiver="UA"/>
        <expression op="weather">
          <expression op="and">Cold Windy</expression>
        </expression>
      </content>
</message>
```

Figure 7 - The sequence of KXML messages corresponding to the brokering protocol.

5.2.3 Workflow for scripting and choreography of multi-agent systems

A convenient way of choreographing these interactions between collaborating agents in a multi-agent society is to use workflow as a form of next generation scripting language[Griss99]. Either a new workflow/scripting language will be developed, or workflow primitives will be added to an existing scripting language, such as TCL, VB Script or Java Script.

The definition or specification of a process (or workflow) can be an effective basis for the specification of agent coordination as well. The process defines steps that are executed by the coordinated actions of a variety of agents ; these individual steps of the process are allocatable to the different agents.

As we stated in the introduction, many E-commerce applications intrinsically involve cross organizational workflow, and thus some form of workflow technology will probably be used for certain parts of the system. At the very least, agent technology and workflow technology need to cooperate more effectively.

Several researchers and practitioners have noticed several interesting connections between agents and workflow. Workflow is a method and mechanism to allow the explicit definition and control of a group of participants in enacting a (business) process of some sort. Workflow automates (part of) a business process, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules. The workflow management system defines, creates and manages the execution of workflows.

A variety of different workflow languages and systems have been developed (e.g., ActionWorks Metro, Endeavors, HP ChangeEngine, IBM FlowMark, Little-JIL, , Verve Workflow, or Worklets) many of which

involve a graphical interface to describe allowed connections between participants, desired and exceptional processing conditions, the assignment of roles, and the request and allocation of resources.

Workflow and agents can be combined in several ways, illustrated with the following examples.

- Agents can collaborate to perform a workflow, *e.g., telecom provisioning (BT), service provisioning (HP), auctions, scheduling, shopping, ...*

- Agents can be used to make workflow more intelligent, *e.g., by adding negotiation, reasoning at decision points.*

- Workflow used to choreograph a set of agents, *e.g. application management (Utah/HP), GPGP*

- Workflow can be used to coordinate interactions between people and agents, having agents delegate to people or people to agents; *e.g. a telecom management system alerting a human operator, or assigning a repair or provisioning engineer.*

An interesting way of creating a workflow system is to have agents represent the participants and resources, and to have the society of agents collaborate to enact the workflow. In some workflow systems, the participants (human or machine) are in fact referred to as agents.

Agent-based systems have been used to implement new workflow systems[Jennings96] or augment existing workflow systems[Shepherdson99] to create more flexible workflow systems and more effective interoperation between different workflow systems. A group of agents in an agency associated with one workflow engine will interact to negotiate with agents in a different agency associated with a different workflow engine, to provision the workflow (that is to allocate or reserve the needed resources), and to mediate and monitor the flow between regions. Work at British-Telecom [Shepherson99] uses the Zeus agent toolkit to provide an agent enhanced workflow, that is an agent level mediating between the workflow engines.

It is thus convenient to view some form of workflow as a next generation scripting language for certain classes of multi-agent component coordination, offering ways of providing explicit control of these multi--agent interactions at higher level.

UML can also be used to model processes and workflow. When UML is used for business modeling, business usecases are used to model processes. A particular usecase is essentially a class of alternative scenarios, covering the interactions of one or more external actors with this process. Each scenario represents a particular set of steps, a flow through the steps of the usecase. Exceptional conditions typically map to another "extension" usecase that extends/modified the normal flow of work through the usecase.

Each actor plays a particular role in its interaction with the process. A business usecase can be modeled at several levels of precession: natural language narrative, natural language pseudo code, state machine, or interaction diagram. Each usecase maps to a collaboration of business objects. Each actor and each business object plays a distinct role on the process.

Protocols might be less constraining or rigid than a fully specified traditional process/workflow. Instead, the process/workflow can be cascaded hierarchically, as sub processes/workflows, loaded dynamically following negotiations.

6 Experience and Related Work

The following activities illustrate some of the current activities at HP and our partners, and some other related work.

6.1 Work at HP

We have several experimental agent projects at HP labs. In collaboration with the University of Utah, one group has worked on distributed measurement involving the use of mobile agents deployed at startup to the measurement site [Griss96]. These agents, implemented as COM or OCX components in the CWave system, utilize a publish/subscribe bus for message-based communication between agents. As part of this infrastructure, the agents include standard libraries of measurement objects and methods in support of the

task at hand; in particular, support for loading and running multi-threaded scripts in VBScript, JavaScript or Perl. Measurements are then initiated, changed, and updated by dynamically sending scripts which are evaluated in the context of the local agent. The script is responsible for performing the measurements and providing any necessary actuation/control of the attached devices. Agents may be logically grouped together to provide a higher-level functionality, which is undertaken via communication and negotiation amongst the agents within the group.

A second group at HP labs has implemented a dynamic agent infrastructure[Chen98], used for data-mining and dynamic workflow integration and provisioning. The infrastructure is Java-based, light-weight, and extensible. It differs from other agent platforms and client/server infrastructures in its support of dynamic behavior modification of agents. A dynamic-agent is not designed to have a fixed set of predefined functions but instead, to carry application-specific actions, which can be loaded and modified on the fly. This allows a dynamic-agent to adjust its capability for accommodating environment and requirement changes, and play different roles across multiple applications. The system can also load new interpreters for new domain-specific languages/ontologies, such as using an XML parser and dispatcher for an XML-based language, or a Java workflow engine to coordinate agent interactions[Chen99]. Part of the dispatcher and interpreter can be generated from the DTD.

A third group, based at HPLabs Bristol, has focussed primarily on the economic aspects of agent mediated electronic commerce [Preist99]. Early work looked at the role of negotiation in task allocation. A prototype system in the domain of computer support was developed, using BDI agents and a new form of facilitator architecture [Pearson97]. One member of the team, Ian Dickinson, played a key role in the FIPA97 ACL specification, chairing the relevant subcommittee and authoring the final document. The team has developed negotiation algorithms for agents participating in electronic marketplaces and analyzed the dynamics of the resulting behavior [Cliff98, Preist99a]. This work suggests that simple, adaptive, negotiation algorithms can be very efficient in marketplaces where price is the only deciding factor. Furthermore, agents can be used to design new kinds of markets which are more secure and fairer than existing ones [Preist99b]. Recent work together with the economist Nir Vulkan of the University of Bristol has focused on negotiation algorithms which combine belief-based and reinforcement learning, switching between the two techniques when statistical analysis determines it is appropriate. This team has also looked at human/agent interaction. If an agent is to negotiate on your behalf, how can you specify your preferences to it in an efficient way? Recent experiments have investigated the effect of using different agent interfaces to control an agent in an electronic market.

HP's technology for constructing E-services, E"Speak, (www.e-speak.hp.com) provides support for naming, communication and XML-based interaction in a very agent-oriented way. It defines vocabularies and messages using an XML encoding as described above.

6.2   Work elsewhere

The University of Massachusetts at Amherst's Little-JIL process programming language demonstrates how a workflow language can be used to coordinate agents[Sutton97], such as those of the GPGP multi-agent scheduling system. In Little-JIL a process is specified as a hierarchical decomposition into steps and substeps. This process specification can be also used as a specification of the way in which the available agents are to be coordinated to execute the defined process, determining the order and interactions of the agents.

At UCI, small, intelligent data agents have been defined to perform a range of simple or complex tasks, such as automatic notification via email of the availability of a report, sending a reminder of a scheduled meeting, advising a user of alternate meeting times as the result of a scheduling conflict, or even actively going out and doing research based on some specified criteria. Agent's activities, from the viewpoint of the workflow system, should allow the activity to be accomplished by either a human or non-human agent or both[Bolcer97]. People participating in the workflow should be allowed to hand off or delegate activities to an agent or take ownership of the activity back from one[Bolcer99]. Similarly, agents can delegate tasks to sub-agents or request the help of human participants as needed. The workflow-based management of multiple agents allows end-users to accomplish tedious or repetitive tasks with minimal effort and avoids the "keep checking back" syndrome.

The University of Ottawa's Iconic Modeling Tool (IMT) for the Agent Inception System [Falchuk99] uses visual techniques to assemble and control mobile agent programs. It uses UML interaction diagrams to specify agent flow (itinerary) between sites.

Other interesting work includes Columbia's Worklets[Kaiser99], a light-weight process workflow system implemented in Java and the JPython scripting language, and the agent enhanced workflow at BT [Shepherdson99], implemented using the Zeus toolkit[Nwana99]. Also COBALT [Bénech97] developed jointly by IRIT and HP Labs Bristol, uses KQML, CORBA, CIM, IDL and XML in various combinations for cooperative service and network management.

Microsoft® BizTalk™ is an XML-based platform neutral e-commerce framework (see www.microsoft.com/Industry/biztalk/default.htm, biztalk.org). The BizTalk framework is composed of: schema, products, and services.

7   Conclusions

As agent technologies and multi-agent systems become more widely used for constructing E-commerce systems, carefully constructed XML-based agent communication languages, more formal ontologies, modeling techniques to specify properties of individual agents and groups, and workflow methods and technologies to provide agent choreography will provide significant advantages.

Technologies such as HTTP, XML, agents, KQML and workflow will combine to produce the next generation of flexible, component software technologies, appropriate to rapid construction of these new applications. The rapid evolution of the XML/HTTP-based eCo, BizTalk and E'Speak specifications, communication and vocabulary frameworks give the flavor of how the future will unfold.

So how will we describe services using XML, and use our agents to mediate between them? That is another story, to be covered in a later article.

8   Acknowledgements

References

[Bach79]   *K Bach and RM Harnish, "Linguistic Communication and Speech Acts." MIT press, 1979.*

[Bénech97]   *D Bénech, T Desprats, and Y Renaud, "KQML-CORBA based architecture for intelligent agents, communication in cooperative service and network management", Proc 1st IFIP Conference on Management of Multimedia Networks and Services, July 1997.*

[Bolcer96]   *G Bolcer and R Taylor, "Endeavors: A Process System Integration Infrastructure," International Conference on Software Process (ICSP4), December, 2-6, 1996, Brighton, U.K.*

[Bradshaw97] JM Bradshaw, "Software Agents," MIT Press, 1997.

[Chen98]   *Q Chen, P Chundi, U Dayal, and M Hsu, "Dynamic Agents for Dynamic Service Provisioning," accepted for publication, Intl. Conf. on Cooperative Information Systems, August 1998.*

[Chen99]   *Q Chen, M Hsu, U Dayal, and M Griss, "Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation," HP Laboratories Report, Oct 1999. (Submitted to Autonomous Agents 2000)*

[Cliff98]   *D Cliff and J Bruten, Market Trading Interactions as Collective Adaptive Behaviour,, in 'Proceedings of 5th International Conference on Simulation of Adaptive Behaviour, MIT Press 1998.*

[Falchuk98]   *B Falchuk and A Karmouch, "Visual Modeling for Agent-Based Applications". IEEE Computer, Vol. 31, No. 12, December 1998. Pp. 31 - 37.*

[Finan97]   *T Finin, Y Labrou and J Mayfield, "KQML as an Agent Communication Language," in Software Agents, J.M.Bradshaw (ed), MIT Press, Cambridge, Mass., p291-316, 1997.*

[Flammia99] G Flammia, "The Skinny on Meta-Data," IEEE Intelligent Systems, July/August, 1999, pp.20-22.

[Genesereth94]   MR Genesereth, and SP Ketchpel: "Software Agents", Communications of the Association for Computing Machinery, July 1994, pp 48-53.

[Glushko99]   *RJ Glushko, JM Tenenbaum, and B Meltzer. An XML framework for agent-based E-commerce. Communications of the ACM, Vol.42, March 1999.*

[Griss99]   *ML Griss, G Bolcer, L Osterweil, Q Chen, and RR Kessler "Agents and Workflow -- An Intimate Connection, or Just Friends?", Panel, TOOLS99 USA, Aug 1999.*

[Griss96]   *ML Griss and RR Kessler, "Building Object-Oriented Instrument Kits," Object Magazine, Apr 1996.*

[Gray95]   *RS Gray. Agent Tcl: A transportable agent system. In Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95), Baltimore, Maryland, December 1995*

[Huhns98]   *MN Huhns and MP Singh, "Readings in Agents", Morgan-Kaufman, 1998*

[Jablonkski96]   *S Jablonski, C Bussler Workflow Management : Modeling Concepts, Architecture and Implementation by (October 1996) International Thomson Publishing; ISBN: 1850322228*

[Jennings98]   *NR Jennings and MJ Wooldridge, "Agent Technology," Springer, 1998*

[Kaiser99]   *G Kaiser, A Stone and S Dossick, A Mobile Agent Approach to Light-Weight Process Workflow, In Proc. International Process Technology Workshop, 1999.*

[Kimbrough97]   *SO Kimbrough and SA Moore, "On Automated Message Processing in Electronic Commerce and Work Support Systems: Speech Act Theory and Expressive Felicity," Trans. On Information Systems, 14:4 (October 1997, pp. 321-367.*

[Maes99]   *P Maes, RH Guttman, and AG Moukas. Agents that buy and sell. Communications of the ACM, Vol.42, No.3, March 1999, pp.81-91.*

[Meltzer98]   *B Meltzer and R Glushko. XML and Electronic Commerce, ACM SIGMOD. 27.4 (December 1998)*

[Moore98]   *SA Moore, "KQML and FLBC: Contrasting Agent Communication Languages," proceedings. 32nd Hawaii international conference on system sciences, 1998.*

[Nwana99]   *H Nwana, D Nduma, L Lee, J Collis, "ZEUS: a toolkit for building distributed multi-agent systems", in Artificial Intelligence Journal, Vol. 13, No. 1, 1999, pp. 129-186.*

[O'Brien98]   *PD O'Brien and R Nicol, FIPA: Towards a standard for intelligent agents. BT Technical Journal, 16(3), 1998.*

[Pearson97]   *S Pearson, CW. Preist, TS. Dahl, E de Kroon, An Agent-Based Approach to Task Allocation in a Computer Support Team, in Proceedings of the 2nd international conference on practical application of intelligent agent and multi agent technology, 1997.*

[Petrie96]   *C Petrie, " Agent-Based Engineering, the Web, and Intelligence," IEEE Expert December 1996.*

[Preist99]   *CW Preist, Economic Agents for Automated Trading, in 'Software Agents for Future Communications Systems', ed A. Hayzelden & J.Bigham, Springer 1999*

[Preist99b]   *CW Preist, Commodity trading using an agent-based iterated double auction, in 'Proceedings of the third international conference on Autonomous Agents', 1999.*

[Preist99a]   *CW Preist and M van Tol, Adaptive Agents in a Persistent Shout Double Auction, in Proceedings of the 1st International Conference on Information and Computation Economies, Elsevier 1999.*

[Schmidt99]   *MT Schmidt, " The Evolution of Workflow Standards," IEEE Concurrency, July-September 1999, Vol. 7, No. 3, p. 44-52.*

[Sharma99]   *T Sharma, "E-Commerce Components," Software Development, Vol 7, August 99*

[Shepherdson99]   *JW Shepherdson, SG Thompson & BR Odgers, "Cross organizational Workflow Coordinated by Software Agents," BT Laboratories, United Kingdom, February 15, 1999. (WACC '99 (Work Activity Coordination and Collaboration) Workshop Paper, February 1999) . (http://www.labs.bt.com/projects/agents/index.htm)*

[Shoham93]   *Y Shoham, "Agent-Oriented Programming," Artificial Intelligence, Vol. 60, No. 1,1993, Pp. 139-159.*

[Singh98]   *MP Singh, "Agent Communication Languages: Rethinking the Principles," IEEE Computer, Vol. 31, No. 12, December 1998, pp. 40-47.*

[Sutton97]   *SS Sutton Jr. and LJ Osterweil, "The design of a next generation process programming language," Proceedings of ESAC-6 and FSE-5, Springer Verlag, 1997, pp. 142-158.*

Some WWW sites

ActionWorks Metro   *www.actionworks.com*
Agent TCL   *agent.cs.dartmouth.edu/*
Biztalk   *www.microsoft.com/industry/biztalk/ and biztalk.org/BizTalk/default.asp.*
Commerce One's Common Business Library   *www.commerceone.com*
CommerceNet eCo framework.   *eco.commerce.net.*
CommerceNet   *www.commercenet.com,*

Common Business Library.	*www.commerceone.com/solutions/xml/cbl/cbl.htm.*
Endeavors	*www.ics.uci.edu/pub/endeavors/endeavors.html, www.endtech.com.*
GRASSHOPPER	*www.ikv.de*
HP ChangeEngine	*www.ebizsoftware.hp.com/main1.html. www.hp.com/e-services*
HP E-services	*www.hp.com/e-services*
HP E''Speak	*www.hp.com/e-speak.*
IBM FlowMark	*www-4.ibm.com/software/ts/mqseries/workflow*
JatLite	*www.java.stanford.edu*
JUMPINGBEANS	*www.jumpingbeans.com*
KIF	*logic.stanford.edu/kif/specification.html.*
Little-Jil	*laser.cs.umass.edu/process.htm*
Ontology.org	*www.ontology.org,*
ONTOLOGY.Org	*www.ontology.org;*
RDF	*www.w3.org/TR/PR-rdf-schema/ for specs, and www.w3.org/RDF/ for tools, etc.*
RosettaNet PIPs	*www.rosettanet.org.*
Verve Workflow	*www.verve.com/workflow*
Worklets	*www.psl.cs.columbia.edu/Worklets*
XML portal	*www.xml.org*
Zeus	*www.labs.bt.com/projects/agents/index.htm*

9 Biography

Martin L. Griss is a Principal Laboratory Scientist at Hewlett-Packard Laboratories, Palo Alto, California. At HP since 1982, he has researched software engineering processes and systems, systematic software reuse, object-oriented development and component-based software engineering. He is currently working on model-driven, agent-based application and e-service management systems. He created the first HP corporate reuse program, and led HP efforts to standardize UML for the OMG. He was previously director of the Software Technology Laboratory at HP Laboratories, and an Associate Professor of Computer Science at the University of Utah. He is co-author of the book "Software Reuse: Architecture, Process and Organization for Business Success". He has written numerous articles on software engineering and reuse, and lectures widely on systematic reuse and software process improvement. He is a member of the ACM SIGSOFT Executive Committee, and of the joint IEEE/ACM software engineering education project. He is a member of the SSR steering committee, the ICSE99, UML99, OOPSLA99, TOOLS99 and ICSE2001 program committees, and the ICSE2002 organizing committee.