Part 2

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.

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:

- **Autonomous -** is capable acting without direct external intervention. It has some degree of control over its internal state and actions based on its own experiences.

© James Odell, 2007

- **Interactive -** communicates with the environment and other agents.
- **Adaptive -** 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 is:

> 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 1.1 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, such as 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

© James Odell, 2007

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 even be designed to learn and evolve.

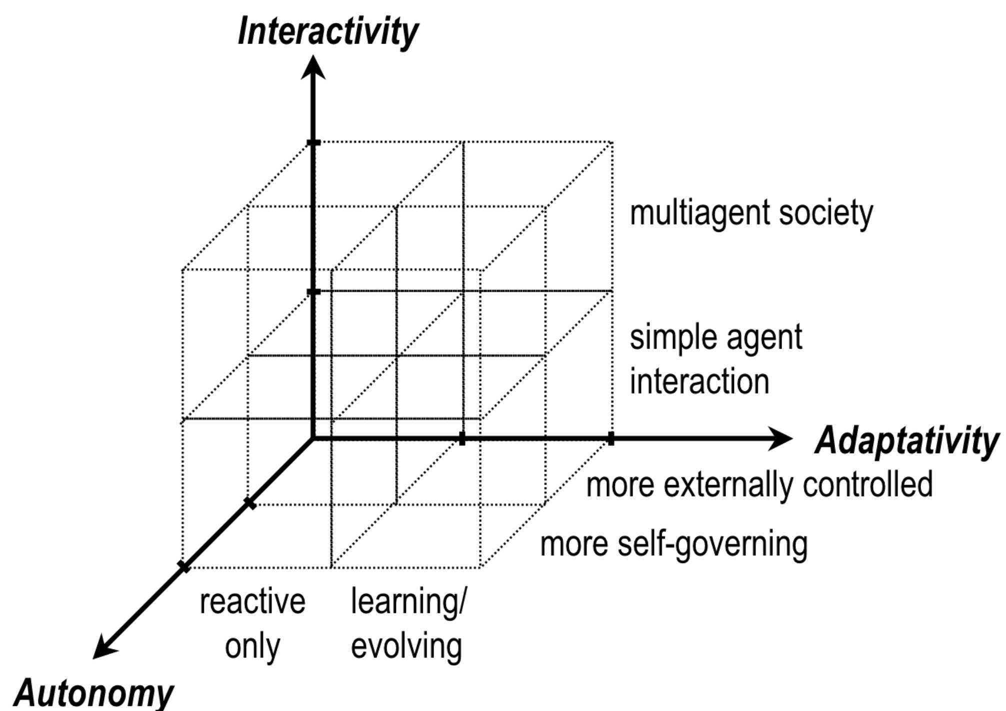

Figure 2.1 – Agents have various degrees of autonomy, interactivity, and adaptivity.

As illustrated in Figure 2.1, agents can be autonomous, interactive, and adaptive to some degree. It is not an all or nothing proposition. In the next few sections, these three key agent properties will be discussed in more detail.

© James Odell, 2007

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.

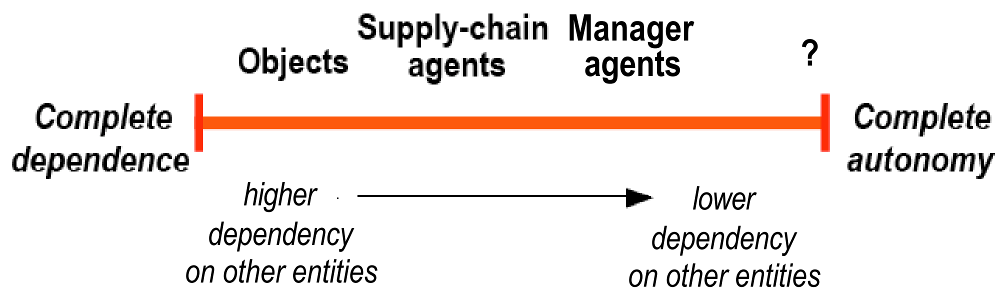

Figure 2.2 – Degrees of autonomy.

Depending on the system developer, an agent's autonomy can be designed to fit some place between these two extremes. For example in Figure 2.2, conventional objects are not completely dependent on outside resources—having some state and behavior of their own. But, they are 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 can not take care of an entire or-

© James Odell, 2007

ganization's supply chain. Yet 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, as well as its ability to make decisions. As depicted in Figure 2.3, 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.

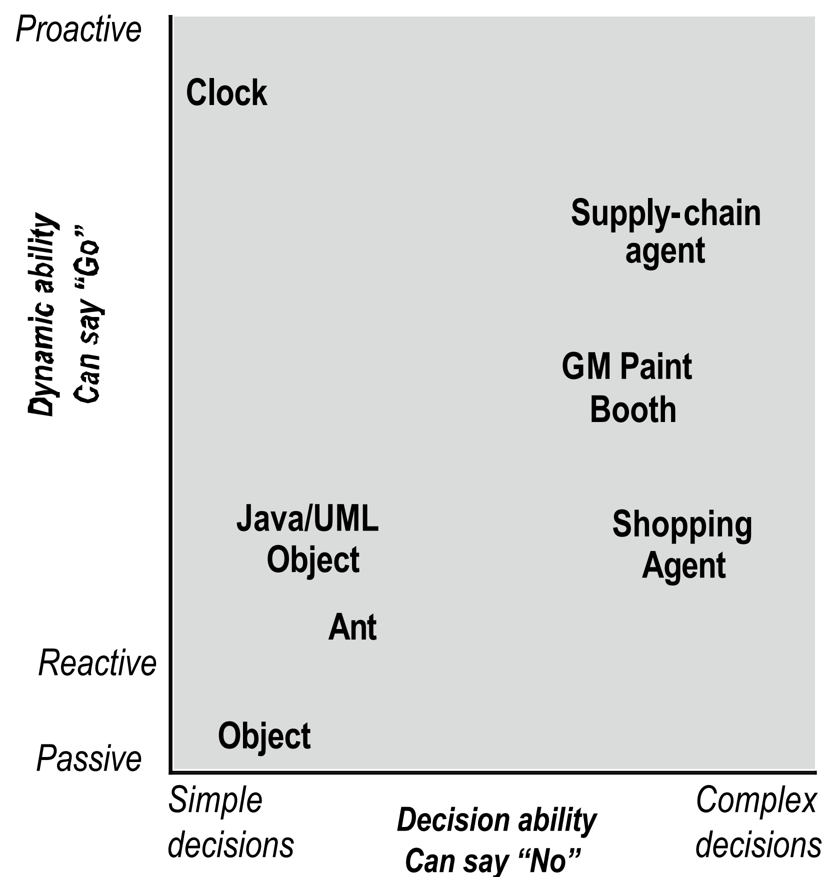

Figure 2.3 – Two aspects of autonomy

© James Odell, 2007

Dynamic Autonomy

Agents can react not only to specific method invocations but to observable events within the environment. Proactive agents will actually poll the environment for events and other messages to determine what action they should take. To compound this, 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."

For example, the GM paint booths described in Chapter 1 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.

© James Odell, 2007

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 booths 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 considered 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 tends 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 who can say "no."

© James Odell, 2007

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.

Agents are Interactive

Interaction implies the ability to communicate with the environment and other entities. As illustrated in Figure 2.4, interaction can also be expressed in degrees.

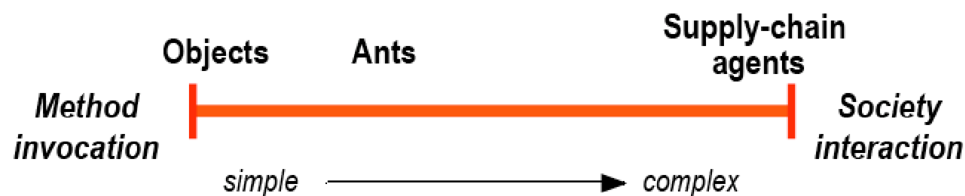

Figure 2.4 – Degrees of interaction.

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 multiagent 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

© James Odell, 2007

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 ever-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. For IT systems, adaptation enables systems to react appropriately to enable such effects as system balancing, integrity assurance, and self-healing systems. When designed properly, the individual parts of the system can be empowered to change based on their 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 by an 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.

© James Odell, 2007

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 would act as a service request, invoking the business process engine to choose the appropriate set of steps that should be executed. The choice would be based on the current context of the process. For example, a request to compute the sales task would trigger the engine to locate the appropriate set of steps that needs to be executed 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

© James Odell, 2007

as the bidder's ability to deliver its goods in the quantity, quality, and time frame requested. Over time, a *purchasing agent* can learn to choose from reliable *vendor 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 techniques 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. Here, agents can literally be bred to fit specific purposes. For example, operation plans, circuitry, and software programs can prove to be more optimal that any product that a human can make in a reasonable amount of time.

Other Agent Properties

In the sections above, 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 that is marked by friendliness or pleasant social relations, that is, where the agent is affable, companionable, or friendly.

© James Odell, 2007

- **Mobile -** able to transport itself from one environment to another.
- **Proxy -** may act on behalf of someone or something, that is, acting in the interest of, as a representative of, or for the benefit of some 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 are often coordinated via plans, workflows, or some other process management mechanism.
- **Cooperative -** able to coordinate with other agents to achieve a common purpose; non-antagonistic agents that succeed or fail together. (*Collaboration* is another term used synonymously with cooperation.)
- **Competitive -** able to coordinate with other agents except that the success of one agent implies the failure of others (the opposite of cooperative).
- **Rugged -** able to deal with errors and incomplete data robustly.
- **Trustworthy -** adheres to Laws of Robotics and is truthful.

© James Odell, 2007

Agents and OO

Just how different—or the same—are objects and agents? Some developers consider agents to be objects, except with more bells and whistles. Then, others see agents and objects as different even though they have many things in common. Both approaches, however, 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 own 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 similar to—the object-oriented way.

	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 2.5 – Evolution of programming approaches [1].

Evolution of programming approaches

Figure 2.5 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 opera-

© James Odell, 2007

tor. 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. Here, 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*) 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.

Object and agents 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, when and how an agent acts can be determined by the agent.

An agent-based approach is employed when a particular situation requires that processing be decentralized and self-organized, instead of centrally organized. While a centrally organized program could have been written to handle the ant simulation (Figure 1.1), the system 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

© James Odell, 2007

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. [3]

Yet, most developers tend to build centrally organized applications. They are also biased towards object-oriented 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 above, agents have such characteristics as autonomy, mobility, and adaptability. Furthermore, business users like to express other concepts, such as rules, constraints, 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.

© James Odell, 2007