Agent Technology:
An Overview

James Odell

22 October 2010 release

This paper was made possible by a grant from the CSC’s Leading Edge Forum (LEF), and the support of CSC’s Federal Consulting Practice (FCP).
# Table of Contents

Preface ........................................................................................................... iii

1. Introduction to Agent Technology ......................................................... 1

2. How can Agents be used? ................................................................. 17

3. What is an Agent? ............................................................................... 55

4. Anatomy of an Agent ......................................................................... 71

5. Agent Communication ....................................................................... 85

6. Agent Interaction .................................................................................. 101

7. Organization and Coordination of Work ............................................ 117

8. Agent-based Manufacturing: A Case Study ..................................... 139

9. Agents, Complexity, and Emergence ............................................... 157

10. Security and Architecture .............................................................. 172

11. OMG Agent Standardization .......................................................... 183

12. Conclusion .......................................................................................... 197
“Let every eye negotiate for itself and trust its agents.
-Much Ado About Nothing, Act II, Scene I (adapted)
Preface

Purpose of the this Paper

Typically, when the word “agent” is used, many people conjure up images from Web spiders and search bots. Yet, these same people feel that there is something more to the concept. For those who wish to know more, the purpose of this paper is to provide:

- a basic understanding of agents and agent-based technology.
- a summary of how agent-based technology is currently being employed.
- brief discussions of the major issues being addressed by agent research and development.
- indications where agent technology is relevant to the OMG—and vice versa.
- suggestions for standardizing areas of this new technology using the OMG process.

Intended Audience(s)

This paper is expected to provide readers at all levels (no prerequisites needed) with an understanding of what agents are, how they can be employed by CSC to build systems that can dynamically manage complex resources. Additionally, the paper is intended for CSC and its customers as a whole and, specifically, those groups that are working in areas that might involve agent technology. The purpose here is to encourage synchronization of those areas employing agents and agent-based technology.
This paper is also intended for the agent technology industries. Their thoughts and suggestions will help address the commercial significance of the concepts and directions established in the paper.

Acknowledgements

The author would like to acknowledge the contributions of the following people: Pascal Gambardella (CSC), Victor Harrison (CSC), David Kerr (Broadcom Eireann Research Ltd.), Heimo Laamanen (Sonera), David Levine (IBM), Gregory Mack (Booz-Allen & Hamilton, Inc.), David Mattox (Mitre Corporation), Francis McCabe (Fujitsu Labs, Stephen McConnell (OSM sarl), Kimmo Raatikaine (University of Helsinki), Kate Stout (Sun Microsystems, Inc.), Craig Thompson (Object Services and Consulting), Fred Thorne (CSC), and The Cutter Consortium.

Questions & Comments

Please address content-related questions or feedback to James Odell (email@jamesodell.com).

James Odell
www.jamesodell.com
October 2010
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 agent 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 1.1 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 orches-
trate rich systems that none of these technologies could provide alone.

Figure 1.1. Agent technology is an evolutionary approach.

**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 fined-grained distributed processing, in which each processing unit, or agent, can interact with others. Here entire
Chapter One

“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.
- **Interactive** – communicates and collaborates with the environment and other agents.
- **Adaptive** – 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.

Software agents (call *active agents in UML*) 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 appro-
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.)

Figure 1.2. Geographically broad interaction is a common way of doing business.

How can Agent Technology help?

Organizational involvement on a geographically broad level is a common way of doing business (Figure 1.2). 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.
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 1.3. Using interaction protocols based on business-based interaction rules.

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 1.3 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.
Chapter One

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 1.4. Agents can provide sophisticated support, such as brokers, aggregators, process managers, and schedulers.](image)

Agents may also act as intelligent brokers and aggregators in SOA and EDA\(^1\) applications (Figure 1.4). 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 boundary-less manner.

---

\(^1\) For EDA agents, see Catalyst paper on Event Driven Architectures. [https://catalyst.amer.csc.com/default.aspx?20_78780]
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.

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 (Figure 1.5):

- 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.

Figure 1.5. Employing agents for complex event processing (CEP) and event-driven architectures (EDA).

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.

**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.

**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 1.6 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.

References


[10] StarLogo - StarLogo is a programmable modeling environment for exploring the workings of decentralized systems -- systems that are organized without an organizer, coordinated without a coordinator. With StarLogo, you can model (and gain insights into) many real-life phenomena, such as bird flocks, traffic jams, ant colonies, and market economies. http://education.mit.edu/starlogo/


Two

How can Agents be used?

Introduction

Many companies have already tested agent-based applications in their research and development facilities. Successful applications are now used commercially throughout the world. Agents are used for both simulation and real-world execution. They are employed to address a variety of application areas—some of which only agent technology can enable. This chapter discusses some of the ways agent technology is assisted organizations to develop applications that were not possible—until now.

Agents in everyday life

Biological agents can produce everyday phenomenon such as ant colonies, traffic jams, stock markets, forest ecosystems. Software agents can enable supply chain systems, planning and scheduling applications, web services, and so on. Software agents need not be complicated. For example, in the ant colony simulation of StarLogo [1][10] or NetLogo [11] illustrated in Figure 2.1, each ant has three simple rules:
1. Wander randomly.
2. If food is found, take a piece back to the colony and leave a trail of pheromones that evaporates over time; then go back to rule 1.
3. If a pheromone trail is found, follow it to the food and then go to rule 2.

Figure 2.1 – Snapshots from an ant colony simulation.
In Figure 2.1(a), the ants 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 (Figure 2.1(b)). Figure 2.1(c) shows the ant colony well underway in retrieving food. Lastly, Figure 2.1(d) depicts two very depleted food sources and one that is exhausted altogether.

**Ant-like Agents at British Telecom**

As it turns out, the foraging technique of the simple ant provides an inspiration to tackling some difficult technological problems. While the insect’s totally random approach to locating food appears to be inefficient, at best, a new area of research known as *swarm intelligence* [2] has been established.

British Telecom (BT) Future Systems uses swarm intelligence for inspiration to make software systems more intelligent and robust, cheaper and simpler. The head of BT Future Systems, Chris Winter, 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.” In other words, the processing is distributed, not centrally controlled.

Winter used ants initially to weed out system bugs. In BT’s development network, little ant-like programs run around finding problems. They leave pheromone-like timestamps 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 Hewlett-Packard Laboratories created the world’s first ant-based systems [3]. 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 above was found both to scale well with the network size and to avoid the any 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 (pheromones) 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 towards 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.
Agent-Based Models (ABM)

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, then add things that are not present in the real world.

Individual ants are not very sophisticated insects. They have limited memory and a largely random element. Acting as a collective, however, ants can perform complicated tasks with reliability and consistency. Ant colonies, however, 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. This broadened approach to agent-based simulation that includes and goes beyond swarm intelligence is called agent-based modeling.

An agent-based model (ABM) is a class of computational models for simulating the actions and interactions of autonomous agents with a view to assessing their effects on the system as a whole. [9]

An agent-based model design is one in which analogs of those real-world entities that are to be modeled are represented as software agents, or objects at a level of detail and resolution necessary to address the questions the model is required to answer. The agents interact with each other as the model runs, producing dynamic information about the system being simulated. (Figure 2.2)
How can Agents be used?

Figure 2.2 – The ABM developer maps the target world to an agent-based model of the target world.[12]

Within an ABM, each agent individually assesses its situation and makes decisions on the basis of its set of rules. Agents may execute various roles, such as: producing, consuming, competing, collaborating, or selling. Repetitive competitive interactions among agents are a feature of ABM that enables dynamic exploration that is out of the reach of pure mathematical methods.

At its simplest, an ABM consists of a system of agents and the relationships among them. Even simple agent-based model will demonstrate complex behavior patterns—providing valuable information about the dynamics of the real-world system that it emulates. As mentioned earlier, agents may be capable of evolving, enabling the emergence of unanticipated behavior. More-sophisticated ABMs incorporate neural networks, evolu-
tionary algorithms, or other learning techniques that empower learning and adaptation.

ABM is more a mindset than a technology, where this mindset consists of describing a system from the perspective of its constituent units of agents and their interactions. A synonym of ABM would be microscopic modeling, and an alternative would be macroscopic modeling. Many researchers think that the alternative to ABM is traditional differential equation modeling or performing an actual real-world experiment. The ABM approach become advantageous when:

- the problem at hand is too difficult to do analytically (e.g., differential equations);
- an approximate theoretical result might not be reliable, and it is necessary to check it with a different method;
- an experiment is expensive or not feasible to perform.

One of the reasons underlying ABM's popularity is its ease of implementation: ABMs are both programmable and a large number of ABM tools exist to build ABM systems.

**Benefits of Agent-Based Modeling.**

The benefits of ABMs are that they provide a natural description of a system, are flexible, and capture emergent phenomena.

ABM provides a natural description of a system. In many cases, ABM is most natural for describing and simulating a system composed of “behavioral” entities. Whether one is attempting to describe a traffic jam, the stock market, voters, or how an organization works, ABM makes the model seem closer to reality. For example, it is more natural to describe how shop-
How can Agents be used?

pers move in a supermarket than to come up with the equations that govern the dynamics of the density of shoppers. ABM also makes it possible to realize the full potential of the customer data a company may have. Knowing the actual shopping basket of a customer makes it possible to create a virtual agent with that shopping basket rather than a density of people with a synthetic shopping basket computed from equations. Also, ABM looks at the organization from the viewpoint not of business processes models but what people inside the organization actually do.

One may want to use ABM when describing the system from the perspective of its constituent units’ activities is more natural, i.e., when:

- Individual behavior is complex. Everything can be done with equations, in principle, but the complexity of differential equations increases exponentially as the complexity of behavior increases. Describing complex individual behavior with equations becomes intractable.
- Activities are a more natural way of describing the system than processes.
- Validation and calibration of the model through expert judgment is crucial. ABM is often the most appropriate way of describing what is actually happening in the real world, and the experts can easily “connect” to the model and have a feeling of “ownership.”

**ABM is flexible.** The flexibility of ABM can be observed along multiple dimensions. For example, it is easy to add more agents to an agent-based model. ABM also provides a natural framework for tuning the complexity of the agents: behavior, degree of rationality, ability to learn and evolve, and rules of interactions. Another dimension of flexibility is the ability to
change levels of description and aggregation: one can easily play with aggregate agents, subgroups of agents, and single agents, with different levels of description coexisting in a given model. One may want to use ABM when the appropriate level of description or complexity is not known ahead of time and finding it requires some tinkering.

**ABM captures emergent phenomena.** Emergence results from agents and their interactions. (For more information on emergence, see the chapter titled, Emergence.) By definition, they cannot be reduced to the system’s parts: the system’s behavior is greater than the sum of its parts’ behaviors—because of the interactions between the parts. For example, a traffic jam, which results from the behavior of and interactions between individual vehicle drivers, may be moving in the direction opposite that of the cars that cause it. This characteristic of emergent phenomena makes them difficult to understand and predict: emergent phenomena can be counterintuitive. Numerous examples of counterintuitive emergent phenomena will be described in the following sections. ABM is, by its very nature, the canonical approach to modeling emergent phenomena.

Using ABM is appropriate when there is potential for emergent phenomena, i.e., when [13]:

- Individual behavior is nonlinear and can be characterized by thresholds, if-then rules, or nonlinear coupling. Describing discontinuity in individual behavior is difficult with differential equations.
- Individual behavior exhibits memory, path-dependence, and hysteresis, non-Markovian behavior, or temporal correlations, including learning and adaptation.
- Agent interactions are heterogeneous and can generate network effects. Aggregate flow equations usually assume global
homogeneous mixing, but the topology of the interaction network can lead to significant deviations from predicted aggregate behavior.

- Averages will not work. Aggregate differential equations tend to smooth out fluctuations, not ABM, which is important because under certain conditions, fluctuations can be amplified: the system is linearly stable but unstable to larger perturbations.

According to Bonabeau [13], “[ABM] may change the way we think about explanation in the social sciences. What constitutes an explanation of an observed social phenomenon? Perhaps one day people will interpret the question, ‘Can you explain it?’ as asking ‘Can you grow it?’.”

### Areas of Application

Running an agent-based model in a computer provides a formal proof that a particular micro-specification is *sufficient* to generate the global behavior that is observed during the simulation. “Agent-based models have been used since the mid-1990s to solve a variety of business and technology problems. Examples of applications include supply-chain optimization and logistics, modeling of consumer behavior, including word of mouth, social network effects, distributed computing, workforce management, and portfolio management. They have also been used to analyze traffic congestion and the traveling-salesman problem. Agent-based modeling tools can be used to test how changes in individual behaviors will affect the system's emerging overall behavior.”
“Other models have analyzed the spread of epidemics, the threat of biowarfare, the growth and decline of ancient civilizations, evolution of ethnocentric behavior, forced displacement/migration, language choice dynamics, and biomedical applications including inflammation and the human immune system agnet-based models have also been used for developing decision support systems such as for breast cancer.”

Multi-Agent Systems MAS

Above, we have limited the discussion of agents and agent systems to simulation. Simulation is process that runs agents systems using non-real-world environments. Here, the agent processing remains basically the same (or at least very similar).

However, substituting a real-world environment for a simulated one environment, we will be performing computations using actual existing agents: that is, input, processing, and output will occur on data based in real life. In other words, our agent application will go “live.” When this occurs in a world with multiple agent interacting among and with other agents, this is called a multi-agent system (MAS).

A multi-agent system (MAS) is a system composed of multiple interacting intelligent agents. Multi-agent systems can be used to solve problems that are difficult or impossible for an individual agent or monolithic system to solve.[9]

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

**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 expected defects, and so on.

Dick Morley, a technology visionary and the father of the programmable controller, swept away old assembly line schedules and developed a better system for painting trucks at GM’s trucks 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 the 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 bid-
ding, 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 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, broken down, or 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 a million dollars in nine months and reduced the lines of computer code from hundreds to four. Dick 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. The larger and more complex the organization, however, the more difficult it is to support centrally managed operations—particularly, large suppliers with volatile and demanding conditions that include unscheduled resource failures, periodic surges in new orders, and changes in requirements and priorities. Here, using agents 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 solution, two primary forms of agents were used to gain distributed control:\footnote{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.}

- **Process-based agents** – have the knowledge of how to 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, machines, materials, and facilities.
Business processes and resources using agents

Figure 2.3 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. 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.
Figure 2.4 – A call for proposal (CFP) to the Install Pole agent results in a CFP to all of its resources and preceding operations.

Such knowledge can be used in several ways. For instance in Figure 2.4, 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 then send a CFP to those resources and preceding services that would be 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 requestor wanted a Utility Pole in two hours, but 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 all might be busy on other assignments—but would do the requested job at double-time rates.

Once the resources had responded to the CFP with a bid, the Install Pole agent would tally up the cost for the service as a

---

2 Agents are a useful way to extend service-oriented architectures (SOA).
whole and send 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 of the award and will confirm (or decline) the award.

**Agent negotiation**

This interaction protocol just described is expressed using the sequence diagram depicted in Figure 2.5. 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.

![Sequence Diagram](image)

**Figure 2.5 – 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 and global one.

![Diagram](image.png)

Figure 2.6 – Renegotiation example for Job 1 by process and resource agents. (based on [5])

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 Figure 2.6, 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 2.7 – Renegotiation example for Job 2 by process and resource agents. (based on [5])

Resource A’s agent must now examine the remaining option of changing Job 2. (Figure 2.7) 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 and reducing costs as well as time-to-market.
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. [4]

Since that time, agent-based schedulers have been designed for suppliers in many industry sectors: manufacturing, finance, energy, and transportation. NewVectors (www.newvectors.net), Magenta Technology (www.magenta-technology.com), Cougaar Software, and Intelligent Automation, Inc. (www.i-a-i.com) have been active in this area.

Magenta Technology has developed an agent-based system that dynamically schedules the deployment of ships and cargo. Intelligent Automation 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 the:

- Components (e.g., resources, work center, parts, tasks/operation, and jobs) required for the scheduling application, using the component libraries
- 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
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. A brief overview of some of them include the following:

- **Logistics, Material Management and Transportation** – High fuel costs, increasing competition and rising custom demands for flexibility and visibility has created challenges for the existing IT infrastructure of transportation companies. Costs must be managed with the best possible routing, taking into account environmental, traffic, fuel price, and other issues on each possible route and that planning must be dynamic and re-planning must be continuous even during execution. Less than full loads must be optimized. And all of this must be done without scrapping your existing IT investments. What is needed is an intelligent a layer of intelligence to put you in control of your organization.

Agent-oriented solutions help provide more visibility with integrated, comprehensive and local dynamic views of the distribution network, enable more effective management of local distribution operations, and provide robust and secure distributed collaboration and execution. Its situational awareness and representation agent-oriented layer gives companies the ability to collaborate and partner effectively with supporting distribution agencies. With it, the user can create an accurate, global, and understandable operational picture of the current transportation situation as well as provide advanced event management, execution monitoring, and collaborative decision support. Users can monitor activities
during execution and dynamically re-plan in the face of external changes or modified operational requirements.

- **Smart and Green Buildings** – With rising energy costs and the desire to support environmental concerns, businesses and building managers need to find every way to improve the efficiency of their facilities. However this cannot be done at the expense of employees’ comfort or productivity. How to manage this trade off space between efficiency and employee choice is key. Similarly within the efficiency tradeoff space, there are trade-offs within every decision – such as between artificial light that uses energy and natural light that might add heat when the air conditioning is on – that must be managed in real-time as conditions change. Finally, security is becoming more complicated with too much data from sensors and cameras is being delivered to be processes by humans. Security systems must take the load off of humans so they can focus where human decision making is best used. To be as green, efficient and safe as possible, buildings must be as smart as possible.

Agent-oriented technology can provide this intelligence by adding a thin layer of intelligence to integrate and enhance existing building systems. This lets you harness you existing investments to support new intelligent capabilities without disruption. Intelligent agents are distributed to the places in the building where they can do the most good from sensor to mainframe, providing a virtual nervous system. Using an agent-oriented cognitive model also provides the “brain” as well. This creates a common operating picture to integrate all building operations from lights, to networks to security. This gives building managers the ability to monitor and control all aspects of building operation so to optimize its operation while maintaining worker control. It can automate certain decisions while watching for unexpected event to bring to
the attention of decision makers. It can also negotiate with suppliers such as telecommunications and electricity suppliers for the best pricing based on parameters you set. (Cougaar Software)

- **Smart Manufacturing - Autonomic Manufacturing Solutions** – Manufacturers today face both competition from globalization but also opportunity. But the benefits of globalization can only be met if the value global supply chains can be maximized. Central to success is processing the vast amount of information to create situational awareness and common operating picture of the supply chain. Companies must move beyond the obsolete and risk fraught Just-in-Time Paradigm to the dynamic, agile and less fragile Sense and Respond Paradigm that is replacing it. Managing uncertainty and risk and balancing those against efficiency is the key. Within the factory, whether it is an assembly line or a traffic management system, adaptive capabilities are key to handling the demands of networked manufacturing equipment. Self-management of the system is key, while at the same time using complex even processing to alter managers of unexpected events.

Solutions built with agent technology can manage, track and audit supplier relationships while maintaining accurate and appropriate (and minimal) inventories to ensuring production processes have the right material at the right time and avoid interruptions, delays, and errors in production processes. This will allow you to gain competitive advantage while increasing customer loyalty and retention by increasing supply chain velocity reducing time to market, decreasing costs of materials and of managing operating processes and managing complex intelligent machines – and the data they generate for prognostics. (Cougaar Software)
• Energy, Oil, Gas and Refining – Energy businesses are perhaps the most global companies in the world – and they are the most harsh and demanding locations. Mistakes can cost huge losses and even lives. Managing these far flung operations is a logistical and operational challenge second only to the military. Add to these issues of deregulation, M&A, global compliance, safety and security and there is a need for distributed intelligent solutions that tie all aspects of operations to one common operating picture. Such a solution would provide global or localized visibility and of assets, supply chain, and operations both throughout the company and partners as well as process automation and decision support.

Solutions built with agent technology puts intelligence where it is most needed whether it is in your organization, your partners or suppliers with our highly distributed architecture. Most importantly, agent environments act as a fabric of intelligence between your existing systems so your current IT investment is not only re-used but enhanced.

• Just-in-time delivery – When operating in a highly competitive market, delivering just-in-time 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) enabled the company to reduce inventory levels by 35% while maintaining its delivery commitments. [7]

• Adaptive Logistics (AL) have been developed to prototype and demonstrate different disciplines of Logistics, Distribution, Maintenance, and Supply. In particular, the Army’s Adaptive Logistics (AL) project now integrates the use of intelligent agent technology and cognitive decision support tools as key drivers of Sense and Respond Logistics (S&RL). The intelli-
gent agents monitor and synthesize large volumes of data from disparate sources to improve situational understanding and rapidly provide COAs to facilitate decision-making, as illustrated in Figure 2.8. The AL project by demonstrates how the net-centric decision support capabilities and intelligent agent technologies can be extended to the strategic level. (Cougar Software, www.cougaarsoftware.com)

Figure 2.8 – ALCT Task Planning Display (Cougar Software)

- **Production Planning** – Production planning is a logistics approach that provides intelligent maintenance estimation and recommendations on optimal allocations, capabilities, and performance analyses, “What if” simulations, and dynamic monitoring and alerting. Cougar Software developed the MyProductionPlanner (MPP) tool has been developed by LOGSA for Army Sustainment Command (ASC), Distribution Management Center (DMC). Figure 2.9 illustrates the MPP process flow.
How can Agents be used?

Figure 2.9 – Army MPP process flow (Cougaar Software)

**Distribution optimization** – Delivery of goods and services between site and customer is a problem for many industries. Companies such as British Telecom, DHL, and Air Liquide 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: Eurobios, Whitestein Technologies (www.whitestein.com), and Cougaar Software.

- **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, Intelligent Automation 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 demand. 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 include Magenta (www.magenta-technology.com), Cougaar Software (www.cougaarsoftware.com), and Intelligent Automation (www.i-a-i.com).

Architectures and ontologies — and agents

In addition to specific applications, the IT developer must address current architectural approaches. A brief overview of those particularly appropriate for agents include the following:

- **Ontologies** – Agents require a common vocabulary and set of concepts to communicate effectively with other agents. Cybele (www.cybelepro.com) and Cougaar (www.cougaar.org) 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.

- **Peer-to-peer (P2P)** – 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 applica-
tions are “social” in nature, they will require increasingly sophisticated approaches both to trust and reputation and to the application of social norms, rules, and structures. Social simulation would be particularly appropriate here to better understand the population dynamics of independent agents. [7]

- **Web Services (WS) and Service Oriented Architectures (SOA)** – Agents can be used to support WS and SOA in many places. For example, Verizon chose a decentralized agent-based approach to SOA, rather than a centralized approach where all services are handled by a serve or set of servers. Additionally, they employ agents to support the subscription, management, and dashboard layers of their SOA, using their IT Workbench software. By the end of 2004, Verizon was handling nearly three million service transaction per day.

Another goal of companies like Verizon is not only to reduce redundancies by building reusable services, but to put the services together to create more sophisticated and valuable services. *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. The W3C Web Services Architecture study [6] recommends an agent-based approach to using service-oriented architectures—both on the service-requestor side and the service-provider side. Such an approach is not necessary or useful for simple services. However, an agent-based approach is useful when the services and their interactions require an active computational entity that: (i) has a persistent identity, (ii) can perceive, reason about,
and initiate in its environment, and (iii) can communicate with other agents, including humans.

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 make it possible to capture the interactions among services and the creation of new services as subtle compositions of others. Agents are not a panacea. However, applying agents in the appropriate places within applications and systems enables us quite naturally to (i) capture deeper constraints on what services are willing to offer, thereby capturing richer requirements for service composition, (ii) discover trustworthy services, (iii) negotiate with external service providers, and 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. [8]

Figure 2.10 – Services can be carried out by many different business processes depending on business conditions and rules.

- **Business Process Management (BPM)** – BPM has come to mean an integrated collection of critical process technology neces-
sary 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, as illustrated in Figure 2.10, multiple business processes can be used to satisfy the service. For example, to request a product’s price, the computation could vary depending: location of order, time and date ordered, quantity, customer, and other items in the same order.

The DHL pricing system had nearly 500 different processes for pricing an order because of the number and combination of variables. To handle this, DHL used agents to choose the appropriate process. In other word, each agent contained a process-execution engine that would select and execute the appropriate process for each pricing-query service request. Furthermore, if a problem arose in the middle of a process, the agent would select an alternate process. DHL found that using 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) would take hours rather than weeks. The software they used employed a graphic editor based on UML activity diagrams, so that even a user could change the process. Again, agents are not always appropriate for controlling business process logic. If the process is simple, straightforward, and does not require frequent changes and processing robustness, a language like BEL could be used. However, for complex processes that involve various operating conditions and require frequent and timely changes, agents should be used.

- **Event-driven architectures (EDA)** – 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 Cougar Soft-
ware) 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) information that changes *constantly* in a variety of ways, (iii) complex exceptions and state changes in real-time, and (iv) the need to deliver and respond appropriately to that information.

Business communications are numerous and often complex. One communication could be monitored by thousands, and then filtered, analyzed, and aggregated with other communications – which in turn can become noteworthy events for others, and so on. As this continues to grow in size and complexity, our ability to understand, analyze, and control can quickly go beyond the abilities of conventional technology. Here is where agents can step in.

For example, Figure 2.11 from Roy Schulte (Gartner) is a token representation of a complex event processing environment. One the left, you can see an even processing agent (EPA) receiving two events. As a result of these two events, the EPA may determine that another event should (or should not) occur. For instance, a Buy and Sell event for a particular customer’s stock might have occurred. If the Sell event occurred prior to the Buy, trading-error event could be generated. This event might be one of three events that another EPA was monitoring, resulting in another event. Which, in turn, are monitored by another EPA, resulting in one of perhaps two events, depending on results of its analysis.
So, why use agents? Why not use a large brain-like approach that would solve the event processing centrally, such as using an expert system or neural network? You could — up to a point. However, when your CEP starts requiring thousands of rules to be processed, maintaining a single expert system would quickly become unwieldy to maintain and test. So, another approach is to divide the expert systems into smaller, manageable — yet federated — expert systems. This would be tangible up to a dozen or so expert systems, but what happens then? Would you understand and manage them? If your expert systems each had a particular business-based theme, you might be able to manage them because there would be some rhyme or reason for the rule grouping. The result, then, is a set of smaller yet federated set of event-processing business-organized “brains.” But again, as each expert system becomes too large, it too must be partitioned in multiple, business-based, smaller ones.
So, where does this all stop? As an organization — particularly a global one — expends or changes, this constant divide-and-conquer approach will not stop. The agent approach, then, offers a more sane alternative: start with small business-based units to begin with.

Agents are autonomous and interactive entities that are defined on the basis of a specific need. They can play multiple roles, can learn, and evolve if need be. They behave socially and can self-organize. In other words, they are adaptable so that they can service specialized needs and communicate with others. Just like social beings, they have the ability to understand, analyze, and control their specific niche.

Agents, then, are a way to divide and conquer the problem, much as humans (and other organisms) do. It’s still software, but it’s modularized based on a business-based need and can be thought of as small-grained “intelligent objects.” They can mimic social configurations and interactions and can scale and perform as an organization. Leveraging agents in this manner, then, enables event processing that is concurrent and distributed. Does this sound like the living, breathing world around us? It should. This is why agent-based applications are able to better manage complex systems and be adaptable.

So, what is the relationship between agents and events? Agents enable event processing networks to scale in size and complexity -- and be designed in a more human-understandable manner. Agents can be used for monitoring, filtering analyzing, and aggregating event and notifying other
agents. As Paul Vincent indicated in his earlier blogs, agents can also be used within the CEP middleware.³

- **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, as illustrated in Figure 2.12.

![Figure 2.12 – Agents can be used to integrate SOA, BPM, and EDA.](image)

**Conclusion**

The list above provides a sampling of uses for agent-based systems—happening now within the IT community. The underlying question, though, is why did these companies decide to use agent technology? Adopting any new technology is disruptive. As stated in the previous chapter, the reasons that agents were chosen by these early-adopter companies include 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
- Provides foundation for future projects
- Reduces time to market
- Increased agility to respond to business needs
- Enabled an organization to use/provide services and resources in a way that was not previously possible

In the coming chapters, this paper will provide you with more insight about what agents are and how they can be used. It is one thing to say that “agents can provide a helpful, distributive approach to software applications”; it is another to understand what that means and entails. The next chapter begins that journey by answering the questions: “What is an Agent?”

References


How can Agents be used?


[10] StarLogo - StarLogo is a programmable modeling environment for exploring the workings of decentralized systems -- systems that are organized without an organizer, coordinated without a coordinator. With StarLogo, you can model (and gain insights into) many real-life phenomena, such as bird flocks, traffic jams, ant colonies, and market economies. [http://education.mit.edu/starlogo/](http://education.mit.edu/starlogo/)

[11] NetLogo - NetLogo is a multi-agent programmable modeling environment. It is used by tens of thousands of students, teachers and researchers worldwide. You can download it free
of charge. http://ccl.northwestern.edu/netlogo/

http://jasss.soc.surrey.ac.uk/12/1/1.html

www.pnas.org/content/99/suppl.3/7280.full


How can Agents be used?
What is an Agent?

Introduction

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. 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 enable them to act without any external choreography. As such, they can act in reactive and proactive ways. 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. In UML terms, agents can have classifier behavior that governs the lifecycle of the agent.

**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. In the Woodridge-Jennings definition of agency, this property is referred to as social ability.

**Adaptive** - capable of responding to other agents and/or its environment. Agents can react to messages and events and then respond in a timely and appropriate manner. Agents can be designed to make difficult decisions and even modify their behavior based on their experiences. They can learn and evolve. In the Woodridge-Jennings definition of agency, this property is referred to as reactivity and proactivity.
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 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.
What is an Agent?

Figure 3.1 – Agents have various degrees of autonomy, interactivity, and adaptivity.

As illustrated in Figure 3.1, 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.

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—primarily because it would be unmanageably large and unnecessarily complex for any common role. At the other extreme, an agent that is barely able to per-
form the simplest self-management is of limited use in complex systems. Even traditional objects have some degree of autonomy within their methods.

![Diagram of autonomy levels](image)

**Figure 3.2 – Degrees of autonomy.**

An agent’s level of autonomy can be designed depending on its role. For example in Figure 3.2, conventional objects are not completely dependent on outside resources—having some state and behavior of their own. However, they are traditionally simple in that they are far from being completely independent. Because a “supply-chain agent” is able to reach certain conclusions and make decisions on its own, it has more autonomy than an “object”. By their very nature, supply-chain agents require interactions with other entities to enable a fully functional supply-chain system, since one agent can not adequately support an entire organization’s supply-chain processing – interactions are also a characteristic for the class of agents called event processing agents (EPA). Even more autonomous are manager agents, because their role can involve a high degree of internal decision making through activities such as monitoring and delegation, 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 3.3, 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 3.3 – Two aspects of autonomy](image)

**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.

**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.”

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 3.4, interaction can also be expressed in degrees.

![Figure 3.4 – Degrees of interaction.](image)

On one end of the scale, object messages (method invocation) can be seen as the most basic form of interaction. A more complex degree of interaction would include those agents that can react to observable events within the environment. For example, food-gathering ants do not invoke methods on each other. Their interaction is indirect through the direct physical effects of their pheromones on the environment. Even more complex interactions are found where agents derive information — complex events — from multiple events over time. Here agents begin to act with “intelligence”. And then there are multiagent systems (MAS) 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 cooperatively and/or competitively through such mechanisms 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 to other agents and/or its environment. Entities that fail to adapt to the ever-changing world become extinct: organisms die, companies go out of business, and so forth. 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. The goals and motivations for adaptation are around remaining competitive and useful, and deriving value. When designed properly, individual parts of systems can be empowered to change based on their environment and market conditions.

Simple Reaction Rules

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 events can 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. Simple reactions can also include business decision rules — such as event-condition-action (ECA) rules.

Inference 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, certain kinds of data-mining agents and identifying complex events. Such inferences are computed by following a chain of inference rules, which can be described as goal-directed or data and event-driven.

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. Popular learning techniques 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.

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 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 than 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 they are essential for agents deployed by IT systems. However, agents may possess various combinations of other properties whose usefulness depends on the application requirements and the agent designer. Here, agent characteristics may include being:

- **Sociable** - interaction that is marked by friendliness or pleasant social relations, that is, where the agent is affable, companionable, or friendly.
- **Mobile** - able to transport itself from one environment to another.

- **Proxy** - may act on behalf of someone or something, that is, acting in the interest of, as a representative of, or for the benefit of some 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.
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.

Figure 3.5 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-
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.

**Conclusion**

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 2.1), 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 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. [2]

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.

References


Introduction

As we discussed in the previous chapter, 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.

Agent as a black box

At its simplest, 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 4.1 depicts this by illustrating that agents have input from, and output to, the environment — i.e., it receives and emits events. 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 constraints of motion and control, and other agents. For software agents, the environment would consist of the hosting platform, its operating systems and supporting middleware, and other agents.
Anatomy of an Agent

Input can be whatever the agent “perceives,” such as messages, requests, commands, and other events. Based on such input, the agent can decide how to act, which typically results in some output from the agent, such as communicating with, or making a change to, its environment. For instance, our Robotic agent could receive a move command, and then change its state through movement (events). An Inventory agent could receive a request for the quantity-on-hand for a given commodity, causing it to determine the requested quantity and send a response event to the requestor. The Inventory agent would also receive notification events regarding inventory being removed and sent to a customer. The agent process would then recompute the quantity-on-hand and decide if a reorder point has been reached, and if so, would send out requests-for-bid to appropriate suppliers.

Agent processing overview

For an agent to interact with its environment, it must not only detect and effect its environment but also understand what it is detecting, determine an appropriate response, and 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 can be visualized as illustrated in Figure 4.2.

![Diagram of an agent's structure](image)

Figure 4.2 – The basic internal architecture of an agent.

**State**

Just like objects, agents can have a form of memory, referred to here simply as *state*. State includes the data that it needs to operate and is maintained or can accessible in some manner (such as, via a database or automated blackboard) by the agent. For example, the 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 the list of valid product vendors that it can contact for reordering—or at least know where the information might be found, or which other agent it can delegate that request to. The agent could even make inferences about its own 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 agent’s interface to the environment. They receive the input events that are of interest to the agent. Detectors can both receive input sent to them (i.e., “push” model) or actively scan the environment for it (i.e., “pull” model). Detectors can receive simple messages and event notifications, or they can scan message logs or monitor event occurrences. In particular, event-monitoring agents (in event-driven architectures) may be specialized to recognize particular kinds of state changes in the environment. For example, master data agents could watch for certain kinds of changes in a database such as metamodel modifications, customer-address changes, or even the database going offline. If any of these events were detected, the agent could notify other agents that can react appropriately, such as the system monitor agent that notifies the DBA. In event-driven architectures (EDA), this is particularly useful. Our robotic agents may employ such sensors as cameras, global positioning hardware, or infrared devices for detectors, and directly process and interpret the associated images and event streams.

Parse input

Once the detector receives input, the agent needs to analyze and “understand” the input. This action typically involves recognition, normalization, and exception handling. For conventional object-oriented software, there is virtually no parsing: the OO “signature’s” the first parameter identifies the object, the second specifies the requested operations, and the remaining input contains the data required for the operation execution. Because the signature syntax is already defined in this manner, the message does not need to go through any “determine ac-
tion” 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 IT agents need to process complex syntax and receive input in one or more languages, such as SQL, Prolog, BPEL4WS, BPML, or ebXML. They may need to understand complex ontologies expressed in OWL before any further action can be determined against some RDF data. For complex-event-processing agents, this parsing can mean identifying event patterns in the input. Such event pattern recognition is important for agents, especially those that need to recognize many potential new state situations (e.g., self-healing systems).

Granted, activities like parsing require greater complexity than a traditional object. In fact, not all agents will require such sophisticated behaviors. Yet, when system requirements demand richer behavioral control at execution time, agents need to be able to accomplish what traditional objects cannot. Activities like parsing or accepting rule updates 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 is determined before a message is sent to the object. This is 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 needs the ability to determine the appropriate method after it receives a request for service. In this way, a Policy agent can decide on the appropriate method to rate a policy based on the requestor’s needs, Order agents can decide 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 would detect each time a product was pulled from a warehouse bin, and for each such instance it might need to first decide if the product should be reordered or, if it is selling too slowly, discontinue the product 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, and await responses.

The Determine Action process might involve a high degree of complex processing in its own right. Can this be accomplished using an OO approach? Absolutely. However, some agent platforms have mechanisms that support higher order models of decisions directly— for example using rule engines instead of having the programmer do extra work in design, construction, testing, and maintenance.

**Invoke action**

Once selected, the appropriate action needs to be invoked. The action execution may involve actions internal to the agent, in which case, message events can be sent directly to the agent’s own detectors. If the action involves other agents, the message
events 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 some Determine Action step determine which action should be taken next. In other words, the agent can have its own self-aware processes. In a business process management system, such mechanisms can be handled in a distributed manner using events and agents, rather than by some centralized controller 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 multiprocessing**

Agents can be single threaded or designed to process input concurrently (Figure 4.3). Agents that can support concurrency might also have a process controller that coordinates the various internal actions in some manner.
Figure 4.3 – Concurrent processing within the agent is possible.

Reasoning, planning, and scheduling

A common approach for agents can involve reasoning, planning, and scheduling. 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 requestor). Additionally, the agent reasoner 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 conflicting goals to choose from). 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 constraint. A planner then determines the actual steps needed to execute the agent’s intension. 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 in-house 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.¹

Figure 4.4 – Determining the appropriate action can involve reasoning, planning, and scheduling.

¹ 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 the goals the agent could attempt to achieve, and intentions refers to the set of goals the agent actually intends to achieve under a given circumstance.
On top of this, an agent may also employ a *coordinator* that ensures that the agent’s concurrent processing works harmoniously: managing conflicts and optimizing the overall functioning of the agent.

**Agents and aggregate agents**

One of the most difficult challenges for automated systems is scalability. Here, life-as-an-analogy brings with it many useful concepts, including some excellent examples on how to scale up. In the physical systems leading up to life, for example, subatomic particles form atoms, and atoms cluster to become molecules in solid, liquid, and gaseous form. Continuing up this hierarchy, molecules can be organized to form organelles and cells, cells can aggregate to form organisms, and so on. In other words, living systems and their components emerge in a hierarchy of interlocking mechanisms.

![Diagram of Eukaryotic Cell](image)

*Figure 4.5 – Eukaryotic cells provide a good example for composite agents.*
If such an emergent approach works in nature, it can also provide a good model for developing agents and agent-based systems. For example, Figure 4.5 depicts the classical biologic cell structure. Notice that the cell is consists of other bodies. The mitochondria, for example, is a cell in its own right. It is an important energy source for the eukaryotic cell that contains it. Mitochondria, then, provide a good example of an agent inside of an agent, where the whole depends upon the parts for its complete functioning.

Agents, too, can be aggregated to form variable structures. These aggregates can be colonial in nature (such as sponges and coral reefs) or metazoan (that is, multi-cellular animals). Agents that adhere to one another can behave in a unified manner and still maintain their own autonomy. Automated agents might choose to aggregate for various reasons, such as protection, resources, or improvement. In other words, the agents might decide that they are better off together than apart. Furthermore, the agents may adapt to serve the aggregation as a whole.

![Figure 4.6 – Agent aggregation](image-url)
Figure 4.6 depicts several aggregation options. [9] The first option, of course, is that there is no aggregation. The second option is that two or more agents can aggregate as a single unit. The containment boundary indicates that the entire construct can be treated as an agent in its own right. In option 1, the agent and the boundary are the same. In option 2, the boundary can be treated as an agent separate from the two agents it contains. The term boundary is chosen because it also provides an interface barrier much like that of a cell membrane. Agents on the membrane (as in option 2) are still accessible from the outside as individuals while benefiting from the proximity to chosen neighbors. In contrast, agents that are contained completely within the boundary, as illustrated in option 3, are encapsulated. Only those agents on or immediately within a boundary may communicate with encapsulated agents. In a complex aggregate configuration (option 4), the nesting continues.

Figure 4.7 – Ant-like simplicity is still very successful—and has been for a very long time.
Conclusion

Agent-based systems do not require complex forms of agents like those described above. For example, ant colonies have been very successful for a long time (7 million years). An estimated 10% of the Earth’s animal biomass is composed of ants. (Much more than humans). An ant has about 250,000 brain cells; humans have about 10 billion. (Simple rules) 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 (Figure 4.7)? 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.
Introduction

Many of the early commercial agents were developed for information search. Here, individual agents were launched on a tether to gather predefined kinds of information and return them to the human requester. In other words, these agents were solo operations that had very little—if any—interaction with other agents. Such an approach certainly has its many uses. However, if we look at the human world, such an approach alone could not build the societies or support the organizations that humans have come to enjoy. Instead, we set up networks of people that interact for various purposes. Interaction among agents, then, is not sufficient to build agent societies, we need agents that can coordinate—either through cooperation, competition, or a combination of both. These agent "societies" are called multiagent systems (MAS).

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

Some of the rationale for multiagent systems are as follows:

- One agent could be constructed that does everything, but fat agents represent a bottleneck for speed, reliability, maintainability, and so on (i.e., there are no omnipotent agents). Dividing functionality among many agents provides modularity, flexibility, modifiability, and extensibility.

- Specialized knowledge is not often available from a single agent (i.e., there are no omniscient agents). Knowledge that is spread over various sources (agents) can be integrated for a more complete view when needed.

- Applications requiring distributed computing are better supported by MAS. Here, agents can be designed as fine-grained autonomous components that act in parallel. Concurrent processing and problem solving can provide solutions to many problems that, up until now, we handled in a more linear manner. Agent technology, then, provides the ultimate in distributed component technology.

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

- provide an infrastructure specifying communication and interaction protocols.
- are typically open and have no centralized designer or top-down control function.
- contain agents that are autonomous, adaptive, and coordinative.
Clearly, single-agent environments are much simpler, because designers do not have to deal with such issues as cooperation, negotiation, and so on. However, the large-scale requirements of industry necessitate approaches that employ coordination and distribution. As such, FIPA and the OMG Agent Special Interest Group are focusing primarily on multiagent systems rather than single agents.

Many Agent Systems

As we discussed in the previous chapter, 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.

Useful systems can be built involving multiple agents that act on their own behalf, yet do not communicate or interact with each other. For example, a pricing system can use agents to determine the price of some commodity, or an insurance policy rate estimator agent can determine provide a quote for obtaining an insurance policy based on the insured’s needs. Such agents are able to provide the service for which they were programmed, yet have no need to interact with other agents. These kinds of systems are called multiple-agent systems (which are different than multiagent systems).

However for more complex systems, agents typically need to communicate and interact with other agents to accomplish their goal. For example in an agent-assisted service-oriented architecture, a Service Requestor agent seeks a service to satisfy a particular need, and a Service Provider agent offers to provide a particular service. Each agent has something the other needs.
To bring the two together, some form of communication and interaction is required. In some cases, a requestor agent may already have a reference to the provider agent it needs, so it negotiate directly with the provider agent. Service directories provide another means for requestors to locate provider that registers their services. Here, once an appropriate provider is located, its directory listing contains the provider’s address, and again the requestor can then negotiate directly with the provider agent. Also, a Service Broker agent can be added to the mix that enables requestors and providers to communicate and negotiate.

When many agents communicate and interact in a system, this is known as a multiagent system (MAS). This chapter discussed the ways in which agent communication and interaction can be accomplished in multiagent systems.

**The Semantics of Communication and Interaction**

If every designer developed a different means of communicating between agents, our agent systems would become an agent-based tower of Babel. Not only would the content and meaning of a communication likely be different, but the mechanism of communication could occur in a variety of ways. For small, local agent systems, the communication content and mechanism can be specialized to the particular group of agents. The communication content and mechanism would still need to be carefully defined, but would not require great elaboration.

However in sophisticated agent environments, agents can employ both simple and complex means of communicating with
other entities in their environment—including software, machines, and humans.

Furthermore, agents can employ interactive mechanisms that enable agent coordination, cooperation, persuasion, negotiation, competition, coalition, and so on. Without interaction, agents would not be able to reach joint agreements, and work together to accomplish a business function.

Probably the most important feature of agent communication and interaction protocols is that they be machine readable. This section discusses the basics required for machine-readable communication and interaction: agent communication languages (including message syntax and ontology) and agent interaction protocols.

Agent Communication Languages

\textit{Agent communication} is the act of agents conveying facts or information.

When two people want to communicate, they need to choose a common language and interchange medium—though even then misunderstandings can occur. Simple gestures and a handful of words can communicate a great deal. Yet, to produce societies and great building, we need to employ a ricer form of communication.

The same is true of agents. Self-contained systems that only need to communicate in a basic and non-standards manner. However, when agents interact widely and need to effectively communicate with agents in other systems and organizations, agents need a standard language—a language with a set of conventions that support agents in identifying, connecting with, and exchanging information with other agents. Agent communication languages (ACLs) enable agents to communicate in a clear
and unambiguous manner. By standardizing these ACLs, different parties can build their agents to interoperate both within and between applications — and organizations.

An example of a simple point-to-point communication between two agents is illustrated in Figure 5.1, where one agent is asking another for the current price of IBM stock. This message specifies: an ask performative, the sender's identity (Joe), the message content (PRICE IBM ?price), the address of the communication (stock-server), the name of the reply expected from the responding agent (IBM-stock), the language in which the content is specified (LRPROLOG), and the set of the agreed-upon terms, or ontology, that will be used in the content exchange (NYSE-TICKS). The responding agent replies with the requested stock price — along with its associated ACL parameters.

![Diagram of agent communication](image)

**FIPA ACL examples:**

```plaintext
(ask
  :sender joe
  :receiver stock-server
  :content (PRICE IBM ?price)
  :language LRPROLOG
  :ontology NYSE-TICKS
  :reply-with ibm-stock)

(reply
  :sender stock-server
  :receiver joe
  :content (PRICE IBM 140)
  :language LRPROLOG
  :ontology NYSE-TICKS
  :in-reply-to ibm-stock)
```

Figure 5.1 – Agent communication language (ACL) examples using FIPA ACL.

Currently, two primary standards for ACLs exist:

- **Knowledge Query and Manipulation Language (KQML)**— Network environments that support plug-and-
play processes are still quite rare. Most distributed systems are implemented with ad hoc interfaces between components. KQML is a language and set of conventions that support network programming specifically for knowledge-based systems and agents. It was developed by the ARPA-supported Knowledge Sharing Effort. (See www.cs.umbc.edu/kqml.)

- **Foundation for Intelligent Physical Agents (FIPA)** - FIPA has been working to develop and promote standardization in the area of agent interoperability since 1996. FIPA’s ACL is a high-level agent communication language that is based on speech acts and is perceived by many as an improvement on KQML. (See www.fipa.org.)

Every FIPA ACL message has the following structure:

<table>
<thead>
<tr>
<th>Element</th>
<th>Category of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>performative</td>
<td>type of communicative act</td>
</tr>
<tr>
<td>sender</td>
<td>identity of the sender</td>
</tr>
<tr>
<td>receiver</td>
<td>identity of the intended recipient(s)</td>
</tr>
<tr>
<td>reply-to</td>
<td>recipient of subsequent messages</td>
</tr>
<tr>
<td>content</td>
<td>content of message</td>
</tr>
<tr>
<td>language</td>
<td>language in which the content is expressed</td>
</tr>
<tr>
<td>encoding</td>
<td>actual representation (such as XML)</td>
</tr>
<tr>
<td>ontology</td>
<td>names of ontology used in message</td>
</tr>
<tr>
<td>protocol</td>
<td>interaction protocol employed</td>
</tr>
<tr>
<td>conversation-id</td>
<td>identity of the ongoing conversation</td>
</tr>
<tr>
<td>reply-with</td>
<td>expression identifying response</td>
</tr>
<tr>
<td>in-reply-to</td>
<td>expression referencing earlier response</td>
</tr>
<tr>
<td>reply-by</td>
<td>latest expected response date</td>
</tr>
</tbody>
</table>

The communicative acts that may be specified (such as ask and reply in Figure 5.1) are defined by the ACL. Some exam-
amples of other FIPA ACL and KQML communicative acts (called performatives) include:

- **achieve or request** - A wants B to perform a certain task.
- **advertise** - A registers as suitable for a particular request.
- **ask** - A requests information from B (ask-one or ask-all).
- **broker** - A wants B to find help to answer something.
- **failure** - A tells B that an action was attempted, but failed.
- **not understood** - A did not understand what B just did or said.
- **recommend** - A wants the name of an agent that supplies an answer.
- **recruit** - A wants B to request an agent to perform a given task.
- **reply** - A answers B.
- **sorry or refuse** - A cannot provide the requested information.
- **subscribe** - A wants any messages from B when they occur.
- **tell or inform** - A sends information.

Most of the ACL elements self explanatory. However, the language, ontology, and protocol parameters are particularly noteworthy.

### Language

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

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

Ontology

The syntax rules of a language are not enough to ensure clear communication; an agreed-upon set of terms is also required. Certainly, the syntax would define some terms, but there are also user-defined terms. For example, to ask for the number of clients that Fujitsu has could be expressed as: \texttt{COUNT FUJITSU client ?integer}. The syntax is well defined. But, if one agent uses the term "client" and the other knows only "customer," the two will not communicate effectively—even though both know that something is supposed to be tallied for Fujitsu. Agents can have different terms for the same concept and identical terms for different concepts. A common ontology, then, is required for representing the knowledge from various domains of discourse. The purpose of the ontology parameter in an ACL is to define the set of terms that will be used in an agent communication.

Although the need for terminology standards is not new, the dependence of agent communications on ontology makes such standards more critical than ever. Many organizations and consortia are now being set up to establish industry vocabularies. Common ontology representations using UML and XML schema are now being enriched with RDF, and OWL syntax.
Protocols

Agent interaction involves \( n \)-way communications among agents for some purpose.

\begin{quote}
*Agent interaction* is the action or influence of agents on each other.
\end{quote}

The purpose of the protocol parameter is to enable a way for agents to interact in a clear — and machine-readable — manner. The *protocol* parameter specifies the name of a particular pattern of communication. Such conversational patterns are called *agent interaction protocols* (or sometimes just *protocols*). Each protocol specifies a pattern of interaction that is formally defined and abstracted from any particular sequence of execution steps. As such, *agent interaction protocols* enable a way for agents to interact in a clear defined—and machine-readable—manner.

Figure 5.2 depicts a three simple interaction protocols using requester and provider agents. Figure 5.2(a) illustrates a simple query protocol, where the requestor places a query with a provider and the provider responds in a synchronous manner.

Figure 5.2(b) contains an example of the kind of extended query that is common for Google or database queries. A requester makes a query and the provider responds with the first block of a multi-block answer. The requester can then ask for the next sequential block in the query with the provider responding. This next/reply dialogue can go on as long as there are more blocks. This compound query requires a synchronous response, just as the simple query does. However, to maintain the progress of the query, the provider must maintain a state of the query until the interaction is completed. The simple synchronous query had no such requirement.
The simplified subscribe-reply protocol in Figure 5.2(c) depicts a requester agent sending a “subscribe” request to a publish/subscribe provider. Here, the provider notifies the requester with a “reply” message each time the provider learns of new information to which the requester has subscribed.

(a) Simple synchronous query protocol.

(b) Compound synchronous query where the provider must maintain state of query.

(c) Asynchronous replies sent as available to a subscribe request.

Figure 5.2 – Simple agent interaction protocols

The next chapter will provide more examples of agent interaction protocols.
Objects and Messages

If agents use ACL and objects use message passing to communicate, then it is still not quite clear how agents and objects can coexist. In fact, this difference can be viewed as a defining characteristic of how agents and objects are different. Objects are objects if they can communicate via object messaging and agents are agents if they can communicate via agent communication language.

This observation holds a possible key to how to simultaneously permit both, namely, some entities can do both so they are both objects and agents. This might permit a system to contain agents that use ACL to communicate with other agents, objects that use message passing to communicate with other objects, and some entities that can do both so they are simultaneously objects and agents.

```
agent --> || ACL | object || <-- object
```

There are several reasons why it is not simple to just model agents as objects directly.

Expressive limitations

When we model any software entity as an object, we focus on its interfaces and its state. In general, the single key characterization of an agent's behavior is that it accepts messages from other agents, performs some processing and (possibly) responds. This behavior, in terms of interface, may be represented in two forms. One would be to provide a method for every message that the agent is capable of processing. The other would be to provide a single method, `AcceptCommunicativeString` which would permit the agent to accept arbitrary messages. (A small set of multiple methods
messages. (A small set of multiple methods could be permitted, as well—particularly as separate channels for different roles. This can also be thought of as specialized ports for the agent.)

It is tempting to imagine providing an agent with a method for each message it can accept. However, we generally want every agent to be able to be able to receive any message, then decide on its own how to process that message. Certainly, the messages it knows how to and decides to process can change over time.

[On the other hand, the messages agents send and receive are not arbitrary - to be understood, they are described in a specific agent communication language with agreed upon semantics and structure. So it might actually make sense for objects that were also agents to be modeled as including the method AcceptACLString. At the same time, it is not satisfying to imagine a large application composed of agents implemented as objects each with one generic method just as it is not satisfying to imagine replacing object methods with a single agent method - too much is lost in this mapping in both directions.]

**Autonomy**

Method invocation also has the wrong basic connotations in terms of autonomy. In general, method invocation is done either on the caller's thread of control, or on the thread of a remote proxy for the caller, in the case of remote method invocation. Agents, on the other hand, aren't running as part of the caller, nor are they running with the caller's identity and privileges. In fact, quite the contrary, an agent runs on its own behalf, and with its own identity and privileges While one can imagine doing things with access control such that the additional rights and proper identity were associated with each invo-
cation, it adds steps that are outside the normal notions of method invocation.

Beyond identity and access control rights, we also run into the notion of autonomy. Unlike a normal object, where access control rights and visibility largely encapsulate notions of whether a method should run, an agent is an autonomous element of the software world. Thus, for every message that the agent receives, it may determine, based on its own goals and state, and the state of the ongoing conversation, whether to process the message and how to respond if it does. This implies that every method, will, in effect, begin with the idiom of making such a determination, something which, in general, is beyond what the majority of objects do at method invocation time. (Parameter checking, and comparison to internal state is fairly normal, but knowing the identity of one's caller, and basing decisions on that, is not)

Also, agents are active, not passive entities within a software system. Traditionally, an object is passive, with its methods being invoked under the control of another component of the system. An agent, in general, can react not merely to specific method invocations, but to observable events within the environment. Further, as part of the basic structure of multi-agent systems, it is natural for agents to be engaged in multiple, parallel interactions with other agents.

**Conversations, and long-term associations**

Another problematic issue for thinking of agent messages as method invocation is the question of long-term conversations and associations. Agents, unlike simple objects, may engage in multiple transactions concurrently, either through the use of multiple threads, or similar mechanisms. This is expressed naturally in a messaging environment, as each conversation can be
assigned a separate identity, and either a unique message destination or a unique identifier can be used to sort out the threads of discourse.

Similar mechanisms can be built out of objects, of course, but they would bypass, or be in addition to, normal method invocation. One could imagine producing multiple cloned "copies" of the agent object and passing one reference to each counterpart, so as to provide separate communication contexts for each conversation. But the very need to invent this kind of mechanism implies that the agent isn't acting as a simple object.

**Conclusion**

As mentioned earlier, agent agents can employ both simple and complex means of communicating with other entities in their environment—including software, machines, and humans. Additionally, agents can employ interactive mechanisms that enable agent coordination, cooperation, persuasion, negotiation, competition, coalition, and so on. Without interaction, agents would not be able to reach joint agreements, and work together to accomplish a business function.

The most important, however, of agent communication and interaction protocols is that they be *machine readable*. This chapter primarily dealt with the basic notions of communication. The next chapter continues by discussing ways that agents can interact using their communication abilities.
Mechanics of Agent Interaction

Many agent interaction mechanisms exist, such as point-to-point interaction, blackboards, agent mobility, and remote procedure calls. But, there are many more. This chapter describes the commonly used techniques employed by agents and agent systems.

Point-to-Point interaction

Agent communication can be achieved in two ways. The first way involves agents communicating directly with each other (logical connection). This approach provides flexibility and freedom but bypasses control and security. The other way involves agents communicating through an agent platform (Figure 6.1). This resolves control and security problems but requires logical communications to be physically resolved via the agent-based software. Using an agent platform is especially important when communicating among different organizations or the agents are mobile.
Most agent systems use the second option, because the agent platform enforces control and security at a system level. With this approach, the look and feel of agents directly addressing other agents can still exist, even though all communications are still processed through the agent platform—even communications to traditional, or legacy, systems.

In agent environments, messages should be schedulable, as well as event driven. They can be sent in synchronous or asynchronous modes.

Furthermore, agents can know the addresses of the other agents with which they wish to communicate, either because the agent has a record of them or because the agent uses a broker agent that has a record of them. However, a directory service should also be provided with the platform. Using this, and agent can find another using techniques such as white pages, yellow pages, or some other approach.

Lastly, the transportation mechanism must support unicast, multicast, and broadcast modes and such services as broadcast
behavior, nonrepudiation of messages, and logging. Although a commercialized message transportation does not yet exist for agent-based systems, it does exist in an object-oriented form. With some enhancements for the requirements of agent-based environments (directly or indirectly), the following technology could be used: CORBA, OMG Messaging Services, JAVA messaging service, RMI, DCOM, and Enterprise Java Beans Events.

**Blackboards**

Generally speaking, a blackboard is a data-structure that stores information used by multiple different systems. Like a real-world blackboard:
- Anyone can read information from the blackboard at any time.
- Theoretically, anyone can write down some facts too.

For agent systems, the blackboard is ultimately a way for different agents to communicate together. The advantage of the blackboard is that it provides modularity; the various agents don’t depend on each other, instead they just exchange the data via the blackboard. This reduces the level of coupling between the different behaviors, which makes the system easy to work with.

A static blackboard stores only certain types of information, and it cannot change at runtime (as opposed to a dynamic blackboard). The information within the predefined slots of blackboard can change at runtime of course, but no new facts can be added.

The structure of the blackboard is designed during development. The designers must go through an iterative process of figuring out what information is required to support intelligent selection of the behaviors and assets.
Using this approach has several advantages and disadvantages:

✔ Simple to create when the game isn’t overly complex, the behaviors are well specified.
✔ Very efficient in terms of performance and storage requirements.
✔ Quick to build, and simple to implement in all programming languages.

✗ Can’t store extra information without up-front design.
✗ Suffer from slow turn around times, as change to the knowledge structure requires a recompile.

So, in essence, it’s a trade-off between minimizing the time spent upfront on the blackboard, and the time required to create additional facts during development.

An agent system may communicate via one or more blackboards.

**Mobility and Remote Procedure Calls**

Stationary agents exist as a single process on one host computer; mobile agents can pick up and move their code to a new host where they resume executing. Mobile agents are able to change platforms and environments; stationary agents are not. From a conceptual standpoint, such mobile agents can also be regarded as itinerant, dynamic, nomadic, wandering, roaming, or migrant. The rationale for mobility is the improved performance that can be achieved by moving the agent closer to the services available on the new host.
Stationary agents must use the network to exchange information primarily using the remote procedure call (RPC) as illustrated in Figure 6.2(a). When a stationary agent requires processing on a different platform, it must employ the services of another agent. Here, a communication (or request) conveys the intention to invoke a specific operation (via an RPC). The operation is then executed and the results (or reply) returned to the requesting agent. Using stationary agents, then, has the following pros and cons:

- ✔ reduces the complexity required for mobility.
- ✔ encourages specialization within platforms.
- ✔ employs well-established protocols.
- ✔ supports closed-environment philosophy.

- ✗ results in performance problems in those situations requiring high volume or frequency.
- ✗ results in processing inefficiencies because having many specialized agents create more work than having a single mobile agent.
- ✗ reduces effectiveness when a connection is lost.
In contrast, mobile agents use the network to exchange information primarily by changing platforms and environments using the *remote programming* (RP) technique. When a mobile agent requires processing on a different platform, it physically relocates to the desired server as illustrated in Figure 6.2(b). This requires that all structural and behavioral properties of the agent must be transferred during migration and that any environmental differences be changed or accommodated. The big issues here are how much time is required to prepare for migration, how much data is actually transferred, and the performance of the transfer communication.

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

✔ reduce network load.
✔ reduce network-related delay.
✔ reduce resource usage of clients.
✔ enable distributed problem solving.
✔ support asynchronous, autonomous processing.
✔ promote reconfigurable or customized services.
✔ make active behavior scenarios conceivable.
✔ enhance decentralization options.

✗ involve a number of security issues such as the identification and authentication of agents, protection from destructive agents, as well as the assurance of the agent’s willingness and ability to pay.

✗ require transport/migration mechanisms be added to software environments—thus increasing their complexity.
have no industry standards for agent environments, migration approaches, or for measuring and billing resource consumption.

have not yet been used in an environment containing a large number of mobile agents.

Basic Interaction Patterns among Agents

As suggested in Chapter 2, interaction is a fundamental property of agents. To enable a group of interactive software agents to coordinate effectively and efficiently, several basic techniques can be used.

Central interaction via single agent

Figure 6.3 depicts a star-wheel connection of agents. Such a pattern is very common for satellite and single-server networks. Agents can use them as well for a simple way of communicating and coordinating with each other through a global coordinator agent. The approach is particularly useful when agents require a level of interoperability based on a central knowledge in order to secure task planning, resource allocation, execution control, monitoring, and intervention. Although depicted as agents, the entities in Figure 6.3 could be a combination of objects and agents.
Figure 6.3 – Centrally coordinated interactions.

The pros and cons include:

✔ Fast because all agents are connected via one or two steps.
✔ Easy to understand, intuitive.
✔ Overall coordination is possible.
✔ Traceable. All coordination processing is in one place.

✗ Overhead to support overall coordination.
✗ Agents may not communicate directly.
✗ Very fragile. All communications needs to go through the center.
✗ Not scaleable. Raising the number of satellite nodes creates a bottleneck.

Overall, the feasibility of central-coordination approach depends on the coordinator’s ability and the number of satellite nodes. For small star-wheel agent networks with competent central agents, the technique can work. Otherwise, one of the remaining techniques should be chosen.

**Top-down interaction via hierarchy**

A common extension to the star-wheel approach is the hierarchy in which each outlying agent in a star-wheel can become a “hub” agent that centrally controls its own star-wheel. For example in Figure 6.4, the top agent in the hierarchy provides a
central hub for two agents, each of which becomes the hub for the agents “below” them, and so on.

![Diagram](image)

Figure 6.4 – Interaction via hierarchy.

This technique has the following pros and cons:

- ✔ Ordered and traceable.
- ✔ Easy to understand, intuitive.
- ✔ Scaleable. No one node in the network has to do too much.

- ✗ Not flexible. Cannot be reconfigured easily.
- ✗ Fragile. If one parent node becomes unavailable, the network below is cut off from the rest.
- ✗ Slow. Users must navigate up and down through the hierarchy to reach most of the organization. Information exchanges involving so many steps will be slow.
- ✗ Prone to distortion. The greater the number of steps in traversing the hierarchy, the greater the likelihood of compounding errors.

As one of the most common approaches in IT systems, hierarchies have been used in the development of OO software, business process definitions, supply chain structures, and Internet networks. This top-down centralized control proved useful for organizing and scaling software efforts.

The entities in Figure 6.4 can be a combination of objects and agents. However, while agent-based systems can employ star-
wheel and hierarchical techniques, they can also use distributed approaches. In other words, agent systems can offer both central and distributed approaches, whereas conventional approaches tend to limit themselves to star-wheel or hierarchical interaction.

**Group coordination via hierarchy**

The group coordination illustrated in Figure 6.5 is almost identical to the top-down interaction via hierarchy, above. A group is just a collection of agents and may itself be an agent. Organizations, software components, and biological cells are some examples of groups.

![Figure 6.5 – Interaction via group hierarchy.](image)

Groups are encapsulated structures, so that to interact with a group, an external agent must communicate via an *interface agent*. An interface agent acts as the doorkeeper for the group, where only the doorkeeper can interact with the agents that are internal to the group. For example, for an external person to purchase a new computer from Apple, the transaction proceeds via an interface point in the Apple sales organization. For software components, you use the interface.
As indicated in Figure 6.5, groups can have more than one interface agent and groups themselves can contain groups. In this mode, group interactions essentially act like hierarchies. As such they have the same pros and cons as the Top-down interaction via hierarchy, above.

**Peer-to-Peer Interactions**

The classical “me client, you server” architecture is too restrictive for agents. As illustrated in Figure 6.6, the peer-to-peer metaphor is closer. Agents can provide resources at one moment and request resources at another. Furthermore, they can provide multiple services to multiple agents at the same time.

![Peer-to-Peer Metaphor](image)

Figure 6.6 – The peer-to-peer metaphor is closer for agents.

Interaction could simply involve two or more agents exchanging simple information such as ants do. Ants can communicate with other ants indirectly by leaving pheromones. As described in Chapter 2, if another ant detects a pheromone, it reacts by following the pheromones to food. People, on the other hand, communicate in many ways: through voice, writing, gestures, or just their general behavior.

**Individual interaction over a network**

Many examples of hierarchy can be observed in nature: the food chain, social groups, pecking orders, and reductionist views of decomposition. However, the predominant pattern in nature is a distributed—yet, interacting—network of entities: subatomic particles in an atom, electrical particles in a storm cloud creating lightening, flocking birds, ants searching for
food, the stock market, and so on. This distributed approach in nature is very useful because it enables a degree of synergy that would not be possible otherwise. The interaction among the agents, then, is vital for certain kinds of IT systems, such as planning and scheduling, resource routing, logistics, and order-processing systems.

![Figure 6.7 – Individual interaction via a network using messages and dialogues.](image)

Distributed networks may or may not require any kind of controlling authority. For example, ant colonies have no “controller ants” that manage food gathering activities. In contrast, crowd-control personnel at football games are frequently useful. For agent-based systems, coordinators can be embedded as peer agents within the system if needed (Figure 6.7).

The pros and cons of the network interaction approach include:

✔ Greater flexibility for each agent.
✔ No overhead caused by central points of control.

✗ Information exchange is limited.
✗ Information completeness is costly.
✗ Overall coordination is difficult.
Individual interaction using groups

Group interaction (mention earlier in Section 3.3) requires a certain degree of hierarchy because they must be accessed via interface agents. However, the agents that are part of a group can interact in a network form. If the group has a sub-group (like that depicted in Figure 6.8) the subgroup, then, has its own encapsulated boundaries and interface agents. Therefore, the agents that are part of a subgroup can interact in a network form. Only interface agents are permitting to cross boundaries to the next higher aggregation level.

Figure 6.8 – Individual interaction within groups, but not across groups.

By their nature, group interaction is hybrid. It certainly requires a hierarchical approach for inter-group communication. However, within groups any of the above forms can be used, as well. In other words, the internal group could be constrained to be centrally coordinated, hierarchical, or in this case network.

The pros and cons are a mixture of the technique used and presented above.
Individual interaction using a blackboard

In the previous interaction techniques, messages exist only transiently, as they travel over the network. However, interacting agents may record messages so that they may persist for a period after they are sent. This term blackboard refers to a common work area where one or more agents can post and retrieve shared data. Using a network approach would be intensely interactive and time consuming if all the agents needed to be properly informed. In the blackboard approach, shared information can be immediately available for each agent (Figure 6.9). Blackboards can be used for many kinds of situations. They can provide a short-term data persistence between two agents; they can also provide a long-term common work area for many agents simultaneously.

![Diagram of blackboard interaction]

Figure 6.9 – Individual interaction via blackboard.

In the ant example, the environment can be considered a kind of blackboard, where pheromones become the information that is posted and queried. Whereas, in the flocking-bird example, no common work area was employed—network interaction was all that was necessary.
Multiple blackboards are possible and can be implemented using a variety of techniques, including shared memory and database management systems. Additionally, dispatchers can be added to inform agents of changes. The pros and cons include:

✔ Each agent has greater flexibility.
✔ Information exchange is facilitated by a central source.
✔ Information completeness is possible.
✔ Each agent determines what is posted or retrieved.
✔ No overhead to support overall coordination.

✗ Overall coordination is difficult.
✗ Agents may not communicate directly.
✗ Doesn’t scale well: grows exponentially.

Conclusion

Since each of the interaction approaches described above has their pros and cons, clearly no one approach will suit every need. Choosing a mixture of approaches is a common solution. For example, in most large human organizations, the structure typically supports a management hierarchy, where a manager’s employees often organize in networks.

For multiagent system designers, the situation is similar. In Figure 6.10, the agent configuration contains elements of various techniques.
Figure 6.10 – Interaction using a variety of techniques.

Agent interaction, therefore, is not limited in its techniques; many others are possible. For example, desktop agents can watch how a person interacts with a system and try to remember common usage patterns for the future. Behavior, then, is being communicated to the agent—either directly or indirectly. Eavesdropping agents listen in on the interactions of other agents without their knowledge or consent. Agent interaction is defined as the action or influence of agents on each other. Action and influence can come in many forms, through electronic messages, signaling devices, voice, visual observation, pheromone detection, and so on. They can be direct (messages between agents) or indirect (non-message interaction); they can be centralized or decentralized.
Seven

Organization and Coordination of Work

Introduction

Human organizations exist primarily to coordinate the actions of many individuals for some purpose. That purpose could be to create such structures as profitable business units, charitable organizations, ballet companies, or Little Leagues. Using human organizations as an analogy, systems involving many agents whose coordination benefits from the same patterns.

Coordination – the combination of agents or functions to act together harmoniously. [OED]

Common agent applications that involve coordination include supply chains, scheduling, vehicle planning, problem solving, contract negotiation, and product design. Without some degree of coordination, such systems could not be possible—in either human or agent-based systems.

Furthermore, the analogy requires that we consider a heterogeneous population of agents. Human organizations are not constructed with a population of identical individuals doing the
same thing. Instead, we diversify, delegate, negotiate, manage, cooperate, compete, and so on. The same approach needs to be employed in agent-based systems. Careful consideration in the design and structure of these systems will increase the likelihood that agents will be coherent in their behavior. While this limits and controls agent spontaneity, it still preserves agent-level flexibility.

Organizing Agents

One of the most difficult challenges for automated systems is designing and constructing large-scale systems. Here, life-as-an-analogy brings with it many useful concepts, including some excellent examples on how to scale up. In the physical systems leading up to life, for example, subatomic particles form atoms, and atoms cluster to become molecules in solid, liquid, and gaseous form. Continuing up this hierarchy, molecules can be organized to form organelles and cells, cells can aggregate to form organisms, and so on. In other words, living systems and their components emerge in a hierarchy of interlocking mechanisms.

If such an emergent approach works in nature, it can also provide a good model for developing agents and agent-based systems. For example, Figure 7.1 depicts the classical biologic cell structure known as a eukaryote. Notice that the eukaryotic cell consists of many interacting component. The mitochondria, for example, is a cell in its own right. It is also an important energy source for the cell that contains it. Mitochondria, then, provide a good example of an agent inside of an agent, where the whole depends upon the parts for its functioning.
To live and maintain life, eukaryotic cells do not consist of just any matter inside its membrane. To function properly, the eukaryote needs certain roles to be played by the internal components. For example, every eukaryote needs a component that provides energy. Mitochondria play that role. Maintaining the cell's shape and size is enabled by the cytoskeleton, and protein transport is provided by the endoplasmic reticulum.

Furthermore depending on the kind of eukaryote, chloroplast can play the role that converts solar energy to organic compounds and oxygen, flagella and cilia provide a means of locomotion; multi-rolled heliozoa with their many stiff, microtubule-supported projections are used for capturing food, sensation, movement, and attachment. In short, eukaryotes contain a number of components—all which play important roles in defining the cell's organization and existence. Some roles can be played by multiple components; some components can play multiple roles.
In reality, Nature has no construct called role—it is a human concept. We use the notion because it useful to group certain capabilities that can be thought of as a unit, independently of any entity that actually ends up playing the role. For example, the roles of customer or supplier exist independently of any entity filling them, because the set of behavior involved in being a customer or a supplier is important to the success of a business. Roles, then, provide an intuitive mechanism for modularity, flexibility, reusability, modifiability, and extensibility.

Agents can not only play roles, they can be grouped to form variable structures. These groups can be colonial in nature (such as sponges and coral reefs) or multi-cellular animals (such as dinosaurs and humans). Agents that adhere to one another can behave in a unified manner and still maintain their own autonomy. Automated agents might choose to group for various reasons, such as protection, resources, or improvement. In other words, the agents might decide that they are better off together than apart. Furthermore, the agents may adapt to serve the group as a whole.

Roles

Agents in a multiagent system may have been designed and implemented by different parties, each with different assumptions and goals. Although these agents may not share the same goals, they may need to interact to achieve the preferred results. In more constrained applications, all decisions can be hardwired into the agent at design time. However, the more adaptive a system is, the more its agents must be capable of coordinating dynamically at runtime. This latter aspect is what makes the agent approach more effective and efficient than conventional software development.
Using roles

In the ant colony example (Chapter 2), all the agents were of the same type. They were uniform, because each had the same goals and the same programming. Even the ant-swarming example was constructed with uniform agents. So, building an effective agent-based system is possible using a uniform agent architecture (Figure 7.2).

![Uniform agent architecture](image)

Figure 7.2 – Uniform agent architecture

However, one agent could be constructed that does everything (as mentioned in Chapter 1). Unfortunately, these “fat” agents represent a bottleneck for speed, reliability, maintenance, and so on. (Alas, there are no omnipotent agents.) Dividing functionality among many agents provides modularity, flexibility, reusability, modifiability, and extensibility. Furthermore, specialized knowledge is not often available from a single agent. Knowledge that is spread over various agents can be integrated for a more complete view when needed (as we will see in the coming sections titled Task Sharing and Result Sharing). In other words, a multiagent system is typically designed using various roles.

A role is a defined set of behavioral capabilities and other features that an entity can possess.

Facilitator roles

One of the most common roles used in agent systems is the facilitator roles. A facilitator functions as a middleman, by dis-
tributing and coordinating resources among a number of client agents (Figure 7.3).

![Facilitated architecture](image)

Figure 7.3 – Facilitated architecture

The facilitator can be used for most applications and possesses a number of services and domain-independent strategies for coordinating resources. Figure 7.4 depicts five facilitator-related roles commonly used along with their interaction protocols: the subscriber, broker, recruiter, matchmaker, and broadcaster roles.

![Facilitation roles and their interaction protocols](image)

Figure 7.4 – Facilitation roles and their interaction protocols.
A subscribe agent receives a subscribe request communication from a requestor who wishes to receive messages on a particular topic. Anytime a provider agent sends ("tells") a communication to the facilitator that fits the subscriber's topic, the subscribe facilitator passes on the communication to the requester.

A broker agent receives both broker communications from requesters and advertise communications from providers. When the facilitator finds a match, it notifies the requestor directly. The recruiter is similar to the broker, except that it notifies the provider who contacts the requestor directly.

The matchmaker works in much the same way as the broker and recruiter by putting together, or matching, agents that can work together. Once this match is made, the requester and provider agents can communicate with each other directly without depending on the matchmaker.

In the broadcaster protocol in Figure 7.4(c), an agent requests the facilitator to broadcast a message to a number of agents.

Facilitator agents can apply whatever "intelligence" they possess (e.g., neural networks, learning, planning, rules) to help them make key decisions. Their source of provider agents can come from their direct knowledge or via agent directories that use extended forms of UDDI.

Roles, in general

Roles identify activities and services necessary to achieve objectives that enable the coordination of an agent-based system. Just as roles are appear in nature—particularly in human-based organizations (Figure 7.5)—employing roles for the “society” of agents is also useful.
For example, a manager agent could coordinate a series of activities in a complex business process management (BPM) system. The manager agents could themselves take their strategic directions and goals from a “higher” agent—with a hierarchy all the way up to a conceptual top-level agent, much like a CEO. Librarian agents could both maintain “libraries” of information and provide a facilitation role for agents requesting information.

Specialized roles can be used for two different purposes. First, agents can be defined that correspond to actual roles in a human organization. Here, the agent enables the various role-based services that are permitted and required for a given organizational role. When such role-based interfaces need to be adaptive and autonomous, agents provide a good implementation mechanism.

The second usage is to provide a technique for defining, organizing, and reusing common groups of structure behavior. In object orientation, the class provided this function. Since agents are perceived more dynamically—even to the point of developing agent social systems within IT—the notion of role is more familiar to the humans who analyze and design such systems. In other words, roles are a familiar metaphor for humans to organize the active entities in their world. Roles and classes can
be used for agents—where the agents can be software, hardware, or human.

Groups

Imagine unleashing thousands or millions of agents in a system. Would it be reasonable to expect all these individuals to coordinate in some manner? In ant colonies and bird flocks, this seems to work very effectively. For building cities, transportation infrastructures, and large companies, this will not work. Instead, agents and roles need to be grouped in some manner. From organization theory, we know that organization structures can be defined that group employees and employee roles in ways that can—to put it in IT terms—maximize cohesion and minimize coupling. Groups can be seen as active components or modules that have their own internal processing and can interact with other groups. In human society groups can be defined for alliances, teams, coalitions, conventions, markets, and organization units.

Furthermore, these groups themselves can become agents. For example, IBM, HP, the United Nations, the Yankees baseball team, and the US Supreme Court are all groups that are recognized agents in their own right—which also consist of agents that are groups, and so on down to “atomic” level agents. In this way, the Requester or Provider in Figure 7.7 could be an atomic agent or a legally recognized organization like DaimlerChrysler that is requesting bids from Provider companies. Yes, a person in each of these organizations might actually process the bid, but such knowledge is unnecessary. In this way, groups as social structures reduce the danger of combinatorial explosion of agent interaction. Instead, they provide a structured scheme for focusing and enhancing the coordina-
tion of agent activities, making richer, more complex structures possible—while hiding unnecessary details. In short, they enable scalability technique for large populations to produce significant deliverables. The case study in Chapter 6 will describe how this technique can be used in a large manufacturer—that literally is designed using structures called *manufacturing cells*.

**Figure 7.6 – Agent grouping approaches**

Figure 7.6 depicts several grouping options. [1] The first option, of course, is that there is no group. The second option is that two or more agents can group as a single unit. The containment boundary indicates that the entire construct can be treated as an agent in its own right. In option 1, the agent and the boundary are the same. In option 2, the boundary can be treated as an agent separate from the two agents it contains. The term *boundary* is chosen because it also provides an interface barrier much like that of a cell membrane. Agents on the mem-
brane (as in option 2) are still accessible from the outside as individuals while benefiting from the proximity to chosen neighbors. In contrast, agents that are contained completely within the boundary, as illustrated in option 3, are encapsulated. Only those agents on or immediately within a boundary may communicate with encapsulated agents. In a complex group configuration (option 4), the nesting continues.

**Agreement**

**Reaching agreements**

In human society, individuals and organizations often have differing viewpoints and interests, and we use various techniques for reaching agreements. If we use software agents to represent us in marketplaces, web services, and other automated situations, some of these same techniques can be used. For example, a manufacturer may be running low on a particular part and send out a notification to its suppliers. The suppliers send in their bids, and the manufacturer can then choose among the suppliers. With an agent-based software system in place, this kind of activity can occur among software agents, instead of human ones.

Software and hardware agents do not need to represent individuals or organizations to use this approach, they can represent themselves or other entities, as well. The Painting Trucks at General Motors’ example in Chapter 2 or the manufacturing cells described in the case study of Chapter 8 are both examples of this. Service-oriented architectures could include software agents that select service providers based on auction-related techniques. Techniques such as auction, negotiation, and delib-
Auctions

Prior to the Internet and Web, auctions were relatively rare in everyday life. Now, individuals and business can participate in auctions at a very low cost with one of the best examples being eBay. Besides being inexpensive, auctions are popular because they have simple and well-defined interaction protocols. This also means that auctions are easy to automate, making them a good technique for automated agents.

Auctions take place between an auctioneer agent and a group of bidder agents. The auctioneer’s purpose is to allocate an auctioned item to one of the bidders. The auctioneer tries to maximize the price, while the bidder tries to minimize it. The actual auction process can have several variations.

- **English auction** – This is the most common form where the auctioneer might start out with a minimum price. Bids are then invited from agents, where each must offer a higher bid than the previous bidder. Eventually, the auction item is awarded to the agent with the highest bid.

- **Dutch auction** – The Dutch flower auction was the inspiration for this form. Here, the auctioneer starts with a high price and then lowers it until a bidder accepts the price. The auction item is awarded to the agent with the first bid.

- **Single-round auction** – Each agent bids only once in this sealed-bid auction. The agent with the highest bid is the winner. This option has two variations: one where the winner pays the amount they bid, and the Vickrey variation where the highest bidder actually pays the second-highest bid amount for the auctioned item. (The Vickrey auction is un-
usual and unintuitive but is found to encourage bidding the true valuation.)

Other auctions exist, as well. For example, the reputation-based single-round auction may not award the auctioned item to the highest bidder, if the second- or third-highest bidder has a close bid on a better reputation. Another variant is the eBay auction that uses an English auction but can offer an additional approach that bypasses further bidding to the first bidder that offers to pay a predetermined “buy it now” price.

![Agent interaction protocol for contract negotiation.](image)

**Figure 7.7** – Agent interaction protocol for contract negotiation.

**Negotiation**

Negotiation is a process by which two or more parties can reach an agreement, each attempting to achieve different goals. A common example is the simplified contract negotiation (illustrated in Figure 7.7), where a requester agent sends out invita-
tions to bid on a job contract. If a provider agent wishes to participate in the negotiation, it can respond with a bid. Since many provider agents might respond, the requester agent has to decide which provider should be awarded the contract. Once the contract has been sent to the provider, the provider can choose to confirm. If the provider declines, the requester must choose another provider. This pattern could support various negotiation scenarios, such as ordering supplies, requesting equipment, or obtaining human resources.

Common variants to the contract interaction protocol can include:

- Bidding can invite a single bid or multiple bids. Awarding a bidder in multiple bidding negotiations brings in one of those auction variations mentioned above.
- Contracts can be awarded to zero, one, or many providers.
- Bidding processes can be interactive, open, secretive, etc.
- The contract process can have its own sub-negotiation and counter offers.
- Contracts can specify penalties—and either party in the contract may choose to violate their contracts by weighing costs and benefits.
- Intermediate negotiation brokers can be used, as illustrated in Figure 7.8. This offers centralization of a specialized agent, when there is a way or need to do so.
Chapter Seven

Deliberation

Deliberation involves deciding what a group of agents wants to achieve. The aim is to jointly arrive at a decision or form a plan of action and may or may not be initiated from a conflict. Decisions could be made by a “judge” agent that takes into account arguments presented by various agents. This could be performed by software mechanisms such as rules engines and neural networks. Democratic means is another technique, where a simple or some form of “super” majority vote determines the outcome. Majority can also be determined by other factors that include weighting of a vote. For example, the votes of some agents might have a higher valuation than others, due to a number of factors such as importance, credibility, and so on. An automated delivery system could determine the next location in a delivery through agent voting—or though some sort of auction.

Figure 7.8 – A negotiation exercise between two equal sets of parties.
Others ways of reaching agreement

Auction, negotiation, and deliberation are the primary techniques currently employed for software agents to reach agreement. Researchers are working on others that are inspired by nature such as argumentation, intimidation, bribing, and brute force. While these approaches may not enter into commercial systems in the near future, agent system designers need to consider both existing and novel techniques.

Beyond Coordination

This chapter has discussed coordination as a property of agent systems performing activities in a shared environment. As designers of agent-based systems, we want the combination of agents or functions to act together harmoniously. The degree of coordination, then, is the extent to which the agents: avoid extraneous activity, reduce contention, avoid deadlocks and livelocks, maintain applicable security and quality of service.

To further aid in accomplishing this, extended forms of coordination are possible [2]:

- **Cooperation** - the process whereby agents coordinate their actions to achieve a common purpose. This suggests: non-antagonistic agents, sociability, and goal sharing. Cooperation is coordination, but not necessarily the other way around.

- **Competition/contention** - The act of agents that coordinates their actions where the success of one agent implies the failure of others. Such agents can be antagonistic agents, yet still perform activities in a shared environment.

- **Coherence** – The behavior of agents that align with system-level goals.
Policies

In our every day lives, our actions are subject to norms and social laws which provide a means for regulating social behavior. A policy is a constraint on the behavior of agents as they perform actions or access resources. In short, they are “business rules” for agents.

Various kinds of policies exist. Some relate to accessing resources in particular ways, others relate more generally to the allowable actions agents may perform, both as provider agents and as requester agents. For IT, they are particularly important for security and quality of service. Security is fundamentally about constraints on the behavior of an action and on accessing resources. Similarly, quality of service is also about constraints on service.

Two types of policy are commonly identified: permissions and obligations.

A permission is a kind of policy that defines the allowed actions and states of an agent and/or resource. An example of a permission policy would be: every Order must have at least one line item and may not have a negative total charge. Such a rule is common in database systems. It also applies to agents because it constrains how an order-placing service must behave.

An obligation requires an agent to perform particular actions or be in particular states. When an agent has an obligation to perform some action, it is required to do so. For instance, when a customer agent and supplier agent have made a contract, they have mutual obligations. The supplier has an obligation to deliver a specific commodity on an agreed-upon date; the customer has an obligation to pay for the commodity if delivered as agreed. Not all obligations relate to actions. For example, an agent providing a service may have an obligation to
maintain a certain state of readiness. Quality-of-service policies are often expressed in terms of obligations.

Closely associated with policies are the mechanisms for establishing policies and enforcing them. Permission-related policies can be enforced proactively. Obligations, by their nature, cannot be proactively enforced though they can be monitored. Agents that violate their obligations can be subject to retribution mechanisms.

Problem Solving and Distributed Problem Solving

As seen above, roles and interaction protocols provide basic building blocks that enable agents to coordinate. By themselves, they can be used to solve simple problems. When an application becomes more complex, however, other techniques can be employed on top of these basic building blocks. For example, one Order Processing agent can be built to handle an entire order-processing activity by itself. In a more effective approach for complex orders, the Order Processing agents would delegate some of the Order Processing activity to other roles. Or in another example, one agent could be designed to handle a massive database search or solve a large problem. A more efficient approach might decompose the problem into smaller parts and have multiple agents find solutions concurrently.
When using agents, this divide-and-conquer approach to solving difficult problems is often appropriate (Figure 7.9). Conceptually, problem solving can be viewed as a three-stage activity:

1. **Division of a problem into subproblems** - In this step, a problem is decomposed into smaller subproblems. Determination and assignment of subproblems can be done by a manager agent or by each agent individually.

2. **Solution of the subproblems** - Here, each subproblem is individually solved. One or more agents can produce solutions to the same (and multiple) problem—which can conflict. The agents might communicate to solve problems or resolve solutions.

3. **Combination of the subsolutions** – Solutions to the individual subproblems are integrated into an overall solution. This integration can be performed by each agent or a consolidation agent.
Grid applications or large database searches are obvious applications for such a problem-solving approach. Business process management (BPM) is another application for this approach. The order processing activity in Figure 7.10 provides a good example of this. One activity (or component) could be constructed to implement all actions within this process. However, other receivables and payables services might already exist in different locations—and most probably different organizations.

In reality, an Order agent would instead delegate some or all of the actions to other agents playing other roles. This “delegation” could be hard wired or requested at runtime using a service-oriented approach (SOA). When SOA is used, the Order agent would become the service requester and the supporting agents would become the service providers. Finding appropriate service providers could be accomplished by using directory services, service broker agents, auction interaction protocols, and so on. If the service providers are few and well known (or even hard wired), an agent-based approach may not be neces-
sary. However, for complex environment requiring adaptive solutions, the agent-based approach is the only viable solution.

**Conclusion**

Imagine unleashing thousands or millions of agents in a system. Would it be reasonable to expect all these individuals to coordinate in some manner? In ant colonies and bird flocks, this seems to work very effectively. For building cities, transportation infrastructures, and large companies, this will not work. Instead, agents and roles need to be organized into groups in some manner. From organization theory, we know that organizational structures can be defined to include roles that have processing capabilities that can—to put it in IT terms—maximize cohesion and minimize coupling. Communities can be seen as active components or modules that have their own internal processing and can interact with other communities. In human society, communities can be defined for alliances, teams, coalitions, conventions, markets, and organizational units.

Furthermore, these communities *themselves* can become agents. For example, IBM, HP, the United Nations, the Yankees baseball team, and the US Supreme Court are all communities that are recognized agents in their own right—which also consist of agents that are communities, and so on down to “atomic” level agents. In this way, a Consumer or Provider could be an atomic agent or a legally recognized organization like DaimlerChrysler that is requesting bids from Provider companies. A person in each of these organizations might actually process the bid, but such knowledge is unnecessary. In this way, communities as social structures reduce the danger of combinatorial explosion of agent interaction. Instead, they pro-
vide a structured scheme for focusing and enhancing the coordination of agent activities, making richer, more complex structures possible—while hiding unnecessary details. In short, they enable scalability technique for large populations to produce significant deliverables.

References


Agent-based Manufacturing: A Case Study

Introduction

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

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

Another problem with traditional schedulers is that they try to anticipate and plan for every possible change that may occur. Unfortunately, the range of scenarios and the possible combinations of parameters are infinite, because manufacturing is so
complex. Even if it were possible to pre-code all possible scenarios, the cost of considering and programming all possible combinations is prohibitive. An unanticipated scenario could cause the system to fail.

In short, traditional manufacturing facilities have shortcomings that affect their ability to compete in today’s constantly changing marketplace.

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

These shortcomings cause problems such as reduced productivity, increased costs, and missed market opportunities. To remain competitive in today’s marketplace, manufacturing must change its approach. In response, a major automotive company is building an agent-based manufacturing system under the direction of Dr. David Greenstein. [1] Here, the agents not only adapt to their environment but can also evolve by learning from the environment. Such an approach prepares manufacturing enterprises for the increasingly complex marketplace and enables them to respond rapidly to change.
This section provides a case study describing how agent-based technology can be applied in business applications. While the example is for an automotive company, the general concepts are applicable to many other industries.

**An Agent-Based Solution**

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

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

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

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

As David Greenstein states, "For a manufacturer to succeed in today’s competitive world, it must have the optimal mix of people, equipment, and knowledge to make the product. The AMIS architecture provides the flexibility and agility in a software system, which enables a manufacturer to monitor, evaluate, and adjust the mix of resources, people, and tools required as market demands change." [1]

The AMIS Agents in more Detail

Cell Agents

In living systems, a cell is a self-contained unit that has its own structure and behavior. It consists of other self-contained structures that interact to support the cell. How well the cell and its components work together determines whether the cell lives or dies. In a manufacturing system, each cell agent is a business unit representing a collection of physical resources, including machines, tools, and people. The cell operates as a self-contained business unit and only continues to exist if it meets its profit and production goals and its responsibilities. The cell also controls its own size—it changes the mix and number of
resources over time to maintain its profitability and competitiveness in the marketplace. The architecture of a cell is summarized in Figure 8.1.

![Figure 8.1 – Typical cell agent architecture for a manufacturing system.](image)

Common Function Agents

The *common function agents* interact with each other and with the resource agents. They provide the complete set of business functionality required to operate the cell as an independent business unit. Each common function agent is responsible for a different business or manufacturing function within the cell. Some of these agents contain information about the resources within the cell, such as the capabilities of the resources. Other agents provide interfaces to the people working in the cell, such as process planners and machine operators.
For example, the maintenance manager schedules and directs maintenance activities whether they are scheduled, opportunistic, or reactive. It also keeps track of the maintenance history.

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

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

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

Negotiator agents (at the top of the cell architecture diagram in Fig. 1) communicate with the outside world on behalf of the cell. The negotiator provides an interface between the cell and the outside world. It routes messages received from the outside world to the appropriate common function agent. When preparing quotes for new jobs, the negotiator assembles the quote information provided by the other agents and summarizes the final quote information for the customer. Similarly, when the cell receives quotes from subcontractors, the negotiator works with other agents to select the winning quote.

Resource Agent

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

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

Resources keep track of their assigned jobs by maintaining a prioritized list of jobs that the resource wins. Each job defines its job type, the earliest start time for the job, the expected job duration, the latest finish time, and the estimated cost.

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

Job Agent

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

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

Broker Agent

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

broker forwards the request to each automobile-producing cell that has registered with the broker.

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

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

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

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

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

**Adaptation in Natural Systems**

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

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

**Learning**

Learning is change that occurs during the lifetime of an agent and can take many forms. The most common techniques enable rules and decisions to be weighted based on positive (or
negative) reinforcement. For example, in a basic bidding system, a bid could be selected simply on the basis of bid price. However, other considerations might also be appropriate, such as the bidder's ability to deliver its goods in the quantity, quality, and time frame requested. Over time, a purchasing agent can learn to choose from reliable vendor 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.

**Evolution**

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

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

**Best Interest**

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

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

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

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

**Seven-Step Negotiation Process**

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

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

1. **Request for quote**—The Request for Quote (RFQ) is the first step in the AMIS bidding process. A customer (or customer agent) creates an RFQ that specifies the desired products or services—along with a response date—and sends it to a broker agent. The broker acts as a liaison, forwarding the RFQ to each cell that has “subscribed” to provide the requested product.

2. **Receive quotes**—Each cell agent determines its ability to complete the job according to the customer's RFQ specifications. If a cell is able to meet the customer’s requirements, it creates a quote for the job. Quotes contain estimated information on the delivery date, price, quality, and special characteristics related to completing the job. Cells return their quotes to the broker, which holds the quotes until the RFQ response date has been reached and then returns all the quotes to the customer.

3. **Select winner**—When time has expired for the cells to submit quotes, the customer (or customer agent) begins the selection process. The customer selects the winner from the submitted quotes by finding the most desirable mix of cost, time, quality, and special characteristics, based on its requirements for the job. The customer sends an award notification to the cell with the best quote.
4. **Winner confirms**—The winning cell accepts or rejects the job depending on whether it still has the capacity to do the job. The cell might reject the job if it has accepted other jobs between the time it prepared the quote and received the award. If the winner rejects the job, the customer offers it to the cell with the next best quote. This continues until a cell accepts the job. After the winner accepts the job, the other cells that submitted quotes are informed of the decision. At that time, losing cells are able to access data on the quotes, which help them evaluate why they lost the job—and perhaps learn and modify their behavior for future quotes.

5. **Issue purchase order**—After a cell has confirmed the customer order, the customer authorizes the cell to begin production by issuing a purchase order to the cell.

6. **Generate product**—The cell completes the work on the product, delivers the product to the customer, and sends an invoice to the customer.

7. **Make payment**—The customer ends the process by paying the cell for the work done.

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

**Summary for Agent-Based Manufacturing Case Study**

**Distributed Organizational Control**

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

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

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

**Capacity Management**

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

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

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

**Market-Driven Economy**

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

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

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

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

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

The author would like to acknowledge Dr. David Greenstein (dgreenst@tir.com) for his providing a comprehensive description of his agent-based AMIS case study.

References

Introduction

Complexity is all around us. It is part of our life; it is the nature of life. Complexity is caused by the collective behavior of many basic interacting agents. Such agents can produce everyday phenomenon such as ant colonies, traffic jams, stock markets, desert ecosystems, and supply chain systems. However, complex systems need not be complicated as the ant colony simulation illustrated in Figure 2.1 demonstrates. A system is complicated if it can be completely described in terms of its huge number of components. On the other hand, a system is complex if the system cannot be fully understood by analyzing its components; here, the interaction among the components also must be considered.

Often, complex systems cannot be fully comprehended, because we do not precisely understand the component (or agent) interaction. For example, we might be able to identify the components involved in the New York Stock Exchange, yet we cannot accurately predict when we will have a bull market or a bear market, or when a market bubble will burst. In such situa-
tions, our knowledge of the interaction among the various components is not well understood.

**Complex Systems**

The science of complex systems was first made popular by the Santa Fe Institute (SFI). This group of complex systems’ theorists and practitioners offers the following definition:

"Complexity refers to the condition of the universe which is integrated and yet too rich and varied for us to understand in simple common … ways. We can understand many parts of the universe in these ways, but the larger more intricately related phenomena can only be understood by principles and patterns—not in detail."

This concept is important to us because business and IT systems are becoming more complex. For almost a century our manufacturing model was reasonably stable. Rules for productivity, market dominance, and company success were well understood. However, the Information Age has turned previous "knowns" on their head. Today, no one can predict what or how rapidly new technologies will be developed. No one can accurately predict how supply chains will be affected or how customers will respond. A good business plan is only a guess.

Even small-to-medium-sized businesses are not insulated from this effect. Satellite communications, the Internet, and air transportation commonly provide ways of rapidly moving corporate resources from place to place. All of us are now connected through a global market of online customers and suppliers, supply chain partnerships, and international franchise competition. As Susan Kelly and Mary Allison [1] suggest, busi-
nesses that don't understand the nature of complex system thinking and take advantage of it "will be at the mercy of an increasing number of sudden and unexpected shifts in the marketplace. As uncertainty grows exponentially with today's high rate of technological change and the fallout from it, so does the pressure of global markets." Many executives try to respond to this with yesterday's mindset and linear cause-and-effect thinking which will often intensify an already downward spiral. Despite well-conceived plans and well-intended actions, a company that operates without complex-systems thinking will find itself unable to respond to the ever-increasing complexity of the business world.

**Primary Issues**

Common to any discussion of complex systems are several fundamental ideas.

- First and most basic among these are *agents*. In complex systems, these are the autonomous entities that interact to carry out their particular tasks.
- Another fundamental concept is the adaptive character of these agents, reacting to their environment and possibly changing their behavior based on what is learned.
- Complex systems are also characterized by *emergence*. Emergence is a coherent pattern that arises out of interactions among agents. For example, the process of an entire ant colony being fed was not programmed. It emerged from some very simple rule programmed into each ant. In other words, emergence is a by-product of individuals—not a choreographed result. Emergent results can be good as well as bad and, therefore, must be considered when developing agent-based systems.
Agents, Complexity, and Emergence

- Successful emergent systems often exist between order and chaos. For any organism or organization to be in a continuous state of order or chaos would be fatal. However, dancing between order and chaos makes possible fitness and continued existence.

- Lastly, nature can teach us a lot about designing complex systems, since it has been solving large combinatorial problems for billions of years. It makes sense, then, for us to consider notions such as parasitism, symbiosis, reproduction, genetics, mitosis, and survival-of-the fittest when developing our agent-based systems. For example, British Telecom is using the model of ants and pheromones in its call-routing network where successful calls leave an equivalent of pheromones to guide future calls.

Agents and Emergence

Agents can work as non-interactive individuals or as a collective. When agents work as individuals with little or no interaction, the result is just that: agents simply doing what they are asked to do. For example, a single "bot" agent sent out to find the cheapest airline fare can be expected to simply return with the requested information. As a collective, however, something new and different can result—something that is more than the sum of the individual participants. The stock market, immune systems, and ant colonies are all examples of agents acting individually, yet from the interactions of these agents a new phenomenon arises. With the stock market, thousands of agents act independently to buy and sell shares of particular stocks and bonds. Yet from this independent behavior, an organism-like product called the stock market emerges. In other words, the rise and fall of the market is not controlled by a central process: it results from agents interacting. The stock market crash of '29
was a result of individual human agents—not a central controller. The crash of October ’87 partly resulted from individual software agents that buy and sell securities according to programmed rules. The stock market, its crashes, temporary bubbles, dead-cat bounces, and all are more than the sum of the parts; it is an entity in its own right. Such entities are called emergent structures. Ant colonies are emergent structures that arise from individual ants acting interactively. The immune system emerges from the collective behavior of agents such as antigens, T cells, B cells, NK cells, immunoglobulin, lymph nodes, and the spleen. Other examples of emergent structures include families, organizations, societies, markets, flocks of birds, and traffic jams. In IT systems, this can include supply-chain, scheduling, trading-floor, and e-commerce systems.

Figure 9.1 — Local interaction can give rise to global dynamics, creating a coherent structure.
Emergence

Emergence is the existence of a coherent pattern that arises out of interactions among agents. (The diagram in Figure 9.1 illustrates this definition.) Emergence embodies several properties:

- **In emergent structures, agents organize into a whole that is greater than the sum of its parts.** In other words, the parts alone do not produce emergent structures—their interaction is required. For example, single bot agents will not result in emergence, no matter how many are scurrying around a system. However, a multiagent supply-chain system can have emergent properties, because it involves multiple interacting agents.

- **Rules that are almost absurdly simple can generate coherent, emergent phenomena.** For example, each ant in the ant colony described earlier has some very simple rules, yet a well-fed colony can emerge from these simple rules. A purchaser or supplier agent, too, can have simple rules resulting in an emergent inventory system.

- **Most emergent systems become apparent from the bottom up.** This contrasts with human engineers’ tendency to develop top down, an approach which is useful under some circumstances. However, in the world of living systems, most emerge from a population of simpler systems. Developing daily operational plans in a top-down manner works well for fairly predictable organizations. In less predictable environments, top-down plans can become obsolete in minutes. Here, a bottom-up agent-based approach could result in a very effective, emergent operation.

- **Persistent emergent structures can serve as components of more complex emergent structures.** In other words, hierarchies of emergent structures can be formed. This is how nature obtains scal-
ability; any IT organization could employ the same mechanism.

- Agents and their emergent structure can form a two-way link. Agents can give rise to an emergent structure; the emergent structure can influence its component agents. For example, the stock market is an emergent structure of individual buyers and sellers. Yet, the rise and fall of such markets can in turn affect the buying and selling habits of individual participants.

- *Emergent phenomena are typically persistent patterns with changing components.* While the specific birds in a flock or vehicles in a traffic jam can change, the flock and traffic jam can remain. Buyers and suppliers in a company's supply chain change frequently, and participants in a scheduling system can differ on a daily basis.

- *Collections of agents can be homogeneous or heterogeneous.* While emergence can occur through similar agents interacting, more often, it results from a diversity of agents that function in a society or ecosystem. Large organizations employ heterogeneity by specializing corporate resources into different roles and business units.

In the following sections, these properties will be explored in more detail.

**Greater than the sum of its parts**

Simple-minded reductionism states that the whole is simply the sum of its parts, and each part can be studied in isolation. However, the parts alone cannot produce emergence which requires the collective behavior and interaction of its components. Emergent structures, then, are a *process*—and the essence of the process is its form, not its parts. Families, organizations, socie-
ties, financial markets, schools of fish, and traffic jams are all examples of this phenomenon. A horde of non-interacting web spiders will not produce an emergent structure. However, when an ecosystem of supply-chain software agents can buy and sell goods and services over the web, a “supply web” can emerge. [2] The supply web, then, behaves like a financial market—which does give rise to an emergent structure. Agent-based modeling and bidding systems make this possible.

**Simple agent rules can produce emergent structures**

A common example of simple rules leading to emergence is that of a flock of birds. Here, each movement of a flock 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. In the flocking simulation of StarLogo (http://education.mit.edu/starlogo), the birds obey only three rules:

1. If you are far away from other birds, head towards the nearest bird.
2. If you are about to crash into another bird, turn to avoid the collision.
3. Otherwise, fly in the same direction as the bird next to you.

Using these three simple rules, no one bird has a sense of an overall flock. The “bird in front” is merely a position of a given bird. It just happens to be there—and will be replaced by others in a matter of minutes. Flocks of birds are not the only things that work like this. Beehives, ant colonies, freeway traffic, the Web, and the phenomenon of Silicon Valley are all ex-
amples of patterns that are determined by local component interaction, instead of a centralized authority. Complex behavior need not have complex roots.

**Top-down versus bottom-up approaches**

Performances of classical music or ballet are obviously orchestrated or choreographed, and the centralized, or top-down, development of these kinds of performances is both obvious and necessary. Many of the products we use in everyday life require top-down engineering to be effective. However, most of the emergent phenomena we experience do not occur as a result of top-down efforts; instead, they are the result of decentralized, or bottom-up, processes. For example, the flock of birds mentioned above emerges without an organizer and behaves without a coordinator. So, too, does the applause that follows a classical concert or ballet.

Today, many resource providers and manufacturers are exploring the possibilities of employing a decentralized approach. Many of these organizations are already adopting solutions that will replace their central, globally optimized operations with a distributed, self-organizing, local one. John Holland, a professor at the University of Michigan, is fond of pointing out that New York City maintains a two-week supply of food with only locally made decisions. Companies, such as Boeing, John Deere & Company, and Detroit Edison, are beginning to do this.

Both centralized and decentralized approaches are useful techniques. Using one technique and not the other limits the possibilities of a system. Often, our human bias toward centralization precludes the consideration of decentralized solutions. For example, three-year old Rachel developed the theory that clouds rain when the thunder commands them. At four, she developed a new theory: “The clouds get together at
night and decide whether or not it should rain the next day.” People resist decentralization. When they see a pattern, they often assume a centralized control. This does not mean that centralized theories are wrong, they are just not always appropriate:

- A central agent is a single point of failure that makes the system vulnerable to accident.
- Under normal operating conditions, a central agent can easily become a performance bottleneck.
- Even if adequately scaled for current operations, a central agent provides a boundary beyond which the system cannot be expanded.
- For software agents, it tends to attract functionality and code as the system develops, pulling the design away from the benefits of agents and in time becoming a large software artifact that is difficult to understand and maintain.

Emergent structures can themselves be components

One of the most difficult challenges for automated systems is scalability. Living systems provide some excellent examples
of scaling up. In the physical systems leading up to life, for example, subatomic particles form atoms, and atoms cluster to become molecules in solid, liquid, and gaseous form. Continuing up this hierarchy, molecules can be organized to form organelles, organelles can group to form cells, cells can aggregate to form organisms, and so on. In other words, living systems and their components emerge in a hierarchy of interlocking mechanisms. In the domain of human organization, similar hierarchies occur where economies emerge from markets, which emerge from enterprises, and so on (as illustrated in Figure 9.2).

Emergence provides the mortar between the bricks to construct viable structures. Furthermore, the new structures can become building blocks for even larger structures—in which each level of the hierarchy differs from the one before and the one after it. Such a hierarchy of interlocking mechanisms is also an appropriate technique for automated agents.

**Agents and their emergent structure can form a two-way link**

Applause occurs when spectators join in what appears to be spontaneous synchronized clapping, even though there is no conductor that coordinates this. When everyone starts, the clapping is totally unorganized—each person’s tempo wildly out of phase with the next. Eventually groups of people begin clapping at the same tempo. People in the audience sense the emerging rhythms and adjust their clapping to join it. The emerging applause rhythm grows even stronger and more people conform to it. Eventually, the entire audience is clapping in a synchronized pattern. This entire process can take place in a matter of seconds with even thousands of individuals.
We have been exploring how the local interaction and behavior of agents can produce global dynamics of emergent structures. In the example of spectator applause, the interaction of humans in the audience produced the dynamics of applause. However, there was another phenomenon occurring in this example: individually, the spectators adjusted their applause rhythm based on the applause that they heard. In other words, local interaction can give rise to global dynamics; and the global dynamics, in turn, can influence the local interaction.

Such an effect is evident in business systems, as well. The stock market both results from and affects buyers and sellers of securities. The auto parts market set up as a joint venture of GM, Ford, and DaimlerChrysler will result from and affect the buyers and sellers of automotive parts. As illustrated in Figure 9.3, emergent structures can be linked to their local agent interaction:
This link influences the boundary conditions of the local agents.

Local agents can then adjust to the presence of global dynamics.

Consequently, the conditions under which the agent behaves might change.

**Emergent structures can have components that change**

As we have seen earlier, the birds in a flock or the vehicles in a traffic jam can change, yet the flock and traffic jam phenomena can remain. The same applies to the stock market and supply-chain webs. Just because an emergent structure exists and is stable does not mean that its components cannot change over time. All the atoms in our body are replaced every three years, yet we are still considered to be the same being. Many Silicon Valley companies have more turnover than this, while still being recognized as the same organization. Change in the underlying agent population is not required for emergence, but it is a common phenomenon.

**Homogeneous and heterogeneous collections of agents**

Some emergent structures consist of a single kind of agent. For example, a flock of birds consists only of birds. Here, each agent plays the same role. Homogeneous agent collections can still have a single kind of agent, yet its agents can play different roles. For instance, an ant colony can have ants that play different roles. An ant can be a patrolling ant that guards the nest, a nest-maintenance worker, forager, brood-care worker, and so on. Furthermore, an ant can change its role depending on the requirements of the colony. For instance, a nest-maintenance worker can become a forager or a patrolling ant when the need for food or security becomes more important.
Heterogeneous collections of agents also play different roles because they contain different kinds of agents. The major difference is that in a heterogeneous collection, agents are different in both structure and behavior. For example, the immune system emerges from the collective behavior of various kinds of agents such as antigens, T cells, B cells, NK cells, immunoglobulins, lymph nodes, and the spleen. T cells and B cells not only play different roles, their structure and behavior is also different enough to be considered heterogeneous.

When we construct complex business systems, we need to think of agents as functioning as a society or ecosystem. In designing such systems, we need to consider how we can effectively employ homogeneous and heterogeneous agents.

**Conclusion**

When constructing agent systems, emergence is an important concept to consider. On the one hand, emergence is also something that can happen to you without your consent or predictive ability. This can be good or bad. On the other hand, as a system developer, you can try to “design in” the emergence that you want. In other words, you can try to design the agents in such a way that the desired structure emerges. In summary:

- You control the action of the parts, not the whole.
- You act as a designer, but the resulting pattern is not designed.
- Self-organizing patterns are created without a central designer.
- You must have enough agents acting in parallel to get a “critical mass.” A colony of ten ants will not suffice.
• The parts must be interacting—parallelism is not enough. Without interactions, interesting colony-level behaviors will never arise.

• Remember: a flock is not a big bird and a traffic jam is not just a collection of cars.

References


Agent security

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

Various types of security risks are described below. Many of these risks are inherent to distributed computing environments, particularly when software passes messages which can be intercepted, modified, or destroyed. While this is a threat to agent systems, it is also a threat to any software system that depends on messages being passed reliably. Another risk centers on whether or not the software can assume that it is using trustworthy services.

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

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

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

Identity is based on a *credential*, which is a set of data that can be validated by a third-party to prove that the entity is what it says it is. For example, when a user logs into a computer system, he often enters both a username and a password, which is the credential that is validated to indicate that he really is that username. Other common mechanisms for identity and credentials are X5.09 certificates and PGP keys.
Types of security risks

Here are some security threats that could happen in multi-agent systems.

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

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

- **Damage** - Destruction or subversion of a host's files, configuration, or hardware, or of an agent or its mission. Such hacking may even harm or destroy an agent.

- **Unauthorized copy and replay** - An attempt to copy an agent or a message and clone or retransmit it. For example, a malicious platform creates an illegal copy or a clone of an agent, or a message from an agent is illegally copied and retransmitted.

- **Denial of service** - An attack that attempts to deny resources to the platform or an agent. For example, one agent floods another agent with requests, and the second agent is unable to provide its services to other agents.

- **Repudiation** - An agent or agent platform denies that it has received or sent a message or taken a specific action. For example, a commitment between two agents as the result of a contract negotiation is later ignored by one of the agents. That agent denies the negotiation has ever taken place and refuses to honor its part of the commitment.
• **Spoofing and masquerading** - An unauthorized agent or agent platform claims the identity of another agent or agent platform. For example, an agent registers as a directory service agent and therefore receives information from other registering agents.

**Message passing**

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

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

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

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

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

**System components dealing with one another**

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

In the damage scenario, the agent may do malicious or inappropriate things to the host system, such as corrupting or deleting files. Therefore, the agent platform may want to control which agents can take which actions. Typically, the agent would offer a credential that identifies it to the agent platform. After validating the credential, the agent platform would use a security policy, based on the agent’s identity to determine what actions the agent could take, and would enforce that policy. This is very much like the access control lists found in most operating systems. However, agent systems probably want to control much more that simply reading, writing and running files. They might want to control message sending, utilization of various resources, when and where an agent can move, and whether a moving agent can run on this platform.
Just as the agent platform may want to validate what entity it is dealing with, an agent may want to validate that it is dealing with an agent platform it knows to be genuine. Agent platforms and services could pretend to be “legitimate” but in fact have some dangerous behavior, such as recording message transmissions before encryption, cloning copies of the agent for its own purposes, or providing false information.

Other risks to which agents can be exposed

In constructing software for an agent, certain types of risks must be addressed—ensuring that the following things can not occur:

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

More security considerations

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

- Agents and agent platforms can have multiple credentials. Multiple credentials reflect the reality that we have multiple roles. Users may have credentials as part of several organizations, as an individual, as the owner of multiple credit cards, and so on.
- Agents can have their own credentials. They may also have credentials for the user that they represent in an e-commerce application.
- Agents should not be created that can act anonymously. For example, a user may want to get data about drug or alcohol
treatment without revealing his or her identity. Obviously, sites can choose to reject these agents, if their security policies do not allow interaction with anonymous entities.

- All aspects of security need to be managed.
- Traceability of actions can be useful.
- Techniques for backup and recover can be useful for recovering from security breaches that harm or destroy an agent.
- Using a lease model on any credential can be helpful. In a lease model, credentials expire after a certain period of time, but can often be renewed from a credential authority. This control can be a very effective way to clean up credentials in a system which uses relatively short-lived agents. Requiring long-lived agents to renew their credentials is also useful, because when an entity with bad credentials is forced to renew, it will be rejected and shut down.
- Identity and credentials are also useful for building reputation services. Such services provide a way of determining whether a particular entity has behaved responsibly.

### Agent architecture

For the most part, agents will be deployed within conventional enterprises and will draw on the enterprise for many services. CORBA provides a rich source of services and a proven architecture. This section provides a framework for considering how a system supporting agents might draw on CORBA services and facilities. The architectural basis for this discussion will be the IEEE FIPA architecture. [www.fipa.org](http://www.fipa.org) The FIPA Agent Platform provides a good construct from which to discuss
the enterprise-related issues in agent deployment and is summarized in Figure 10.1.

![Figure 10.1 — The agent platform—an infrastructure for deploying agents.](image)

**Agent Platform**

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

FIPA does not specify the physical nature of a platform. However, two cases should be considered, that of a single host and that of multiple processors deployed as a “virtual” platform. If the platform is virtual, having it fulfill several requirements would be wise. It should have: 
- high-speed communications.
- a single agent management system.
- a single agent platform security manager.

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

**Agent Management System**

The agent management system (AMS) can be implemented as a single agent that supervises access to and use of the agent platform. The AMS maintains a directory of logical agent names and their associated transport addresses for an agent platform. The AMS is responsible for managing the lifecycle of the agents on the platform and its actions include:

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

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

**Agent Platform Security Manager**

The agent platform security manager (APSM) is responsible for maintaining security policies for the platform and infrastructure. The APSM is responsible for run-time activities, such as communications, transport-level security, and audit trails. Security cannot be guaranteed unless, at a minimum, all communication between agents is carried out through the APSM.

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

**Agent Platform Communication Channel**

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

In chapters 6 and 10, several of the agent standards from IEEE FIPA (Foundation for Intelligent Physical Agents) were introduced (www.fipa.org). Chapter 6 described the Agent communication Language (ACL) standard; and Chapter 10 discussed the FIPA architectural elements.

This specification introduces a standard that is currently in progress by the OMG: the Agent Metamodel and Profile (AMP)\(^1\). The objective of AMP is to provide a metamodel and profile for extending UML with core capabilities to enable agents and agent-based software. At a minimum, these core capabilities include the notions of agent, role, and community—and the structural and behavioral patterns defined by such constructs. Given these capabilities, this submission attempts to:

- Clarify semantics concerned with modeling agents.
- Establish Agent modeling best practices utilizing OMG technologies.

\(^1\) To review the full standard, see:
http://www.omg.org/techprocess/meetings/schedule/
Agent_Metamodel_and_Profile_%28AMP%29_RFP.html
- Develop a MOF-compliant agent metamodel to be used either standalone or via extending the existing UML metamodel with agent modeling capabilities.
- Enable agent model interchange between tools via XMI.
- Optionally facilitate modeling of Peer-to-Peer, Grid and Cloud computing, and other technologies in terms of a collection of Agents.

The primary goals that this submission fulfills are to provide a foundation to enable the use of agent technology that can:
- Model agents and agent-enabled constructs that can aid in the design of agent-based systems and emphasize how they will interact and collaborate.
- Be used in conjunction with existing and upcoming OMG technologies, such as: UML, the UML Profile and Metamodel for Services (UPMS) and the Event Metamodel and Profile (EMP).
- Be completed in a timely manner (approximately one year). Multiple follow-on agent-related RFPs can be planned and issued over time. Here, each RFP needs to be tangible and deliverable in a timely manner and carefully coordinated with the other agent-related RFPs.

This submission, then, is expected to be the first in a series of agent-related submission. As such, it seeks to address those basic foundational elements of agent technology that are both commonly used and can be defined in a reasonable amount of time.
Core Concept Overview

In general, agents can be software agents, hardware agents, firmware agents, robotic agents, human agents, and so on. While software developers naturally think of IT systems as being constructed of only software agents, a combination of agent mechanisms might in fact be used from shop-floor manufacturing to warfare systems.\(^2\)

These properties are mainly covered by a set of core aspects each focusing on different viewpoints of an agent system. Depending on the viewpoint of an agent system, various aspects are prominent. These aspects can be specified within the AMP metamodel

- **Agent aspect** – describes single autonomous entities and the capabilities each can possess in order to solve tasks within an agent system. In AMP, the stereotype Agent describes a set of agent instances that provides particular service capabilities.

- **Collaboration aspect** – describes how single autonomous entities collaborate within the multiagent system (MAS) and how complex organizational structures can be defined. Collaboration can involve situations such as cooperation and competition.

- **Role aspect** – covers feasible specializations and how they could be related to each role type. The concept of a role is especially used in the context of agent service contracts and interactions. Like in agent systems, the role indicates which responsibilities an actor has to take on.

---

\(^2\)For a further history and description of agents, see:
http://eprints.ecs.soton.ac.uk/825/05/html/chap3.htm,
http://en.wikipedia.org/wiki/Software_agent,
http://www.sce.carleton.ca/netmanage/docs/AgentsOverview/ao.html.
- **Interaction aspect** – describes how the interactions between autonomous entities or communities /organizations take place. Each interaction specification includes both the actors involved and the order in which messages are exchanged between these actors in a protocol-like manner.

- **Behavioral aspect** – describes how plans are composed from complex control structures and simple atomic tasks such as sending a message and specifying information flows between those constructs.

- **Organization/Community aspect** – Agents can form social units called communities. A community can be formed to take advantage of the synergies of its members, resulting in an entity that enables products and processes that are not possible from any single individual.

**Agents and Agent classes**

An agent is an autonomous, interactive entity. Agents, then, have the following three important properties. They are:

- Autonomous
- Interactive
- Adaptive

An agent’s foundational features are defined by an Agent Class. For example in Figure 11.1 below, depicts individual agents (e.g., Agent1 or “Bob”, Agent2 or “Betty” are Person agents; Agent3 is a neural network Manager agent called NN237, and Agent4 is a black-box Stock Trader processor called BB32).

An agent class is a classification of agents. In particular, each agent class provides the main foundational features, constraint, and semantics in common for a given set of associated agent individuals. The primary purpose of the agent class is to specify
what a particular kind of agent will be—indeed of its roles. While roles can change over the lifetime of an individual agent, its agent class does not. The purpose for this is to specify an immutable and defined structure. For example, the agent class Person would define an entity with certain structural and behavioral characteristics, which are independent of the roles played.

Three common approaches to defining agent classes exist:

- For software engineering, modeling Agents as instances of Agent Classes is useful because it can enable an MDA approach to creating the agents. For instance, a Cougaar Agent class would create agents that embodied the features that are associated with Cougaar Software agents. Similarly with other platforms, you could have JADE agents, JACK agents, Cybele agents, and so on. Although each produces agents, they agents will have different features for communication, internal processing, and so on.
Agent classes are also used to specialize agents for application purposes. Instead of having generic agents, a Stock Trader or Manager Agent may be the “foundational” form. While Stock Trader and Manager can also be thought of as roles, agent designers are free to specialize their agents for any purpose. The disadvantage is that a Manager will always be a manager. If dynamic flexibility is not a design concern, then pre-defined role-based agents can be useful because they can be created and used immediately. If dynamic flexibility is a design concern, a more generic agent class can be defined and role association can be adopted (described further, below).

A hybrid approach can also be employed so that the agent designer can specify Cougaar Manager or JADE Stock Traders. This approach is quite common, because it enables the MDA approach for creating agents from known agent software platforms and specializes the “role” for immediate deployment.

**Agents and Roles**

The notion of *role* is fundamentally a thespian concept, and attention to how it functions in the theater can reinforce our intuitions and provide useful metaphors for application to multiagent systems (MAS).

As humans, we find the perspective and language of the theater a useful analogy for describing and understanding many of the same complex aspects of individual behavior. The *role* perspective consists of those factors presumed to be influential in governing human behavior. Since we commonly employ this device in real life for conceptualizing human behavior, it may also serve as a useful device for other kinds of individuals in a
MAS—be they life forms, active software constructs, or hardware devices.

In the theater, an individual assigned to play a part, or role. In a MAS, an actor is an agent. Roles define normative repertoires of behavior and other features for agents. They provide both major building blocks for agent social systems and the requirements by which agents interact. Each agent is linked to other agents by the roles it plays by virtue of the system’s functional requirements—which are based on the expectations that the system has of the agent.

Figure 11.2 – Individual agents have multiple ways for classification. Role classes provide multiple and changeable roles during the lifetime of an agent; agent classes are permanent classifications.
As depicted in Figure 11.2, agents can be associated with more than one role class at the same point in time (multiple classification) and can change roles over time (dynamic classification). For example, Betty (Agent2) is a Person. Yet, dynamically, she has acquired the features to play multiple roles: Buyer and Seller. (Note she has two Buyer roles because she is a buyer for two different companies.) Later, she can take on additional roles or lose her current roles. The ability to acquire or dispose of roles is based on system requirements. For example, she could have been assigned or designated to play these roles by another agent with the ability to maintain roles. Or, she could have acquired the roles based on need, desire, or attained proficiency. In other words, role modification in the agent world can mirror the same approaches employed in the real world.

**Role class and role assignment**

A role class define a normative repertoire of behavior and other features that may be played by agents. In short, it is a social position that may be played by an agent; whereas the agent class specifies the physical definition of an agent. This figure depicts instances of role class (e.g., Broker, Buyer, Trust Manager, Buyer, …) and their general social relationship with agents – which also have the permanent foundational classifications via the agent class.

Each role class is a class, or kind, of role that may be played. As a class, then, it specifies features, constraints, and semantics in common for a given set of associated agent individuals. For example, the Seller is a role class that would contain methods such as Request Bid Proposal (from a seller), Decline Seller Proposal, Buy Goods or Service, and so on. It also has attributes, such as Maximum Authorized Purchase and Buyer ID.
In contrast, a role assignment is an instance of a role class. Here, each role will have values, processes, and policy specifications associated with each feature. For example in Figure 11.3, the role class of Buyer assigned to the Person Betty could have a Buyer ID of #9927, Maximum Authorized Purchase is $5,000, and plays role within the Think Box Corp, and so on. Since Better is a Buyer for Technika, as well, she has different role assignment values where her Buyer ID of #63661, Maximum Authorized Purchase is $22,000, etc.

Figure 11.3 – A way of graphically representing multiple role assignments for an agent.

In Figure 11.2, the assignment of roles is depicted using an association. Here, Betty has three associations to two different Role Classes. Bob has an association to the same class as Betty. Role Assignments, as illustrated in Figure 11.3, clarify the actual role assignment to an agent by:

- Providing a separately entity to record the data for each role assignment.
- Enabling multiple role assignments for the same kind, by ensuring that each role assignment has its own unique existence and identification.
Differentiating those apparently “identical” role assignments (e.g. Betty’s two Buyer role assignments by distinguishing their context (e.g., Technika and ThinkBox organizations).

Figure 11.4 – An agent and its roles are treated—together—as the same entity.

Figure 11.4 expands and clarifies the initial depiction in Figure 11.2 by inserting the role assignment between the Role Class and Agent. (While this could have been accomplished by associating a class with the association, a standard class with two associations was chosen for clarity.)

**Agent and role assignments – and the “same as” association**

Agents and Role Assignments are technically different set of objects. Conceptually, however, the agent and the roles assign-
ments that it plays can be thought of as occurring within the same entity. For example, the Person Bob and the Seller role that he plays are not two separate things. Bob plays the role and the role is played by Bob. We do not think of the part that is playing Bob’s role as not being Bob. Bob is taking the responsibility for playing the role assignment, as well as seen as being held responsible for how he plays his role. As such, the agent and its role assignments are related using a “same as” association. In Owl, the Same As association treats the linked entities as the same entity. In other words, Bob the Person and Bob playing the Seller position #21 are the same entity.

Figure 11.5 – Using Community to provide role context.
Community and Community Class

As a way to provide role context in the previous section, the role assignments are qualified by a community. A community is an individual community that comprises a collection of agents that are related via their roles. Each community can be treated as a composite structure—enabling a hierarchy of communities. A community class is a metaclass class that defines the basic kinds of groups, or communities, that may be defined (e.g., Organizations, Institutions, Teams, Military, Manufacturing, Banking, and so on).

A useful property of community is that they are agents in their own right. For example in Figure 11.5, Murks Inc is an agent because it is perceived as a legal entity that has its own autonomy and can interact with other agents. Yes, the community can and does consist of other agents (which themselves can be communities). However, from the outside world and the legal system, a community has its own rights, obligations, and prohibitions. It can be sued and subpoenaed. Due to these kinds of characteristics, communities are agents. And when communities can consist of communities to any number of levels, a rich degree of scalability can be achieved. If agents were only atomic-level entities, the social structure of the world as we now know it would not be possible.

Conclusion

Essentially, the AMP RFP requests a metamodel and profile for extending UML with capabilities applicable to agents and agent-based software. Submissions developed in response to this RFP will achieve the following: Clarify semantics concerned with modeling agents. Establish Agent modeling best practices utilizing OMG technologies. Develop a MOF-compliant agent
metamodel to be used either standalone or via extending the existing UML metamodel with agent modeling capabilities. Enable agent model interchange between tools via XMI. Optionally facilitate modeling of Peer-to-Peer, Grid and Cloud computing, and other technologies in terms of a collection of Agents. It is expected that responses to this RFP will make good use of agent modeling capabilities already supported by the OMG.
Twelve
Conclusion

Introduction

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.

We are no longer in the era of mainframe 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 changed to more side by side—as individuals, small groups, and organizations interact around the world. As a result, organizations are now demanding and cultivating new business practices that encourage less command and control and more horizontal connecting, collaborating, and competing.

Agent technology is a primary enabler to support this new era. In fact, without agent technology, our current technology will not scale to support the ever-increasing global interaction.
Global collaboration requires new IT approaches – using agents

This talk of “tearing down the walls and silos” is not new, and it has been a hot topic, particularly in recent years. So, what is different now? Increasingly, we find that we must pull together services from other organizations to form a collaborative team for just one consumer. This solution will create value for the consumer that does not exist in any one application or service or from any one organization. The underlying effort must support fine-grained, individualized resources and be able to address the personal level as well as organizational. For this to scale adequately, it must be automated. How can this be accomplished?

Figure 12.1 Service architectures enhanced using agents for interaction (from [2]).
One such approach is suggested in both the W3C Web Services Architecture [2] and the OASIS Service Oriented Architecture (SOA) Reference Model [3]. (Figure 12.1) Here, a service provides some functionality on behalf of its owner, which is a person or organization. The provider entity is the person or organization that supplies an appropriate agent to implement a particular service. A requester entity is a person or organization that wishes to make use of a provider entity's service. It will use a requester agent to exchange messages with the provider entity's provider agent. The figure illustrates this further. In step 3, the service description and semantics are realized by the requester and provider agents, and in step 4, the requester and provider agents exchange messages, resulting in some task being performed on behalf of the requester by one or more provider entities. This reduces the information and processing overload on human operators.

When the links between requesters and providers are few and change is not an issue, agents are not needed and conventional IT techniques can be employed. However, when the links require complex, dynamic binding and are subject to rapid change, agent-based approaches should be considered. Prior to the demand for global collaboration, we developed a centralized controller to ensure that all interactions would be appropriately managed. Now, this is no longer possible. The world is too big and a central bottleneck would paralyze the effort. Furthermore, the access techniques now require a more horizontal style of interaction—rather than one that is centrally administered.

Both collaboration and competition are supported by an agent-based world. Among the techniques using agents are auctions which link a requester’s service requirement to a rich world of service providers (e.g., British auction, Dutch auction,
eBay auction, or contract net). In the contract net protocol, for example, requester agents send out requests for quotes (RFQs) and award the winning provider(s) with a “contract” to provide a resource. For instance, an energy company like DTE has a finite number of resources (e.g., vehicles and crew members) to handle all the energy related services that are required for a power outage system. Under normal conditions, such planning and scheduling could be handled adequately by human schedulers and dispatchers with automated support. However, an ice storm, knocking out the power at 100,000 locations, will tax even the best human schedulers. Here, agents can be employed to automate the process by bidding on behalf of the various resources, including those from power companies in neighboring states and countries.

Cape Canaveral is now installing a similar system for its launch control services and resources using an agent-based approach developed by Intelligent Automation, Inc. [5] NASA is investigating using this technique for planning and scheduling sensor-based services and resources. These systems would use agents to identify, assemble, and execute the services that would form a virtual service for the end consumer. Other agents plan and schedule those resources required.

Managing services and resources in this manner enables and promotes global interaction and collaboration. At the heart of all these systems, agents digitize and decompose the supply chain and move the work around to anywhere in the world. [6]
For example, how can agents enable SOA to work in a complex world?

Agent technology is an appropriate supplement to many technologies. Service Oriented Architecture (SOA) is just one. SOAs have the flexibility and responsiveness to enable organizational priorities to finally drive technology decisions. Many companies are also expanding SOA beyond IT resources to include hardware (such as sensors, electronic equipment, and robotics, as well as human resources). By adding agent-based functionality to the SOA, an even greater flexibility and agility is possible. [4] In particular, agent technology can be used to enable the following features:

- **Resource provision and request** – In both the W3C and OASIS approaches to Web Services Architecture, the actual service requester and provider entities will agree on the service description and semantics that will govern the interaction between the requester and provider agents. The agents will then automate performing services on behalf of the actual requesters and providers. The agents can choose the smartest and most efficient way to accomplish a task by connecting different nodes in a network—whether horizontally or vertically.

- **Processing support** – For complex processes, agents can be used to orchestrate and “team” software processes to solve problems collaboratively or compete intelligently. Services result from self-assembly. The provider agents determine the value proposition for the consumer agents and then identify and assemble the services that will form a virtual service for the end consumer.
- **Resource discovery** — *Dynamic service selection* is increasingly common as organizations recognize the benefits of its flexibility. If an agent has flexibility in picking its business partners, it can select them to optimize any kind of quality-of-service criteria, including performance, availability, reliability, and trustworthiness. A requester agent can be chartered to do this, but specialized *service-discovery agents* are sometimes useful for facilitating this.

- **Middleware support** — Within SOA middleware, agents should be used for functions that require flexibility, autonomy, and robustness. For example, managing application-level fault tolerance, security, performance, and QoS are common uses for agents. In addition, agents can be used as a more “intelligent” way of handling and propagating changes in ontology and dealing with incompatibilities in vocabularies, semantics, and pragmatics among service providers, brokers, and requesters.

**Where to go from here**

Adopting any new technology is disruptive, and agent technology is no exception. Early-adopter companies are currently choosing agents for 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

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 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. The typical organization probably has both of these situations in one form or another, and the savvy organization will have a mixture of technologies: object-oriented (OO), relational, and agent-based. OO and relational technologies enable a top-down and centralized solution to business applications. Agents provide one more tool that conventional IT shops do not have: a bottom-up and distributed approach. The real benefit, then, comes when an organization can choose the appropriate mix of technologies for a given application—providing a balance of both centralized and distributed approaches.

**Conclusion**

As Friedman said so eloquently [1], 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.”

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


