

# Agent Technology: What Is It and Why Do We Care?

by James Odell, Senior Consultant,  
Cutter Consortium

Agent technology is now necessary to reduce costs; to improve efficiency and effectiveness; and to support the requirements of individuals, groups, companies, and universities as they collaborate globally. More importantly, this type of technology enables us to create and support a whole class of IT applications and approaches that we previously could not have developed.

In this *Executive Report*, we discuss what agent technology is as well as how and where it is currently being used in various industries.

# Cutter Business Technology Council



Rob Austin



Tom DeMarco



Christine Davis



Lynne Elyn



Jim Highsmith



Tim Lister



Lou Mazzucchelli



Ken Orr



Sheleen Quish



Ed Yourdon



## About Cutter Consortium

Cutter Consortium is a unique IT advisory firm, comprising a group of more than 150 internationally recognized experts who have come together to offer content, consulting and training to our clients. These experts are committed to delivering top-level, critical, and objective advice. They have done, and are doing, groundbreaking work in organizations worldwide, helping companies deal with issues in the core areas of software development and agile project management, enterprise architecture, business technology trends and strategies, enterprise risk management, business intelligence, metrics, and sourcing.

Cutter delivers what no other IT research firm can: We give you Access to the Experts. You get practitioners' points of view, derived from hands-on experience with the same critical issues you are facing, not the perspective of a desk-bound analyst who can only make predictions and observations on what's happening in the marketplace. With Cutter Consortium, you get the best practices and lessons learned from the world's leading experts, experts who are implementing these techniques at companies like yours right now.

Cutter's clients are able to tap into its expertise in a variety of formats including print and online advisory services and journals, mentoring, workshops, training, and consulting. And by customizing our information products and training/consulting services, you get the solutions you need, while staying within your budget.

Cutter Consortium's philosophy is that there is no single right solution for all enterprises, or all departments within one enterprise, or even all projects within a department. Cutter believes that the complexity of the business technology issues confronting corporations today demands multiple detailed perspectives from which a company can view its opportunities and risks in order to make the right strategic and tactical decisions. The simplistic pronouncements other analyst firms make do not take into account the unique situation of each organization. This is another reason to present the several sides to each issue: to enable clients to determine the course of action that best fits their unique situation.

**For more information, contact Cutter Consortium at +1 781 648 8700 or [sales@cutter.com](mailto:sales@cutter.com).**

# Agent Technology: What Is It and Why Do We Care?

## ENTERPRISE ARCHITECTURE ADVISORY SERVICE

Executive Report, Vol. 10, No. 3

---

by James Odell, Senior Consultant, Cutter Consortium

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, we are no longer in the era of main-frame computing, when both companies and applications were typically command-and-control-oriented and organized in vertical silos. With the combination of the Internet, fiber optics, and PCs, the business and technology playing field has been flattened. No longer primarily top-down, it has shifted more to a side-by-side approach — as individuals, small groups, and organizations interact around the world. Furthermore, 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 new markets will be left behind.

Agent technology is a primary enabler to support this new era. In fact, *without* it, our current technology will not scale to support the ever-increasing global interaction. 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 not only adapt to their environment but also 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.

The biggest breakthrough with agents is that they are an *evolution* of existing technologies. What is *revolutionary* is the way we think about and use agents to design IT systems. They are being built *from* today's technology and will work together *with* today's technology. While agents, objects, relational databases, legacy systems, service-oriented architectures (SOAs), event-driven approaches, and so on, each have their own niche, together they can orchestrate rich systems that none of these technologies could provide alone.

In this *Executive Report*, we first discuss how agent technologies — including ant-based systems — are improving operations in different industry sectors. We then explore agent technology in more

---

detail by defining it, examining its internal architecture and properties, and understanding how various agent-based systems adapt and interact within environments.

### WHY SHOULD WE CARE ABOUT AGENT TECHNOLOGY?

#### *First, An Example from Everyday Life*

Living creatures and plants can produce complex everyday

phenomena (e.g., ant colonies, traffic jams, stock markets, and prairie ecosystems). Similarly, automated entities — called *agents* — can enable supply chain systems, planning and scheduling applications, Web services, and so on. Such agents need not be complicated. For example, the ant colony simulation of StarLogo<sup>1</sup> (see Figure 1) uses software agents, where each ant follows three simple rules:

1. Wander randomly.
2. If food is found, take a piece back to the colony and leave a trail of pheromones, which evaporates over time; then go back to rule one.
3. If a pheromone trail is found, follow it to the food and then go back to rule two.

In Figure 1a, the “ant” agents are just emerging from the anthill to begin their random walk. Eventually, an ant discovers a food source and returns some to the colony, leaving a trail of evaporative pheromones, as shown in Figure 1b. Figure 1c shows the ant colony well underway in retrieving the food. Lastly, Figure 1d depicts two depleted food sources and one that is exhausted altogether.

#### *Ant-Like Agents at British Telecom and HP Labs*

The ant colony is organized without an organizer, coordinated without a coordinator. As it turns out, this simple ant foraging technique provides an inspiration to tackling some difficult technological problems. While the insect’s totally random approach to locating food may appear to be inefficient, the study of this technique has led to a new area of research known as “swarm intelligence” [2].

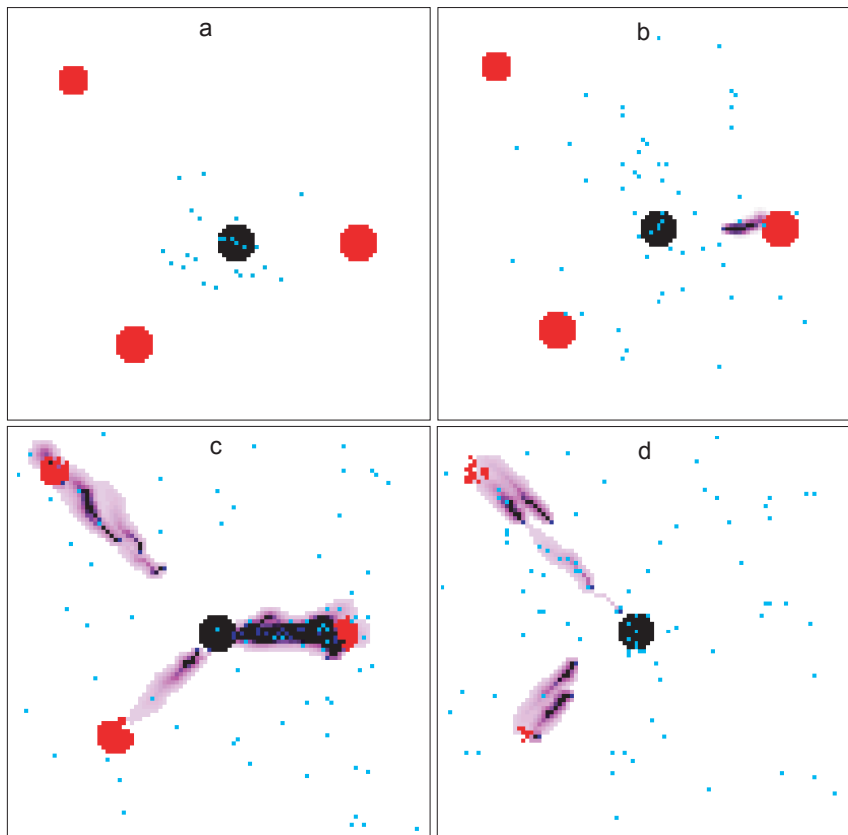


Figure 1 — Snapshots from an ant colony simulation using software agents.

<sup>1</sup>StarLogo software is available for free at <http://education.mit.edu/starlogo>.

For example, a decade ago, British Telecom (BT) used swarm intelligence for inspiration to make software systems simpler, cheaper, and more intelligent and robust. At the time, Chris Winter, then head of BT Future Systems, was concerned that the trend of IT systems was toward central control. “Ants walk around at random and there’s no intelligent controller telling them what to do,” said Winter. “When they find food, they simply run back to the nest laying a little pheromone trail that other ants can smell. The rest of the colony need only follow that path to fetch the remaining food” [3]. In other words, the processing is distributed, not centrally controlled.

Winter used ants initially to weed out system bugs. Today, in BT’s development network, little ant-like programs run around finding problems. They leave pheromone-like time stamps that highlight the fault and draw help from other software. In other words, BT is using the ants as a first step in self-healing systems software. Many software companies are now embedding ant-like agents in systems software and middleware for similar reasons.

In another application, BT and HP Labs created the world’s first ant-based systems [9]. Both telephone networks and the Internet typically route connections through a number of intermediate switching stations. When the

network is large, many possible routes exist for each connection. Using a centrally controlled routing approach, however, scales badly and can lead to failure of the entire system. A decentralized mechanism like the ant-based approach was found to both scale well with the network size and avoid the possibility of a system-wide failure.

The approach here is to send ant-like agents periodically between nodes. At each node, the agent updates the node’s *routing table* with information (“pheromone”) indicating how long the journey took from its origin and which nodes the agent used along the way. The routing table contains a list of the node’s immediate neighbors and probabilities associated with using that neighbor as the next step on the journey toward a target node in the network. The fastest ants will have a positive effect on the probability scores of the nodes they used, while slow ants will have a negative effect.

Network congestion is another problem that was addressed by BT and HP. They considered congestion in a particular section of the network to be analogous to the depletion of a food source near an ant colony. Here, ant-like agents would search for new routes and dynamically update the virtual pheromone trail recorded in routing tables.

### **Other Uses of Ant-Based Systems**

In ant-based systems, an ant colony of a finite size searches collectively for a good solution to a given optimization problem. The optimal solution can only be found through the global cooperation of all the colony’s ants, where the ants only communicate indirectly by adding pheromones to the environment. The BT applications above are only two examples of where techniques start with models of ant behavior and then add things that are not present in the real world. However, there are many more [2, 8], some of which include:

- **Traveling salesman problem.** A salesman must find the shortest route by which to visit a given number of cities, each city exactly once.
- **Quadratic assignment problem.** This involves the problem of assigning facilities to locations so that the costs of the assignment are minimized.
- **Job-shop scheduling problem.** For a set of machines and a set of jobs, operations must be assigned to time intervals in such a way that (1) no two jobs are processed at the same time on the same machine, and (2) the maximum of the completion times of all operations is minimized.
- **Vehicle routing problem.** This involves finding minimum cost vehicle routes where (1) every

customer is visited exactly once by exactly one vehicle; (2) for every vehicle, the total demand does not exceed the vehicle capacity; (3) the total tour length of each vehicle does not exceed a given limit; and (4) every vehicle starts and ends its tour at the same position (the depot).

### **Beyond Ants**

Individual ants are not very sophisticated insects. They have limited memory and a largely random element to their behavior. However, acting as a collective, ants can perform complex tasks with reliability and consistency. Ant colonies are not the only things that work like this. Beehives, flocks of birds, freeway traffic, national and global economies, societies, and immune systems are all examples of patterns that are determined predominantly by local component interaction instead of 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 where local components also have control — instead of limiting system-design approaches *solely* 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?

### **Painting Trucks at General Motors**

Traditionally, assembly line schedules are centrally developed and controlled. Any change in the schedule must be centrally reconfigured. When the line is small and has few unplanned stoppages, centrally controlled schedules work well. However, scheduling for most real-world assembly lines can be a nightmare: work stations break down, personnel get sick, environmental conditions are not always within acceptable limits, products coming down the line have or acquire unexpected defects, and so on.

Dick Morley, a technology visionary and father of the programmable controller, swept away old assembly line schedules and developed a better system for painting trucks at GM's assembly plant in Fort Wayne, Indiana. "How do I schedule the non-schedulable?" Morley wondered. "Trucks do not come down the line in order of their color, and frequently no paint booth is available with the correct color." Morley also discovered that many of the paint booths were typically broken down or being repaired.

In his technique, the scheduling program interacts with each paint booth. Instead of assigning unpainted trucks to booths, GM's solution was to have the booths bid on the paint jobs [4]. To accomplish this, each booth was

equipped with a simple software agent that was programmed to keep its booth busy and bid on each paint job. The amount of the booth's bid was based on how busy the booth was at the moment of bidding, whether it had to change to a different paint, and whether the booth was functioning properly.

To coordinate the various bids for each paint job, a scheduler agent acts as a broker. For example, when a truck arrives to be painted, the scheduler agent tells the booths, "I have a truck that needs to be painted red." A vacant paint booth already loaded with red paint will bid very high. However, a vacant booth with a different color would bid lower because of the extra labor and time to clean and reload the paint gun. A booth that has just started to paint a truck, has broken down, or is otherwise less suited for the job would bid even lower. Based on the outcome of the bidding activity, the scheduler assigns the truck to the highest-bidding paint booth.

In a top-down planned "push-through" world, if one booth malfunctioned, a centrally controlled system would require immediate recomputing. With bottom-up, "pull-through" paint booth agents, other booths were ready to pick up the bidding slack at a moment's notice. This new design saved \$1 million dollars in nine months and reduced the lines of computer code from hundreds to four [4].

Morley and GM tackled a problem where centralized scheduling did not work efficiently by adopting an agent-based approach where each booth acts on its own behalf using a market-based bidding system. Even though the scheduler was a centralized element, it deferred to distributed booth agents. Agent-based solutions do not remove centralization; instead, they try to balance it with distributed solutions *wherever it makes sense*.

### Dynamic Scheduling

Current market trends are driving organizations from mass production (where the supplier tells the customer what to buy) to mass customization (where the customer tells the supplier what to provide). For small supplier organizations where resources and requirements are reasonably stable, this does not present a problem. However, in larger and more complex organizations, supporting centrally managed operations is more difficult. This is particularly true of large suppliers with volatile and demanding conditions that include unscheduled resource failures, periodic surges in new orders, and changes in requirements and priorities. Using agents here can change the perspective to a more *distributed* approach, making the solution more scalable, adaptable, and robust. In fact, many organizations have already realized their limit and

can no longer scale using a centralized approach. Organizations like DHL and Credit Suisse have found that an agent-based approach is their *only* option for successfully managing the previously unmanageable size and complexity within their businesses.

At the heart of the agent-based solution, two primary forms of agents were used to gain distributed control:<sup>2</sup>

- 1. Process-based agents.** These have the knowledge of how to combine resources and create products as part of a workflow in a supply chain.
- 2. Resource-based agents.** These manage the capacity-constrained resources of the systems, such as people, vehicles, tools, machinery, materials, and facilities.

### Business Processes and Resources Using Agents

Figure 2 contains an example of a workflow diagram for a process that installs utility poles for an electric company. The round-cornered rectangles represent processes, and the square-cornered rectangles are those resources required by the processes.

In business process management (BPM) systems, such activities

are usually centrally managed. By using agents, however, the process and resources can be managed in a distributed manner. For example, an Install Pole agent is a specialized entity that knows its Install Pole process needs a utility pole, pole-installer equipment, a person to operate the pole installer, and the prior completion of a Dig Pole Hole process.

Such knowledge can be used in several ways. For instance, a requester might be interested in an Install Pole service but would like to know its cost before the service is invoked (see Figure 3). Here, a call for proposal (CFP) would be sent to an agent that provides such a service.<sup>3</sup> The agent would send a CFP to those resources and preceding services required for the Install Pole process. Each resource agent would then calculate the charge for its usage based on the time and duration specified by the requester. If a resource is urgently needed and is in short supply, the cost might be high; if not, the price would be low. For example, if the requester wants a utility pole in two hours and only an expensive provider has one available, the price would be higher than if the requester could wait for two days and obtain it from a cheaper supplier. In contrast, the agent representing pole installers might be able to provide a person immediately. In two days, however,

<sup>2</sup>Several other types of agents were used as well (e.g., order agents, dispatch agents, and supervisor agents). However, the process and resource agents are key to understanding the approach.

<sup>3</sup>Agents are a useful way to extend SOAs.

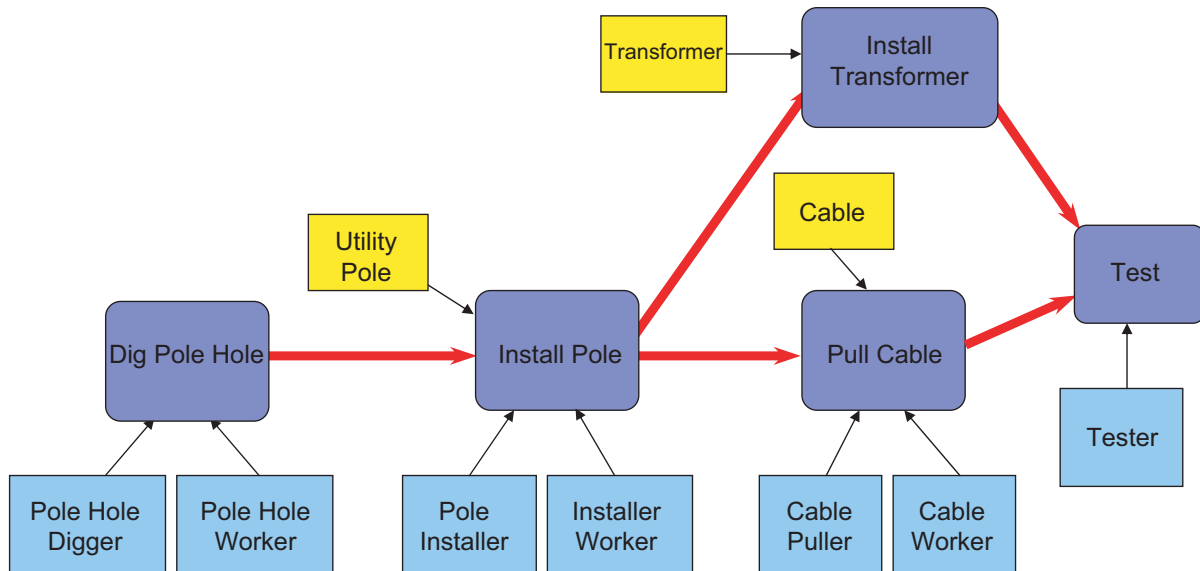


Figure 2 — Example of a business process where processes and resources are managed using agents. (Source: [7].)

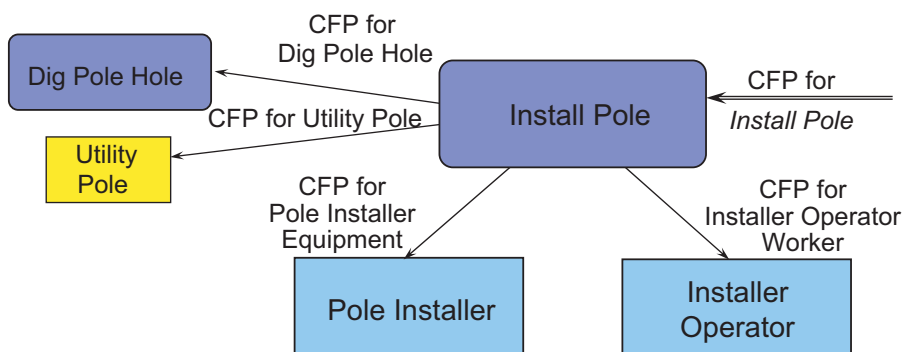


Figure 3 — A call for proposal (CFP) to the Install Pole agent results in a CFP to all of its resources and preceding operations. (Source: [7].)

they could all be busy on other assignments but would do the requested job at double-time rates.

Once the resources have responded to the CFP with a bid, the Install Pole agent tallies up the cost for the service as a whole and sends a consolidated bid to the requester for the Install Pole service.

In other words, the Install Pole agent acts like a broker on behalf of the process. If the requester accepts the bid, the broker is notified of the award and will confirm (or decline) the award.

#### Agent Negotiation

The interaction protocol just described is expressed using the

sequence diagram depicted in Figure 4. The requester is playing the role of customer and the provider is playing the role of supplier. Interaction protocols are useful because they define the expected behavior between interacting agents — whether either can be a resource agent or a process agent.<sup>4</sup>

#### Agent Renegotiation

In the real world, suppliers can overcommit, have resource failures, and experience process delays. Using a top-down, centralized approach, the schedule changes tend to be reoptimized from a global level. Such a technique would work in a small operation but would not scale to a large one. However, using an agent-based approach — which is

<sup>4</sup>It should be noted that the notion of *requester* and *provider* agents are at the very heart of the W3C's Web Services Architecture study [11].

distributed by nature — enables more dynamic scheduling by adapting to these unexpected situations on an individual and local basis, rather than a massive and global one.

For example, an important job request is received for Resource A, which is *already* allocated to Job 1 and Job 2. The agent representing Resource A needs to interact with the process agents for each job to determine whether the extra work can be accommodated. The agent for Resource A asks the agent for the first step in Job 1 about any extra (or slack), time (see Figure 5). It turns out that if Resource A takes on the new job, it will delay Step 1 by four hours. Now, Step 1’s agent can ask Step 2 if a four-hour delay would affect its processing. Since Step 2 has lots of slack time, it can absorb the delay without affecting the overall schedule.

Resource A’s agent must now examine the remaining option of changing Job 2 (see Figure 6). Step 1 would be delayed by three hours. The Step 1 agent then asks Step 2 if a three-hour delay would impact its processing. The Step 2 agent indicates that such a slippage would also cause a three-hour delay for it. Step 3 has a similar result and therefore its agent must ask the end customer if this slippage presents a problem. In this example, the customer has a large penalty clause built into the contract for late shipments. In summary, Job 2

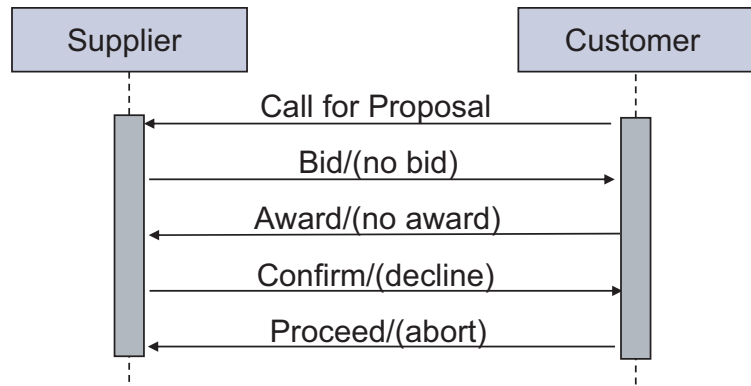


Figure 4 — An interaction protocol for making a contract.

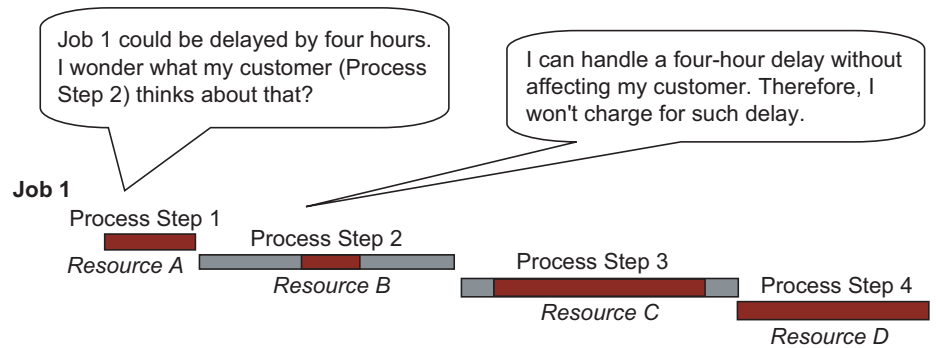


Figure 5 — Renegotiation example for Job 1 by process and resource agents. The bar represents the total time allocated to perform the process; the segment within the bar indicates the actual amount of time needed by the process. (Source: [1].)

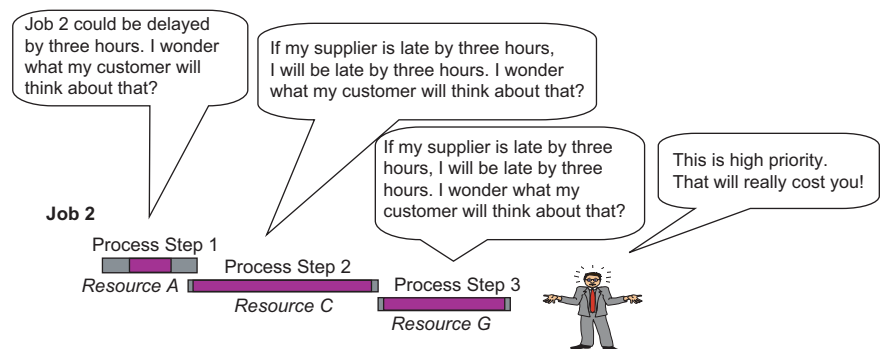


Figure 6 — Renegotiation example for Job 2 by process and resource agents. (Source: [1].)

would result in a penalty if modified to handle the extra requirement for Resource A. However, adapting Job 1 would enable Resource A to be assigned to a new job without affecting the overall schedule. All decisions in this example were based on a local optimization rather than a global one — enabling decisions to be made in seconds instead of hours or days and reducing costs as well as time to market.

### Agents and Dynamic Scheduling

The agent-based approach described above was developed with support from the Rock Island Arsenal as part of the US Department of Defense's Defense Advanced Research Projects Agency (DARPA) contract to build an agent-based, factory-scheduling prototype, named Autonomous Agents for Rock Island Arsenal. In one benchmark test using these techniques, inventory costs were cut by 47%, lead times were cut 59%, and schedule-reducible costs (such as overtime and inventory holding charges) were cut by 93% [4].

Since that time, agent-based schedulers have been designed for suppliers in many industry sectors: manufacturing, finance, energy, and transportation. NewVectors LLC (<http://www.newvectors.net>), Magenta Technology ([www.magenta-technology.com](http://www.magenta-technology.com)), and Intelligent Automation, Inc. — IAI ([www.i-a-i.com/view.asp?tid=7](http://www.i-a-i.com/view.asp?tid=7)) — have all been active in this area.

Magenta Technology has developed an agent-based system that dynamically schedules the deployment of ships and cargo. IAI is developing a generic scheduler that rapidly generates tailored and optimized scheduling engines. Using their software, an end user who has only domain expertise can define:

- **Components (e.g., resources, work center, parts, tasks/operation, and jobs)** — are required for the scheduling application, using the generic scheduling engine builder (GenSEB) component libraries
- **Protocols (interactions/constraints between components)** — use a standardized representation language called an agent interaction protocol (AIP)
- **Rules/policies** — define the order of interactions and the content of an interaction based on a user-defined scheduling algorithm

### Other Agent-Based Uses

#### Some Common Applications

Many companies have already tested agent-based applications in their research and development facilities. Successful applications are now used commercially throughout the world. Here is a brief overview of some of them.

#### *Just-In-Time Delivery*

When operating in a highly competitive market, delivering

just-in-time (JIT) products and services is vital. SCA Packaging turned to an agent-based modeling solution that explored different strategies for reducing inventory levels without compromising its delivery commitments. An agent-based simulation developed by Eurobios ([www.eurobios.com](http://www.eurobios.com)) enabled the company to reduce inventory levels by 35% while maintaining its delivery commitments [6].

#### *Insurance Claims Processing*

Acklin B.V. ([www.acklin.nl](http://www.acklin.nl)) developed an agent-based, international vehicle-claims processing system for three European companies. The agent system ensures EU regulations for confidentiality and enables a robustness that can survive broader system shutdowns and failures. As a result, the total time for client and claim identification is reduced from six months to two minutes [6].

#### *Distribution Optimization*

Delivery of goods and services between site and customer is a problem for many industries. Companies such as BT, DHL, and Air Liquide ([www.airliquide.com](http://www.airliquide.com)) require transportation networks that can flexibly adapt in real time to changing and unforeseen run-time conditions, fluctuations, and requirements. Not surprisingly, at least three companies provide agent-based solutions to this area: IAI, NuTech Software Solutions

(www.nutech.com), and Whitestein Technologies (www.whitestein.com).

#### *Onsite Maintenance*

Any industry that employs JIT processes relies on the availability of manufacturing equipment. Faulty equipment can have catastrophic consequences on productivity. In this context, both preventive maintenance and corrective maintenance become key activities. In the agent system developed by Oslo Software (www.oslo-software.com), its automotive customer handled 150,000 pieces of equipment with two million maintenance operations per year. The automotive company was originally using the maintenance module of a well-known enterprise resource planning (ERP) software vendor. While the ERP package could provide preventive-maintenance schedules, it could not handle the corrective-maintenance operations. Here, team managers had to deal with exceptional corrective-maintenance operations using the less practical Excel format. This led to canceling an average of 1,000 maintenance operations every month, increasing the risk of equipment failure. The maintenance problem is complex because many pieces of equipment are involved and the skills required of the personnel are varied. New technologies are needed to automate the maintenance process. These technologies need to automate the

exception-handling process in order to provide better support to business users. Agent technology, by its unique exception-handling capability, provides the solution to this problem.

#### *Healthcare*

By modeling the stakeholders in primary care systems as agents, Calico Jack (www.calicojack.co.uk) has developed a way to integrate health systems for the Scottish Executive Health Department.

#### *Simulation*

Since multiagent systems involve many interacting agents, predicting the whole system's success or failure is difficult. Defining the behavior of each agent is one thing; knowing what will emerge from their joint interaction is another.

Multiagent models can be used to simulate the behavior of complex computer systems, including multiagent computer systems. Such simulation models can assist designers and developers of complex application systems and provide guidance to software engineers. Agent-based simulation is also an important technique that offers strong models for representing complex and dynamic real-world environments. Agent systems simulating real-world domains may provide answers to certain physical or social problems that would be otherwise unobtainable due to their

complexity. Agent-based simulation spans many areas. For example, it can aid social structures and institutions in developing plausible explanations of observed phenomena; assisting in designing organizational structures; and informing policy or managerial decisions. Agent-based simulation can help model physical systems, including intelligent buildings, traffic systems, biological populations, and software systems of all types, including e-commerce and information management systems [6]. The two most popular agent simulators are Swarm (www.swarm.org) and CybelePro (www.cybelepro.com). Recently, CybelePro was used to simulate air transportation systems involving flights, airports, terminal areas, controllers, and airline operation centers with varying levels of fidelity. Here, an individual simulation is comprised of tens of thousands of dynamically interacting agents configured to represent a concept/scenario.

#### *Collaborative Decision Making and Distributed Control*

As systems become increasingly decentralized and autonomous, technology needs to support collaborative, team-based decision making and control. Agents can work individually and socially in teams to aid and accomplish this. In particular, IAI has developed applications for team formation, cooperative sensing, tracking, monitoring, and traffic management.

### *Supply Chains and Logistics*

Today's supply networks need to be flexible, responsive, adaptive, and able to cope with the variability of demands. Traditional supply chain and logistics optimization systems are not designed to handle volatility and complexity or to function in a real-time environment. Using an agent-based approach is now a common solution for many companies. Software vendors utilizing this type of approach include Magenta Technology, IAI, and Cougaar Software, Inc. ([www.cougaarsoftware.com](http://www.cougaarsoftware.com)).

### *Autonomic Computing*

In today's global networks, we need computer systems that regulate themselves much in the same way our autonomic nervous system regulates and protects our bodies. Self-healing systems require an agent-based approach rather than a centralized one.

### **Architectures and Ontologies — and Agents**

In addition to specific applications, the IT developer must address current architectural approaches. Here is a brief overview of those approaches particularly appropriate for agents.

#### *Ontologies*

Agents require a common vocabulary and set of concepts to communicate effectively with other agents. Agent-system developers

have been adapted to use the semantic Web language (OWL), the semantic Web rule language (SWRL), and the services ontology (OWL-S) for the specification of agent systems. In addition, agents can be used to manage and maintain large distributed ontologies as well as perform data mining functions. Since ontologies are both syntactically and semantically defined, agents not only intelligently query, infer, and reason over ontologies, their behavior can also be guided by an ontology. These are the so-called *ontology-driven* approaches. Here, the agent code does not have to be modified and recompiled for every change. Instead, the ontology can be changed, resulting in a dynamic change in the agents' behavior based on the modified ontology.

#### *Peer-to-Peer*

Peer-to-peer (P2P) applications display agent-like characteristics that include both applying self-organization techniques that ensure continuous operation of the network and employing interaction protocol designs to enforce correct behavior among interacting nodes. For example, commercial e-marketplace systems like eBay include simple credit reputation systems to reward socially beneficial behavior. As P2P systems become more complex, increasing the use of agent technologies will be appropriate. For example, the auction-based mechanisms and negotiation

techniques used by agents could be used to enhance the level of automation of peers in popular applications. Since complex P2P applications are social in nature, they require increasingly sophisticated approaches to trust and reputation and to the application of social norms, rules, and structures. Social simulation would be particularly appropriate here in order to better understand the population dynamics of independent agents [6].

#### *Web Services and SOA*

The W3C Web Services Architecture study recommends an agent-based approach to using SOAs — both on the service requester side and the service provider side [11]. Agents act with varying levels of autonomy, depending on environmental constraints and their ongoing interactions. Because services are often best modeled as autonomous and heterogeneous, they can naturally be associated with agents. Agents are not a panacea. However, when applied appropriately, they enable us to (1) define elaborate constraints on what services are willing to offer, thereby enabling us to define richer requirements for service composition; (2) discover trustworthy services; (3) negotiate with external service providers; and (4) evaluate the compliance of service providers within their contracts. They can also provide a more natural, human-like way of interacting with service

requesters, as well as ensuring the entire process is carried out effectively and efficiently [10].

Verizon chose a decentralized agent-based approach to SOA rather than a centralized approach where all services are handled by a server or set of servers. Additionally, Verizon employs agents to support the subscription, management, and dashboard layers of its SOA, using its IT Workbench software. By the end of 2004, Verizon was handling nearly three million service transactions per day.

Service composition aggregates services to create new functionality. Often, the composed functionality would itself be exposed as a new service with a standard interface. If this happens, a new service could be composed that would intelligently guide the service requester throughout the lifetime of a particular business process. Such an intelligent service could be implemented as an agent. Agents also make it possible to capture the interactions among services and to create new services as subtle compositions of others.

#### BPM

BPM has come to mean an integrated collection of critical process technology necessary to support the business process life-cycle (also called the BPM suite). The actual business process logic can be centralized in one location, as opposed to being distributed

across and embedded within multiple services. However, multiple business processes can be used to satisfy the service (see Figure 7). For example, to request a product's price, the computation could vary depending upon location of order, time and date ordered, quantity, customer, and other items in the same order.

The DHL pricing system, for example, had nearly 500 different processes for pricing an order because of the number and potential combinations of variables. To handle this, DHL used agents to choose the appropriate processes. In other words, each agent contained a process-execution engine that would select and execute the appropriate process for each pricing-query service request. If a problem arose in the middle of a process, the agent would select an alternate one. DHL found that an agent-based approach not only executed faster, but was easier to maintain. Changes to a process or process rule (such as adding a fuel surcharge) took hours rather than weeks. In a situation like this, software from companies like OSLO can be employed, which uses a graphic editor based on UML activity diagrams — enabling a user to change the process. Again, agents are not always appropriate for controlling business-process logic. If the process is simple and straightforward and it does not require frequent changes and processing robustness, a language like BPEL could be used. However, for complex processes that involve

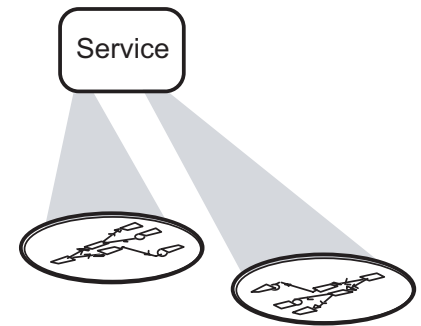


Figure 7 — Services can be carried out by many different business processes depending on business conditions and rules.

various operating conditions and require frequent and timely changes, agents should be used.

#### Event-Driven Architectures

Real-time response is often critical to customer satisfaction. Abnormal events or a combination of events can have a significant impact on an organization. More and more companies, such as IAI and Rhysome ([www.rhysome.com](http://www.rhysome.com)), are using agents along with software and physical sensors to detect changes in the business environment. (Examples include RFID tags for retail supply chain optimization, medical monitors, physical sensors that detect changes in air quality, and electronic data capture tools for patient trials at pharmaceutical companies.) Companies that derive the greatest advantages from agent-based, event-driven architectures (EDAs) have the following characteristics: (1) large and heterogeneous environments, (2) information that changes *constantly* in various ways,

(3) complex exceptions and state changes in real time, and (4) the need to deliver and respond appropriately to that information.

#### *Integrating SOA, BPM, and EDA*

These last three architectural approaches can be considered separately for including agent technology. Additionally, agent technology can be used to link business processes and services to facilitate a single architecture that integrates SOA, BPM, and EDA (see Figure 8).

#### **Why Use Agent-Based Technology?**

Thus far, we've examined a sampling of uses for agent-based systems — happening now within the IT community. Given that any new technology is disruptive, why did these early adopter companies decide to use agent technology? The reasons can be summarized by one or more of the following benefits:

- Faster ROI
- Lower maintenance

- Higher productivity
- Leverage of existing infrastructure
- Reuse of processes and services
- Foundation for future projects
- Reduction of time to market
- Increased agility to respond to business needs

Is an agent-based approach useful for every application and usage? Does it always provide the benefits listed above? Certainly not. If your business is predictable and stable and if your processes are centralized and scalable for the foreseeable future, adopting an agent-based approach is not necessary. However, for those applications that must support complexity and change in a scalable and timely manner, agents will likely be a necessary technology.

In the typical organization, both of these situations probably coexist in one form or another. Savvy organizations will respond with a

mixture of technologies: object-orientated (OO), relational, *and* agent-based. OO and relational technologies enable a top-down and centralized solution to business application. Agents provide one more tool that conventional IT shops do not have — a bottom-up and distributed approach. The real benefit comes when an organization can choose the appropriate mix of technologies for a given application, which provides a balance of both the centralized and distributed approaches.

It is one thing to say that agents can provide a helpful, distributive approach to software applications; it is quite another to understand what that means and entails. The next section begins that journey by answering the question “What is an agent?”

#### **WHAT IS AN AGENT?**

Conventional objects can be thought of as passive, because they wait for a message before performing an operation. Once invoked, they execute their method and go back to “sleep” until the next message. A current trend in many systems is to design objects that both react to events in their environment and are proactive. In UML 2.0, these are known as *active objects*; in the agent community, they are known as *agents*. Whether they are called active objects or agents, this new direction is going to change radically how we design systems.

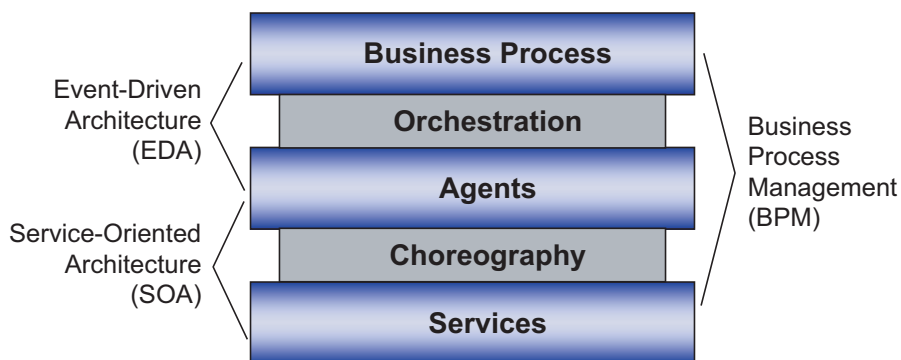


Figure 8 — Agents can be used to integrate SOA, BPM, and EDA.

### **The Basic Properties of Agents**

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 “something that acts.” However, for developing business and IT systems, such a definition is too general. While an industry-standard definition of agent has not yet emerged, most agree that agents deployed for IT systems are not useful without the following three important properties:

- 1. Autonomous.** The agent is capable of acting without direct external intervention. It has some degree of control over its internal state and actions based on its own experiences.
- 2. Interactive.** The agent communicates with the environment and other agents.
- 3. Adaptive.** The agent is capable of responding to other agents and/or its environment. An agent can modify its behavior based on its experience.

Based on this approach, a basic working definition for an agent is “an autonomous entity that can adapt to and interact with its environment.”

Agents are commonly regarded as autonomous entities because they can be thought of as having their own set of internal responsibilities and processing. For example, each ant in Figure 1 has its own self-contained processing that enables searching for and

gathering of food without any external choreography.

Agents are interactive entities because they are capable of exchanging rich forms of messages with other entities in their environment. These messages can support requests for services and other kinds of resources, as well as event detection and notification. They can be synchronous or asynchronous in nature. The interaction can also be conversational in nature — negotiating contracts, marketplace-style bidding, or simply making a query. In the ant colony example, ants interact via the pheromones that they deposit in the environment. The pheromones act as information signposts for other agents, providing a simple yet highly effective means of communication.

Lastly, agents can be thought of as adaptive because they can react to messages and events and then respond appropriately. In the example above, each ant adapts to its environment by continuing to wander randomly if it fails to find food or pheromones. However, when pheromones are detected, an ant reacts by changing its behavior to track the pheromones to the food source. Once the food is found, the ant again adapts by picking up the food and carrying it back to the colony. If the food is moved, the adaptive ants will locate the new food source and notify others while bringing it to the colony.

Ants provide a good example of simple reactive adaptation. However, agents can also be designed to learn and evolve.

Overall, agents can be autonomous, interactive, and adaptive to some degree (see Figure 9). It is not an all-or-nothing proposition. In the next few sections, these three key agent properties will be discussed in more detail.

### **Agents Are Autonomous**

Agents can be thought of as autonomous because each is capable of governing its own behavior to some extent. Autonomy is best characterized by degrees. At one extreme, an agent could be completely self-governing and self-contained. However, an agent that does not require the resources of or interaction with other entities is very rare. Even a database access or Web query agent requires external resources. At the other extreme, an agent that is barely able to perform the simplest of actions is impractical for the pragmatic system developer. Even traditional objects have some degree of self-contained lines of code within their methods.

Depending on the system developer, an agent’s autonomy can be designed to fit some place between these two extremes. For example, conventional objects are not completely dependent on outside resources; they have some state and behavior of their own (see Figure 10). But they are

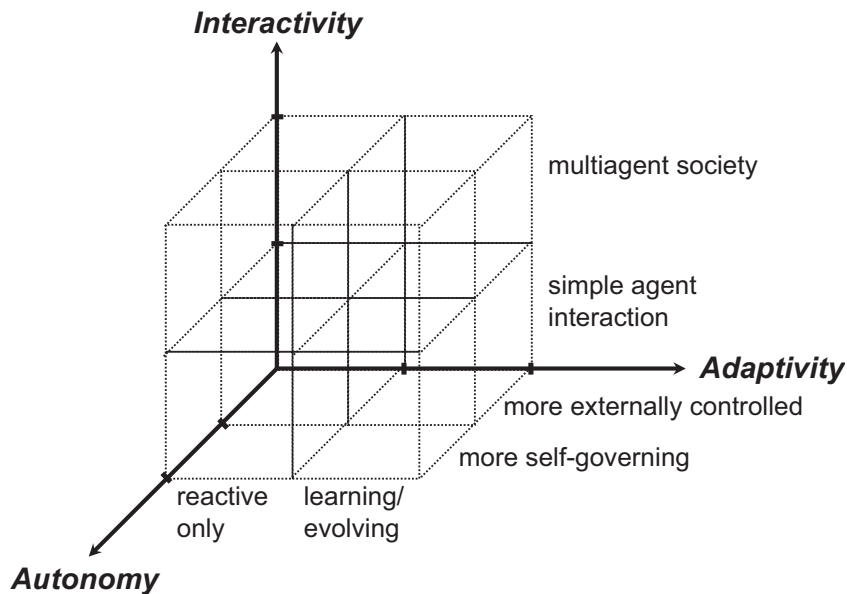


Figure 9 — Agents have various degrees of autonomy, interactivity, and adaptivity.

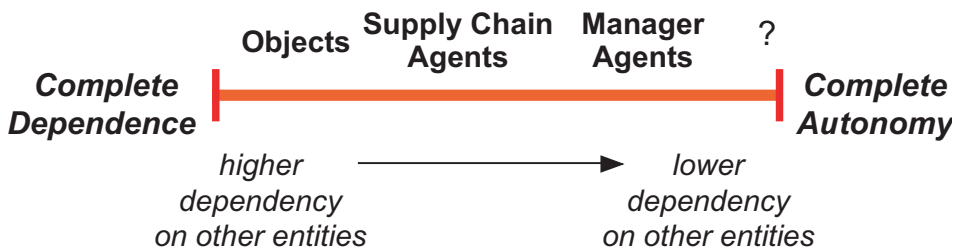


Figure 10 — Degrees of autonomy.

traditionally simple, which means that they are far from being completely independent. Because supply chain agents are able to reach certain conclusions and make decisions on their own, they have more autonomy than objects. By their very nature, supply chain agents require interactions with other entities to enable a fully functional supply chain, since one agent cannot take care of an entire organization's supply chain. More autonomous are manager agents because their role can

involve a high degree of internal decision making though activities such as monitoring and delegating, which depend on outside resources.

Two major aspects of an agent's autonomy involve its capacity to be dynamic and its ability to make decisions. As depicted in Figure 11, autonomy can be considered on two axes. On one axis, the dynamic aspect of an agent's autonomy can range from being simply passive in its action to

entirely proactive. On the other axis, an agent's decision-making aspect ranges from being limited to simple decisions to being capable of making complex decisions (see Figure 11).

### Dynamic Autonomy

In addition to reacting to specific method invocations, agents can respond to events within their environment. Proactive agents will actually poll the environment for events and other messages to determine what action they should take. To compound this, in multiagent systems, agents can be engaged in multiple parallel interactions with each other — magnifying their dynamic nature. In short, an agent can decide when to say “go.”

As described earlier, the GM paint booths had both reactive and proactive features. Information about an unpainted car or truck coming down the line was posted in an automated form accessible to all paint booths. When a paint booth had nearly completed its current job, it basically said, “Hmmm, I’m running out of work, I’ll look over at the jobs posted.” As stated earlier, if the booth was applying the color of paint required by an upcoming job, it would bid more for the job than would a booth having a different color. Other bidding criteria could include how easy or important the job was. In other words, the booth was reactive in whether to bid on a paint job, but proactive in that it

would check the paint job list and determine what to do.

Objects, on the other hand, are conventionally passive — with their methods being invoked under a caller’s thread of control. The term “autonomy” barely applies to an entity whose invocation depends solely on other components in the system. However, UML and Java have recently introduced event-listener frameworks and other mechanisms for allowing objects to be more active. In other words, objects now have some of the dynamic capability of agents.

### Decision Autonomy

Agents can be designed to make a few simple decisions. However, the more decisions an agent is capable of making and the more complex they are, the more autonomous the agent appears. A completely autonomous agent will do whatever it wishes.

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 choices for its behavior are predetermined. In contrast, the GM paint booth can decide on how much to bid based on the booth’s ability to efficiently take on a new paint job. Certainly, the choices are limited for a paint booth, but the ant has no choice on how to act when food is found. In contrast, shopping agents can have very

complex criteria for selecting a gift for a specific individual. In fact, the agent might return empty handed because it decided no gifts were appropriate. A contract-negotiation agent may decide the contract is not worth pursuing for a number of reasons. In other words, the agent can also say “no” to performing a requested service.

Conventional objects certainly have the ability to make decisions. However, the typical usage and direct support with OO languages tend toward a less complicated approach to decision making. For instance, when a message is sent to an object, the method is always invoked. The contract-negotiation agent could refuse to invoke a requested bid method; the object cannot. Yes, an object may determine whether or not to process

the message and how to respond if it does. In common practice, however, the refusal of an object to execute its method is typically considered an error situation. With agents, this is not the case. Manager and quality assurance agents are good examples of agents that can say “no.”

Object classes are usually designed to be highly predictable in order to facilitate buying and selling reusable components. Agents are commonly designed to determine their behavior based on their individual goals and states as well their contexts within ongoing conversations with other agents. While OO implementations can certainly be developed to include nondeterministic behavior, this is common in agent-based thinking.

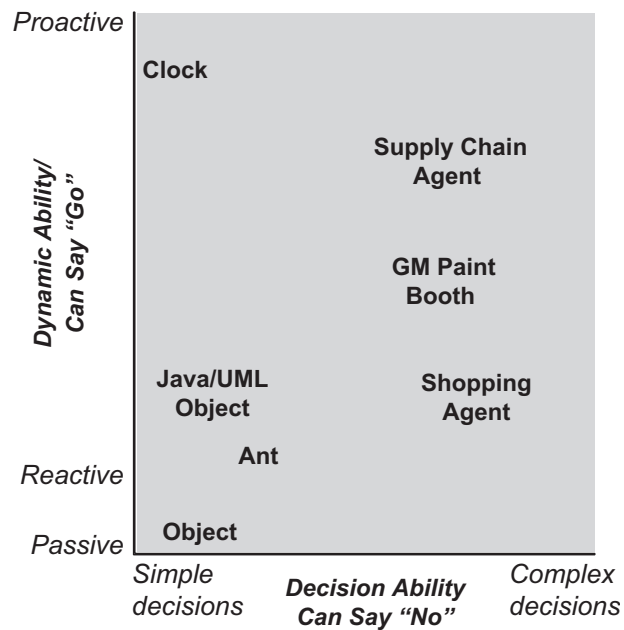


Figure 11 — Two aspects of autonomy.

### Agents Are Interactive

Interaction involves the ability to communicate with the environment and other entities. It can be expressed in degrees (see Figure 12).

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 don't invoke methods on each other; their interaction is indirect, through direct physical effects on the environment. Even more complex interactions are found in multi-agent systems where agents engage 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).

While 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.

### Agents Are Adaptive

An agent is considered adaptive if it can respond appropriately to other agents and/or its environment to some degree. Autonomous entities that fail to adapt rapidly to the changing world become extinct: organisms die; companies go out of business. For an organization, adaptation enables the system to react effectively to changes in areas such as the market and business environment. In IT systems, adaptation enables systems to react appropriately enabling, for example, system balancing, integrity assurance, and self-healing systems. When designed properly, the individual parts of the system can be empowered to change based on environment and market conditions.

### Simple Reactivity

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. Such a response is usually expressed in “if-then” form. Thermostats, robotic sensors, and simple search bots fall into this category.

From atoms to ants, the reactive mode is quite evident. A carbon atom has a rule that states in effect: “If I am alone, I will only bond with oxygen atoms.” An ant has a rule that if it finds food, it should return the food to its colony while leaving a pheromone trail.

### Rules

Beyond the simple reactive agent is the agent that can appear to *reason* by following chains of rules. For example, agents can react by making inferences and include patient diagnosis agents and certain kinds of data mining agents. These inferences are computed by following a chain of inference rules.

Business process engines can use a similar approach. Here, libraries of processes are maintained, which indicate the circumstances under which a particular process might be appropriate. Each step within a process acts as a service request, invoking the business process engine to choose the appropriate set of steps that should be executed, based on the business context. For example, a request to compute the sales task would

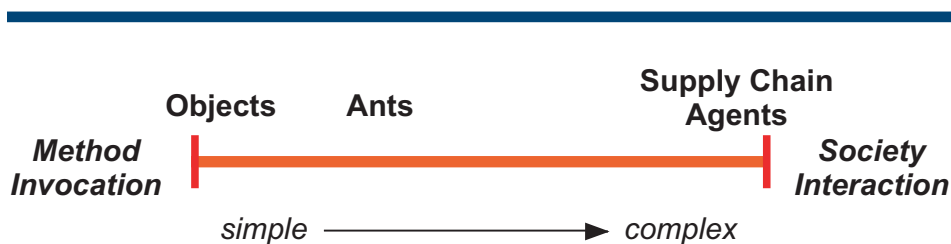


Figure 12 — Degrees of interaction.

trigger the engine to locate the appropriate set of steps needed for a given location. The taxes in Michigan would be computed differently than in Brussels.

Rules do not change in and of themselves. Instead, 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. However, other considerations might also be appropriate such as the bidder’s ability to deliver its goods in the quantity, quality, and timeframe requested. Over time, a *purchasing agent* can learn to choose from reliable *vendor agents* 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 tech-

niques that employ reinforcement learning include credit assignment, Bayesian and classifier rules, and neural networks. Examples of learning agents would be agents that can approve credit applications, analyze speech, or recognize and track targets.

### Evolution

Evolution is change that occurs over successive generations of agents. A primary technique for agent evolution usually involves genetic algorithms and genetic programming. Agents can literally be bred to fit specific purposes. For example, operation plans, circuitry, and software programs can prove to be more optimal than any product that a human can make in a reasonable amount of time.

### Other Agent Properties

In earlier sections, three key agent properties were discussed: autonomy, interactivity, and adaptivity. These properties are important because agents deployed for IT systems are not useful without them. However, agents may possess various combinations of other properties that may be useful — depending on the application requirements and the agent designer. Here, agents may be:

- **Sociable** — interaction marked by friendliness or pleasant social relations (i.e., 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 (i.e., acting in the interest of, as a representative of, or for the benefit of some other entity)
- **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
- **Temporally continuous** — is a continuously running process
- **Credible** — believable personality and emotional state
- **Transparent and accountable** — must be transparent when required, yet must provide a log of its activities upon demand
- **Coordinative** — able to perform some activity in a shared environment with other agents; activities 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 (also known as *collaboration* )

- **Competitive** — able to coordinate with other agents, but the success of one implies the failure of others (opposite of cooperative)
- **Rugged** — able to deal with errors and incomplete data robustly
- **Trustworthy** — adheres to Laws of Robotics and is truthful

### WHAT GOES ON INSIDE AN AGENT?

As we discussed in the previous section, agents are autonomous entities that can adapt to and interact with their environment. To

accomplish this, agents must have some means by which they can perceive their environment and act upon those perceptions (see Figure 13).

#### **Agent as a Black Box**

In its simplest form, each agent can be thought of as an interactive black-box process, where no indication of an agent's internal structure and behavior is given other than that which is visible from outside the agent. Figure 14 depicts this by illustrating that agents have input from, and output to, the environment. An agent's environment can be thought of as everything outside of the agent. For robotic agents, the environment would consist of the

terrain, the laws of physics, and other agents. For software agents, the environment would consist of the hosting platform, its operating systems and supporting middle-ware, and other agents.

Input can be whatever the agent “perceives,” such as messages, requests, commands, and events. Based on this input, the agent can choose how to act on it, which typically results in output from the agent such as communicating with, or making a change to, its environment. For instance, an Inventory agent could receive a message that requests the quantity on hand for a given commodity. The agent would then determine the requested quantity and send a response to the requester. The Inventory agent could also be notified of the event that some of the inventory has been removed and is being sent to a customer. The agent process would then recompute the quantity on hand and determine whether a reorder point has been reached. If so, the Inventory agent would send out requests for bid to the appropriate vendors.

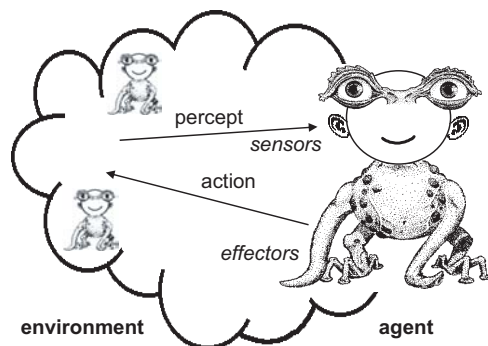


Figure 13 — Agents interact with their environments.

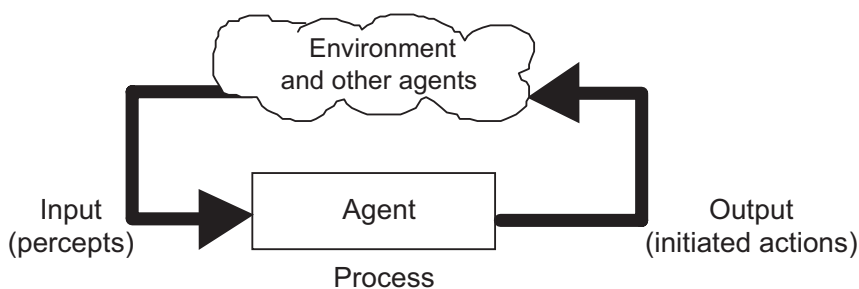


Figure 14 — An agent and its environment.

#### **Agent Processing Overview**

For an agent to interact with its environment, it must be able to detect and effect its environment but also to understand what it is detecting, to select an appropriate response, and to act on it. In other words, for the agent to be an interactive entity, it also requires some degree of autonomy and adaptivity. The

internal structure of an agent is shown in Figure 15.

### State

Just like objects, agents can have a form of memory, referred to here simply as *state*. State includes data that it needs to operate effectively, which the agent maintains or can access in some manner (such as via a database or automated blackboard). For example, an Inventory agent would need to maintain information, such as the reorder point for a particular product or the algorithm needed to compute the reorder point. The agent might also maintain a list of valid product vendors that it can contact for reordering — or at least know where the information might be found. The agent can even make inferences about its state or the state of other entities in its environment. State, then, can be metaphorically interpreted as anything that the agent can “know” or “believe” about itself or its environment.

### Detectors

*Detectors* provide the interface to the agent’s environment. They receive input that may interest the agent. Detectors can both receive input sent to them or actively scan the environment for it. For the more passive software agents, this could include receiving messages and event notifications. Active agents might scan message boards or monitor event occurrences. In particular,

event-monitoring agents (in EDAs) are specialized to recognize particular kinds of state changes in the environment. For example, database event agents could watch for certain kinds of changes in a database such as metamodel modifications, customer address changes, or the database going offline. If any of these events are detected, the event monitor could notify the appropriate agents that can react appropriately to the event. In EDAs, this is particularly useful. Robotic agents may employ such mechanisms as cameras, global positioning hardware, or infrared devices for detectors.

### Parse Input

Once the detector receives input, the agent needs to analyze and understand the input. This action typically involves recognizing the input, normalizing it, and dealing with any inconsistencies in the content. For conventional OO software, there is virtually no parsing. In the OO “signature,” the first parameter identifies the

object, the second specifies the requested operations, and the remaining input contains the variables required for the operation execution. Because the signature syntax is already defined in this manner, the message does not need to go through the Determine Action step. Instead, it goes immediately to invoke its specified method.

In a system with a more complex style of input, the OO approach cannot provide a satisfactory solution. For example, some agents need to process complex syntaxes and receive their input in one or more languages, such as SQL, Prolog, BPEL4WS, BPML, or ebXML. Furthermore, they may need to understand ontologies expressed in RDF or OWL before any further action can be determined. For event-monitoring agents, this parsing can mean looking for patterns in the input. Event pattern recognition is important for agents, especially for those that participate in mechanisms for self-healing systems.

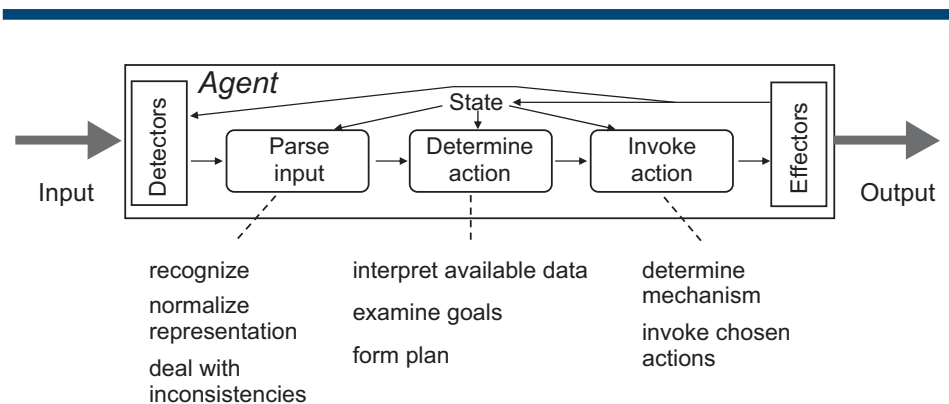


Figure 15 — The basic internal architecture of an agent.

Granted, a parsing activity requires greater complexity than the traditional object. In fact, not all agents will require sophisticated parsers. Yet, when system requirements demand richer behavioral control at execution time, agents will be able to accomplish what traditional objects do not. Parsing extends the notion of the semantic net to enabling the semantic approach to system interaction, in general.

### Determine Action

As suggested earlier, traditional OO recommends that the method requested of an object be determined *before* a message is sent to the object, because each object class defines separate methods, one for each kind of message “signature.” If the number of possible methods is tangible, this is not a problem; for those situations requiring hundreds or thousands of possibilities, such an approach is impractical. For example, insurance companies have hundreds of ways of rating a policy; retailers and shipping companies have an equally large number of rules and procedures for pricing a product or service. To support this need in an agile and adaptive manner, the agent approach also has the option of determining the appropriate method *after* it receives a request for service. In this way, a policy agent can select the appropriate method to rate a policy based on the requester's needs. Order agents can select

the appropriate pricing algorithm when the customer submits order.

Another function within the Determine Action step could be to determine the appropriate goal for further consideration. For example, an Inventory agent could detect each time a product was pulled from a warehouse bin. For each instance, the agent might need first to decide whether the product should be reordered or, if it is moving too slowly, discontinued altogether. Each of these goals requires a different process. One might involve sending out and processing bids from other agents to refill the bin; the other might notify customer agents that the product will no longer be available.

The Determine Action process might involve a high degree of complex process in its own right. Can this be accomplished using an OO approach? Absolutely. However, some agent platforms have mechanisms that support action determination *directly* — instead of having the programmer do extra work in design, construction, testing, and maintenance. Agents, by nature, are adaptive.

### Invoke Action

Once selected, the appropriate action needs to be invoked. The action execution may involve other actions *within* the agent, in which case, messages can be sent directly to the agent's own detectors. If the action involves other agents, the messages need to be

sent to them. The agent controls its autonomy by using its own resources or having them provided by other agents. It determines what it can do and what it must delegate. In this sense, the agent manages itself — rather than just being managed.

In addition, some agents can determine when their internal actions are not functioning as expected. The agent can also communicate to itself that the situation needs to be remedied — by sending itself a message and letting the Determine Action step determine which action should be taken next. In other words, the agent can have its own processing engine. For BPM systems, this mechanism can then be handled in a distributed manner using agents, rather than by centralized controllers that will not scale under large processing volumes.

### Effectors

When messages need to be sent or events posted outside the agent, effectors are the connection with the agent's environment. It is the mechanism an agent uses to interact with and mobilize within the environment. This could involve sending messages through a message bus or activating a hardware device such as a printer or robot arm.

### Agents Can Be Multiprocessing

Agents can be single threaded or designed to process input concurrently (see Figure 16).

Agents that can support concurrency might also have a process controller that coordinates the various internal actions in some manner.

### Reasoning, Planning, and Scheduling

A common approach for agents can involve reasoning, planning, and scheduling (see Figure 17). In this approach, an agent can determine the appropriate action by having a *reasoner* step consider what it knows about its state and the state of its environment (including that of the service requester). Additionally, the reasoner agent can identify the appropriate goals for acting based on the detected input and state knowledge. Here, the agent may optionally weigh the importance of certain goals (when there are several goals to choose from or conflicting goals). For example, a scheduling agent may have a goal to ensure that all the tasks in its care are executed on time. Another goal might be that certain priorities must be given to important tasks. The two goals, however, may conflict when priority tasks cause the delay of other tasks. Here, the agent may have to weigh the cost of both and determine which service it intends to perform under a given circumstance. A *planner* then determines the actual steps needed to execute the agent's

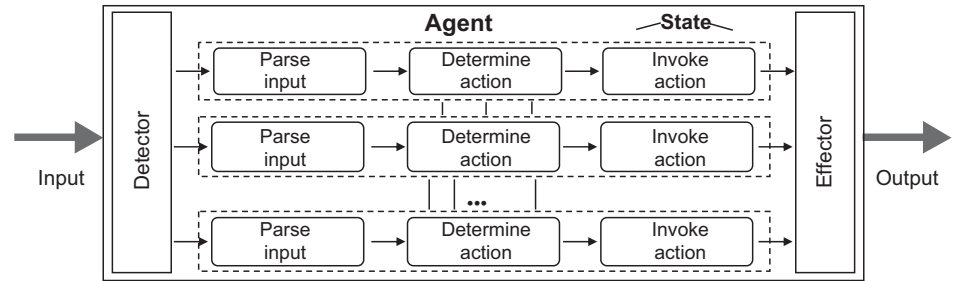


Figure 16 — Concurrent processing within the agent is possible.

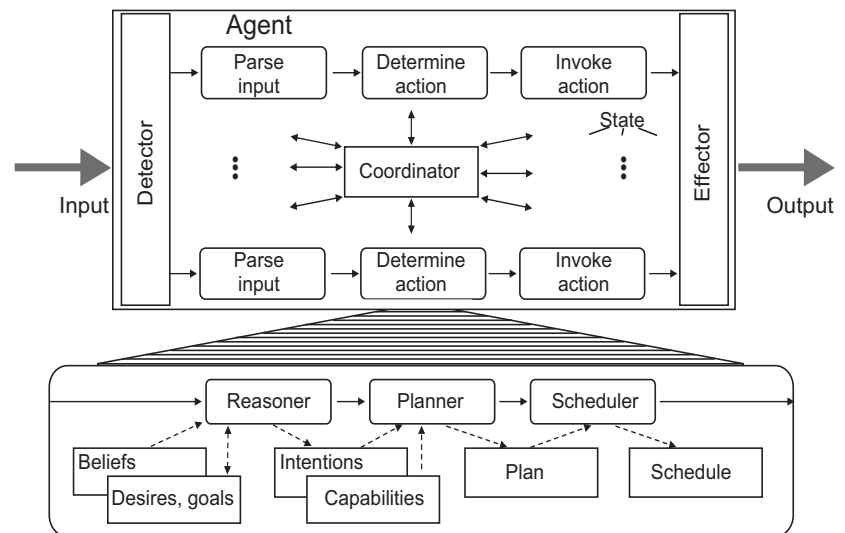


Figure 17 — Determining the appropriate action can involve reasoning, planning, and scheduling.

intention. For example, the scheduling agent may need to ensure a particular product is created for given order. The agent may need a make-or-buy decision resulting in a process that either fabricates the product inhouse or purchases a fabricated item from an external source. Some agent planners dynamically assemble the planned set of steps to fit each occasion; others have a library

of prefabricated processes from which to choose.

The planned process is then sent to the agent's *scheduler* to determine which resources will be assigned and when the process will be scheduled.<sup>5</sup>

On top of this, an agent may also employ a *coordinator* that ensures that the agent's

<sup>5</sup>Agents using this approach are sometimes referred to as “BDI” agents because they employ beliefs, desires, and intentions. *Beliefs* refer to what the agent “believes” the state of its environment and other agents to be; *desires* basically refer to the goals the agent could attempt to achieve; and *intentions* refer to the set of goals the agent actually intends to achieve under a given circumstances.

concurrent processing works harmoniously: managing conflicts and optimizing the overall functioning of the agent.

**But Keep In Mind ...**

Agent-based systems do not require complex forms of agents like those previously described. For example, ant colonies have been very successful for seven million years. An estimated 10% of the Earth’s animal biomass is composed of ants, which vastly exceeds the human population. An ant has about 250,000 brain cells; humans have about 10 billion. Ants can lift 20 times their own body weight. (They may be small, but they’re efficient!)

What if it were as hard to get your deliveries off schedule, as it is to keep ants out of your kitchen? (See Figure 18.) In other words, agent systems do not require sophisticated agents to yield impressive results. With simple rules and high fault tolerance, the colony thrives — a great first-system model.

**Agents and Objects**

Just how different — or similar — are agents and objects? Some developers consider agents to be objects, except with more bells and whistles. Others see agents and objects as different even though they have many things in common. However, both

approaches envision using objects and agents together in the development of software systems. In other words, objects and agents are two distinct notions — each having its particular place in software development. The important point here is that the agent-based way of thinking brings a useful and important perspective for system development, which is different from (while also similar to) the OO way.

**Evolution of Programming Approaches**

Figure 19 illustrates one way of thinking about the evolution of programming languages. Originally, the basic unit of software was the complete program where the programmer had full control. The program’s state was the responsibility of the programmer and its invocation determined by the system operator. The term “modular” did not apply because the behavior could not be invoked as a reusable unit in a variety of circumstances.

As programs became more complex and memory space grew, programmers needed to introduce some degree of organization to their code. Modular programming employed smaller units of code that could be reused under a variety of situations. Structured loops and subroutines were designed to have a high degree of local integrity. While each subroutine’s code was encapsulated, its state was determined by externally supplied arguments, and it gained

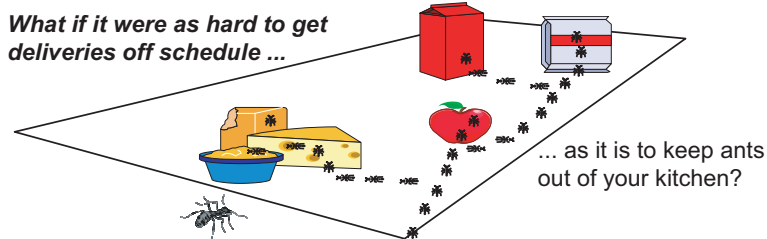


Figure 18 — Ant-like simplicity is still very successful. (Source: [7].)

	Monolithic Programming	Modular Programming	Object-Oriented Programming	Agent Programming
Unit Behavior	Nonmodular	Modular	Modular	Modular
Unit State	External	External	Internal	Internal
Unit Invocation	External	External (CALLed)	External (message)	Internal (rules, goals)

Figure 19 — Evolution of programming approaches (Source: [7].)

control only when invoked externally by a CALL statement. This was the era when the primary programming unit was based on modularizing procedures.

In contrast, object orientation added to the modular approach by maintaining its segments of code (or *methods*) as well as by gaining local control over the variables manipulated by its methods. However, in traditional OO, objects were considered passive because their methods were invoked only when some external entity sent them a message.

#### Agents and Objects Together

Software agents have their own thread of control, localizing not only code and state but their invocation as well. Such agents can also have individual rules and goals, making them appear like “active objects with initiative.”

In other words, the agent can determine when and how it acts. An agent-based approach is employed when a particular situation requires that processing be decentralized and self-organized, instead of centrally organized. Any centrally organized program written to handle the ant simulation (refer back to Figure 1) would have been far too cumbersome. It would have required a single set of top-level rules telling each ant 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 an animated cartoon than animated life [9].

Yet most developers tend to build centrally organized applications. They are also biased toward OO notions, such as *class*, *association*, and *message*. While these constructs are useful for a certain category of applications, they do not directly address the requirements of agents. As we have seen, agents have such characteristics as autonomy, mobility, and adaptability.

Furthermore, business users like to express other concepts, such as rules, constraints, and goals and objectives as well as 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.

#### CONCLUSION

##### ***Making Things Possible with Agents — Not Just Faster, Better, and Cheaper***

Using agents has produced applications faster and cheaper than other approaches do. For

example, DHL found that its pricing and tracking applications could be developed 50%-80% faster using agent development tools like those available from OSLO or Agentis Software ([www.agentissoftware.com](http://www.agentissoftware.com)). Time to market was reduced and ROI increased. This does not mean that agent development tools will always reduce function points, but it does for some classes of application.

The most compelling case for using agent technology is when agents can enable solutions that were either impossible or very difficult using conventional mechanisms. At manufacturing companies like Deere & Company, several hours each night are typically dedicated to computing optimized production schedules for the shop floor’s use the next day. When an organization has the time to use techniques like these, agent technology is probably unnecessary. However, what if John Deere’s IT department delivered its shop-floor schedule at the beginning of a day when several key employees were out sick or a vital piece of machinery in the assembly line broke down? If they needed to stop the assembly line for several hours to recreate the production schedule, this solution clearly is not the answer. Under these circumstances, an agent-based approach should be considered. Instead of starting over to reoptimize the schedule for all the resources, agents can be used to produce a local optimum. If one

resource is no longer available, agents can look for a replacement and “contract” with alternative resources. If the resources are going to be delayed, agents in a supply chain can determine which way to solve the delay with the least impact — using automated speeds.

Even techniques such as linear programming, neural networks, and genetic algorithms are sometimes used for similar situations. Again, when these provide accurate results in a timely matter, agents are not necessary. However, for large complex systems, such approaches can sometimes become costly to configure or reconfigure and also involve scalability problems. In such situations, they are often run offline and do not respond rapidly to dynamic changes in environment. When this is the case, the horizontal approach of agent technology can significantly outperform a centralized scheduling approach.

Companies like IAI have developed systems involving over 100,000 agents that can handle the resource rescheduling in a manner of seconds — instead of hours. General-purpose agent platforms, such as the OSLO Suite, can be used, or a more specialized package can be chosen. For example, agent-based logistics software is available from Cougaar and Whitstein, and event-driven sense-and-respond solutions from Rhysome. Many other large software companies

already employ agents inside their offerings but do not refer to them by name. Agent technology makes it possible to do things that were previously either impossible or could not be accomplished in an acceptable time frame.

#### **Where To Go From Here**

If you think agent technology might be appropriate for your organization, start by reading some books on agents (such as [10]). Then, talk to companies that provide or use agent technology (such as those mentioned in this report). In doing so, you will get an idea of how and where to use agents. Remember, though, that the biggest breakthrough with agents is that they are an *evolution* of existing technologies. What is *revolutionary* is the way we think about and use agents to design IT systems.

#### **Going Forward to the Global IT Organization**

Agent technology is now necessary to reduce costs; to improve efficiency and effectiveness; and to support the requirements of individuals, groups, companies, and universities as they collaborate globally. More importantly, it will enable us to create and support a whole class of IT applications and approaches that we previously could not have developed.

As Pulitzer Prize-winning author and columnist Thomas Friedman said so eloquently:

[We need] a global, Web-enabled platform for multiple forms of collaboration. This platform enables individuals, groups, companies, and universities anywhere in the world to collaborate — for the purposes of innovation, production, education, research, entertainment, and, alas, war-making — like no creative platform before. This platform now operates without regard to geography, distance, time, and, in the near future, even language. Going forward, this platform is going to be at the center of everything. [5]

This emergence of new business processes is resulting in new IT practices and approaches — where the two mutually reinforce each other. Now that individuals, groups, companies, and universities are interacting on a global scale, IT must not forget its supporting role. Agents are a key ingredient in IT globalizing itself.

#### **REFERENCES**

1. Baker, Albert D., H. Van Dyke Parunak, and Kutluhan Erol. “Agents and The Internet: Infrastructure for Mass Customization.” *IEEE Internet Computing*, Vol. 3, No. 5, September 1999, pp. 62-69.
2. Bonabeau, Eric, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

3. British Broadcasting Corporation (BBC) News. "Humble Ant Is a Software Saviour." 28 January 1998 (<http://news.bbc.co.uk/2/hi/science/nature/51337.stm>).
4. Ernst & Young Center for Business Innovation. "Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business." Christopher Meyer (ed.). *Proceedings from the 1996 Colloquium on the Business Application of Complexity Science*, Boston, Massachusetts, USA, 17-19 July 1996.
5. Friedman, Thomas L. *The World Is Flat: A Brief History of the Twenty-First Century*. Farrar, Straus and Giroux, 2005.
6. Luck, Michael, Peter McBurney, Onn Shehory, and Steven Willmott. "Agent Technology Roadmap: Computing as Interaction (A Roadmap for Agent Based Computing)." AgentLink, 2005.
7. Parunak, H. Van Dyke. "DESK: Density-Based Emergent Scheduling Kernel." Slide presentation to DTE Energy, material adapted in collaboration with NewVectors, 2000.
8. Resnick, Mitchel. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. The MIT Press, 1997.
9. Schoonderwoerd, Ruud, Owen Holland, Janet Bruten, and Leon Rothkrantz. "Ant-Based Load Balancing in Telecommunications Networks." *Adaptive Behavior*, Vol. 5, No. 2, 1996, pp. 169-207.
10. Singh, Munindar P., and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. Wiley, 2005.
11. W3C. "Web Services Architecture." W3C Working Group Note, 11 February 2004 ([www.w3.org/TR/2004/NOTE-ws-arch-20040211/](http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/))

#### ABOUT THE AUTHOR

James Odell is a Senior Consultant with Cutter Consortium's Enterprise Architecture practice and a regular contributor to the Enterprise Architecture advisory service. He is a consultant, writer, and educator in the areas of agent-based and object-oriented (OO) systems. Throughout most of his 35-year career, Mr. Odell has been heavily involved in developing better methods to understand, communicate, and manage system requirements. He was one of the early innovators of information engineering methodologies.

Mr. Odell is one of the first practical implementers of OO analysis and design. Working with the OMG and other major methodologists, he continues to innovate

and improve OO methods and techniques. He participated in the development of UML, and is the cochair of the OMG's Analysis and Design Task Force as well as the Agents Special Interest Group.

Mr. Odell conducts international seminars and workshops and provides consulting to major companies worldwide. Formerly, Mr. Odell was the principal consultant for KnowledgeWare, Inc., where he pioneered and taught the concepts of data modeling, information strategy planning, and CASE technology application. He now works as an independent consultant to advance the state of agent technology within the OMG and the computer industry.

Mr. Odell has coauthored, with James Martin, several books including *Object-Oriented Analysis and Design*, *Object-Oriented Methods: Pragmatic Considerations*, and most recently, *Object-Oriented Methods: A Foundation, UML Edition*. He also has published three books on agent-oriented software engineering.

Mr. Odell's clients include Amazon.com, Netscape, Chrysler, Capgemini, DHL, NASA, and other major companies across various business sectors in 19 different countries. He can be reached at [email@jamesodell.com](mailto:email@jamesodell.com).