Software Agents as Next Generation Software Components

Martin L. Griss, Ph.D.
Laboratory Scientist
Software Technology Laboratory
Hewlett-Packard Company, Laboratories
Palo Alto, CA, USA

(*Chapter 36 in **Component-Based Software Engineering: Putting the Pieces Together,***
Edited by George T. Heineman, Ph.D. & William Councill, M.S., J.D., May 2001,
Addison-Wesley

Introduction

The increasing volume of Business-to-Consumer (B2C) and Business-to-Business (B2B) Internet traffic provides great opportunity to delegate information search, analysis, and negotiation to automated assistants. These new classes of applications demand flexible, intelligent solutions. Distributed software agents offer great promise, building on increasingly pervasive message-based middleware and component technology, Java, the Extensible Markup Language (XML), and the Hypertext Transfer Protocol (HTTP). Agents are specialized kinds of components, offering greater flexibility than traditional components. As will be explained, agents use dynamically adaptable rich message-based interaction, and flexible knowledge-based techniques to make it much easier to build and evolve systems as requirement and technologies change. At HP Laboratories, our E-commerce research focuses on simulation, agent communication, mobile appliances and personal agents, and multi-agent systems (Griss, 2000b; Chen, 1999). We are interested in how agents are defined and constructed, how they communicate, and how collaborations between groups of agents can be established and controlled. We combine agents and workflow to provide significant benefits beyond those traditionally associated with components and scripting. There are many kinds of software agents, with differing characteristics such as mobility, autonomy, collaboration, persistence and intelligence. This chapter surveys some of these agent capabilities. More details can be found in books (Huhns, 1998; Jennings, 1998; Bradshaw, 1997), papers (Genesereth, 1994; Maes, 1999; Griss, 1999, 2000) and web sites (http://agents.umbc.edu/, http://www.hpl.hp.com/reuse/agents/, http://www.agent.com/).

E-Commerce Requires Flexible Implementation

To integrate B2B business processes across enterprises, the next generation of E-commerce applications will need to be larger, more complex, and flexible (Sharma, 1999). Because these applications will be assembled using components written at different times by different developers, the developers need powerful ways to quickly build flexible systems and services to provide a more compelling user experience to more users.

Agent-oriented E-commerce systems have great potential for these E-commerce applications. Agents can dynamically discover and compose E-services and mediate interactions. Agents can serve as delegates to handle routine tasks, monitor activities, set up contracts, execute business processes, and find the best services (Maes, 1994; Chen, 1999). Agents can use the latest web-based technologies, such as Java, XML, and HTTP. These technologies are simple to use, ubiquitous, heterogeneous and platform neutral. XML will likely become the standard language for agent-oriented E-commerce interaction to encode exchanged messages, documents, invoices, orders, service descriptions, and other information. (Glushko, 1999; Meltzer, 1998). HTTP, the dominant WWW protocol, provides many services, such as robust and scalable web servers, firewall access, and levels of security.

Agent Types

There are many definitions of agents, but many people agree that: "*an autonomous software agent is a component that interacts with its environment and with other agents on a user's behalf."* Some definitions emphasize one or another characteristics such as mobility, collaboration, intelligence or flexible user interaction. Organizations such as the Foundation for Intelligent Physical Agents (FIPA) define reference models, mechanisms, and agent communication language standards (O'Brien, 1998; http://www.fipa.org/). There are several different kinds of agents:

- **Personal agents** interact directly with a user, presenting some "personality" or "character", monitoring and adapting to the user's activities, learning the user's style and preferences, and automating or simplifying certain rote tasks. Toolkits such as Microsoft® Agent (http://www.microsoft.com/msagent) which is a set of software services that supports the presentation of software agents as interactive personalities, includes natural language and animation capabilities. Microsoft's Agents "Bob" or "Paper Clip" are simple examples built using this technology. The site www.bottechnology.com provides a survey of some personal agent technologies and www.redwhale.com describes personalizable interfaces using aspects of agent technology. Many of the information agents explored at MIT fall into this category.
- **Mobile agents** are sent to remote sites to collect information or perform actions and then return with results. "Touring" agents visit sites to aggregate and analyze data, or perform local control. They are typically implemented in Java, TCL (Ousterhout, 1990), VB Script, Perl or Python (http://www.python.org/). Such data intensive analysis is often better performed at the source of the data rather than shipping raw data; examples include network management agents and Internet spiders.
- **Collaborative agents** communicate and interact in groups, representing users, organizations and services. Multiple agents exchange messages to negotiate or share information. Examples include online auctions (e.g. Michigan Auction Bot or Pricebots at priceline.com), planning, negotiation, logistics and supply chain and telecom service provisioning. COBALT (Bénech, 1997) uses KQML, CORBA, CIM, IDL and XML for cooperative service and network management.

More than one hundred agent system and toolkits have been developed and described in the literature or on the web, ranging from A-Z (e.g., Agent-TCL, Aglets, Concordia, FIPA-OS, Grasshopper, Jackal, JADE, JATLite, Jumping Beans, Voyager, and Zeus), with different mixes of mobility, adaptability, intelligence, ACL and multi-language support. Agents are often implemented using distributed objects, active objects, and scriptable components. Agents are usually driven by goals and plans (rather than procedural code) and have "business" or "domain" knowledge. Often, agents differ more from each other by the knowledge they have and roles they play, than from differences in their implementing classes and methods.

In the rest of the chapter, I explain how agents are really next-generation components and discuss some features and variants of typical agent systems. I then show how agents communicate using XML encoded messages, and finally how agent conversations can be "choreographed."

Agents Are Next Generation Components

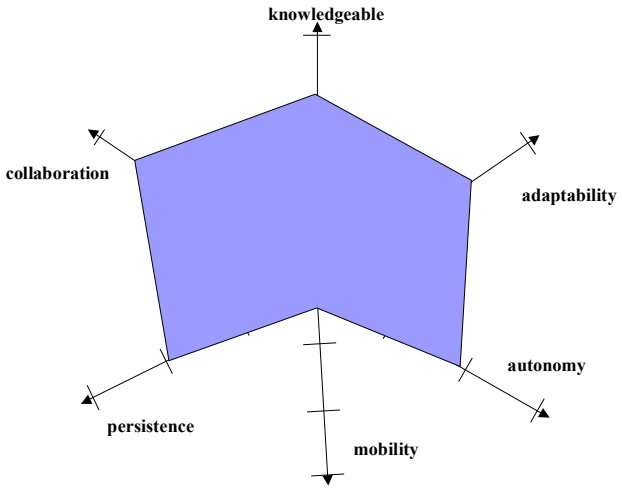
Agent-oriented software development extends conventional component development, promising more flexible componentized systems, less design and implementation time, and decreased coupling between agents. In the same way that models, collaborations, interaction diagrams, patterns and aspect-oriented programming help build more robust and flexible components and component systems, the same techniques can be applied to agents and agent systems (Kendall, 1999; Gschwind, 1999; Griss, 2000b).

Agents rely on an infrastructure that provides services and mechanisms, allowing agents to have simpler interfaces and be more composable. Agent-oriented programming (AOP) (Shoham, 1993) decomposes large complex distributed systems into relatively autonomous agents, each driven by a set of *beliefs*, *desires* and *intentions* (BDI). An agent-oriented developer creates a set of agents (each with their own beliefs and intentions) that collaborate among themselves by exchanging carefully structured messages. Some agents may be implemented using Artificial Intelligence (AI) technology; others will use conventional component technology. As we shall see, agent technology based on autonomous, communicating, adaptable agent components promises quicker, more flexible development of complex, distributed, evolving systems.

Major Characteristics of Different Kinds of Agents

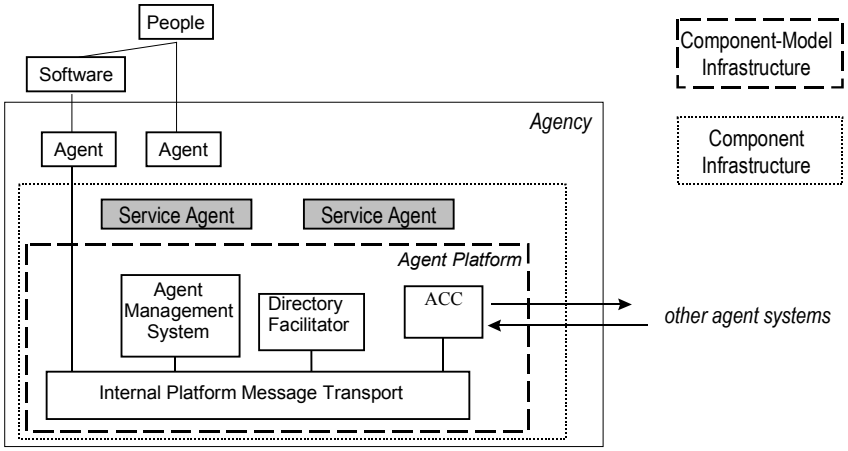
An agent system is essentially a component system exhibiting several of the characteristics pictured in Figure 1. The greater the area enclosed on the diagram, the more "agent-like" the component system. These six orthogonal characteristics work together to make agent-oriented systems more flexible and robust to change. These characteristics are supported by mechanisms and interfaces in the underlying infrastructure and by models and policies configured in each agent or group of agents.

Figure 1 - Agent Dimensions



1. **Adaptability -** the degree to which an agent' s behavior may be changed after it has been deployed.
2. **Autonomy-** the degree to which an agent is responsible for its own thread of control and can pursue its own goal largely independent of messages sent from other agents.
3. **Collaboration -** the degree to which agents communicate and work cooperatively with other agents to form multi-agent systems working together on some task.
4. **Knowledgeable -**the degree to which an agent is capable of reasoning about its goals and knowledge.
5. **Mobility** - the ability for an agent to move from one executing context to another, either by moving the agent's code and starting the agent afresh, or by serializing code and state, allowing the agent to continue execution in a new context, retaining its state to continue its work.
6. **Persistence -** the degree to which the infrastructure enables agents to retain knowledge and state over extended periods of time, including robustness in the face of possible run-time failures.

Figure 2 - Agency Reference Model



The Structure of an Agent System

The term *Agency* refers to the conceptual and physical location in which agents reside and execute. Using the terms of this book, the heart of the agency is the *agent platform*, the component model infrastructure that provides local services for agents and includes proxies to access remote services. An agent system can provide particular services using *service agents*, such as a Broker, Auctioneer or Community Maker agents (For example, see HP E-Speak http://www.hp.com/e-speak); these service agents form the component infrastructure. As described in *ChapterWeinreicherSametinger*, the interaction standard of a component model defines the nature of an interface. An agent system is composed of components with simple interfaces, but complex behavior results from the messages that the components process and communicate with each other. The agents are only able to communicate with each other because they all conform to some common, well-defined interaction standard.

The agency serves as a "home base" for locating and messaging mobile and detached agents, and collecting knowledge about groups of agents. Services include agent management, security, communication, persistence, naming, and agent transport in the case of mobile agents. In addition to basic agent infrastructure and agent communication, a FIPA compliant agent system, such as Zeus (Nwana, 1999), provides additional services in the form of specialized agents, residing in some (possibly remote) agency. Figure 2 summarizes the overall agent system architecture:

1. Agent Management System (AMS) - controls creation, deletion, suspension, resumption, authentication, persistence and migration of agents. Provides "White Pages" to name and locate agents.
2. Agent Communication Channel (ACC) - routes messages between local and remote FIPA agents, realizing messages using an agent communication language.
3. Directory Facilitator (DF) - provides "Yellow Pages" service for FIPA agents that register agent capabilities so an appropriate task-specific agent to handle the task can be found.
4. Internal Platform Message Transport - provides communication infrastructure.

Many agent capabilities for naming and communication can be implemented by using HTTP and XML, such as XML structured documents for messages, and URL/URIs for naming and locating. Agents may have a globally unique name or they may have one or more local names based on the agency or agent group. To provide a systematic and consistent way for agents to exchange messages with each other, most agent systems use a special Agent Communication Language (ACL); the ACL becomes associated with the

component model. The agent platform and additonal service agents can monitor and control these message exchanges to insure that they conform to a desired protocol, or at least do not violate "rules of engagement.".

The Language Problem

The interaction that a component supports is generally described by interfaces, defined using an IDL (see Chapter 31), that contains a syntactic description of each method, its parameter names and types, return values, and possible exceptions. The semantic knowledge of these interfaces, however, must be understood from documentation, though a protocol may be used to provide precise specification of the ordering and interaction of methods. As a component-based system increases in size and complexity, the designer often creates standardized interfaces in the component infrastructure, making it increasingly rigid and difficult to extend to meet needed change.

Instead of defining many interfaces and methods, the agent approach is to use a simple interface with more complex, structured messages. These messages can be extended dynamically as the system evolves, avoiding a costly re-design and re-implementation. XML provides an attractive way to encode these messages with structure defined by an XML Document Type Definition (DTD). It is easy to define an *ad hoc* XML-based language for each application or domain. However, these *ad hoc* languages have no well-understood relationship to each other, are hard to extend, and quickly lead to a large number of incompatible languages.

Multi-agent Communication Languages

A standard *agent communication language* (ACL) carefully defines the overall structure and standard patterns of interaction between agents in an extensible way. An ACL factors messages into several relatively independent parts: the message type, addressing, context, and the content of the message. Some parts of the message indicate the domain being described; others described the expected conversation pattern. This approach makes it easier to dynamically extend agents to new problem areas, yet still have the system check conformance to expectations, and enable the component model infrastructure to manage messages and agents.

```
<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 3: Sample KXML message

Most ACLs are loosely based on Speech Act Theory (SAT) (Bach, 1979), a linguistic theory concerning how people communicate and engage in stylized conversations using typed messages. The type indicates the key intent of a message (e.g., request, command, information or error), suggesting how messages should be processed individually, and how they should be handled as part of a multi-agent conversation. There are a number of agent communication languages, with KQML (Finin, 1997) and FIPA ACL (http://www.fipa.org/), the most well known. 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 (Moore, 1998). Each ACL factors messages in different ways, and provides different standard reusable parts. KQML has over 30 standard types, while FIPA ACL has fewer but they can be combined in more ways.

We use an XML encoding, combining aspects of several ACLs, which we call *KXML 1.0*. Figure 3 contains a sample message. Each part of the message has a distinct role:

- **Message type** – These building blocks of conversations, such as ADVERTISE, BROKER, ASK-IF, TELL, SORRY, ERROR, REQUEST, INFORM, REFUSE, FAILURE, are used with specified conversation protocols to allow the system to monitor and control the progress of conversations, and confirm the compatibility with conventions, with a need to examine detailed content.
- **Address** - *Sender* and *receiver* identify the participants in the conversation. *Originator* is used for forwarded or brokered messages.
- **Context** - *Reply-with* and *in-reply-to* are used to connect a series of messages to a specific conversation and to each other. *Conversation-id* connects a set of related messages; *reply-by* sets a deadline for a timeout and *protocol* identifies a specific conversation type, which details an expected pattern message exchange between agents.
- **Content** - *Vocabulary* (also called ontology), identifies the domain of the message, for example "buying," "payment," or "banking". The Vocabulary defines objects and legal data types, action words, and attributes. *Language* specifies the how the content is expressed, using an AI language (LISP, KIF, or PROLOG), an agent language (FIPA-SL or KQML), or a scripting language (TCL, Perl or VB Script), perhaps encoded in XML. Finally, the content is a statement or an expression in the chosen language using terms from the chosen vocabulary, or a nested message.

KXML 1.0 provides the syntax of the messages and lists all the legal message types, additional constraints on the attributes, and the explicit expression and statement structure (Griss, 1999a).

Vocabulary - What the Words Mean

Agents communicate using words that are meaningful and understood in the given context. The set of words, their relationships, and their meanings is known as an ontology or vocabulary. 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 understand what the words mean. Some words can be defined formally in terms of other words, but others must be described in a standard "dictionary," telling the agent program(mer) 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 in each case. Similar words can have distinct meanings; for example, in banking, the words "charge", "fee", "levy", "settlement", "payment", and "interest" have distinct meanings and relationships.

The vocabulary specifies the concepts in a domain and defines the meaning and intent of each word. The vocabulary defines key relationships between words, such as synonyms or generalizations. A vocabulary can be created by using agreed terminology from an international standards committee (e.g., UML, CYC), or created for specific markets by industry groups (e.g. "banking," or "auto parts exchange"). Vocabularies from different sources will be used, and so the ACL needs to specify the selected vocabulary using the *vocabulary* attribute. Several organizations are standardizing vocabularies and related E-commerce frameworks. For example, BizTalk (www.biztalk.org), Ontology.org (www.ontology.org), CommerceNet (www.commercenet.com), CommerceOne's (www.commerceone.com), and RosettaNet (www.rosettanet.org). HP E"Speak provides a basic vocabulary for describing and managing other vocabularies (www.hp.com/e-speak).

There are several ways in which a useful vocabulary may be represented:

1. A natural language "dictionary," listing terms, their meanings and intended use.
2. An XML DTD defining terms, and some relationships (attribute syntax and some typing). Comments or an external dictionary explain the meaning, additional relationships and intent. A DTD alone can not completely define meaning, nor fully model all data types or multiplicity.
3. An object oriented XML Schema (see http://www.w3c.org/) can more precisely define structured datatypes, can use element inheritance, and provide more precise control over how many of each type of element and attribute are permitted. CommerceOne has used this approach.

4. A full modeling language and tool, such as UML or Ontolingua, defining vocabulary elements (classes or types,) relationships (inheritance or associations) and semantics and constraints using OCL and comments. For example, RosettaNet uses UML to precisely model elements in its technical dictionary.

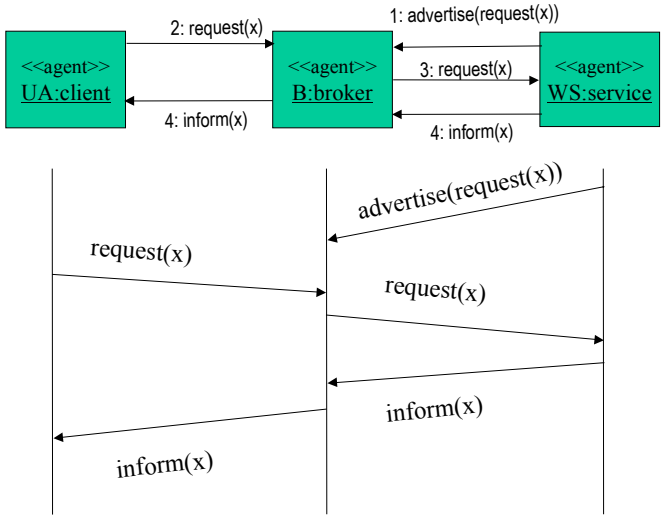
Multi-agent coordination

While some agents are used individually, several agents can collaborate to perform more complex tasks. For example, to purchase books, a group of agents will exchange messages in a conversation to find the best deal, bid in an auction, arrange financing, select a shipper, and track the order.

Multi-agent systems (groups of agents collaborating to achieve some purpose) are critical for large-scale E-commerce, especially B2B interactions such as service provisioning, supply chain, negotiation, and fulfillment. The grouping can be static or dynamic. The conversation can include interactions between people and agents, or between agents. We need to coordinate the interactions between the agents to achieve a higher-level task, such as requesting, offering and accepting a contract for some services. We call this "choreography" or "conversation management." An agent group will have a series of stylized message exchanges to accomplish this task, perhaps by advertising and using a brokered service (Figure 4), bidding during an auction, responding to a call for proposals, or negotiating a price.

There are several possible levels of choreography of the expected message sequences, depending on the need for a relatively "loose" or more "tight" control of allowed interactions (Griss 1999a). These include:

Figure 4 - Message Sequencing



1. *Built-in message type assumptions* - The message type itself directly suggests a standard expected message sequence. The agent system can monitor compliance. For example, an agent might issue a "request" to one or more agents. Other agents would respond, either to "inform" of their ability to provide this service, or to indicate that they "did not understand" the message. The system can also monitor timeouts, using the "reply-by".
2. *Rules* - A set of rules can be defined for any agent, or community of agents. For example, one could customize an agent's time to wait for a response, the number of other agents it can talk to at one time, and what sort of responses to make to a specific message from various types of agent. Not all request/response patterns are constrained by the rules (Moore, 1998). A standard rule language can be used, such as the forward chaining rule system provided by Zeus (Nwana, 1999).
3. *Conversation protocols* - Often a group of agents must "lock-step" through a standard protocol of messages; if A says "x" then B says "y"; for example, bidding during an auction, or responding to a

call for proposals. These protocols can be expressed as finite state machines, UML state charts or petri-nets. Each participant can have the same or a compatible model, stepped through explicitly to determine the action and response any incoming message. Figure 4 shows UML diagrams of the "broker" message type, used by a weather service agent, WS. First, WS advertises to broker B that it can respond to certain requests by sending the message "`advertise(request(x))`" to B. User agent, UA, issues a "`broker(request(x))`"; it will obtain the answer from WS, via a "`request(x)`" which then responds with "`inform(answer)`". B relays this to UA with its own "`inform(answer)`".

4. *Workflow* - When the conversation coordination for interactions of multiple agents and humans needs to be more complex and precise, a workflow system can be used. A workflow system allows the explicit definition (e.g. as a graphical model) and control of a group of participants that execute a (business) process (Schmidt 1999). Many E-commerce applications involve some form of inter- or intra- organizational workflow. A workflow system automates (part of) a business process, passing documents, information or tasks from one participant to another for action, according to a set of rules. Workflow systems such as ActionWorks Metro, Endeavors, HP ChangeEngine, IBM FlowMark, Little-JIL, or Verve Workflow describe allowed connections between participants, desired and exceptional processing conditions, the assignment of roles, and the request and allocation of resources. For example, a telecom management system alerting a human operator, or assigning a repair or provisioning engineer. This use of workflow can be seen as a "next generation scripting language" (Griss, 1999). A single coordinating agent ("workflow manager") with full workflow, could assign tasks and monitor progress for each member of the group; alternatively a piece of the workflow structure could be assigned to an agent with the role it is to play (say as "buyer" or "seller").

Workflow and agents can be combined in several ways (Chen, 1999; Griss 1999). Agents can collaborate to perform a workflow, *e.g., telecom provisioning (BT), or service provisioning (HP).* Agents can represent the participants and resources, the society of agents collaborate to enact the workflow. Agent systems have been used to implement new workflow systems (Jennings, 1996) or augment existing workflow systems (Shepherdson, 1999). Agents can be used to make workflow more intelligent, e.g., by adding negotiation, reasoning at decision points. Other examples include the Little-JIL process programming language demonstrates how a workflow language can be used to coordinate agents (Sutton, 1997). Worklets is an agent-like light-weight process workflow system implemented in Java and the JPython. (Kaiser, 1999).

Related Work

We have several experimental agent projects at HP Labs in Palo Alto and Bristol, focusing on E-commerce and application management. The CWave light-weight mobile agent system and visual toolkit was developed for application management and process control using COM/OCX, a publish-subscribe bus and VB Script (Griss, 1996; Mueller-Planitz, 2000). Agents include standard libraries of measurement objects and methods to initiate, change, and dynamically update measurement.(See http://beast.cs.utah.edu). We have implemented a light-weight, dynamic agent infrastructure in Java (Chen, 1998, 1999), used for data-mining and dynamic workflow integration and provisioning. Dynamic agents support "on demand" dynamic loading of new classes to extend agents with domain-specific XML interpreters, new vocabulary parsers or workflow. Several projects have focussed on the agent mediated E-commerce, looking at negotiation in task allocation (Preist 1999), implementing a new facilitator architecture (Pearson, 1997), and playing a key role in the FIPA97 ACL specification. The team has developed and analyzed negotiation algorithms for agents participating in electronic marketplaces (Cliff, 1998; Preist, 1999a). My group is developing an experimental web-based economic simulation environment for a shopping mall, and integrating that with personal agents and mobile appliances based on HP CoolTown (http://www.cooltown.com) (Griss, 2000), using Zeus (Nwana, 1999) as base. Wealso plan to incorporate HP's technology for constructing E-services, E-Speak, (http://www.e-speak.hp.com/).

Agents can perform a range of simple or complex tasks, such as automatic notification via email of the availability of a report, sending a reminder or re rescheduling a meeting (Bolcer, 1996), or negotiating on a users behalf and at (Maes, 1999). The Iconic Modeling Tool (Falchuk,1999) uses visual techniques and UML to assemble and control mobile agent programs and itinerary. Microsoft® BizTalk™ is an XML-

based platform neutral E-commerce framework (see http://biztalk.org/). The BizTalk framework is composed of: schema, products, and services.

Conclusions and Future Work

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 vocabularies, 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, e-services, KQML/FIPA ACL 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, UUDI, .NET and E'Speak specifications, communication and vocabulary frameworks give the flavor of how the future will unfold.

Research is needed to make it easier to define and implement different agent systems directly in terms of their capabilities. An agent, or set of compatible agents, will be constructed by combining aspects and components representing key capabilities. UML models of vocabularies, workflow, role diagrams, patterns, and feature trees will drive aspect-oriented generators to create highly customized agent systems (Kendall, 1999; Gschwind, 1999; Griss, 2000b). See Chapter *HoustonNorrisChapter* for discussion of some of these technologies.

Multi-agent systems can reveal interesting kinds of "emergent behavior," in which autonomous agents can work together in ways that have not been explicitly programmed, potentially producing unpredictable or unexpected results. Agents can discover other agents and form dynamic groups, new behavior modules can be downloaded to adapt the basic behavior of an agent, new vocabularies and protocols can be installed, and knowledgeable, adaptive agents can use machine learning techniques to change their responses to events and messages. Research is being directed at ways of predicting, simulating, exploiting and controlling this emergent behavior. Techniques based on rules of engagement, monitoring agents, and simulators are being explored to understand and manage these effects. For example, see the Swarm multi-agent simulation system, www.swarm.org .

In summary, agent technology has the potential to be more flexible than programming with traditional, more static components. Most agent and e-service systems offer several capabilities that work together to provide this flexibility, that promises quick development of flexible, complex systems, and more effective handling of evolution, distribution, and complexity. First, all agents use the same, simpler interfaces, with major differences in behavior provided by richly structured messages, and dynamically grown databases of knowledge within the agents. Second, message structure and interaction between agents is factored into relatively independent pieces, described and controlled by dynamically loadable protocols and vocabularies. This means that a particular agent or group of agents can be changed dynamically, adding new capabilities, without changing other agents. Finally, agents register and discover each other dynamically using the white pages and yellow page naming services, which means it is easy to add a new agent with new capabilities into a system, and have other agents find when requesting services from directory and facilitator agents. Furthermore, some of the agents can act as mediators, or intermediaries, transforming and delegating requests to other agents, and transforming and aggregating responses. All of these capabilities make it much easier possible to grow and evolve a system as requirements, technology and business change.

Acknowledgements

I greatly appreciate the many suggestions from my colleagues, David Bell, Pankaj Garg, Jon Gustafson, Reed Letsinger, Harumi Kuno, Vijay Machiraju, Chris Preist and Nanjangud C. Narenda.

References

K Bach and RM Harnish, *Linguistic Communication and Speech Acts.* MIT press, 1979.

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.

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

JM Bradshaw, *Software Agents*, MIT Press, 1997.

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

Q Chen, M Hsu, U Dayal, and M Griss, "Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation," *Autonomous Agents 2000*, June, Barcelona.

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

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

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.

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

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

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

ML Griss, "My Agent Will Call Your Agent … But Will It Respond?" Hewlett-Packard Laboratories, *Technical Report, TR-HPL- 1999-159*, Dec 1999.

ML Griss, "My Agent Will Call Your Agent, *Software Development Magazine,* Feb 2000.

ML Griss and R Letsinger, "Games at Work - Agent-Mediated E-Commerce Simulation, in *Workshop on Industrial Applications of Agents Workshop, Autonomous Agents 2000*, Barcelona, Spain, June 2000a *(To appear)*

ML Griss, "Implementing Product-Line Features By Composing Component Aspects, ", *Proceedings of 1st International Software Product Line Conference,* Denver, Colorado, August 2000 b (*To appear*)

T Gschwind, M Feridun and S Pleisch, "ADK - Building Mobile Agents for Network and Systems Management from Reusable Components," in *Proc. of ASA/MA 99*, Oct, Palm Springs, CA, IEEE-CS, pp 13-21. *(See http://www.infosys.tuwien.ac.at/ADK/)*

MN Huhns and MP Singh, *Readings in Agents*, Morgan-Kaufman, 1998

NR Jennings and MJ Wooldridge, *Agent Technology,* Springer, 1998

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

EA Kendall, "Role Model Designs and Implementations with Aspect-oriented Programming," in *Proc. of OOPSLA 99*, Oct, Denver, Co., ACM SIGPLAN, 353- 369.

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.

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

SA Moore, "KQML and FLBC: Contrasting Agent Communication Languages," in P*roceedings of 32nd Hawaii International Conference on System Sciences*, 1998.

C Mueller-Planitz, "CWave 2000 - *A Visual Workbench for Distriibuted Measurement Agents*, Phd Thesis, CS Dept, University of Utah, May 2000 (in process).

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.

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

Ousterhout, J, "TCL: An Embeddable Control Language", USENIX conference, 1990.

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.

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

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.

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

T Sharma, "E-Commerce Components," Software Development, Vol 7, August 99

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*)

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

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.