Distributed Computing Architecture/e-Business Advisory Service

Agents: Technology and Usage (Part 1)

Executive Report, Vol. 3, No. 4

by James Odell

Centralizing a corporation was once considered an efficient way to run an enterprise. Decisions and information processing occurred in an orderly, top-down, hierarchical manner. However, it is now clear that this type of system only works in a reasonably stable market. Globalization and changes in technology are causing today's market to be in a state of constant flux. Companies that cannot adapt fast enough to thrive in these new markets will be left behind.

In response, many companies are now building agent-based systems. These systems employ agents that can distribute functionality across a vast computing network. Furthermore, agents can adapt to their environment and evolve by learning from the environment. In short, they are the ultimate in distributed computing. Such an approach prepares enterprises for an increasingly complex marketplace and enables them to respond rapidly to change.

However, agents and agent-based technology are an evolution, not a revolution. They are being built from, and will work in combination with, today's technology. While agents, objects, relational databases, legacy systems, embedded systems, and so on each have their own niche, together they can orchestrate rich systems that none of these technologies could provide alone.

WHAT IS AN AGENT?

A Flock Is Not a Bird

Imagine you are sitting in the park on a nice summer day and a flock of birds sweeps the sky. One moment the birds are circling, another they dart to the left or drop to the ground. Each move is so beautiful that it appears choreographed. Furthermore, the movements of the flock seem smoother than those of any one bird in the flock. Yet the flock has no high-level controller or even a lead bird. The phenomenon is a result of what is often called self-organization.[1] Each bird follows a simple set of rules that it uses to react to birds nearby. When animator Craig Reynolds of DreamWorks developed his bird simulation (www.red.com/cwr/boids.html), each bird behaved according to two basic rules:

- Maintain a minimum distance from other objects, including other birds.
- Be sociable (i.e., try to match velocities with other birds if

they are nearby, and move toward the perceived center of their group).

Orderly flocks emerge from simple rules like these. No one bird has a sense of an overall flock. The "bird in front" is merely the position of a given bird. It just happens to be there — and will be replaced by others in a matter of minutes. "The flock is organized without an organizer, coordinated without a coordinator," as Mitchell Resnick states.[1]

Flocks of birds are not the only things that work this way. Beehives, ant colonies, freeway traffic, national and global economies, societies, and immune systems are all examples of patterns that are determined by local component interaction rather than centralized authority. For IT applications, this can include order processing, supply chain, shop floor control, inventory management, message routing, and management of multiple databases. In other words, a decentralized approach should be considered wherever local components have control — instead of limiting your approach to the centrally organized one traditionally employed by IT. After all, if New York City can maintain a two-week supply of food with only locally made decisions, why can't a supply-chain system perform in a similar manner? As Figure 1 suggests, if we could develop IT systems using even the simplicity of an ant colony, we would have very robust and adaptable systems indeed.

Definition of Agent

Another name for the "local component" described above is *agent*. An agent can be a person, a machine, a piece of software, or a variety of other things.

The basic dictionary definition of agent is *one who acts*. However, for developing IT systems, such a definition is too general; IT-related agents need additional properties. Some of the properties that agents may possess in various combinations include being:

- **Autonomous** — capable of acting without direct external intervention. It has some degree of control over its internal state and actions based on its own experiences.
- **Interactive** — communicates with the environment and other agents.
- **Adaptive** — capable of responding to other agents and/or its environment to some degree. More advanced forms of adaptation permit an agent to modify its behavior based on its experience.
- **Sociable** — interaction that is marked by friendliness or

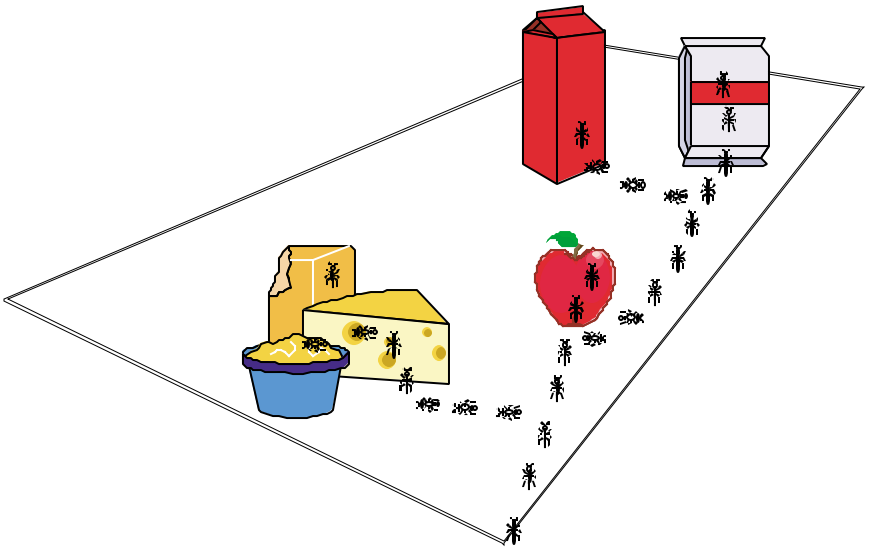


Figure 1 — What if we could build IT systems that are as robust and adaptable as those found in nature? (Source: Van Parunak, ERIM, Inc.[2])

Distributed Computing Architecture/e-Business Advisory Service Executive Report is published by the Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552. Tel: +1 781 641 5118 or, within North America, +1 800 964 5118, Fax: +1 781 648 1950 or, within North America, +1 800 888 1816, E-mail: consortium@cutter.com, Web site: www.cutter.com/consortium/. Client Services: Megan Nields, Tel: +1 781 641 5118, or Christine Doucette, Tel: +1 781 641 9768, E-mail: info@cutter.com.

pleasant social relations; that is, the agent is affable, companionable, or friendly.

- **Mobile** — able to transport itself from one environment to another.
- **Proxy** — may act on behalf of someone or something, that is, acting in the interest of, as a representative of, or for the benefit of, some entity.
- **Proactive** — goal-oriented, purposeful. It does not simply react to the environment.
- **Intelligent** — state is formalized by knowledge (i.e., beliefs, goals, plans, assumptions) and interacts with other agents using symbolic language.
- **Rational** — able to choose an action based on internal goals and the knowledge that a particular action will bring it closer to its goals.
- **Unpredictable** — able to act in ways that are not fully predictable, even if all the initial conditions are known. It is capable of nondeterministic behavior.
- **Temporally continuous** — is a continuously running process.
- **Credible** — believable personality and emotional state.
- **Transparent and accountable** — must be transparent when required, but must provide a log of its activities upon demand.
- **Coordinative** — able to perform some activity in a shared environment with other agents. Activities are often coordinated via plans, workflows, or some other process management mechanism.
- **Cooperative** — able to coordinate with other agents to achieve a common purpose; nonantagonistic agents that succeed or fail together. (Collaboration is a term used synonymously with cooperation.)
- **Competitive** — able to coordinate with other agents except that the success of one agent implies the failure of others (the opposite of cooperative).
- **Rugged** — able to deal with errors and incomplete data robustly.
- **Trustworthy** — adheres to Laws of Robotics and is truthful.

An industry-standard definition of agent has not yet emerged. Most agree that agents bound for IT systems are not useful without at least the first three of the above properties. Others require IT agents to possess all of the properties listed above to varying degrees. At a minimum, an IT agent is generally regarded to be *an autonomous entity that can interact with its environment*. In other words, it must be able to perceive its environment through sensors and act upon its environment with effectors.

Agents and OO

An agent-based approach is employed when a particular situation requires that processing be decentralized and self-organized, instead of centrally organized. Although a centrally organized program could have been written to handle the bird simulation, the system would have been far too cumbersome. It would have required a single set of top-level rules telling each bird precisely what to do in every conceivable situation. Not only would such an application be touchy and fragile, it would likely end up looking jerky and unnatural, more like a poorly animated cartoon than animated life.[3]

Most developers tend to build centrally organized applications. They are also biased toward object-oriented (OO) notions such as *class, association,* and *message*. Although these constructs are useful for a certain category of applications, they do not directly address the requirements of agents. As we saw above, agents have characteristics such as autonomy, mobility, and adaptability. Furthermore, business users like to express other concepts, such as rules, constraints, goals and objectives, and roles and responsibilities. In short, the agent-oriented approach distinguishes between autonomous,

interactive, mobile entities (agents) and the passive ones of conventional OO (objects). This does not mean that object orientation is dead or passé. A well-designed agent-based system uses both objects and agents — just as real-life organizations employ a balance of both active and passive elements. Furthermore, object technology can be used to enable, rather than drive, agent-oriented technology. (In the coming months, Part 2 of this report will discuss the differences and similarities of agents and objects.)

Important Forms of IT Agents

The agents found in IT systems have special requirements: they must execute as software, hardware, robotics, or a combination of these. Agent developers have identified several forms of agents that are important for IT system development. The list of agent characteristics presented earlier addresses some of these requirements. Additionally, since IT systems have special needs, software- and hardware-related forms must be considered. Those forms considered the most important to agent developers today are discussed below.

Software Agents

Software agents are a more specific kind of agent. At a minimum, software agents are defined as *autonomous software entities that can interact with their environment*. In other words, they are agents that are implemented using software. This means that they are autonomous and can react with other entities, including humans, machines, and other software agents, in various environments and across various platforms.

At a minimum, software agents are defined as autonomous software entities that can interact with their environment. In other words, they are agents that are implemented using software. This means that they are autonomous and can react with other entities, including humans, machines, and other software agents, in various environments and across various platforms.

Basically, software agents are design patterns for software. Tools, languages, and environments can be specifically developed to support the agent-based pattern. However, the agent design pattern can also be implemented using OO tools, languages, and environments — or any other tool, language, or environment that is capable of supporting software entities that are autonomous, interactive, and adaptive. Agent-based tools are preferable primarily because the agent design patterns are inherent in the software, rather than explicitly programmed. In other words, object technology can be used to *enable* agent-based technology, but the autonomous, interactive, and adaptive nature required by agents is not currently supported within OO technology. These properties can be (and are being) added to the OO approach, but, currently, the design patterns for agents and agent-based software are not fully and directly supported.

(Author's Note: From this point on, the term *agent* will usually mean *software agent*. Although many of the ideas and technology discussed here can be generalized to support machines and people, the emphasis in this *Executive Report* will be on software implementation.)

Autonomous Agents

When an agent has a certain independence from external control, it is considered autonomous. Autonomy is best characterized in degrees, rather than simply being present or not. To some extent, agents can operate without direct external invocation or intervention. Without any autonomy, an agent would no longer be a dynamic entity, but rather a passive object such as a part in a bin or a record in a relational table. Therefore, autonomy is considered by the Foundation for Intelligent Physical Agents (FIPA) and the Object Management Group's (OMG) Agent Working Group to be a required property of agents.

Autonomy has two independent aspects: *dynamic autonomy* and *unpredictable autonomy*. Agents are dynamic because they can exercise some degree of activity — ranging from simply passive to entirely proactive. For example, although ants are basically reactive, they do exhibit a small degree of proaction when they choose to walk, rest, or eat. A supply-chain agent can react to an order being placed, yet be proactive about keeping its list of suppliers up to date.

Agents can react not only to specific method invocations but also to observable events within the environment. Proactive agents will actually poll the environment for events and other messages to determine what action they should take. (To compound this, many agents can be engaged in multiple parallel interactions with other agents, magnifying the dynamic nature of the agent system.) In short, an agent can decide when to say "go."

Agents may also employ some degree of unpredictable (or nondeterministic) behavior. When observed from the environment, an agent can range from being totally predictable to completely unpredictable. For example, an ant that is wandering around looking for food can appear to be taking a random walk. However, once pheromones or food are detected, the ant's behavior becomes quite predictable.

In contrast, the behavior of a shopping agent might be highly unpredictable. Sent out to choose, negotiate, and buy a birthday present for your mother-in-law, the agent might return with something odd indeed or with nothing at all. In fact, the agent can even say "no."

The behavior of a shopping agent might be highly unpredictable. Sent out to choose, negotiate, and buy a birthday present for your mother-in-law, the agent might return with something odd indeed or with nothing at all. In fact, the agent can even say "no."

Interactive Agents

Interactive agents can communicate with both the environment and other entities and can be expressed in degrees. On one end of the scale, object messages (method invocation) can be seen as the most basic form of interaction. A more complex degree of interaction would include those agents that can react to observable events within the environment. For example, food-gathering ants do not invoke methods on each other; their interaction is indirect through physically affecting the environment. In other words, ants do not receive method invocations; they receive events regarding the state of the environment. Even more complex interactions are found in systems where agents can be engaged in multiple, parallel interactions with other agents. Here, agents begin to act as a society. Finally, the ability to interact becomes most complex when systems involving many heterogeneous agents can coordinate through cooperative and/or competitive mechanisms (such as negotiation and planning).

Although we can conceive of an agent that cannot interact with anything outside of itself, the usefulness of such an entity for developing agent-based systems is questionable. Therefore, like autonomy, interaction is considered by FIPA and OMG's Agent Working Group to be a required property of agents.

Adaptive Agents

An agent is considered adaptive if it is capable of responding to other agents and/or its environment to some degree. At a minimum, this means that an agent must be able to react to a simple stimulus — to make a direct, predetermined response to a particular event or environmental signal. Thermostats, robotic sensors, and simple search bots fall into this category.

Beyond the simple reactive agent is the agent that can reason. Reasoning agents react by making inferences and include patient diagnosis agents and certain kinds of data mining agents.

More advanced forms of adaptation include the capacity to learn and evolve. These agents can change their behavior based on experience. Common software techniques for learning are neural networks, Bayesian rules, credit assignments, and classifier rules. Examples of learning agents are agents that can approve credit applications, analyze speech, and recognize and track targets. A primary technique for agent evolution usually involves genetic algorithms and genetic programming. Here, agents can literally be "bred" to fit specific purposes. For example, operation plans, circuitry, and software programs can be bred, proving to be more optimal than any human could make in a reasonable amount of time.

An agent that cannot respond to its environment or to other agents is another kind of agent whose usefulness is questionable for developing agent-based systems. Therefore, adaptation is considered by FIPA and OMG's Agents Working Group to be a required property of agents.

Mobile Agents

Stationary agents exist as a single process on one host computer; mobile agents can pick up and move their code to a new host where they can resume executing. From a conceptual standpoint, such mobile agents can also be regarded as itinerant, dynamic, wandering, roaming, or migrant. The rationale for mobility is the improved performance that can sometimes be achieved by moving the agent closer to the services available on the new host.

Human organizations are not constructed with a population of identical individuals doing the same thing; instead, we diversify, delegate, negotiate, manage, cooperate, compete, and so on. The same approach needs to be employed in multiple agent systems.

For example, if an agent wants to obtain information from several sources on different platforms, it could send information requests to each of the platforms using the equivalent of a remote procedure call (RPC). However, if the volume of information exchanged with the remote site is large, issues of traffic and bandwidth must be considered. Also, the agent might be able to process the remote data more effectively than those services offered at the remote site. In either or both of these cases, relocating the agent to each of the various platforms could be a more efficient way of processing. One disadvantage of such mobility is that the remote sites must provide the CPU cycles to support the mobile agent's processing. This brings an additional processing burden to the remote site. It also raises three issues: security, billing the agent for its CPU time, and unanticipated scalability problems.

The ability of an agent to transport itself from one environment to another is not a requirement for agenthood. Nevertheless, mobility is an important property for many agent-based systems — and necessary for a certain class of application.

Coordinative Agents

Human organizations exist primarily to coordinate the actions of many individuals for some purpose. That purpose could be to create such structures as profitable business units, charitable organizations, ballet companies, or Little Leagues. Using human organizations as an analogy, systems involving many agents could benefit from the same pattern. Some of the common coordinative agent applications involve supply chains, scheduling, vehicle planning, problem solving, contract negotiation, and product design. Without some degree of coordination, such systems could not be possible — in either human or agent-based systems.

Furthermore, the analogy requires that we consider a heterogeneous population of agents. Human organizations are not constructed with a population of identical individuals doing the same thing; instead, we diversify, delegate, negotiate, manage, cooperate, compete, and so on. The same approach needs to be employed

in multiple agent systems. Careful consideration should be given to designing and structuring agent-based systems, however. This will increase the likelihood that agents will be coherent in their behavior. While this limits and controls agent spontaneity, it still preserves agent-level flexibility.

Intelligent Agents

After decades, the term *intelligent* has still not been defined (or understood) for artificial systems, and applying the term now to agents may not be appropriate. Most people tend to regard the terms *agent* and *intelligent agent* as equivalent. Perhaps this is just an attempt to communicate that agents have more power than conventional approaches. For example, in comparison to relational tables or objects, agents can be thought of as somewhat "smarter " — or it could just be marketing hype. However, it would be fair to say that the notion of intelligence for agents could very well be different than for humans. We are not creating agents to replace humans; instead, we are creating them to assist or supplement humans. A different kind of intelligence, then, would be entirely appropriate.

The current wisdom is that whatever the term *intelligent agent* means, such agents will require a basic set of attributes and facilities. For example, an intelligent agent's state must be formalized by knowledge (i.e., beliefs, goals, desires, intentions, plans, assumptions) and must be able to act on this knowledge. It should be able to examine its beliefs and desires, form its intentions, plan what actions it will perform based on certain assumptions, and eventually act on its plans. Furthermore, intelligent agents must be able to interact with other agents using symbolic language. All this sounds like a model of rational human thinking — but we should not be surprised. Once again, agent researchers are using our understanding of how humans think as a model for designing agents.

FIPA and OMG's Agent Working Group do not have a standard definition of an intelligent agent. However, the description above provides a general idea of the group's current direction.

Wrapper Agents

This agent allows another agent to connect to a nonagent software system/service uniquely identified by a software description. Client agents can relay commands to the wrapper agent and have them invoked on the underlying services. The role provided by the wrapper agent provides a single generic way for agents to interact with nonagent software systems. (Note: the wrapper agent would probably not be necessary for objects; that is, objects should still be able to coexist in the same environment.)

The wrapper is considered important by FIPA and OMG's Agent Working Group. It provides a bridge to legacy code and facilitates the reuse of code for an agent's process.

Other Forms of Agents

The kinds of agents listed above are the predominate forms considered for every agent-based system. More detailed examination of an application can identify other forms, such as facilitator agents, broker agents, manager agents, and so on. These forms can be more easily thought of as roles that an agent can play, rather than the fundamental approach designed into an agent.

Single Versus Multiagent Systems

Many of the early commercial agents were developed for information searching. Here, individual agents were launched on a tether to gather predefined kinds of information and return them to the human requester. In other words, these agents were solo operations that had very little, if any, interaction with other agents. Such an approach certainly has its many uses. However, if we look at the human world, such an approach alone could not build the societies or support the organizations that humans have come to enjoy. Instead, we set up networks of people that interact for various purposes. Interaction among agents, then, is not sufficient to build agent societies, we

need agents that can coordinate either through cooperation, competition, or a combination of both. These agent "societies" are called *multiagent systems* (MAS).

Multiagent systems are systems comprising agents coordinated through their relationships with one another. For example in a kitchen, the toaster "knows" when the toast is done, and the coffeemaker "knows" when the coffee is ready. However, there is no cooperative environment here — only an environment with single agents. In a multiagent environment, the kitchen becomes more than just a collection of processors. The appliances would be interconnected in such a way that the coffeemaker and the toaster would ensure that the coffee and toast are ready at the same time.

Some of the rationale for multiagent systems is as follows:

- One agent could be constructed that does everything, but fat agents represent a bottleneck for speed, reliability, maintainability, and so on (i.e., there are no omnipotent agents). Dividing functionality among many agents provides modularity, flexibility, modifiability, and extensibility.
- Specialized knowledge is not often available from a single agent (i.e., there are no omniscient agents). Knowledge that is spread over various sources (agents) can be integrated for a more complete view when needed.
- Applications requiring distributed computing are better supported by MAS. Here, agents can be designed as fine-grained autonomous components that act in parallel. Concurrent processing and problem solving can provide solutions to many problems that, up until now, we handled in a more linear manner. Agent technology, then, provides the ultimate in distributed component technology.

To support MAS, an appropriate environment needs to be established. MAS environments:

1. Provide an infrastructure specifying communication and interaction protocols
2. Are typically open and have no centralized designer or top-down control function
3. Contain agents that are autonomous, adaptive, and coordinative

Clearly, single-agent environments are much simpler because designers do not have to deal with such issues as cooperation and negotiation. However, the large-scale requirements of industry necessitate approaches that employ coordination and distribution. As such, FIPA and OMG's Agent Working Group are focusing primarily on multiagent systems rather than single agents.

AGENT-BASED MANUFACTURING: A CASE STUDY

The Traditional Manufacturing Approach

In traditional manufacturing, information systems mimic organizational structures, using a top-down, command-and-control structure. Communicating decisions and information down through the organization is time consuming — making it impossible to respond and adapt quickly to external forces.

Furthermore, traditional manufacturing relies on schedules as a means of forecasting what needs to be produced. Schedulers sequence jobs based on the assumption that the environment will not significantly change during the schedule's time span. This approach works adequately in a predictable market. In a turbulent marketplace, a schedule is impractical. Any small, unanticipated change in demand or factory floor conditions can substantially affect the schedule, rendering it obsolete.

Another problem with traditional schedulers is that they try to anticipate and plan for every possible change that may occur. Unfortunately, the range of scenarios and the possible combinations of parameters are infinite because manufacturing is so complex. Even if it were possible to precode all possible scenarios, the cost of considering and

programming all possible combinations is prohibitive. An unanticipated scenario could cause the system to fail.

In short, traditional manufacturing facilities have the following shortcomings that affect their ability to compete in today's constantly changing marketplace:

- They do not have mechanisms in place to accommodate rapid changes in business conditions caused by global competition and changing market demands.
- They do not have mechanisms in place to modify systems while they are executing.
- They are rigid and slow to make significant organizational or functional changes.
- They do not have a mechanism to recover gracefully from partial failures on the factory floor.
- They are unable to form or to participate in virtual enterprises.
- They are not scalable for changes in the market.
- The business model and the operational philosophy are not customer driven.

These shortcomings cause problems such as reduced productivity, increased costs, and missed market opportunities. To remain competitive in today's marketplace, manufacturing must change its approach. In response, a major automotive company is building an agent-based manufacturing system under the direction of Dr. David Greenstein.[4] Here, the agents not only adapt to their environment but can also evolve by learning from the environment. Such an approach prepares manufacturing enterprises for the increasingly complex marketplace and enables them to respond rapidly to change.

Since cells are agents in their own right, they can form virtual organizations able to adapt constantly to the changing global marketplace. This dynamic structure enables each cell to remain agile. Rather than being constrained by a fixed hierarchy, the cells (and therefore the overall business) can thrive in a continuously changing and unpredictable environment.

This section provides a case study describing how agent-based technology can be applied in business applications. Although the example is for an automotive company, the concepts are applicable to many other industries.

An Agent-Based Solution

The Agile Manufacturing Information System (AMIS) is a new approach and operational model that addresses the problems of traditional manufacturing practices. Because today's dynamic marketplace is similar to ecosystems, AMIS is modeled after the behavior of the natural world — an approach that is agile, adaptive, and dynamic. It can adjust to changes in the marketplace and in technology, making it effective and competitive.

Traditional manufacturing systems rely on a rather rigid, top-down structure to represent a manufacturing enterprise. AMIS uses a loose aggregation of software agents to represent a manufacturing entity. For example, resource agents represent the capabilities and capacity of the various resources available, such as machines, tools, people, and computers. The work performed within a facility is represented by job agents. In a small system, the interaction of the resource agents with job agents manages the manufacturing process.

However, in systems involving many jobs and resources, the interaction could tax even modern information systems. Here, resource agents can be grouped into cells. Since cells are agents in their own right, they can form virtual organizations able to adapt constantly to the changing global marketplace. This dynamic structure enables each cell to remain agile. Rather than being constrained by a fixed hierarchy, the cells (and therefore the overall business) can thrive in a continuously changing and unpredictable environment.

Each cell can be treated as a manufacturing business unit. Since it is responsible for its own bottom line, each cell must be profitable over time. When a cell is consistently unprofitable, it is dissolved and other cells absorb its resources. Similarly, each resource in a cell is responsible for maintaining a positive bottom line and contributing to the cell's overall profit. This distributed profit responsibility allows the cell to maintain a suitable size and the right mix of resources for the current workload, while maintaining the flexibility to address future needs.

As Greenstein states, "For a manufacturer to succeed in today's competitive world, it must have the optimal mix of people, equipment, and knowledge to make the product. The AMIS architecture provides flexibility and agility in a software system, which enables a manufacturer to monitor, evaluate, and adjust the mix of resources, people, and tools required as market demands change."[4]

The AMIS Agents in More Detail

Cell Agents

In living systems, a cell is a self-contained unit that has its own structure and behavior. It consists of other self-contained structures that interact to support the cell. How well the cell and its components work together determines whether the cell lives or dies. In a manufacturing system, each cell agent is a business unit representing a collection of physical resources, including machines, tools, and people. The cell operates as a self-contained business unit and only continues to exist if it meets its profit and production goals and its responsibilities. The cell also controls its own size, changing the mix and number of resources over time to maintain its profitability and competitiveness in the marketplace. The architecture of a cell is summarized in Figure 2.

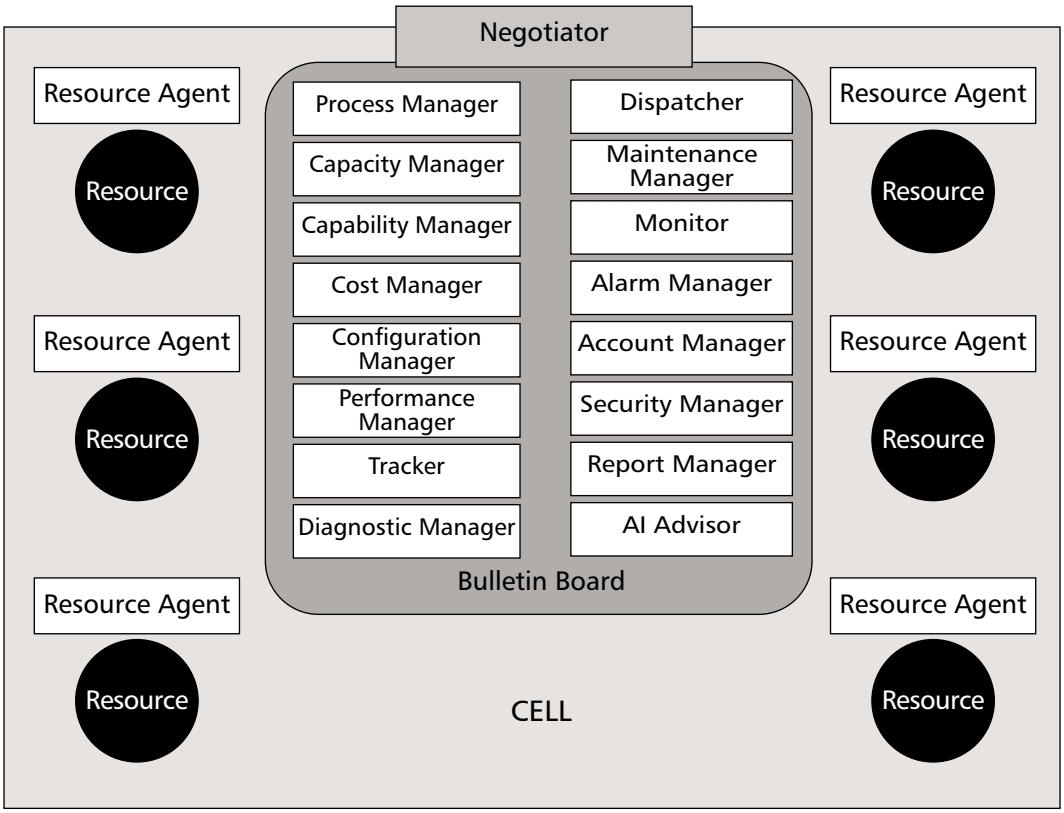


Figure 2 — Typical cell agent architecture for a manufacturing system.

Common Function Agents

Common function agents interact with each other and with the resource agents. They provide the complete set of business functionality required to operate the cell as an independent business unit. Each common function agent is responsible for a different business or manufacturing function within the cell. Some of these agents contain information about the resources within the cell, such as the capabilities of the resources. Other agents provide interfaces to the people working in the cell, such as process planners and machine operators.

For example, the maintenance manager schedules and directs maintenance activities whether they are scheduled, opportunistic, or reactive. It also keeps track of the maintenance history.

The process planner determines whether the cell will bid on requests for quote (RFQs) received by the cell. A broad analysis is made of the cell to judge the cell's ability and desire to produce a quote for this RFQ. The analysis uses criteria associated with the type of product being requested (automotive, pharmaceuticals, electronics, etc.), the processes needed (welding, casting, packing, etc.), and the resources needed (five-axis CNC, drill, sheet metal press, etc.). If it is determined that the cell lacks the appropriate abilities and cannot subcontract them, then the cell will not bid on the RFQ. If the cell does bid on the RFQ, the process planner generates the process workflow (e.g., a Unified Modeling Language [UML] Activity Diagram) that will be used to execute the quote, if selected.

The capability manager uses the process workflow to verify that the cell has the resources needed to carry out the job. It verifies each step in the process (job agent) with the available resources in the cell. The verification of the capability is based on the information contained in the workflow (time, quality, special characteristics, and cost criteria). If the capability is not present in the cell, the capability manager initiates the subcontracting process through the process planner.

Similarly, the capacity manager uses the information from the workflow to provide capacity to the job agents. The jobs currently accepted by the cell are taken into consideration when deciding if the cell has the capacity to take on this new job. If sufficient capacity is not present in the cell, the capacity manager initiates the subcontracting process through the process planner.

Negotiator Agents

Negotiator agents (at the top of the cell architecture diagram in Figure 2) communicate with the outside world on behalf of the cell. The negotiator provides an interface between the cell and the outside world. It routes messages received from the outside world to the appropriate common function agent. When preparing quotes for new jobs, the negotiator assembles the quote information provided by the other agents and summarizes the final quote information for the customer. Similarly, when the cell receives quotes from subcontractors, the negotiator works with other agents to select the winning quote.

Resource Agent

Each resource agent represents a physical resource within the cell: a machine, tool, computer, or person. Each physical resource provides a specialized utility or function to add value to the order-completion process. The resource agent captures the attributes of the physical resource, allowing the agent to represent it in the cell and to coordinate the cell's use of the resource.

Each resource agent lists the capabilities that define those processes that the physical resource can perform. For example, a resource might be able to perform several types of milling operations. The capability list allows the resource agent to determine whether to bid on the various jobs in the cell.

Resources keep track of their assigned jobs by maintaining a prioritized list of jobs that the resource wins. Each job defines

its job type, the earliest start time for the job, the expected job duration, the latest finish time, and the estimated cost.

The resource agent also maintains profit and loss figures for the resource. The best interest of a resource is to maximize profit by working on as many jobs as possible. If the resource does not maintain a profit over time, the cell may sell the resource to another cell. The resource agent is responsible both for ensuring that the resource is optimally utilized and for representing the resource when bidding for new jobs.

Job Agent

The job agent represents the customer through the order placed into the system. The job agent defines the processes needed to complete the final product specified in a customer order. Each node in the process workflow is a subjob and is handled by an agent. Each subjob agent contains information about that specific process, including the type of process, set-up time, runtime, and cost.

The job agent is responsible for monitoring its current status and due dates. As due dates approach for the overall job or for individual subjobs, the subjob agent will raise alarms to initiate corrective action. The subjob also communicates with its neighboring subjobs, passing state information and alarms to allow the previous and following subjobs to monitor more accurately their status and take appropriate action. The job and subjob agents are active agents responsible for making sure that they are completed by the expected due dates and at the lowest cost possible.

Broker Agent

The broker agent helps customers find providers of services and products. In the AMIS environment, each provider is a cell that registers with the broker, specifying the types of products and services it provides. For example, car buyers do not have to contact each automobile manufacturer; instead, they send the attributes of the desired model (including such criteria as price, delivery date, color, and accessories) to the broker. The broker forwards the request to each automobile-producing cell that has registered with the broker.

The customer specifies the date by which all cells must provide quotes. The broker waits until this date and then collects all the cell bids and returns them to the customer. When the customer selects a winning quote, the broker forwards the customer award notification to the winning cell. Losing cells can view the attributes of the winning bid and compare it to their bid, in order to improve their bids in the future.

AMIS organizes brokers in a hierarchy based on geographic regions. First, the local broker forwards the customer request to its local cells that have registered. If no local cell can meet the customer request, the broker forwards the request to the regional broker. The regional broker forwards the request to each local broker within that region. In turn, these local brokers forward the request to every cell within their local area that manufactures the requested product.

The bids from each cell pass back through this hierarchy, going from the local information brokers to the regional broker. The regional broker returns the bids to the customer via the local information broker that originally received the customer request.

If no cells within the region can meet the customer requirements, the regional broker forwards the request to the global broker. The process is repeated, with the global broker forwarding the request to each regional broker, down through the local information brokers, and to each cell worldwide that produces that product.

In some cases, the customer may wish to solicit quotes from cells worldwide without initially limiting the scope to cells registered with the local broker. In this case, the customer sends the requirements directly to the global broker, bypassing the local and regional brokers.

Adaptation in Natural Systems

Adaptation is no stranger to manufacturing operations. Manufacturers that fail to adapt rapidly to changes become extinct; they go out of business. Adaptation enables the system to react to changes in the market or in the manufacturing environment. When designed properly, the individual parts of the system can be empowered to change based on their environment and market conditions.

An adaptive agent is one that responds to its environment. The simplest form of adaptation is reaction, that is, a direct, predetermined response to a particular event or environmental signal usually expressed in an if-then form. From atoms to ants, the reactive mode is quite evident. A carbon atom would have a rule that states in effect, "If I am alone, I will only bond with oxygen atoms." An ant would have a rule that if it finds food, it should return the food to its colony, leaving a trail of pheromones. Reaction rules do not change in and of themselves, but change can come through other mechanisms such as learning and evolution. Without learning and evolution, ants and atoms are still quite able to support complex "societies." With learning and evolution, however, the rules can be changed based on experience — resulting in new and perhaps improved results.

Learning

Learning is change that occurs during the lifetime of an agent and can take many forms. The most common techniques enable rules and decisions to be weighted based on positive (or negative) reinforcement. For example, in a basic bidding system, a bid could be selected simply on the basis of bid price.

Reaction rules do not change in and of themselves, but change can come through other mechanisms such as learning and evolution. With learning and evolution, the rules can be changed based on experience — resulting in new and perhaps improved results.

However, other considerations might also be appropriate, such as the bidder's ability to deliver its goods in the quantity, quality, and time frame requested. Over time, a purchasing agent can learn to choose from reliable vendor agent*s* instead of just choosing the lowest bid. If a vendor's performance improves (or declines), the purchaser's decisions are modified accordingly. In other words, the agent continues to learn. Popular learning techniques that employ reinforcement learning include credit assignment, Bayesian and classifier rules, and neural networks.

Evolution

Evolution is change that occurs over successive generations of agents. For example, cell agents in AMIS continually evolve to address changing market and business needs. Here, the mix of resources within a cell dynamically evolves and changes so the cell can produce the products demanded in the marketplace. Each resource agent in a cell must continue to win jobs and maintain a positive bottom line, thereby contributing to the overall profitability of the cell.

A resource that consistently fails to win jobs will eventually have a negative cash balance. If a resource maintains a negative cash balance long enough, the cell may decide to replace that resource. The nonproductive resource can either "die" due to malnutrition of cash or be sold to another cell. The original cell can then buy a replacement resource possessing capabilities that are better suited to the products made by the cell. In other words, there is a "survival of the fittest" quality to the mechanism, in which each internal change represents a new generation of cell. By evolving in this way, the cell maintains a set of resources that allows it to remain profitable and survive in a dynamic marketplace.

Best Interest

Whether adaptation is by learning or evolution, each agent is

responsible for acting in its own best interests to ensure that its goals are met. The cell agent's best interest is to win as many jobs as possible and keep the cell busy fulfilling customer orders. The cell also generates as much profit as possible, ensuring its continued viability as a virtual business enterprise.

The resource agent's best interest is to win as many jobs as possible and keep busy processing jobs. The resource also generates as much profit as possible, guaranteeing that it remains a viable member of the cell.

The job agent's best interest is to complete the job quickly, making certain that it is finished by the customer's due date. The job agent also looks for the cell and resource that can complete the job at the lowest possible cost.

The best interest concept embodies the metaphor of free market behavior, as the cell, resource, and job interact and compromise to reach a solution that balances each agent's best interests. This balancing of best interests between these three entities and their dynamic interaction allows a dynamic, adaptive, and productive structure to emerge in agent-based manufacturing systems.

Seven-Step Negotiation Process

When you decide to purchase a product, your decision is influenced by certain requirements. For instance, the cost of the product must be within your budget. Another requirement might be how long it takes to receive the finished product. Once all the elements of your decision criteria are met, you award the work to the supplier that best fits your needs.

The best interest concept embodies the metaphor of free market behavior, as the cell, resource, and job interact and compromise to reach a solution that balances each agent's best interests.

AMIS uses a standard seven-step bidding process to form an agreement to provide a product. This bidding process allows customers to obtain products through a common market process, ensuring that they are all purchased at fair market prices. All seven steps of the negotiation process must be completed successfully in order to complete the transaction.

1. **Request for quote.** The RFQ is the first step in the AMIS bidding process. A customer (or customer agent) creates an RFQ that specifies the desired products or services, along with a response date, and sends it to a broker agent. The broker acts as a liaison, forwarding the RFQ to each cell that has "subscribed" to provide the requested product.
2. **Receive quotes.** Each cell agent determines its ability to complete the job according to the customer's RFQ specifications. If a cell is able to meet the customer's requirements, it creates a quote for the job. Quotes contain estimated information on the delivery date, price, quality, and special characteristics related to completing the job. Cells return their quotes to the broker, which holds the quotes until the RFQ response date has been reached and then returns all the quotes to the customer.
3. **Select winner.** When the time has expired for the cells to submit quotes, the customer (or customer agent) begins the selection process. The customer selects the winner from the submitted quotes by finding the most desirable mix of cost, time, quality, and special characteristics, based on its requirements for the job. The customer sends an award notification to the cell with the best quote.
4. **Winner confirms.** The winning cell accepts or rejects the job depending on whether it still has the capacity to do the job. The cell might reject the job if it has accepted other jobs between the time it prepared the quote and received the award. If the winner rejects the job, the customer offers it to the cell with the next best quote. This continues until a

cell accepts the job. After the winner accepts the job, the other cells that submitted quotes are informed of the decision. At that time, losing cells are able to access data on the quotes, which helps them evaluate why they lost the job and perhaps learn and modify their behavior for future quotes.

5. **Issue purchase order.** After a cell has confirmed the customer order, the customer authorizes the cell to begin production by issuing a purchase order to the cell.
6. **Generate product.** The cell completes the work on the product, delivers the product to the customer, and sends an invoice to the customer.
7. **Make payment.** The customer ends the process by paying the cell for the work done.

The seven-step process establishes a common approach for business interaction between cells. The same process is followed when a cell wants to subcontract part of a job to another cell.

Summary for Agent-Based Manufacturing Case Study

Distributed Organizational Control

To be agile, large, centralized manufacturing organizations must be decomposed into simpler, smaller business units that are responsible for their own business, financial, and production success. This distributed organizational control allows these smaller units to reorganize and react quickly to changing market conditions. These smaller units — cells — can easily be reconfigured to maximize efficiency or respond to a change in the market. Distributed organizational control enables the system to respond locally to unexpected failures or shutdowns by quickly reallocating the necessary resources.

Furthermore, distributed organizational control can be based on the concept of survival of the fittest. Therefore, if a cell within an organization consistently fails to contribute to the greater well-being of that organization, that cell ceases to exist. On the other hand, if every cell is successful, the entire operation is successful. Distributed organizational control allows a successful manufacturing operation to emerge from the interaction of smaller units.

Another benefit of a distributed organization is the ability to quickly form ad hoc formations of business units that achieve common business goals. Here, individual cells cooperate as a unit for a common benefit — and then dissolve when no longer needed. The components of such a virtual organization do not have to be aligned with a physical organization, adding another degree of flexibility not found in traditional systems.

Capacity Management

The more unpredictable the manufacturing environment, the more significant the problems associated with advanced scheduling. For that reason, AMIS does not use the concept of scheduling. Instead, it manages the capacity of the resources.

As a business entity, each cell has limited resources that have limited capacity. All jobs in a cell are temporarily put into the holding capacity queue of that cell. Then, just before a job starts, each resource in a cell bids on the job in the queue. Because the bidding is done right before the job starts, the chance of an unexpected event affecting the completion of the job is significantly reduced.

However, if a problem occurs during the production process, the system is not disabled. This is an important benefit of capacity management. Because the resources in the cell are self-loading and balance the load among themselves, a job that cannot be completed by a resource is returned to the cell's queue for rebidding and reallocation. This dynamic allocation of jobs to resources greatly reduces the effects of the unpredictable nature of the shop floor. Although this is not the only

technique for capacity management, it works well in the automotive industry.

Market-Driven Economy

In a market-driven economy, manufacturers build products in response to market demand, rather than in anticipation of demand. Businesses compete for limited resources and customers, but cooperate when it is beneficial. Change is constant as new products emerge and customer demands evolve.

AMIS relies heavily on the economic laws of supply and demand. Rather than try to forecast market demand and schedule production based on rigid plans, AMIS provides an architecture that adapts to the dynamic marketplace. Both inside and outside the cell, agents operate in a profit-driven economy. The competition between cells or resources will drive the market to an equilibrium or market-clearing price. The producer's need for higher profit and faster production times interact with the customer's need for lower prices and higher quality. These opposing forces result in the best prices and products for everyone involved.

Case Study Conclusion

As designed by Greenstein, AMIS provides a means for a manufacturer to be more productive and adaptive in responding to changing market demand. Specifically, it will allow a manufacturer to:

- Increase machine (resource) usage by better matching capacity to workload

Rather than try to forecast market demand and schedule production based on rigid plans, AMIS provides an architecture that adapts to the dynamic marketplace. Both inside and outside the cell, agents operate in a profit-driven economy.

- Increase throughput by making the right products at the right time
- Reduce the number of late jobs by better capacity planning and monitoring
- Utilize/tune the correct resource types and mix by monitoring resource efficiency
- Create a flexible and dynamic architecture that responds rapidly to a continuously changing market
- Enable an activity-based costing (ABC) approach to collect and calculate actual production costs
- Reduce single points of failure in production systems

Agent-based manufacturing is a new way of thinking about and applying information. The primary benefits of the agent-based approach are that they provide dynamic, reliable, and agile systems. As such, this approach will enable organizations of the future to accommodate rapidly changing business conditions, increase market responsiveness, lower cycle times, increase productivity, and better use their resources. Most importantly, it will benefit the bottom line. In other words, the agent-based approach will be the way modern manufacturers develop their systems to compete in the 21st century.

CURRENT STATUS OF AGENT TECHNOLOGY

The emergence of agent technology has stories similar to those behind other technologies, such as relational, OO, and GUI technologies. One can expect some of history's lessons to repeat:[5]

- Agent technology is not a single, new, emerging technology, but rather the integrated application of multiple technologies.
- Agents are not necessarily a new, isolated form of application. Instead, they can add a new set of capabilities to existing applications.
- Initially, agent functions will emerge within applications, but — with experience — will become part of the operating system or application environment.
- Agent applications may strengthen the human-computer interaction.

- Ultimately, applications that do not exploit agent support in the operating system will be severely disadvantaged.

Although destined to achieve lofty goals, the current state of agent technology can be summarized as follows:

- It is still in an active research stage.
- Isolated pioneer products are emerging.
- The full set of technologies is not yet available.
- The technologies are not yet integrated with one another.
- There is no consensus on how to support agents at the operating system level.
- Despite the hype, agent technology is not in widespread use, nor has it been widely accepted as an inevitable trend.
- There are early adopters who can demonstrate the value of agent technology.

Current Usage of Agents

The agent industry is in an embryonic state. Deployments of agent-based systems and technology are isolated and few, but exist nonetheless — and are, in fact, increasing. Today, agents are just beginning to be applied in a wide range of applications.

Personal use. Currently, most agent applications are for individual use and include search-and-retrieval agents, bots (another name for agent), Web site personalization tools, and user-assistance agents. As the speed and capacity of the personal computer increases, agents will be introduced as a way to schedule appointments, perform file management, suggest ways of locating information, manage the PC itself, and perform other tasks.

Interest matching agents. These are probably the most used agents, and most users do not even know they are using them. Interest-matching agents are used by commercial Web sites to offer recommendations, such as, "*If you liked 'Frank Sinatra's Greatest Hits,' you might also like 'Tony Bennett's Songs for a Summer Day.'*" Based on Patti Maes's work at MIT Media Labs and later at Firefly, these agents observe patterns of interest and usage in order to make recommendations. They have been deployed at Amazon.com and various CD and video sales sites.

Network and system management agents. Telecommunications companies have been the most active in this area and are committed to the agent paradigm. Their goal is to use agents to assist in complex system and network management tasks such as load balancing, failure anticipation, problem analysis, and information synthesis.

Information, decision, and logistic support agents. Agents are already providing services to manage, filter, select, prioritize, reroute, discard, monitor, and share many types of information. Many companies (such as utility companies and military organizations) now use agents for information synthesis and decision support. These systems may alert an operator to a possible problem or provide information in support of a complex decision. They are closely aligned to decision support systems from the traditional AI community.

Process control. Process controllers are autonomous reactive agents that ensure an organization's activities are carried out at the level of individual components, rather than by a centralized controller. Electricity transportation management and particle accelerator control are some early examples of these.

Manufacturing. As described earlier in this report, work cells can be grouped into flexible manufacturing systems. Here, agents plan, negotiate, bid for work, carry out the task, and so on.

Air-traffic control. Sydney airport has developed a sophisticated agent-based air-traffic control system known as OASIS. Agents are used to represent each aircraft that enters the airspace as well as the various air-traffic control systems in operation.

E-commerce. Currently, e-commerce is driven almost entirely by human interaction. However, some of the commercial decisionmaking can be placed in the hands of agents. The electronic marketplace could have buying, brokering, bidding, and selling agents for each product being bought or sold. Such an approach will become an integral part of business-to-business applications, not just for individual shopping on the Web.

Business process management. Business processes can be viewed as a community of negotiating, service-providing agents. Each agent can represent a distinct role or department in an organization and be capable of providing a variety of services. For example, multiagent supply-chain systems would have agents playing the roles of buyers, suppliers, brokers, stock, orders, line items, and manufacturing cells. Operations systems would have resource agents, material agents, process agents, and so on.

Product design. Agents can help designers (often in different locations and with different companies) design the components and subsystems of a complex product using many different analysis tools.

KEY ISSUES FOR AGENT TECHNOLOGY

Several important areas must be addressed before a rich and robust agent technology can exist. Currently, one of the most important areas for standardization is agent communication. If every designer developed a different means of communicating between agents, our agent systems would be worse than a tower of Babel. Not only would the content and meaning of a communication likely be different, but the means of communication could occur in a variety of ways. Agent mobility is also important if we wish to benefit from the relocation of agent processing. The issue of security must also be addressed if we are to ensure that both the agents and their environment are free from danger. This section discusses each of these issues and indicates which standard services and specifications might support them.

One of the most important areas for standardization is agent communication. If every designer developed a different means of communicating between agents, our agent systems would be worse than a tower of Babel. Not only would the content and meaning of a communication likely be different, but the means of communication could occur in a variety of ways.

Agent Communication Languages

When two people want to communicate, they need to choose a common language and interchange medium — though, even then, misunderstandings can occur. Agents, too, need a standard language and a set of conventions that support agents in identifying, connecting with, and exchanging information with other agents. However, for agents, it is even more important to minimize any possible misunderstandings; otherwise, our IT systems could get very confused. Agent communication languages (ACLs) enable agents to communicate in a clear and unambiguous manner. By standardizing these ACLs, different parties can build their agents to interoperate both intra- and intercompany.

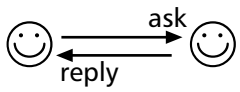
An example of a simple point-to-point communication between two agents is illustrated in Figure 3. Here, one agent is asking another for the current price of IBM stock. This message, or *communication act*, specifies an "ask" speech act, the sender's identity (Joe), the message content (`PRICE IBM ?price`), the address of the communication (stock-server), the name of the reply expected from the responding agent (`IBM-stock`), the language in which the content is specified (`LRPROLOG`), and the set of the agreed-upon terms, or *ontology*, that will be used in the content exchange (`NYSE-TICKS`). The responding agent replies with the requested stock price, along with its associated ACL parameters.

Currently, two primary standards for ACLs exist:

- **Knowledge Query and Manipulation Language (KQML).** Network environments that support plug-and-play processes are still quite rare. Most distributed systems are implemented with ad hoc interfaces between components. KQML is a language and set of conventions that support network programming specifically for knowledge-based systems and agents. It was developed by the ARPA-supported Knowledge Sharing Effort. (See www.cs.umbc.edu/kqml.)
- **Foundation for Intelligent Physical Agents (FIPA).** FIPA has been working to develop and promote standardization in the area of agent interoperability since 1996. FIPA's ACL is a high-level agent communication language that is based on speech acts and is perceived by many as an improvement on KQML. (See www.fipa.org.)

The speech acts that may be specified (such as "ask" and "reply" in Figure 3) are defined by the ACL. Some examples of KQML speech acts (called *performatives* in KQML) include:

- Achieve — A wants B to perform a certain task.
- Advertise — A registers as suitable for a particular request.




```
(ask                                  (reply
:sender Joe                           :sender stock-server
:content (PRICE IBM ?price)           :content (PRICE IBM 140)
:receiver stock-server                :receiver Joe
:reply-with IBM-stock                 :in-reply-to IBM-stock
:language LRPROLOG                    :language LRPROLOG
:ontology NYSE-TICKS)                 :ontology NYSE-TICKS)
```

Figure 3 — Simple point-to-point communication using an agent communication language.

- Ask — A requests information from B (ask-one or ask-all).
- Broker — A wants B to find help to answer something.
- Delete — A wants B to remove specific facts from knowledge-base.
- Recommend — A wants the name of an agent that supplies an answer.
- Recruit — A wants B to request an agent to perform a given task.
- Reply — A answers B.
- Sorry — A cannot provide the requested information.
- Subscribe — A wants any messages from B when they occur.
- Tell — A sends information.

ACLs provide a high-level format for expressing communication acts among agents. The communication detail, however, is embodied in the content parameter, where the agent expresses the actual question, reply, or request. The format of the content parameter must be agreed upon by both sender and receiver(s), otherwise effective communication will not be possible.

The language parameter helps to some extent, because it specifies a registered syntax form. For example, if Prolog were specified, the agent would know that the rules for content syntax must conform to the Prolog language. The Knowledge Information Format (KIF) is one standard for agent language syntax, and was developed by a consortium at the University of Maryland. Another emerging approach is to use and extend XML.

Ontology Communication

The syntax rules of a language are not enough to ensure clear communication; an agreed-upon set of terms is also required. Certainly, the syntax would define some terms, but there are also user-defined terms. For example, to ask for the number of clients that Fujitsu has could be expressed as:

`COUNT FUJITSU client ?integer`. The syntax is well defined, but if one agent uses the term "client" and the other only knows "customer," the two will not communicate effectively — even though both know that something is supposed to be tallied for Fujitsu. Agents can have different terms for the same concept and identical terms for different concepts. A common ontology, then, is required for representing the knowledge from various domains of discourse. The purpose of the ontology parameter in an ACL is to define the set of terms that will be used in an agent communication.

The need for terminology standards is not new; it is a key requirement for Electronic Data Interface (EDI) and KQML. Because agent communications depend on ontology, such standards are now more critical than ever. As such, many organizations and consortia are now being set up to establish industry vocabularies, such as RosettaNet (www.rosetta.net), BizTalk (www.biztalk.org), CommerceNet (www.commerce.net), and Ontology.Org (www.ontology.org). Common ontology representations use UML and XML schema.

Message Transportation Mechanism

Agent communication can be achieved in two ways:

1. Directly with each other (logical connection), which provides flexibility and freedom, but bypasses control and security
2. Through the agent platform (physical connection), which resolves control and security problems but requires logical communications to be physically resolved via the agent base software

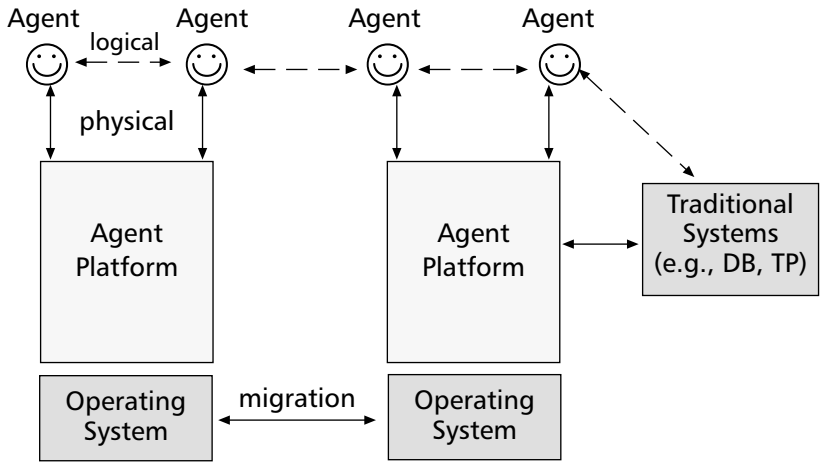


Figure 4 — Using the agent platform for communication and migration.[6]

Most agent systems use the second option (see Figure 4), because the agent platform enforces control and security at a system level. With this approach, the look and feel of agents directly addressing other agents can still exist, even though all communications are still processed through the agent platform, including communications to traditional (legacy) systems.

In agent environments, messages should be able to be scheduled as well as event-driven. They can be sent in synchronous or asynchronous modes. Furthermore, the transportation mechanism should support both unique addressing and role-based addresses (i.e., "white page" versus "yellow page" addressing). Lastly, the transportation mechanism must support unicast, multicast, and broadcast modes, as well as such services as broadcast behavior, nonrepudiation of messages, and logging. (The agent platform will be discussed in more detail in the "Agent Architecture" section on Page 26.)

Although message transportation does not yet exist for agent-based systems, it does exist in an OO form. With some enhancements for the requirements of agent-based environments (directly or indirectly), the following technology could be used: CORBA, OMG Messaging Services, Java Message Service, Remote Method Invocation, Distributed COM, and Enterprise JavaBeans Events.

Agent Interaction Protocols

Agents can interact in various patterns called *interaction protocols* (which are also known as conversation or communication protocols). Each protocol is a pattern of interaction that is formally defined and abstracted from any particular sequence of execution steps.

Figure 5 depicts a few more interaction protocols using multiple agents: requester, provider, and facilitator agents. The facilitator agent functions as a middleman. For example in Figure 5 (a), the facilitator receives a "subscribe" request communication from a requester who wishes to receive messages on a particular topic. Any time a provider agent sends a communication to the facilitator that fits the subscriber's topic, the facilitator passes on the communication.

The facilitator in the recruiter protocol in Figure 5 (b) receives both "recruit" communications from requesters and "advertise" communications from providers. When the facilitator finds a match, it notifies the provider, who then contacts the requester directly. In the broadcaster protocol in Figure 5 (c), an agent requests the facilitator to broadcast a message to a number of agents.

Figure 6 illustrates a negotiation protocol, in which a broker agent sends out invitations to bid on a job contract. If a provider agent wishes to participate in the negotiation, it can respond with a bid. Since many provider agents might respond, the broker agent has to decide which provider should be awarded the contract. Once the contract has been sent to the provider, the provider can choose to confirm. If the provider declines, the broker must choose another provider. Such a pattern could support various negotiation scenarios, such as ordering supplies, requesting equipment, or obtaining human resources. There are many other patterns that provide basic communication protocols.

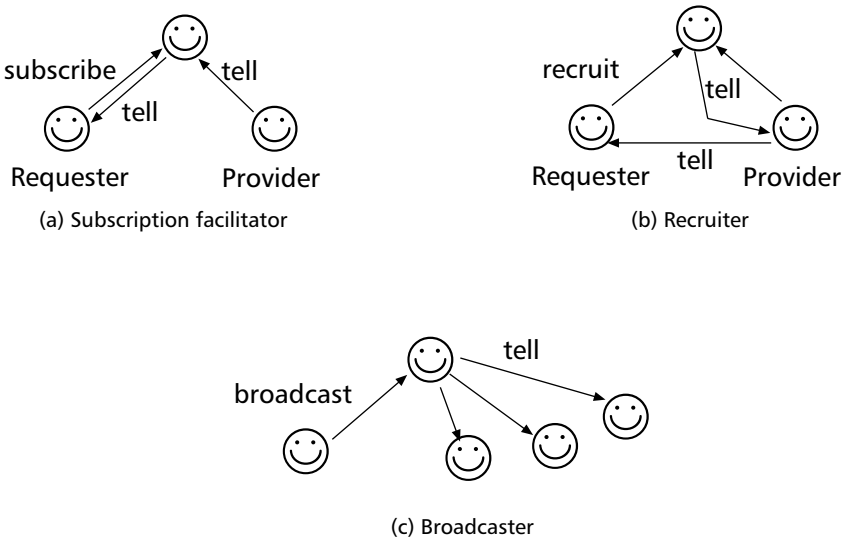


Figure 5 — Interagent communication using facilitator agents.

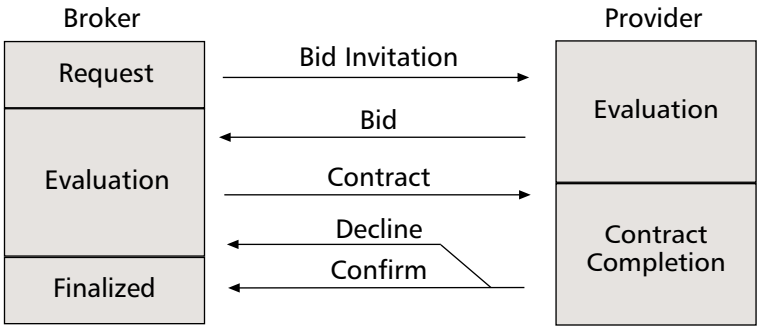


Figure 6 — Agent interaction protocol for negotiation.

Agent Mobility

Stationary agents exist as a single process on one host computer; mobile agents can pick up and move their code to a new host where they resume executing. Mobile agents are able to change

platforms and environments; stationary agents are not. From a conceptual standpoint, mobile agents can be regarded as itinerant, dynamic, wandering, roaming, or migrant. The rationale for mobility is the improved performance that can be achieved by moving the agent closer to the services available on the new host.

Stationary agents must use the network to exchange information, primarily using the *remote procedure call* (RPC), as illustrated in Figure 7(a). When a stationary agent requires processing on a different platform, it must employ the services of another agent. Here, a communication (or request) conveys the intention to invoke a specific operation (via an RPC). The operation is then executed, and the results (or reply) is returned to the requesting agent. Thus, the use of stationary agents:

- Reduces the complexity required for mobility
- Encourages specialization within platforms
- Employs well-established protocols
- Supports closed-environment philosophy

However, the stationary approach also:

- Results in performance problems in those situations requiring high volume or frequency
- Results in processing inefficiencies because having many specialized agents creates more work than having a single mobile agent
- Reduces effectiveness when a connection is lost

In contrast, mobile agents use the network to exchange information primarily by changing platforms and environments using the *remote programming* technique. When a mobile agent requires processing on a different platform, it physically relocates to the desired server, as illustrated in Figure 7(b). This requires that all structural and behavioral properties of the agent be transferred during migration and that any environmental differences be changed or accommodated. The big issues here are how much time is required to prepare for migration, how much data is actually transferred, and the performance of the transfer communication.

Migrations can be handled by the agent. Although this reduces the complexity of the runtime environment, it increases agent complexity. In contrast, migrations can be transparent to the agent, which reduces agent complexity while increasing the complexity of the runtime environment. The advantages of mobile agents are that they:

- Reduce network load
- Reduce network-related delay
- Reduce resource usage of clients
- Enable distributed problem solving
- Support asynchronous, autonomous processing
- Promote reconfigurable or customized services

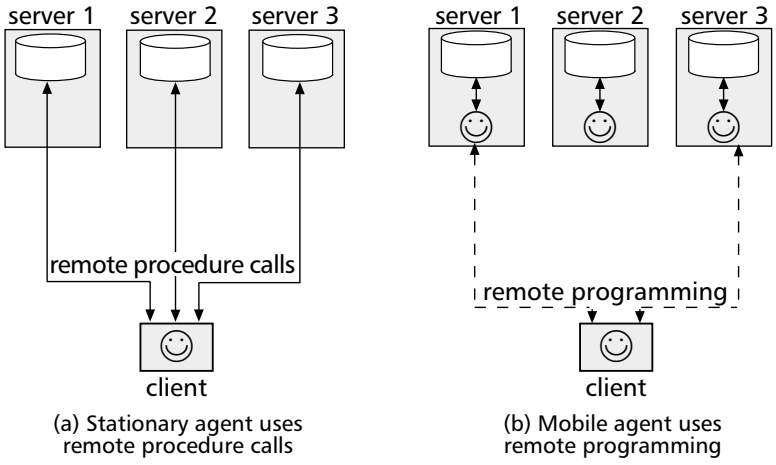


Figure 7 — Stationary and mobile agents.[6]

- Make active behavior scenarios conceivable
- Enhance decentralization options

The disadvantages of mobile agents are that they:

- Involve a number of security issues, such as the identification and authentication of agents, protection from destructive agents, and the assurance of the agent's willingness and ability to pay
- Require transport/migration mechanisms be added to software environments, thus increasing their complexity
- Have no industry standards for agent environments, migration approaches, or measuring and billing resource consumption
- Have not yet been used in an environment containing a large number of mobile agents

Agent Security[7]

Agents are software entities that often run in a distributed computing environment and interact with many other software entities, including other agents. When software runs in a distributed environment, security issues are numerous. The possibility of encountering security problems increases in open environments, such as the Internet or a virtual private network, or in any environment where all the entities are not known, understood, and administered by a single group.

Agents are software entities that often run in a distributed computing environment and interact with many other software entities, including other agents. When software runs in a distributed environment, security issues are numerous.

Various types of security risks are described below. Many of these risks are inherent to distributed computing environments, particularly when software passes messages that can be intercepted, modified, or destroyed. Although this is a threat to agent systems, it is also a threat to any software system that depends on messages being passed reliably. Another risk centers on whether the software can assume that it is using trustworthy services.

Generally, the word security refers to the actions taken to ensure that something is secure. If the item is free from danger, it can not be taken, lost, or damaged. In practice, security is usually applied only to somewhat valuable items, because implementing security has associated costs. This is true both in the everyday world, where we protect our cars or homes from theft (but not a disposable pen) and in the world of computing, where we may protect some, but not all, company resources.

Security policy refers to how access to valuable resources is controlled. For example, a company may have a policy about which groups can access which data, when certain types of processing jobs can run, or whether outside entities can connect to the company network. Agent systems will also require security policies that may control where agents can run, what they can do, and with what other entities they can communicate.

Security policies are usually based on *identity*, something that serves to identify or refer to an entity. In this way, an agent could be referred to by its name, a role that it is playing, the fact that it is a member of some organization, and so on. An agent, then, can have multiple forms of identity. For example, a particular agent could simultaneously be a purchasing agent working on behalf of user Rolf Smith, be playing the role of a bidder in a negotiation with E-Widgets, have its software composed of elements from company ExDeus, and have the serial number 98734501. Each of these identities might be important in different interactions.

Identity is based on a *credential*, a set of data that can be validated by a third party to prove that the entity is what it says it is. For example, when users log into a computer system, they often enter both a username and a password; the latter is the credential that is

validated to indicate that they really are that username. Other common mechanisms for identity and credentials are X.509 certificates and PGP keys.

Types of Security Risks

Here are some security threats that could happen in multiagent systems:

Unauthorized disclosure — a breach in the confidentiality of an agent's private data or meta data. For example, an entity eavesdrops on the interaction between agents and extracts information on the goals, plans, capabilities, or other information that belongs to the agents. Or, an entity can probe the running agent and extract useful information.

Unauthorized alteration — the unauthorized modification or corruption of an agent, its state, or data; for example, the content of messages is modified during transmission, or the agent's internal value of the last bid is modified.

Damage — destruction or subversion of a host's files, configuration, or hardware, or of an agent or its mission.

Unauthorized copy and replay — an attempt to copy an agent or a message and clone or retransmit it; for example, a malicious platform creates an illegal copy or a clone of an agent, or a message from an agent is illegally copied and retransmitted.

Denial of service — an attack that attempts to deny resources to the platform or an agent. For example, one agent floods another agent with requests, rendering the agent unable to provide its services to other agents.

Repudiation — an agent or agent platform denies that it received or that it has sent a message or taken a specific action. For example, a commitment between two agents as the result of a contract negotiation is later ignored by one of the agents. That agent denies the negotiation has ever taken place and refuses to honor its part of the commitment.

Spoofing and masquerading — an unauthorized agent or agent platform claims the identity of another agent or agent platform; for example, an agent registers as a directory service agent and therefore receives information from other registering agents.

Message Passing

In systems where agents pass messages, the importance of avoiding message alteration or disclosure is described above. If a message is altered, it might provide incorrect information or transmit a dangerous action. If a message can be read by or disclosed to other entities, the other entity may use the acquired data inappropriately.

Message alteration is usually avoided by providing a mechanism for authenticating the message. Most of the techniques for doing this are based on public/private key pair technologies, such as X.509 certificates. Additional information is sent with the message that allows the receiver to validate that the message has not been changed. Message disclosure is avoided by encrypting the message, which again is based mostly on public/private key pair technologies.

For both threats, the authentication or encryption can occur either by encrypting the message itself or by sending it through a transport that provides authentication or encryption services.

Other threats related to message passing include copy and replay, spoofing and masquerading, and repudiation. Both in copy and replay and in spoofing and masquerading, an agent may assume the identity of another agent. Using this false identity, it can communicate with another agent in order to request an inappropriate action. Many agent systems use relatively simplistic naming schemes (identities) with no additional credentials. Therefore, a message claiming to be from "Joe" cannot be validated.

This set of problems can be solved in various ways. By tagging messages with credentials, the message can be sent in a way that ensures authentication. Thus, the message can be sent without the possibility of tampering by a third party. Tagging messages with credentials can also help avoid repudiation. If a message is signed using a credential, the signing agent cannot later deny that it sent the message.

System Components Dealing with One Another

Agents can use agent platforms to provide services. They can also interact with well-known services, such as a *directory service* that helps them locate other agents or an *ontology service* that helps them look up ontologies. When two systems components interact, several risks can occur: the two most likely being damage or spoofing and masquerading.

In the damage scenario, the agent may do malicious or inappropriate things to the host system, such as corrupting or deleting files. Therefore, the agent platform may want to control which agents can take which actions. Typically, the agent would offer a credential that identifies it to the agent platform. After validating the credential, the agent platform would use a security policy (based on the agent's identity) to determine what actions the agent could take and would enforce that policy.

This is very much like the access control lists found in most operating systems. However, agent systems probably want to control much more than simply reading, writing, and running files. They might want to control message sending, usage of various resources, when and where an agent can move, and whether a moving agent can run on this platform.

Just as the agent platform may want to validate what entity it is dealing with, an agent may want to validate that it is dealing with an agent platform it knows to be genuine. Agent platforms and services could pretend to be "legitimate" but, in fact, have some dangerous behavior, such as recording message transmissions before encryption, cloning copies of the agent for its own purposes, or providing false information.

Other Risks to Which Agents Can Be Exposed

In constructing software for an agent, certain types of risks must be addressed to ensure that the following things cannot occur:

- Viewing the private security key of the agent
- Viewing the private data of the agent (i.e., the highest bid an agent is willing to make on a product)
- Invoking private methods in the agent
- Designing public methods in such a way that permits security risks

More Security Considerations

When designing agent systems, the following aspects of security, security policy, and identity should be considered:

- Agents and agent platforms can have multiple credentials. Multiple credentials reflect the reality that we have multiple roles. Users may have credentials as part of several organizations, as an individual, as the owner of multiple credit cards, and so on
- Agents can have their own credentials. They may also have credentials for the user that they represent in an e-commerce application
- Agents should not be created that can act anonymously. For example, users may want to obtain data about drug or alcohol treatment without revealing their identity. Obviously, sites can choose to reject these agents if their security policies do not allow interaction with anonymous entities.
- All aspects of security need to be managed
- Traceability of actions can be useful
- Using a lease model on any credential can be helpful. In a lease model, credentials expire

after a certain period of time but can be renewed from a credential authority. This control can be a very effective way to clean up credentials in a system that uses relatively short-lived agents. Requiring long-lived agents to renew their credentials is also useful, because when an entity with bad credentials is forced to renew, it will be rejected and shut down

- Identity and credentials are also useful for building reputation services. Such services provide a way to determine whether a particular entity has behaved responsibly

Technologies Currently Being Used

Almost all of the systems being built today are being built with:

- Programming language: Java or C++
- ACL: KQML or FIPA ACL
- ACL content: represented as strings or XML documents or using a content language such as KIF or SL1
- Movement for mobile agents: most current systems use Java serialization

The first commercial toolsets for building agents have entered the market during the past year. Some can be purchased, others downloaded free. These agent building systems vary widely in functionality and do not adhere to any standards. Agents built in one system will not work in others, nor is there uniform support for communications protocols across these tools. However, the OMG's Agent Working Group and FIPA are currently working on standards for agents and agent-based systems. Table 1 gives an overview of commercial agent systems available at the time of this writing. For more information on the items in this table, see www.agentbuilder.com/AgentTools/index.html. Table 2 (on page 28) lists company involvement by category.

AGENT ARCHITECTURE

For the most part, agents will be deployed within conventional enterprises and will draw on the enterprise for many services. CORBA provides a rich source of services and a proven architecture. This section provides a framework for considering how a system supporting agents might draw on CORBA services and facilities. The architectural basis for this discussion will be the FIPA architecture.[8] The FIPA *agent platform* (AP) provides a good construct from which to discuss the enterprise-related issues in agent deployment and is summarized in Figure 8 (on page 28).

Agent Platform

The key element to the enterprise architecture is the AP, which provides an infrastructure in which agents can be deployed. An agent must be registered on a platform in order to interact with other agents on that or other platforms. Minimally, an AP consists of an *agent management system* and an *agent platform communication channel*.

FIPA does not specify the physical nature of a platform, however, two cases should be considered: that of a single host and that of multiple processors deployed as a "virtual" platform. If the platform is virtual, having it fulfill several requirements would be wise. It should have:

- High-speed communications
- A single agent management system
- A single agent platform security manager

These last two requirements make the agent system easier to use. From the system perspective, the lifecycle and security of all agents in a given platform is controlled by a single entity: the agent management system. From the perspective of a human (or agent proxy), the platform itself should also be controlled by a single entity.

Agent Management System

The agent management system (AMS) can be implemented as a single agent that supervises access to and use of the agent platform. AMS maintains a

Table 1 — Overview of Commercial Agent Systems

Product	Company	Language	Description
AgentBuilder	Reticular Systems, Inc.	Java	Integrated agent and agency development environment
AgenTalk	NTT/Ishida	LISP	Multiagent coordination protocols
Agent Development Environment	Gensym	Java	Environment
Agentx	International Knowledge Systems	Java	Agent development environment
Aglets	IBM Japan	Java	Mobile agents
Concordia	Mitsubishi Electric	Java	Mobile agents
Grasshopper	IKV++	Java	Mobile agents
Infosleuth	Microelectronics and Computer Technology Corporation	Java, Perl, Tk/tcl	Cooperative information agent environment
iGEN	CHI Systems	C/C++	Cognitive agent toolkit
Intelligent Agent Factory	Bits & Pixels	Java	Agent development tool
Intelligent Agent Library	Bits & Pixels	Java	Agent library
JACK Intelligent Agents	Agent Oriented Software Pty. Ltd.	JACK Agent Language	Environment
Jumping Beans Engineering	Ad Astra Engineering, Inc.	Java	Mobile agents
LiveAgent	AgentSoft Ltd.	Java	Internet agent construction
Microsoft Agent	Microsoft Corporation	Active X	Interface creatures
Pathwalker	Fujitsu	Java	Agent development tool
Swarm	Swarm Development Group	Objective C, Java	Agent simulator
Versatile Intelligent Agents	Kinetoscope	Java	Agent building blocks
Voyager	Object Space	Java	Agent-enhanced object request broker

directory of logical agent names and their associated transport addresses for an agent platform. AMS is responsible for managing the lifecycle of the agents on the platform; its actions include:

- Authentication
- Registration
- Deregistration
- Modification
- Query platform profile
- Search
- Mobility requests
- Control of agent lifecycle

AMS also provides two kinds of directory services to other agents: white pages and yellow pages. White page service is simply a way of locating individual agents. Yellow pages offer a way of locating agents for a given category. Agents may register their services with the directory or query it to determine what services are offered by other agents. For example, an agent might register itself as a rare art broker or punch press resource for other agents (including human) to contract its services.

Agent Platform Security Manager

The agent platform security manager (APSM) is responsible for maintaining security policies for the platform and infrastructure. APSM is responsible for runtime activities such as communications, transport-level security, and audit

Table 2 — Some Companies Involved in Agent Software

Languages	❑ Lotus Development Corp.
❑ Microsoft, Inc.	❑ Neuron Data, Inc.
❑ General Magic, Inc.	**Personalization**
❑ Sun, Inc.	❑ Broadvision, Inc.
❑ Vertel, Inc.	❑ Guideware, Inc.
❑ Agentsoft, Inc.	❑ Agnetsoft, Inc.
❑ First Virtual Holdings, Inc.	❑ Wisewire, Inc.
Development environments	❑ Aptex, Inc.
❑ Agentsoft	❑ Vignette, Inc.
❑ Autonomy	❑ Firefly, Inc.
❑ Crystaliz	**Research and development**
❑ Firefly Network	❑ British Telecom
❑ FTP Software	❑ Oracle, Inc.
❑ Fujitsu	❑ Digital Equipment Corp
❑ Gensym	❑ AT&T
❑ IBM	❑ Apple Computer, Inc.
❑ KYMA Software	❑ Logica, Ltd.
❑ MCC	❑ Siemens
❑ Microsoft	**Class libraries**
❑ Mitsubishi	❑ Agentsoft, Inc.
❑ ObjectSpace	❑ IBM, Inc.
❑ Oracle	❑ General Magic, Inc.
❑ Reticular Systems	❑ Objectspace, Inc.
❑ Toshiba	❑ FTP Software, Inc.
❑ Blackboard Technology	❑ Crystaliz, Inc.

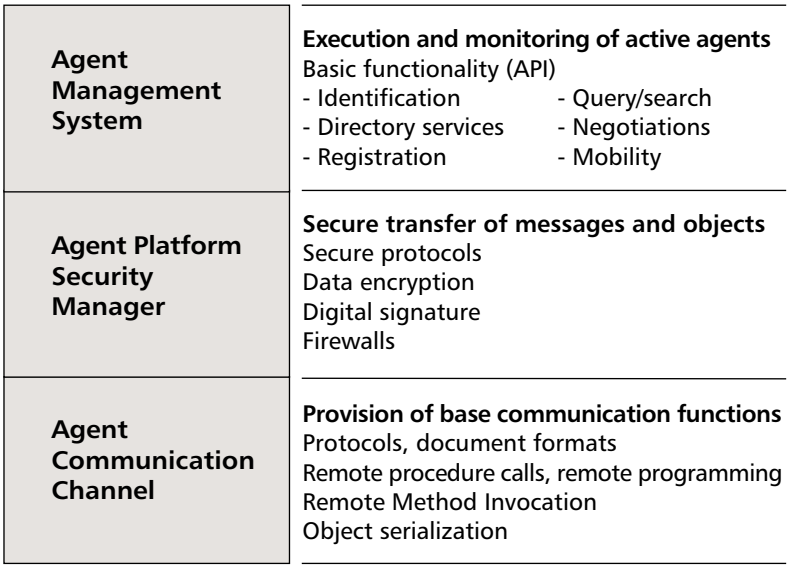


Figure 8 — The agent platform: an infrastructure for deploying agents.

trails. Security cannot be guaranteed unless, at a minimum, all communication between agents is carried out through APSM.

APSM is responsible for negotiating the requested inter- and intra-domain security services with other APSMs in concert with the implemented distributed computing architecture, such as CORBA, COM, and DCE, on behalf of the agents in its domain. APSM is responsible for enforcing the security policy of its domain, and can, at its discretion, upgrade the level of security requested by an agent. The APSM cannot downgrade the level of services requested by an agent, but it must inform the agent that the service level requested cannot be provided.

Agent Platform Communication Channel

All agents have access to the agent platform communication channel, which provides a path for basic interchange among agents, agent services, AMS, and other agent platforms. It must at least support Internet Inter-ORB Protocol. Agents can reach other agents on any number of other platforms through the agent communication channel. Access to agents outside of the local namespace could be supported by the CORBA Trader Services.

CONCLUSION

Agents and agent-based technology are an evolution, not a revolution — emerging both from and with

today's technology. Currently, agents, objects, relational databases, embedded systems, and so on have their own niches. As stated earlier, together they can orchestrate rich systems that none of these technologies alone could provide.

Agents will move into the mainstream of personal and corporate computing during the next three years, particularly in utilities, banking, healthcare, and telecommunications. Although agent technology itself is not a revolution, its usage will be revolutionary. Some of the major opportunities for agents include the following:

- They will constantly scan and collect data on their own internal processes, products, and services to identify new opportunities and markets as well as potential threats, risks, and challenges.
- They will sense changes in the global marketplace with their suppliers, vendors, customers, clients, and competitors; adapt to changes made by regulators; and learn from these changes.
- They will establish huge knowledge pools of products in order to filter, interpret, and present data to management in new ways.
- They will be an engine of continuous change, adapting to the market and rapidly delivering quality products and services at lower cost per unit.
- Their importance in decision-making will grow enormously.
- Solving large combinatorial optimization problems will be one of the most important uses of agents.

Some other issues to consider include:

- First-generation agent products are now being released. Future products will include intelligence, personalities, and interactive features.
- Issues such as privacy, access to data, and other legal elements need to be addressed.
- As agents become more accepted, they will begin to communicate with each other in multiagent agencies.

ACKNOWLEDGEMENTS

The author would like to acknowledge Dr. David Greenstein (dgreenst@tir.com) for providing a comprehensive description of his agent-based AMIS case study. I would also like to thank Kate Stout (kate.stout@sun.com) for sharing her expertise in the area of agent-based security.

REFERENCES

[1]Resnick, Mitchell. *Termites, Turtles, and Traffic Jams: Explorations in Massively Parallel Microworlds*, MIT Press, Cambridge, Massachusetts, 1997.

[2]Figure courtesy of Van Parunak (ERIM, Inc., vparunak@erim.org).

[3]Waldrop, M. Mitchell. *Complexity: The Emerging Science at the Edge of Order and Chaos*. Simon and Schuster, New York, 1992.

[4]Greenstein, David, and Kelly Thomas. "Intelligent Agents for an Emergent Industrial Ecology." *Intelligent Manufacturing Systems, Proceedings of IJCAI*, AAAI 1995.

[5]Caglayan, Alper, and Colin Harrison. *Agent Sourcebook*. John Wiley & Sons, New York, 1997.

[6]Brenner, Walter, Rüdiger Zarnekow, and Hartmut Wittig. *Intelligent Software Agents: Foundations and Applications*. Springer-Verlag, Berlin, 1998.

[7]Stout, Kate, adapted from her contribution to the "Agent Technology Green Paper," Object Management Group Agent Working Group, 2000.

[8]"Foundation for Intelligent Physical Agents FIPA98 Agent Management Specification." Geneva, Switzerland, October 1998. (www.fipa.org)

Cutter Consortium Distributed Computing Architecture/e-Business Advisory Service

Senior Consultants

Douglas Barry

on storing objects using DBMSs

Douglas Barry is principal of Barry & Associates, Inc., which he founded in 1992, and executive director of the Object Data Management Group, an industry standards organization. Mr. Barry has worked with DBMS technology for more than 20 years and with object technology and databases since 1987. He has been actively involved in creating and promoting standards for storing objects in databases. Mr. Barry specializes in the strategies, technologies, and products associated with objects and object-relational database applications.

Peng Boey

on component-based architecture for e-business applications

Peng Boey is vice president of Consulting Services with NetNumina Solutions. With more than 10 years of IT experience, Mr. Boey is an expert in distributed systems architecture. He has provided professional advice to Global 1000 companies on how to design, build, and deploy component-based architectures for e-business applications. He has been a leader in creating the VIEW methodology, a revolutionary architecture process that utilizes the latest enabling technologies for building mission-critical e-business systems for the Internet, as well as for intranets and extranets. Currently, Mr. Boey is conducting research on developing component frameworks for rapidly building e-business solutions.

Theodore R. Burghart

on developing heterogeneous client-server and distributed systems

Theodore Burghart is principal engineer at Quoin, Inc. He has extensive experience with heterogeneous client-server and distributed systems design and development. His projects have included communications, database, technical, and process control services implementations. Mr. Burghart is experienced in cross-platform enabling technologies, such as CORBA as well as with LDAP, and with relational, object/relational, object-oriented, and full-text databases. Currently, Mr. Burghart collaborates with development teams to define and construct CORBA-based infrastructures. In addition, Mr. Burghart provides technology training and consulting services to clients in the healthcare, insurance, and financial services industries.

Richard Dué

on component development methods and project management

Richard T. Dué is president of Thomsen Dué and Associates Limited. He specializes in object and component development methods and in object technology project management. Mr. Dué has developed and presented information technology training courses in 28 countries to participants from hundreds of organizations. He is a member of the OPEN methodology consortium and has been actively involved in developing business object standards. Mr. Dué is a frequent contributor to the *Cutter IT Journal*, and has held various management positions in the public and private sector in the US and Canada.

David Frankel

on Java- and Internet-based component architectures

David Frankel is chief scientist at Genesis Development. He assists clients in developing and customizing advanced component architectures based on CORBA, DCOM, Java, the Internet, and related technologies. Mr. Frankel has been instrumental in formalizing advanced component architecture to support large-scale software development and systems integration. He is a member of the OMG Architecture Board, was a major contributor to the CORBA/COM Internetworking standard, and is cochair of the OMG Business Object Initiative Working Group.

Max Grasso

on distributed secure transaction systems

Max P. Grasso is chief technology officer of NetNumina Solutions. He is a recognized expert on distributed secure transaction systems with a focus on high reliability, mission-critical applications. He also has significant expertise in the management issues involved with deploying such systems. Mr. Grasso has been at the forefront of distributed computing technology since its beginning, both as a member of the Open Software Foundation's team and as a cofounder of the Open Environment Corporation. As CTO of Internet Business Solutions, his mission was building the technology for the execution of secure distributed transactions on the Internet. In that role he designed a framework for business-to-business transactions and interenterprise transactional workflows. Mr. Grasso has overseen the architecture and the design of large systems in the telecommunication, financial, banking and gaming industries.

Michael Guttman

on transitioning to enterprise component technology

Michael Guttman is chief technical officer and cofounder of Genesis Development, where he assists clients in planning their transitions to enterprise component technology. Mr. Guttman, who has been a pioneer in the use of component and object technology for large-scale distributed systems, is a specialist in advanced component architectures. Mr. Guttman has more than 20 years of experience in software development and has been a major contributor to several OMG standards, including CORBA 1.0, CORBA IIOP, and CORBA/COM Internetworking.

Senior Consultants

Curt Hall

on data warehousing and data management strategies

Curt Hall, editor of *Business Intelligence Advisor*, is an expert on data warehousing technologies and products. His recent study on the corporate use of data warehouses and the issues associated with data warehousing projects has resulted in the in-depth report *Data Warehousing for Business Intelligence*. Mr. Hall is the coauthor of *Intelligent Software Systems Development* and a contributing editor to James Martin and James Odell's *Object-Oriented Methods: Pragmatic Considerations*. He is the former associate editor of *Object-Oriented Strategies* and *Application Development Strategies*. Mr. Hall's work has appeared in technical journals such as *IEEE Expert*. He has also been an organizer of and speaker at industry events such as *ObjectWorld*.

Paul Harmon

on distributed computing and component development for business applications

Paul Harmon is a well-known consultant and analyst of software trends. Mr. Harmon has been very influential in the movement to commercialize object and component technologies for business applications. Mr. Harmon recently completed a study of the acceptance of object technology and components in corporate development groups. As editor since 1991 of *Component Development Strategies*, published by Cutter Information Corp., Mr. Harmon has studied the commercial and business applications of object technology. He has also been the editor of three other Cutter Information Corp. newsletters over the years: *Intelligent Software Strategies*, *Application Development Strategies*, and *Business Process Strategies*. Mr. Harmon is a frequent speaker on the strategic impact of new software technologies on business. Mr. Harmon is the coauthor of several books.

Ian Hayes

on e-business strategy

Ian Hayes is founder president of Clarity Consulting, Inc., where he provides strategic consulting on issues affecting the management and support of corporate business systems. Mr. Hayes has advised dozens of *Fortune* 1000 companies on a variety of IT issues, including major Y2000 initiatives, e-business, insourcing, outsourcing, and process improvement. Mr. Hayes is a regular contributor to the *Cutter IT Journal* and is on the editorial advisory board of the *Enterprise Application Integration Journal*. Mr. Hayes was a cofounder of Language Technology, Inc., an early software redevelopment product vendor, and a practice manager at Keane, Inc. before founding Clarity Consulting in 1993.

J. Bradford Kain

on distributed business components

Brad Kain is CEO and cofounder of Quoin, Inc., providing consulting, mentoring, and software development services in object and distributed technology. Mr. Kain has used object-oriented analysis and design since 1987. He has helped define the use of object and distributed technology to realize distributed business components. This work has involved the specification of sophisticated intranet, Java, and distributed applications. Mr. Kain has managed the technical direction and development teams of distributed application infrastructure development projects for managed care, client management, general ledger, securities trading, marketing, engineering and manufacturing design, and other applications. Mr. Kain has participated in the work of the Object Management Group's Technical Committee on CORBA and the specification of domain services.

André Leclerc

on formal specification approaches to the development of information and management systems

André Leclerc is the director of development for Technology Development Associates, Inc., where he is active in developing, training, consulting and mentoring object-oriented information systems. Mr. Leclerc's interest is in formal specification approaches to the development of information and management systems. In 1984, Mr. Leclerc was appointed vice president of Yourdon, Inc. Following his tenure at Yourdon, Inc., he served as vice president of Kenneth G. Moore and Associates. Mr. Leclerc has authored a book on structured PL/1, and a variety of articles, seminars, and tutorials on information systems, including the OO seminars for Ptech, Inc.

Jean Pierre LeJacq

on architecture and implementation of distributed systems

Jean Pierre LeJacq, an experienced architect, designer, and implementer of distributed systems, is CTO and cofounder of Quoin, Inc., providing consulting in object and distributed technologies to clients worldwide. Mr. LeJacq is the architect and technical lead for the development of an infrastructure for distributed application development, and is responsible for the design and implementation of a CORBA-based system. He has extensive experience in Java, C++, and UNIX-based systems, and in a variety of design methods. Mr. LeJacq has been using object-oriented languages and modeling systems since 1984 for clinical, managed care, client management, engineering and manufacturing design, and aircraft control simulation applications.

Jason Matthews

on transitioning to enterprise component technology

Jason Matthews is cofounder of Genesis Development. He has nearly 20 years of technical and management experience in software development and related professional services. Mr. Matthews is a pioneer in the use of component/object technology for large-scale distributed systems and the Internet, and a specialist in the process of transitioning large organizations to component technology. Mr. Matthews has managed end-user information systems organizations and the development of commercial software products. He has been a consultant to a wide range of industries, including financial services, insurance, healthcare, manufacturing, telecommunications, and energy.

SENIOR CONSULTANTS

JAMES ODELL

on object-oriented methodologies and agent technology

James Odell was an early innovator of information engineering methodologies, and has spent most of his 30-year career developing better methods to understand, communicate, and manage system requirements. Working with the Object Management Group (OMG) and other major methodologists, Mr. Odell continues to innovate and improve object-oriented methods and techniques. He participated in the development of the UML, and is the cochair of both the OMG's Object Analysis and Design Task Force as well as the Agents Work Group. Formerly, Mr. Odell was the principal consultant for KnowledgeWare, Inc., where he pioneered the concepts of data modeling, information strategy planning, and CASE technology application. Mr. Odell has coauthored several books with James Martin, including the most recent title *Object-Oriented Methods: A Foundation, UML Edition*.

CHRIS PICKERING

on e-business trends and strategies

Chris Pickering, president of the research and consulting firm Systems Development, Inc., analyzes industry practices. Mr. Pickering's areas of focus include information architecture, business-IT alignment, technology acquisition and deployment, organizational change, system modeling, and software practices. He is the author of the survey-based study *E-Business Trends, Strategies, and Technologies* and the periodic *Survey of Advanced Technology*, which tracks the use of advanced information technologies, assesses the effectiveness of that use, and identifies the benefits and hazards of using the leading technologies. He then applies the lessons learned from the research to helping clients maximize their information technology investments.

JOHN R. RYMER

on tools, middleware, and application development for distributed systems

John Rymer is president of Upstream Consulting, which he founded in 1997. Mr. Rymer is a well-known strategy advisor and a veteran industry analyst. Since 1989, Mr. Rymer has developed a strong track record of helping software companies solve difficult market and technical problems. He specializes in application development technology for distributed systems, including tools and middleware. Mr. Rymer is a former vice president and founding analyst at Giga Information Group, Inc., where he was responsible for tracking application development technology and products. Mr. Rymer has been a keynote speaker at *OOPSLA, Networld + Interop, ObjectWorld,* and other industry conferences.

GREG SABATINO

on architecting and implementing highly scalable distributed e-business solutions

Greg Sabatino, cofounder of NetNumina Solutions, specializes in the architecture and implementation of highly scalable distributed e-business solutions. Mr. Sabatino's career has centered on the training, support, and delivery of distributed computing architectures and applications for IT organizations worldwide. His efforts focus on enabling organizations to successfully integrate and employ emerging technologies in order to realize a strategic advantage. Mr. Sabatino's experience spans the retail, petrochemical, telecommunications, pharmaceutical and, especially, finance industries. Mr. Sabatino contributes to several industry publications and speaks at conferences on a variety of distributed computing issues.

KENT SEINFELD

on enterprise information architecture development

Kent Seinfeld is the founder of Enright Consulting, a small group of senior IT consultants. Mr. Seinfeld specializes in enterprise information architecture development. Mr. Seinfeld is a former senior vice president of IT and served in three different positions with CoreStates Bank. He was the founder and manager of the Technology Planning and Research group at CIGNA, a global insurance and financial service company, where he was responsible for computing standards, security policy, development methodologies, and the research and development program. Mr. Seinfeld was the CIO for Girard Bank. Earlier in his tenure he was the principal architect in the design and implementation of a large-scale highly integrated banking system. This system evolved into the foundation of one of the first large ATM networks.

ROGER SESSIONS

on distributed middle-tier technologies

Roger Sessions is the world's leading expert on Microsoft's distributed middle-tier technologies, including COM, DCOM, and MTS. Prior to starting his company, ObjectWatch, Inc., Mr. Sessions worked at IBM, where he was an architect of one of the CORBA services. He was also a lead architect for IBM's implementation of the CORBA Persistence Service, gaining an unparalleled perspective on middle-tier technologies. Mr. Sessions has written four books; his most recent is *COM and DCOM: Microsoft's Vision for Distributed Objects*. He writes the highly respected and often controversial online *ObjectWatch Newsletter*. In addition to frequent speaking engagements worldwide, Mr. Sessions writes articles for many industry publications.

ED YOURDON

on object-oriented design and analysis

Edward Yourdon is widely known as the lead developer of the structured analysis/design methods of the 1970s. He was a codeveloper of the Yourdon/Whitehead method of object-oriented analysis/design and the popular Coad/Yourdon OO methodology. He is also the editor of the *Cutter IT Journal*. Mr. Yourdon is currently focused on issues of business/IT alignment; mitigating risks of large outsourcing initiatives; auditing of large, risky projects; and the development and implementation of e-business initiatives as well as forecasting and tracking critical business/IT "megatrends" in the coming decade. Mr. Yourdon is currently a member of the Airlie Council, a group of high-end advisors formulating software "best practices" for the US Department of Defense. Mr. Yourdon has authored more than 200 technical articles; he has also written 25 computer books since 1967.