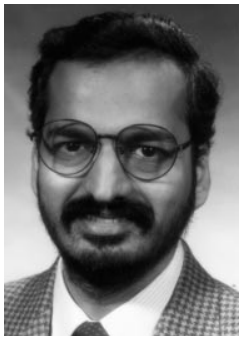

ALL AGENTS ARE NOT CREATED EQUAL

Michael N. Huhns • *University of South Carolina* • *huhns@sc.edu*
Munindar P. Singh • *North Carolina State University* • *singh@ncsu.edu*

In the early 1980s, when we first started building agents, we used Lisp and C, and sometimes a rule-based language such as OPS5 (a predecessor to **CLIPS**). There weren't any specialized agent-building tools available then, because agents were just a research curiosity.

Now we build mobile agents with IBM's **Aglets**, KQML-speaking multiagent systems with Stanford's JATLite, and personalized interface agents with AgentSoft's **Agent Builder**. These new tools have made it a lot easier to develop agents. As the technology advances, we can expect the development of specialized agents to be used as standardized building blocks for information systems.

Two trends lend credence to such a prediction.

First, software systems in general are being constructed with larger components, such as ActiveX and JavaBeans, which are becoming closer to being agents themselves. They have more functionality than simple objects, respond to events autonomously, and, most importantly, respond to system builders at development time, as well as to events at runtime.

Moreover, there is a move toward more cooperative information systems, in which the architecture itself plays an important role in the effectiveness of the system, as opposed to traditional software systems where effectiveness depends on the quality of the individual components. These architectures are generating a set of standardized agents. Architectures based on standardized agent types should be easier to develop, understand, and use. Perhaps most important of all, these architectures will make it easier for separately developed information systems to interoperate.

Agents in a Cooperative Information Architecture

To support an architecture in which heterogeneous components can interoperate and appear homogeneous, a variety of agents are needed. Different agents are needed for each of the different components.

A *user agent* acts as an intermediary between the user and the information system, providing access to such information resources as data analysis tools, workflows, and concept-learning tools. It supports a variety of interchangeable user interfaces (for example, query forms, graphical query tools, menu-driven query builders, and query languages), result browsers, and visualization tools.

User agents maintain models of the other agents in the cooperative information system so that they can interact with them more effectively. For example, a user agent might contain a mechanism to select an ontology from an ontology agent. The ontology would enable the user agent to present a customized interface that contains terminology familiar to the end-user.

Broker agents implement directory services for locating appropriate agents with appropriate capabilities. They manage a namespace service and may store and forward messages and locate message recipients. Brokers might also function as communication aides by managing communications among the various agents, databases, and application programs in an environment.

Resource agents provide access to information stored in legacy systems. The three common types are classified by the resource they represent. *Wrappers* implement common communication protocols and translate commands and results into and from local access languages. For example, a wrapper agent may use a local data-manipulation language such as SQL to communicate with a relational database or OQL for an object-oriented database. *SQL database agents* manage specific information resources, and *data analysis agents* apply machine learning techniques to form logical concepts from data or use statistical techniques to perform data mining.

Resource agents apply the mappings that relate each information resource to a common context for purposes of translating messages meaningfully. At most n sets of mappings and n resource agents are needed for interoperation among n resources and applications, as opposed to $n(n-1)$ mappings that would be needed for direct pairwise interactions among n resources without agents.

Execution agents, which might be implemented as rule-based knowl-

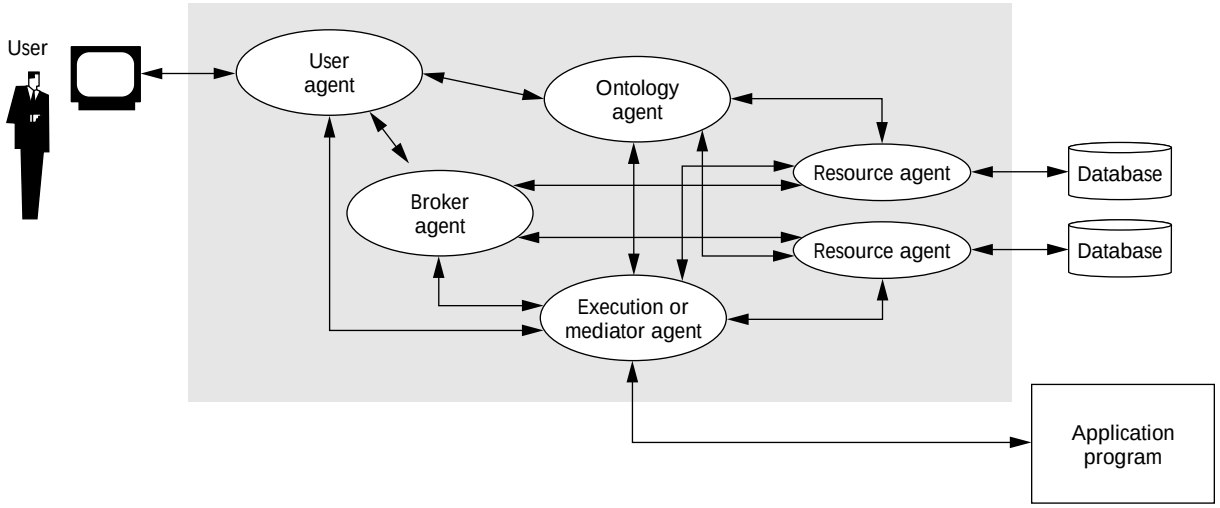

Figure 1. The agents in a cooperative information system first determine a user's request (the user agent's responsibility) and then satisfy it by managing its processing. Under the control of the execution agent, the request might be sent to one or more databases, which are managed by resource agents.

edge systems, supervise query execution, operate as script-based agents to support scenario-based analyses, or monitor and execute workflows. This third function can extend over the Web and be expressed in a format such as the one specified by the **Workflow Management Coalition**. A *mediator agent* is a specialized execution agent. Mediators work with brokers to determine which resources might have relevant information. They also decompose queries to be handled by multiple agents, combine the partial responses obtained from multiple resources, and translate between ontologies.

Security agents provide system-wide authentication and authorization, and can be used to enforce appropriate usage policies for information resources.

Ontology agents manage the distributed evolution and growth of ontologies. (See our column "Ontologies for Agents"[1] for a discussion of ontologies and their uses in information systems.) They provide a common context as a semantic grounding, which agents can use to relate their individual terminologies. A third function of ontology agents is providing remote access to multiple ontologies.

Figure 1 shows a multiagent system architecture in which each agent has a specialized function. The agents communicate using languages such as KQML, FIPA, or SQL. Such an architecture could provide a user with the appearance of homogeneity among heterogeneous resources, and act as a cooperative partner in finding and managing information.

Diversity vs. Complexity

Among the reasons why agents are attractive, two particularly interest us here. First, agents enable us to construct modular systems from heterogeneous pieces that may have been created by any number of vendors. Second, the agents themselves embody diverse knowledge, reasoning approaches, and perspectives. This diversity is sometimes essential, because the agents represent people or business interests that have different goals and motivations. Diversity can sometimes be added by design: it can make an agent system more robust by enabling a variety of viewpoints to be represented and exploited.

However, agents are typically complex pieces of software, so the question arises whether a set of different agents would unnecessarily add to a system's complexity. The more kinds of agents there are, the harder it is to build and maintain them.

Fortunately, this is not the dilemma it seems to be. The agents must be diverse in content (for example, knowledge, reasoning techniques, and interaction protocols), but not in the form in which that content is realized (the language or toolkit with which they are constructed). Problems arise through unnecessary heterogeneity in construction; the cost of necessary heterogeneity in content is more than recovered through the flexibility it offers.

There are three practical ways you can limit the heterogeneity and its

URLs for this column

Agent Builder • www.agentsoft.com
Aglets • www.trl.ibm.co.jp/aglets/
CLIPS • www.ghg.net/clips/CLIPS.html
Java Agent Template • cdr.stanford.edu/ABE/JavaAgent.html
Java Ontology Editor • www.ece.sc.edu/Labs/HIIT/html/imts-new.html
MCC • www.mcc.com/projects/infosleuth/
Workflow Management Coalition • atom.kjist.ac.kr/~jjyoo/wfmc/glossary.html

pernicious effects. One, construct agents using a toolkit, preferably a common toolkit (or as few as possible, because the choice is often based on past practice or local politics). Two, apply agents in the conventional roles outlined above. You will be much happier if you keep your broker conceptually separate from your user agent, for example. You could upgrade each agent independently or, if you like, plug in someone else's improved version for one of yours. And three, use standards wherever appropriate. Public standards can make it easier to construct composite systems from heterogeneous and independently developed parts. The more you and your collaborators can agree on in advance, the fewer problems you will have when you hook up your systems. ■

Systems of the Bimonth

The InfoSleuth project[2] has produced examples of most of the agent types described in this column. You can read about it at the **MCC** Web site.

Check it out!

REFERENCES

1. M.N. Huhns and M.P. Singh, "Ontologies for Agents," *IEEE Internet Computing*, Vol. 1, No. 6, Nov./Dec. 1997, pp. 81–83.
2. M.H. Nodine, "The InfoSleuth Agent System," *Proc. Second Int'l CIA-98 Workshop: Learning, Mobility, and Electronic Commerce for Information Discovery in the Internet*, Paris, July 1998.