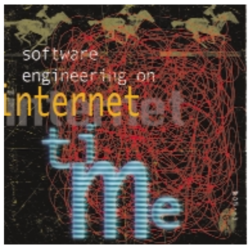

Accelerating Development with Agent Components

This analytical survey of agent components reveals that the technology will likely form the foundation for flexible, intelligent, Web-based enterprise application systems.

Martin L. Griss
Hewlett-Packard Laboratories

Gilda Pour
San Jose State University

As the demand for more flexible, adaptable, extensible, and robust Web-based enterprise application systems accelerates, adopting new software engineering methodologies and development strategies becomes critical. These strategies must support the construction of enterprise software systems that assemble highly flexible software components written at different times by various developers. Traditional software development strategies and engineering methodologies, which require development of software systems from scratch, fall short in this regard.

Component-based software engineering offers an attractive alternative for building Web-based enterprise application systems. CBSE works by developing and evolving software from selected reusable software components, then assembling them within appropriate software architectures. By promoting the use of software components that commercial vendors or in-house developers build, the component-based software development approach promises large-scale software reuse. Component-based software development has the potential to

- significantly reduce the development cost and time-to-market of enterprise software systems because developers can assemble such systems from a set of reusable components rather than building them from scratch;
- enhance the reliability of enterprise software systems—each reusable component undergoes several review and inspection stages in the course of its original development and previous uses, and CBSE relies on explicitly defined architectures and interfaces;
- improve the maintainability of enterprise software systems by allowing new, higher-quality components to replace old ones; and
- enhance the quality of enterprise software systems—application-domain experts develop components, then software engineers who specialize in CBSE assemble those components into enterprise software systems.[1]

Researchers generally agree that developers create software components primarily so that they can be reused in various software systems. They also generally accept that a component is a unit of independent deployment that interacts with its environment through its well-defined interfaces while encapsulating its implementation.

The component-based software development life cycle differs from traditional software development in many ways. The component-based software-system design phase includes new activities such as selection and creation of software architectures, as well as selection and customization of a set of software components. In addition, the implementation phase deals with the integration of a set of software components within appropriate software architectures. This requires developing wrappers that glue reusable components together to build the software system, rather than extensive coding to build a software system from scratch. Late integration of components developed by others eliminates the confidence usually drawn from integration testing in a traditional software engineering model. Developers must architect and design extensibility into a system and all its parts—or the components in the resulting system will not be independently producible and deployable.[1,2]

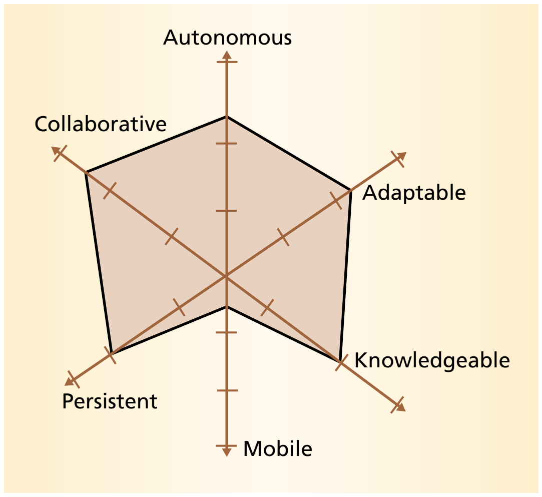

Figure 1. Components as agents. A component can be called an agent if it exhibits a combination of the characteristics shown here.

Software agents offer greater flexibility and adaptability than traditional components. Agent-oriented software engineering allows developers to use a set of high-level, flexible abstractions to represent and understand Web-based enterprise application systems. Rapid integration of distributed agents provides opportunities to build such software systems.

For example, the increasing volume of Internet traffic that various Web-based enterprise applications require has inspired the delegation of information search, analysis, and negotiation to automated assistants, implemented using software agent technology. Examples include Shopbots and Pricebots, which monitor product availability and price, then negotiate and complete sales of goods and stocks to optimize business-to-consumer and business-to-business interactions.

Agents communicate by passing dynamically adaptable rich messages, using flexible knowledge-based techniques to facilitate building and evolving software systems as software technologies and requirements change.[3] Developers use increasingly pervasive message-based middleware and component technologies, Java, Extensible Markup Language (XML), and the Hypertext Transfer Protocol (HTTP) to create agent-based enterprise software systems. New, mobile appliance-oriented application servers and portal technologies based on these technologies, such as HP's "totally e-mobile" Bluestone, provide a base for more robust agent-oriented, Web-based enterprise application systems. These technologies promise to make the use of mobile appliances, adaptive content, and software agents quicker and easier.

Intelligent agents—autonomous components that have their own goals and beliefs and can reason about their present and future behavior—offer ample opportunity for rapid, incremental development of Web-based enterprise application systems. Developers can apply these systems to a variety of complex, dynamic domains, ranging from e-commerce to human planetary exploration.

AGENTS: NEXT-GENERATION COMPONENTS

Although we lack a universal definition of agents, according to the commonly used one, an agent is a proactive software component that interacts with its environment and other agents as a surrogate for its user, and reacts to significant changes in the environment. We call a component an agent if it exhibits a combination of several of the following characteristics, as shown in Figure 1:

- *Autonomous.* The agent proactively initiates activities in accord with its goal, has its own thread of control, and can act on its user's behalf, largely independent of messages other agents send.
- *Adaptable.* Either its own learning, user customization, or downloading new capabilities can change an agent's behavior after deployment.
- *Knowledgeable.* The agent can reason about its goals, acquired information, and knowledge about other agents and users.
- *Mobile.* The agent can move from one executing context to another, either by moving its code and starting afresh or by serializing its code and state, continuing execution in a new context and retaining its state to continue its work.
- *Collaborative.* The agent can communicate and work cooperatively with other agents to form dynamic or static multiagent societies, collaborating to perform a task.
- *Persistent.* The infrastructure enables agents to retain knowledge and state over extended periods of time, including robustness in the face of possible runtime failures.

We view agents as next-generation components and agent-oriented software engineering as an extension of conventional CBSE. Developers can integrate different agent types—personal, mobile, and collaborative—to build agent-based enterprise systems in a wide variety of problem domains. Daemons are simple agents that patrol networks to find available resources. Other, more complex and intelligent agents navigate the Internet to collect relevant data, perform tasks, and even make decisions on behalf of their users. The new generation of intelligent software agents can manage, organize, and sift through enormous amounts of data on behalf of their users. For instance, agents in e-commerce applications can dynamically discover and compose e-services and mediate interactions. Agents can also serve as delegates to handle routine tasks, monitor activities, set up contracts, execute business processes, and find the best services.[4]

Agent-oriented programming[5] decomposes large and complex distributed systems into relatively autonomous agents. An agent-oriented developer creates a set of agents—each driven by its own beliefs, desires, and intentions—that collaborate among themselves by exchanging structured messages. Developers can apply and adapt design methods, architectures, patterns, and generators based on the Unified Modeling Language (UML) to aid in the rapid incremental development of agent-based systems.

Developers often use distributed objects, active objects, and scriptable components to implement agents. Usually driven by goals and plans rather than procedural code, agents encapsulate business or domain knowledge. They often differ more from each other by the knowledge they have and the roles they play than by the differences in their implementing classes and methods. Agents can use different mixes of mobility, adaptability, intelligence, agent communication languages (ACL), and multilanguage support. Developers can use either artificial intelligence or conventional programming technology to implement agents.

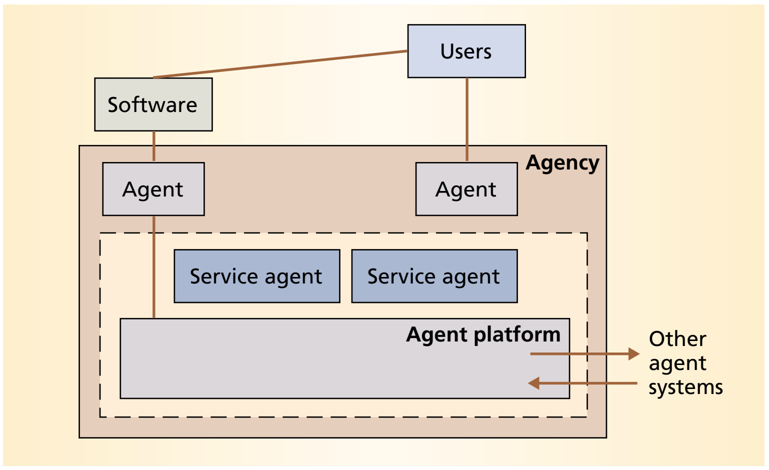

Figure 2. Agent system architecture. The agent platform serves as the system's core, while a component model infrastructure provides agents with local services and proxy access to remote services.

AGENT SYSTEM ARCHITECTURE

Agents reside and execute in a conceptual and physical location called an *agency*. The agency provides facilities for locating and messaging mobile and detached agents and for collecting knowledge about groups of agents. The agency's core is the agent platform, a component model infrastructure that provides local services for agents and includes proxies to access remote services such as agent management, security, communication, persistence, and naming. In the case of mobile agents, the agent platform also provides agent transport. Most agent systems provide additional services in the form of specialized agents that reside in some—possibly remote—agency. Some agent systems also include standard service agents, such as a broker, auctioneer, or community maker. These service agents augment the basic agent infrastructure. The agent platform and additional service agents can monitor and control message exchanges to detect any violation of the rules of engagement. Figure 2 shows the agent system architecture.

Multiagent communication

An agent system consists of components with simple interfaces. Much of the system's power comes from its loose coupling style, in which agents interact dynamically by exchanging asynchronous messages. An agent system's complex behaviors result from the messages the components process and exchange with each other. To communicate with one another, the agents must conform to some common, well-defined interaction standard. An ACL is a specialized declarative language that defines the overall structure and standard patterns of interaction between agents. The agent system associates the ACL with the component model and factors messages into several relatively independent parts: the message type, addressing, context, and message content. Some message parts describe the domain, others describe the expected conversation pattern. This approach makes it easier to dynamically extend agents to new problem areas while the system checks conformance to expectations and lets the component model infrastructure manage messages and agents.[3]

Several well-known agent communication languages, such as KQML[6] and FIPA ACL (http://www.fipa.org/), have been converted to a standard XML form. At HP Labs, developers define KXML 1.0 using an XML encoding that combines aspects of several ACLs.[3] Some ACLs support the beliefs, desires, and intentions model, while others make it easy to confirm that agents conform to organizational conventions and display acceptable behavior. Each ACL factors messages in different ways and provides different standard reusable parts. All ACLs make it possible to write agent components more quickly and more independently of other agents, allowing the agents to exchange messages so they can dynamically discover each other and the services they offer.

Agents interact with one another using a set of vocabularies—also called an ontology—grounded in the application domain. A vocabulary consists of a word set that describes things, attributes, and actions from some domain, their relationships and meanings, and how the agent system uses the vocabulary to structure interactions and access services. An international standards committee such as OMG, EDI, or SWIFT can agree upon terminology to use in creating a vocabulary, or industry groups such as banking or motor parts manufacturers can agree upon terminology for specific markets.

Several organizations are standardizing vocabularies and related e-commerce frameworks, which vary

Agent Types

Depending on their function, we can classify agents in one of several major categories.

Personal Agents

These 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) offer a software services set that supports the presentation of software agents as interactive personalities and includes natural language and animation capabilities. Simple examples built using this technology include Microsoft's Bob or Paper Clip. The site http://www.bottechnology.com provides a survey of some personal agent technologies, while http:// www.redwhale.com describes personalizable interfaces that use aspects of agent technology. Many of the information agents developed at MIT fall into this category as well.

Mobile Agents

Sent to visit remote sites and collect information before returning with the results, mobile agents aggregate and analyze data or perform local control. Developers typically implement these agents in Java, Java-based component technologies, TCL,[1] VBScript, Perl, or Python (http://www.python.org/). Such data-intensive analysis is often better performed at the source because doing so avoids the shipment of bandwidth-consuming raw data. Examples of such applications include network management agents, Internet spiders, and NASA's mobile agents for human planetary exploration.

Collaborative Agents

These agents communicate and interact in groups, representing users, organizations, and services. Multiple agents exchange messages to negotiate or share information. Collaborative agent examples include those that expedite online auctions, planning, negotiation, and logistics, supply-chain, and telecom service provisioning. Cobalt, for example, uses KQML, Corba, CIM, IDL, and XML for cooperative service and network management.[2]

References

1. J. Ousterhout, "TCL: An Embeddable Control Language," *Proc. Usenix Conf.*, Usenix Assoc., Berkeley, Calif., 1990, pp. 133-146.
2. M. Griss, "Software Agents as Next-Generation Software Components," *Component-Based Software Engineering: Putting the Pieces Together*, G. Heineman and W. Council, eds., Addison Wesley Longman, Reading, Mass., to be published June 2001.

from one problem domain to another. If a problem domain is dynamically formed based on a specific application, its ontology is also dynamic. Any of the following can represent an agent vocabulary:

- a natural language dictionary that lists terms, their meanings, and intended use;
- an XML document type definition that defines terms and some relationships—such as attribute syntax and some typing—where comments or an external dictionary explain the meaning, additional relationships, and intent;
- an object-oriented XML schema (see http://www.w3c.org/) that more precisely defines structured data types and attributes, uses element inheritance, and provides more precise control over how many of each element type and attribute are permitted; and
- a full modeling language and tool, such as UML or Ontolingua, for defining vocabulary class and type elements, inheritance or association relationships, and semantics and constraints using OCL and comments.

The interacting agents must understand the vocabulary used to communicate with one another. Some words can be defined formally in terms of other words, but others must be described in a standard dictionary—such as BizTalk (http://www.biztalk.org), Ontology.org (http://www.ontology.org), CommerceNet (http://www. commercenet.com), CommerceOne (http://www.commerceone.com), and RosettaNet (http://www.rosettanet.org)—that tells the agent programmer how the agent system uses the words. HP E-Speak provides a basic vocabulary for describing and managing other vocabularies (http://www.hp.com/e-speak).

Multiagent coordination

The complex tasks an enterprise computing environment requires demand that developers integrate a group of various agents to coordinate this activity. Building reusable multiagent interaction techniques into the integration infrastructure itself facilitates rapid integration. A multiagent system can group agents statically or dynamically and the agents can coordinate with other agents as well as with people. We call the conversation between agents themselves and agents and people *choreography* or *conversation management*.

Several technologies and tools have emerged that make defining these interaction patterns easier. The expected message sequences can have several possible levels of choreography, depending on how loosely or tightly the system must control the allowed interactions.[3] In all cases, rather than directly programming the code to handle message coordination, we use some form of higher-level declarative rules or graphical models. These models make it easier to see how the agents interact. The system can use explicit rules or models to monitor or enforce compliance, easing the programmer's task.

Techniques for agent coordination include the following:

- *Rules*. We can define a set of rules for any agent or community of agents. For example, we 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 agent types.
- *Conversation protocols*. Expressed as finite state machines that use UML state charts, these protocols enable a group of agents to lockstep through a standard protocol of exchanged messages—if A says *x*, then B says *y*—when, for example, bidding during an auction or responding to a call for proposals.
- *Workflow*. The most powerful technique is used when the conversation coordination for interactions of multiple agents and humans must be more complex and precise. A workflow system allows the explicit definition and control of a group of participants that executes a process.

Many e-commerce applications involve some form of inter- or intraorganizational workflow. Using a defined set of rules, a workflow system automates all or part of a business process, document exchange, or transmission of information or tasks to be acted upon from one participant to another. Workflow systems such as ActionWorks Metro, Endeavors, HP Change-Engine, 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, depending on the situation, a telecom management system could alert a human operator or assign a repair or provisioning engineer.

We view this use of workflow as a next-generation scripting language. Workflow functions as a structured scripting or glue language in which agents represent the participants and resources while the society of agents collaborates to enact the workflow. The Little-JIL process programming language demonstrates how we can use a workflow language to coordinate agents. Worklets, an agent-like lightweight process workflow system, uses Java and JPython for its implementation.[7]

Automated workflow systems do not yet adequately address information mobility or tasks disconnected from the rest of a business process. MAGI—the Micro-Apache Generic Interface, based on the Apache HTTP server—provides an open-architecture framework that explicitly addresses e-business messaging and deployment issues across a broad range of computing platforms. Both HP's E-Speak and SUN's Jini map well to MAGI's architecture.[8]

Agents in agent-oriented e-commerce systems dynamically discover and compose e-services and mediate interactions. XML can encode information and services with meaningful structure and semantics that agents can understand.

Using a structure coordination system and either a visual state chart, UML activity, or another workflow model, we can quickly diagram complex coordination patterns and rapidly convert them into an executing system. We can also use other visual and generative techniques to build agent-based systems, such as agent patterns, interaction diagrams, and aspect-oriented programming.[9-11]

Workflow functions as a structured scripting language in which agents represent participants and resources while the society of agents collaborates to enact the workflow.

WEB-BASED DEVELOPMENT TECHNOLOGIES

Developers often use distributed objects, active objects, and scriptable components to implement agents. They can use Java, Java-based component technologies, XML, and HTTP to build agents. These technologies are simple to use, ubiquitous, heterogeneous, and platform-independent. Developers can use HTTP and XML to implement many agent capabilities for naming and communication, and they can use the universal resource locator and universal resource identifier for naming and locating. 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.[12] HTTP provides many services, including robust and scalable Web servers, firewall access, and some level of security.

The available agent systems and toolkits—such as Agent-TCL, Aglets, Concordia, FIPA-OS, Grasshopper, Jackal, JADE, Jatlite, Jumping Beans, Voyager, and Zeus—have different mixes of mobility, adaptability, intelligence, ACL, and multilanguage support.[3]

RELATED AGENT-ORIENTED SYSTEMS

The following examples describe some of the many research projects currently investigating the development of agent-oriented systems for complex, dynamic environments.

HP Labs research prototypes

Several experimental agent projects are under way at HP Labs in Palo Alto, Bristol, and Grenoble. These projects include work on application management, agent-mediated e-commerce, multiagent negotiation and coordination, and personal and workplace assistants.[10,13]

The CWave lightweight mobile agent system and visual toolkit, developed jointly by HP Laboratories and the University of Utah for experimental application management and process control, uses COM/OCX, a publish-subscribe bus, HTTP, and VBScript.[14,15] Agents include standard libraries of measurement objects and methods to initiate, change, and dynamically update measurement (http://beast.cs.utah.edu). HP Labs is also developing an experimental Web-based economic simulation environment for a shopping mall, integrating the environment with personal agents and mobile appli-

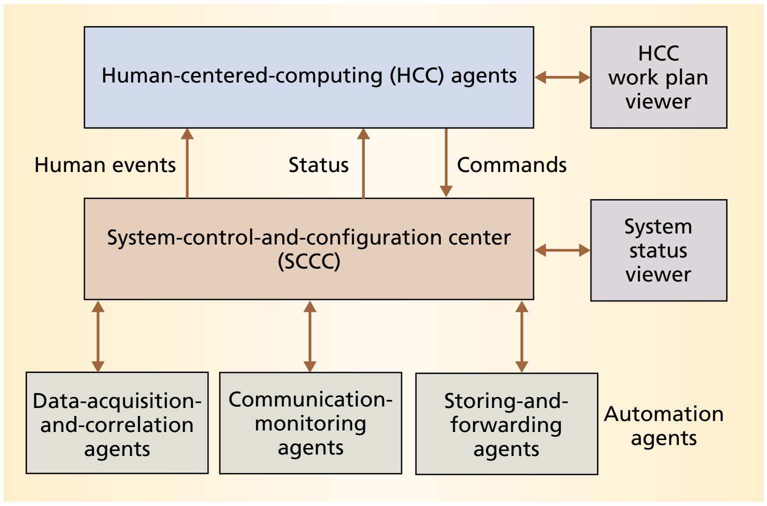
Human-centered-computing (HCC) agents
HCC work plan viewer
Human events
Status
Commands
System-control-and-configuration center (SCCC)
System status viewer
Data-acquisition-and-correlation agents
Communication-monitoring agents
Storing-and-forwarding agents
Automation agents

Figure 3. High-level design of a Web-based, multi-mobile-agent system for human planetary exploration. Built around a system-control-and-configuration model, the system allows multiple people using multiple instruments and computers to automatically create a unique data record of scientific activities.

ances based on HP CoolTown (http://www.cooltown.com)[3] and using Zeus[16] as a base. The most recent work uses the JADE agent system and HP Bluestone to prototype a variety of workplace productivity assistants for mobile workers.

HP's agent projects tackle important issues such as how to effectively define and construct agents, how agents communicate, and how to establish and control collaborations between groups of agents. The combination of agents and workflow provides significant benefits beyond those traditionally associated with components and scripting.

Software agents have differing characteristics such as mobility, autonomy, collaboration, persistence, and intelligence. Some research seeks to simplify the direct definition and implementation of different agent systems' capabilities. Researchers combine aspects and components that represent key capabilities to construct an agent, or a set of compatible agents. UML models of vocabularies, workflow, role diagrams, patterns, and feature trees will drive aspect-oriented generators to create highly customized agent systems.[9,10]

The Iconic Modeling Tool[11] uses visual techniques and UML to assemble and control mobile-agent programs and itineraries. Such agents can perform a range of simple or complex tasks, such as automatic notification via e-mail that a report is available, sending a reminder or rescheduling a meeting, or negotiating on behalf of users.

Web-based multiagent systems for NASA's human planetary exploration

Future NASA missions will require robust and flexible Web-based information systems that designers can rapidly develop, extend, and customize. Thus, the agency can meet these requirements by adopting agent-oriented software engineering and agent components. For example, a new Web-based multiagent system can enhance human performance during planetary exploration in hostile environments. Figure 3 shows this system's high-level design, which consists of mobile intelligent agents.

Such a system must meet the requirements of an extremely dynamic environment in which nodes appear and disappear from clusters, and transient communication links undergo rapid changes. These parameters motivated the design of an agent-based system-control-and-configuration model (SCCM) to allow *multiple* people using *multiple* instruments and computers to automatically create a unique data record of scientific activities.

The SCCM creates a map of the system configuration by listening for the presence of each component on the network. Exploration tools such as digital cameras will use the federation of Jini Lookup services to advertise their presence, function, and status to other network nodes. The data-acquisition-and-correlation agent acquires field data from a digital camera after the explorer presses the shutter. The communication-monitoring agent determines the data transfer path, and the storing-and-forwarding agent stores and forwards the image to the system-control-and-configuration center (SCCC).

The SCCC then passes both the event and location of the new data to the human-centered-computing (HCC) agents to determine what to do with the image, based on the HCC's workflow model. The data is stored in local Web servers, which allow Web browsers to be the primary human-computer interface. Loose coupling of mobile agents and the use of workflow in this project allow rapid development of robust and flexible Web-based systems that NASA can customize and extend to meet the requirements of future human planetary exploration.

Candidate technologies for developing this system include Java 2 Platform, Enterprise Edition, several Jini mechanisms, Java-based component technologies such as Enterprise JavaBeans and JavaBeans, and agent technologies based on XML and HTTP.[17]

AGENT-BASED MIDDLEWARE

As agents become more powerful, message-oriented middleware and agents will share the task of providing a variety of services, including

- location services that enable agents to find and interact with one another in peer-to-peer or client-server configurations;
- application services for functions such as dynamic self-configuration and deployment;
- management services for functions such as registration and life-cycle management; and
- dynamic binding services between agents and hardware.

Middleware can also provide federation, ownership, and mobility services.

We agree with Ebrahim Mamdani and Jeremy Pitt[18] that, as the intelligence of an overall system increases, the distribution of intelligence will inevitably tend toward the agents and away from the middleware. This shift will result in a transition from intelligent networks to networked intelligence in which any middleware functionality will itself be an agent. The availability of E-Speak; the Java 2 Platform, Enterprise Edition; Jini; .NET; and Bluestone will help make this transition possible.

Agent-component technology and agent-oriented software engineering have the potential to be more powerful than traditional, less dynamic components for rapid incremental development. Most agent and e-service systems offer several capabilities that work together to provide unprecedented flexibility and promise to be more effective at handling the resulting software's evolution, distribution, and complexity.

Further, some agents can act as mediators, or intermediaries, transforming and delegating requests to other agents and transforming and aggregating responses. All these capabilities facilitate maintenance and the evolution of agent-based enterprise systems as requirements and technology change. Thus, agent-oriented software engineering offers unique opportunities for developing and maintaining Web-based enterprise systems at Internet speed. ✷

References

1. G. Pour, "Component Technologies: Expanding the Possibilities for Component-Based Development of Web-Based Enterprise Applications," *Handbook of Internet Computing*, B. Furht, ed., CRC Press, Boca Raton, Fla., 2000, pp. 133-156.
2. I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse, Architecture, Process, and Organization for Business Success,* Addison Wesley Longman, Reading, Mass., 1997.
3. M. Griss, "Software Agents as Next-Generation Software Components," *Component-Based Software Engineering: Putting the Pieces Together,* G. Heineman and W. Council, eds., Addison Wesley Longman, Reading, Mass., to be published June 2001.
4. Q. Chen et al., "Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation," *Proc. 4th Int'l Conf. Autonomous Agents 2000*, ACM Press, New York, June 2000, pp. 255-256.
5. Y. Shoham, "Agent-Oriented Programming," *Artificial Intelligence*, vol. 60, no. 1, 1993, pp. 139-159.
6. T. Finin, Y. Labrou, and J. Mayfield, "KQML as an Agent Communication Language," *Software Agents*, J.M. Bradshaw, ed., MIT Press, Cambridge, Mass., 1997, pp. 291-316.
7. G. Kaiser, A. Stone, and S. Dossick, "A Mobile Agent Approach to Lightweight Process Workflow," *Proc. Int'l Process Technology Workshop*, ACM Press, New York, 1999; http://www-adele.imag.fr./IPTW/.
8. G.A. Bolcer, "MAGI: An Architecture for Mobile and Disconnected Workflow," *IEEE Internet Computing*, May/June 2000, pp. 46-54.
9. E.A. Kendall, "Role Model Designs and Implementations with Aspect-Oriented Programming," *Proc. OOPSLA 99*, ACM Press, New York, pp. 353-369.
10. M.L. Griss, "My Agent Will Call Your Agent," *Software Development Magazine*, Feb. 2000, pp. 43-46.
11. B. Falchuk and A. Karmouch, "Visual Modeling for Agent-Based Applications," *Computer*, Dec. 1998, pp. 31-37.
12. R.J. Glushko, J.M. Tenenbaum, and B. Meltzer, "An XML Framework for Agent-Based E-Commerce," *Comm. ACM*, Mar. 1999, pp. 106-114.
13. M.L. Griss and R. Letsinger, "Games at Work—Agent-Mediated E-Commerce Simulation," *Proc. 4th Int'l Conf. Autonomous Agents 2000,* ACM Press, New York, June 2000, HPL-2000-52 for extended version; http://www-adele.imag.fr/IPTW/.
14. M.L. Griss and R.R. Kessler, "Building Object-Oriented Instrument Kits," *Object Magazine*, Apr. 1996, pp. 71-81.
15. C. Mueller-Planitz, "CWave 2000—A Visual Workbench for Distributed Measurement Agents," doctoral dissertation, Computer Science Dept., Univ. of Utah, Salt Lake City, 2000.
16. H. Nwana et al., "ZEUS: A Toolkit for Building Distributed Multi-Agent Systems," *Artificial Intelligence*, vol. 13, no. 1, 1999, pp. 129-186.
17. G. Pour, "A Jini-Based Mobile Agent Architecture for Planetary Exploration," *Proc. Int'l Conf. Technology of Object-Oriented Languages and Systems*, IEEE CS Press, Los Alamitos, Calif., to be published 2001.
18. E. Mamdani and J. Pitt, "Responsible Agent Behavior: A Distributed Computing Perspective," *IEEE Internet Computing*, Sept./Oct. 2000, pp. 27-31.

***Martin L. Griss** is a principal laboratory scientist at Hewlett-Packard Laboratories, Palo Alto, Calif. His research interests include software engineering processes and systems, systematic software reuse, object-oriented development, component-based software engineering, software agent technology, personal agents, mobile appliances, and e-services. Griss received a PhD in physics from the University of Illinois. He is an adjunct professor at the University of Utah and a member of the ACM SIGSOFT Executive Committee. Contact him at martin_griss@ hp.com.*

***Gilda Pour** is a professor of software engineering at San Jose State University. Her research and industrial experience is in distributed object technology and component-based enterprise software engineering, with current emphasis on component-based and agent-oriented development of Web-based enterprise application systems. Pour received a PhD in computer science and software engineering from the University of Massachusetts. She is a member of the IEEE Computer Society. Contact her at gpour@email.sjsu.edu.*