Agent Technology and SOA

Introduction

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 new markets will be left behind.

In response, many companies are now building agent-based systems. These systems employ software agents that distribute functionality across vast computing networks. Furthermore, agents not only adapt to their environment but also evolve by learning from the environment and employ a variety of computational approaches, from OO to expert systems, neural networks, genetic algorithms, and so on. Such an approach prepares enterprises for an increasingly complex marketplace and enables them to respond rapidly to change.

As a result, we are moving from a vertical chain of command to a more horizontal chain of collaboration in both commercial and noncommercial activities. However, we need to adjust to make the transition from vertical to horizontal thinking. Vertical thinking typically involves asking who or what commands and controls the system; horizontal involves a more peer-to-peer way of thinking. Agent technology, then, will enable us to create and support a whole class of IT applications and approaches that we previously could not have developed. For this reason, two new SOA standards have emerged: OMG’s SoaML and OASIS’ Reference Architecture—both explicitly including the use of agents technology for SOA systems.

Agent technology, therefore, is now accepted as a primary enabler to support this new era. In fact, without agent technology, our current technology—including SOA, EDA (Event Driven Architecture), and BPM (Business Process Management)—will not scale to support the ever-increasing complexity and 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 can 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.

As illustrated in Figure 1, agents are an evolution of existing technologies. What makes it 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, 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.
What is a software agent?

Imagine sitting in the park on a nice summer day and a flock of birds sweeps the sky. One moment they 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. Each bird follows a simple set of rules that it uses to react to birds nearby. When Craig Reynolds of DreamWorks developed his simulation, each bird behaved basically according to the following rules (http://www.red3d.com/cwr/boids/):

• 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 towards the perceived center of their group).

The flock is organized without an organizer, coordinated without a coordinator. Flocks of birds are not the only things that work like this. 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 instead of a centralized authority.

In the world of IT applications, similar examples of decentralized coordination of activity can be found in order processing, supply chain, shop-floor control, inventory management, message routing, multiple database management, operating systems, and self-healing middleware. In other words, a decentralized approach is particularly suitable for agents and should be considered where local components have some degree of control (instead of limiting your approach 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? Agents can be designed to behave in a hierarchical manner. However, the strength of agents is that they can used in both hierarchical and collaborative ways; whereas, the conventional approaches lack such flexibility and scalability.
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; whereas, 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. It is a way of providing very fine-grained distributed processing, in which each processing unit, or agent, can interact with others. Here entire “social” systems based on these autonomous and interactive processing units are possible.

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—sometimes on behalf of another. However, for developing business and IT systems, such a definition is too general. While an industry-standard definition of agent has not yet emerged, a basic working definition is as follows:

An agent is an autonomous entity that can adapt to and interact with its environment.

In other words, most agree that agents deployed for IT systems are not useful without three important properties:

- **Autonomous** – is capable acting without direct external intervention. Agents have some degree of control over their internal state and can act based on their own experiences. They can also possess their own set of internal responsibilities and processing that enables them to act without any external choreography. When an agent acts on behalf of (or as a proxy for) some person or thing, its autonomy is expected to embody the goals and policies of the entity that it represents.

- **Interactive** – communicates with the environment and other agents. 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, such as negotiating contracts, marketplace-style bidding, or simply making a query.

- **Adaptive** – capable of responding to other agents and/or its environment. Agents can react to messages and events and then respond appropriately. Agents can be designed to make difficult decisions and even modify their behavior based on their experiences. They can learn and evolve.

Now, imagine that your SOA system has autonomous entities that can both represent you and automatically interact with consumers and providers to match and obtain SOA services. Instead of imagining birds flocking, imagine your SOA system behaving much as businesses do all over the world. In the OMG and OASIS standards, the Agent extends the typical IT OO-like Participant. Because of this extension, they are specialized to have their own thread of control or lifecycle. Another way to think of agents is that they are “active Participants” in a SOA system.
Participants are Components whose capabilities and needs are static. In contrast, Agents are Participants whose needs and capabilities may change over time.

Agents and OO

Just how different—or similar—are objects, components, and agents? Some developers consider agents to be objects or components, except with more bells and whistles. Others see them as different even though they have many things in common. However, both approaches envision using objects, components, and agents together in the development of software systems. 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 similar to—the object-oriented or component-based way.

Figure 2 illustrates one way of thinking about the evolution of programming approaches. 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 control only when invoked externally by a CALL statement. This was the era of procedures as the primary unit of decomposition.

In contrast, object orientation added to the modular approach by maintaining its segments of code (or methods) and 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. (This also includes the component-based approach.)
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 (or active components) 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. Savvy organizations, then, will employ a mixture of technologies in their applications: object orientated (OO), component-enabled, relational, and agent-based. OO, components, 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, that is a bottom-up and distributed approach. The real benefit comes when an organization can choose the appropriate mix of technologies for a given application—providing a balance of both the centralized and distributed approaches. (Note: Currently, few agent languages exist, e.g., JADE, JACK, ZEUS, and RETSINA. Most agent-based systems employ and extend JAVA to include the extra facilities required by agents.)

How can Agent Technology help?

Organizational involvement on a geographically broad level is a common way of doing business (Figure 3). For an organization to operate and compete, it needs to consider a boundary-less approach to collaboration, cooperation, and competition. Most SOA adoptions begin with a dozen or two services. Here, discovering, choosing, and binding your services can employ a basic SOA foundation with human involvement. However, when there are hundreds or thousands of providers offering the same (or similar) services—which can change from moment to moment—the situation becomes more challenging. Furthermore, many services will require the interaction and aggregation of other services. Such a complex and changing world is where agent technology can enhance the SOA environment most dramatically.

Figure 3. Geographically broad interaction is a common way of doing business.

Agents can employ a variety of techniques and roles, because they can mimic the way businesses and people work in the real world. For example, the technique
employed at the center of Figure 4 is an interaction protocol for making a contract between agents playing the roles of consumer and provider. Here, a consumer agent can broadcast a request for a particular service using a “call for proposal.” Those provider agents (on the far right of the figure) that can provide the requested service, respond with their proposal, or “bid.” The consumer agent can then choose from among the possible thousands of responses and “award” a chosen provider to supply the service.

If there are only a few providers and they are already known, using agents in this manner may not be necessary. However, if many providers are possible (and can change quickly over time), the agent approach for SOA is highly effective and efficient. Using agents, then, enables SOA consumers and providers to link up anywhere in the world. In other words, SOA can be enabled to be truly collaborative, dynamic, and boundary-less to the degree permitted by the business organizations.

Figure 4. Using interaction protocols based on business-based interaction rules.

Agents may also act as intelligent brokers and aggregators in SOA and EDA\(^1\) applications (Figure 5). Here, advanced techniques (such as rules engines, Bayesian process, and neural networks) can be used by an agent to become a smarter broker for a consumer or provider. Additionally, agents can “organize” themselves into groups to aggregate services in an effective and efficient way. Agents can even provide process management and scheduler roles as well as sophisticated process management and service-scheduling support that cannot be accomplished using conventional, centralized approaches.

---

\(^1\) For EDA agents, see Catalyst paper on Event Driven Architectures. [add link]
Figure 5. Agents can provide sophisticated support, such as brokers, aggregators, process managers, and schedulers.

An event-driven architecture (EDA) is an architectural style in which components are driven by events and can also communicate by means of events. [1, 2, 3] In particular, event-driven architecture provides the following functionality:

- Monitors and senses for changes and conditions in the system environment/databases.
- Initiates appropriate responses, actions, or processes.
- Modifies processes in real time for optimal response to changing conditions.

Being event driven, then, means that process behavior responds to events arriving from external or internal sources. Such processing behavior can be in the form of conventional processing architectures, SOA (Service Oriented Architecture), or BPM (Business Process Management). Many companies use a hybrid approach.

More and more companies 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 have these characteristics: (i) large and heterogeneous environments, (ii) many kinds of changes can occur frequently within the environment, (iii) the need to deliver and respond appropriately to complex exceptions and state changes in real time. (Figure 6)
SOA, BPM, and EDA

SOA, BPM, and EDA/CEP are three architectural approaches that can be separately considered for including agent technology. However, agent technology can be used to link business processes and services to facilitate a single architecture that integrates SOA, BPM, and EDA/CEP. Additionally, many other environments and architectures can benefit from employing agent technology. In conclusion, agents can be employed either as a stand-alone technology or with other technologies.

Extending SOA to be resource oriented

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 their systems. Conventional centralized or federated systems could no longer provide timely support given the size and complexity of their businesses.

At the heart of the solution, two primary forms of agents were used to gain distributed control:

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

### Managing processes and resources using agents

Figure 7 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 systems, the orchestrations of specific flows of activities (sometimes referred to as plans) are usually centrally managed. The services and resources can be managed in a distributed manner using an agent-oriented approach. 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.

![Workflow diagram for utility pole installation](image)

**Figure 7. Example of a business process where processes and resources are managed using agents.**

Such knowledge can be used in several ways. For instance in Figure 8, a requester might be interested in an Install Pole service but would like to know its cost before the service is invoked. Here, a call for proposal (CFP) would be sent to an agent that provides such a service. The agent would send a CFP to those resources required for

![Call for proposal diagram](image)

**Figure 8. A call for proposal (CFP) to the Install Pole agent results in a CFP to all of its resources and preceding operations.**
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 in short supply, the cost might be high; if not, the price would be lower. For example, if the requestor 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, they could all be busy on other assignments but would be willing to 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 requestor for the Install Pole service. In other words, the Install Pole agent acts like a broker on behalf of the process. If the requestor accepts the bid, the broker is notified and will confirm (or decline) the award.

Agent negotiation

The interaction protocol just described is expressed using the sequence diagram depicted in Error! Reference source not found. 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 or not either can be a resource agent or a process agent.

![Sequence Diagram](image)

**Figure 9.** An interaction protocol for making a contract.

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 re-optimized 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 distributed by nature—enables more dynamic scheduling by adapting to these unexpected situations on an individual and local basis rather than a massive 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. In Error! Reference source not found., the agent for Resource A asks the agent for the first step in Job 1 about any extra, or slack, time. 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. (Note: 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.)

Figure 10. Renegotiation example for Job 1 by process and resource agents.

The agent of Resource A must now examine the remaining option of changing Job 2 (Figure 11). Step 1 would be delayed by three hours. Step 1’s agent then asks Step 2 if a three-hour delay would impact its processing. Step 2’s 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 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) as well as reducing costs and time to market.

Figure 11. Renegotiation example for Job 2 by process and resource agents.
Agents and dynamic scheduling

The agent-based approach described above was developed with support from Rock Island Arsenal (RIA) as part of a DARPA contract to build an agent-based, factory-scheduling prototype, named AARIA. In one benchmark test using these techniques, inventory costs were cut by 47 percent, lead times were cut 59 percent, and schedule-reducible costs (such as overtime and inventory holding charges) were cut by 93 percent.

Using agent-oriented software (e.g., Cougaar Software), dynamic resource scheduling can be developed using:

- COTS and COTS-adapted components (e.g., resources, work center, parts, tasks/operation, and jobs) required for the scheduling application.
- Protocols (interactions/constraints between components) using a standardized representation language called an Agent Interaction Protocol (AIP).
- Rules/Policies that define the order of interactions and the content of an interaction based on a user-defined scheduling algorithm.

Conclusion

By the year 2000, people realized that they were in touch with people with whom they had never been in touch, and they were being challenged by people who had never challenged them. Our competitors were new and our collaborators were new. We were doing things as individuals that we had never dreamt of doing before. In what Pulitzer prize-winner Thomas Friedman referred to as the flat world, this phenomenon has caused, enabled, and empowered small groups and individuals to collaborate and compete globally.

Keep in mind, though, that adopting any new technology is disruptive, and agent technology is no exception. Yet, early-adopter companies are already finding that agent technology can provide one or more of the following benefits:

- Faster return on investment (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

However, is an agent-based approach useful for every application and usage? Does it always provide the benefits listed above? It certainly does not. Figure 12 illustrates one way to think about the choices.
One axis represents the level of adaptability and agility required of an application. If your business application is both predictable and stable and its processes are centralized and scalable for the foreseeable future, it is low on this axis and adopting an agent-based approach is probably not necessary. The other axis represents the level of complexity that will be involved in the application. Here, complexity refers to the amount of interaction among a number of different kinds of processing units. Complexity is high when many different kinds of processing units must interact with many others in a variety of ways. For example, the complexity would be high for a SOA system that federates hundreds or thousands of distributed service registries and databases. If such a system also required it to be agile in adapting to changes in properties, such as size, interaction, data, metadata, rules, then an agent-oriented approach would necessary. Conventional approaches could not handle this requirement and still be responsive in a timely manner.

Organizations all over the world are employing agent technology in their systems and products—to some extent. Some are simple server agents, others are “search bot” agents; but now large organization (e.g., NASA, US Army, DOD, TIBCO, IBM, HP) are employing agents to simulate and/or manage complex systems in a dynamic and scalable manner. Organizations of all sizes should be leveraging agent technology and enabling their customers to deploy adaptive, competitive solutions to challenging, real-world problems.

Figure 12. When to use an agent approach—or not.