CHAPTER 1

# Introduction

In this chapter, we present an introduction to the two major topics covered in this book: artificial intelligence (AI) and intelligent agents. We trace the history of AI research and discuss the basic premises of both the symbol processing and neural network schools. We explore the evolution of AI systems from a promising but largely discredited technology into the basis for today's intelligent agent applications. The simultaneous emergence of network computing and the Web, and their requirements for intelligent software are also discussed. We present key attributes of intelligent agents such as autonomy, mobility, and intelligence and provide a taxonomy for classifying various intelligent agent applications. We also discuss the unique features of the Java programming language that support our agent requirements.

## Artificial Intelligence

The science of AI is approximately forty years old, dating back to a conference held at Dartmouth in 1958. During the past forty years, the public perception of AI has not always matched the reality. In the early years, the excitement of both scientists and the popular press tended to overstate the real-world capabilities of AI systems. Early success in game playing, mathematical theorem proving, common-sense reasoning, and college mathematics seemed to promise rapid progress toward practical machine intelligence. During this time, the fields of speech recognition, natural language understanding, and image optical character recognition all began as specialties in AI research labs.

However, the early successes were followed by a slow realization that what was hard for people and easy for computers was more than offset by the things that were easy for people to do but almost impossible for computers to do. The promise of the early years

has never been fully realized, and AI research and the term *artificial intelligence* have become associated with failure and over-hyped technology.

Nevertheless, researchers in AI have made significant contributions to computer science. Many of today's mainstream ideas about computers were once considered highly controversial and impractical when first proposed by the AI community. Whether it is the WIMP (windows, icon, mouse, pointer) user interface, which dominates human-computer interaction today, or object-oriented programming techniques, which are sweeping commercial software development, AI has made an impact. Today, the idea of intelligent software agents helping users do tasks across networks of computers would not even be discussed if not for the years of research in distributed AI, problem solving, reasoning, learning, and planning. So before we dive into the tools and tricks of the AI trade, let's take a brief look at the underlying ideas behind the intelligence in our intelligent agents.

## Basic Concepts

Throughout its history, AI has focused on problems that lie just beyond the reach of what state-of-the-art computers could do at that time [Rich and Knight 1991]. As computer science and computer systems have evolved to higher levels of functionality, the areas that fall into the domain of AI research have also changed. Invented to compute ballistics charts for World War II-era weapons, the power and versatility of computers were just being imagined. Digital computers were a relatively new concept, and the early ideas of what would be useful AI functions included game playing and mathematics.

After 40 years of work, we can identify three major phases of development in AI research. In the early years, much of the work dealt with formal problems that were structured and had well-defined problem boundaries. This included work on math-related skills such as proving theorems, geometry, calculus, and playing games such as checkers and chess. In this first phase, the emphasis was on creating general "thinking machines" which would be capable of solving broad classes of problems. These systems tended to include sophisticated reasoning and search techniques.

A second phase began with the recognition that the most successful AI projects were aimed at very narrow problem domains and usually encoded much specific knowledge about the problem to be solved. This approach of adding specific domain knowledge to a more general reasoning system led to the first commercial success in AI: expert systems. Rule-based expert systems were developed to do various tasks including chemical analysis, configuring computer systems, and diagnosing medical conditions in patients. They utilized research in knowledge representation, knowledge engineering, and advanced reasoning techniques, and proved that AI could provide real value in commercial applications. At the same time, computer workstations were developed specifically to run Lisp, Prolog, and Smalltalk applications. These AI workstations featured powerful integrated development environments and were years ahead of other commercial software environments.

We are now well into a third phase of AI applications. Since the late 1980s, much of the AI community has been working on solving the difficult problems of machine vision and speech, natural language understanding and translation, common-sense reasoning,

and robot control. A branch of AI known as connectionism regained popularity and expanded the range of commercial applications through the use of neural networks for data mining, modeling, and adaptive control. Biological methods such as genetic algorithms and alternative logic systems such as fuzzy logic have combined to reenergize the field of AI. Recently, the explosive growth in the Internet and distributed computing has led to the idea of agents that move through the network, interacting with each other and performing tasks for their users. Intelligent agents use the latest AI techniques to provide autonomous, intelligent, and mobile software agents, thereby extending the reach of users across networks.

When we talk about artificial intelligence or intelligent agents, the question often arises, what do we mean by *intelligence*? Do we mean that our agent acts like a human, that it thinks like a human, or that it acts or thinks rationally? While there are as many answers as there are researchers involved in AI work, we'll tell you what we think it means. To us, an intelligent agent acts rationally. It does the things we would do, but not necessarily the same way we would do them. Our agents may not pass the Turing test, proposed by Alan Turing in 1950 as a yardstick for judging computer intelligence. But our agents will perform useful tasks for us. They will make us more productive. They will allow us to do more work in less time, and see more interesting information and less useless data. Our programs will be qualitatively better using AI techniques than they would be otherwise. No matter how humble, that is our goal: to develop better, smarter applications.

## Symbol Processing

There are many behaviors to which we ascribe intelligence. Being able to recognize situations or cases is one type of intelligence. For example, a doctor who talks with a patient and collects information regarding the patient's symptoms and then is able to accurately diagnose an ailment and the proper course of treatment exhibits this type of intelligence. Being able to learn from a few examples and then generalize and apply that knowledge to new situations is another form of intelligence.

Intelligent behavior can be produced by the manipulation of symbols. This is one of the primary tenets of AI techniques. Symbols are tokens that represent real-world objects or ideas and can be represented inside a computer by character strings or by numbers. In this approach, a problem must be represented by a collection of symbols, and then an appropriate algorithm must be developed to process these symbols.

The physical symbol systems hypothesis [Newell 1980] states that only a "physical symbol system has the necessary and sufficient means for general intelligent action." This idea, that intelligence flows from the active manipulation of symbols, was the cornerstone on which much of the subsequent AI research was built. Researchers constructed intelligent systems using symbols for pattern recognition, reasoning, learning, and planning [Russell and Norvig 1995]. History has shown that symbols may be appropriate for reasoning and planning, but that pattern recognition and learning may be best left to other approaches.

There are several typical ways of manipulating symbols that have proven useful in solving problems. The most common approach is to use the symbols in formulations of if-then rules that are processed using reasoning techniques called forward and backward

chaining. Forward chaining lets the system deduce new information from a given set of input data. Backward chaining allows the system to reach conclusions based on a specific goal state. Symbol processing techniques may also include constructing a semantic network, in which the symbols and the concepts they represent are connected by links into a network of knowledge that can then be used to determine new relationships. Another formalism is a frame, in which related attributes of a concept are grouped together in a structure with slots and are processed by a set of related procedures called daemons or fillers. Changing the value of a single slot could set off a complex sequence of related procedures as the set of knowledge represented by the frames is made consistent. These reasoning techniques are described in more detail in Chapter 3, "Knowledge Representation," and Chapter 4, "Reasoning Systems."

Symbol processing techniques represent a relatively high level in the cognitive process. From a cognitive science perspective, symbol processing corresponds to conscious thought, where knowledge is explicitly represented, and the knowledge itself can be examined and manipulated. While symbol processing and conscious thought are clearly part of the story, another set of researchers is examining a nonsymbolic approach to intelligence modeled after the brain.

## Neural Networks

An increasingly popular method in AI is called neural networks or connectionism. Neural networks have less to do with symbol processing, which is inspired by formal mathematical logic, and more to do with how human or natural intelligence occurs. Humans have neural networks in their heads, consisting of hundreds of billions of brain cells called neurons, connected by adaptive synapses that act as switching systems between the neurons. Artificial neural networks are based on this massively parallel architecture found in the brain. They process information not by manipulating symbols but by processing large amounts of raw data in a parallel manner. Different formulations of neural networks are used to segment or cluster data, to classify data, and to make predictive models using data. A collection of processing units that mimic the basic operations of real neurons is used to perform these functions. As the neural network learns or is trained, a set of connection weights among the processing units is modified based on the relationships perceived among the data.

Compared to symbol processing systems, neural networks perform relatively low-level cognitive functions. The knowledge they gain through the learning process is stored in the connection weights and is not readily available for examination or manipulation. However, the ability of neural networks to learn from and adapt to their surroundings is a crucial function for intelligent software systems. From a cognitive science perspective, neural networks are more like the underlying pattern recognition and sensory processing performed by the unconscious levels of the human mind. We discuss neural networks and learning in Chapter 5, "Learning Systems."

Whereas AI research was once dominated by symbol processing techniques, there is now a more balanced view in which the strengths of neural networks are used to

counter the weaknesses of symbol processing. In our view, both are absolutely necessary in order to create intelligent applications and intelligent autonomous agents.

## Turing and Connectionist Networks

**Digital computers are based on a theoretical foundation laid out by the English scientist and mathematician Alan Turing in 1935. The Universal Turing Machine, an abstract device comprised of a tape with symbols (the program) and a read/write scanner (the computer), can match the behavior of a human, working with paper and pencil, following a mechanical process or algorithm. The Universal Turing Machine can simulate the behavior of any other processing machine. It defines the limits of computability using conventional computers.**

**In a paper written in 1948, a decade before Frank Rosenblatt's work on the Perceptron, Turing went outside the logical box of the Turing machine and described an extended machine that includes a black box or oracle providing the means for carrying out uncomputable tasks [Copeland and Proudfoot 1999]. In his definition of the O-machine, Turing introduced a type of neural network he called a B-type unorganized machine. His neural network contained a collection of neurons with arbitrary interconnection patterns. Connections pass through modifier devices that serve as dynamic switches, either allowing the signal to pass or blocking it. Each neuron has two inputs and a single output that computes the logical not-AND (NAND) function. He chose the NAND function because all other Boolean logical operations can be constructed using NAND gates. (Turing tended toward the most general solutions to computing problems.)**

**This is interesting because much of the tension between the mathematical logic school of AI and the more empirical soft computing school of AI is based on the heritage of Turing machines. And here was Alan Turing, extending his logical base with neural networks toward computing the uncomputable. His 1948 paper, titled "Intelligent Machinery," was not published until 1968, fourteen years after his death. Although his definition of artificial intelligence, the so-called Turing Test, is widely known throughout the field, his work on connectionist computing is relatively unknown. Connectionism is a natural, logical extension of his work on the Turing machine.**

**In his biography [Hodges 1983] Turing is quoted as saying "if a machine is expected to be infallible, it cannot also be intelligent" (page 361). It seems we want perfection from artificial intelligence, while we expect (and get) far less from human intelligence. People aren't perfect, yet they represent the highest form of intelligence on earth. Why do we expect machine intelligence to be perfect? So connectionists, take heart. You are following in the path of Alan Turing, the father of modern computing.**

## The Internet and the Web

The Internet grew out of government funding for researchers who needed to collaborate over great distances. As a byproduct of solving those problems, protocols that allowed different computers to talk to each other, exchange data, and work together were developed. This led to TCP/IP becoming the de facto standard networking protocol for the Internet. The growth in the Internet is astounding, with the number of sites increasing exponentially. Thousands of new sites are connected to the Internet each month, containing approximately 1 billion static Web pages with over 7 billion hyperlinks among them.

While electronic mail (e-mail) was once the primary service provided by the Internet, information publishing and software distribution are now of equal importance. The Gopher text information service, which gained popularity in the early 1990s, generated the first wave of information publishing on the net. The File Transfer Protocol (FTP) allowed users to download research papers and articles as well as retrieve software updates and even complete software products over the Internet. The Network News Transport Protocol (NNTP) allowed users to exchange ideas on a broad range of topics in Internet news groups. But it was the HyperText Transfer Protocol (HTTP) that brought the Internet from the realm of academia and computer technologists into the public consciousness. The development of the Mosaic browser at the University of Illinois transformed the Internet into a general-purpose communications medium, where computer novices and experts, consumers, and businesses can interact in entirely new ways.

The Web, with its publishing and broadcasting capabilities, has extended the range of applications and services that are available to users of the Internet. The now-ubiquitous Web browser provides a universal interface to applications regardless of which server platform is serving up the application. In the browsing or "pull" mode, the Web allows individuals to explore vast amounts of information in one relatively seamless environment. Knowing that all of the information is out there, but not knowing exactly how to find it, can make the Web-browsing experience quite frustrating. The popular search engines and Web index sites such as AltaVista, Excite, Yahoo, and Lycos provide an important service to users of the Web, by grouping information by topics and keywords. But even with the search engines and index sites, Web browsing is still a hit-or-miss proposition (with misses more likely than hits). In this environment, intelligent agents will emerge as truly useful personal assistants by searching, finding, and filtering information from the Web, then bringing it to a user's attention. Even as the Web evolves into "push" or broadcast mode, where users subscribe to sites which send out constant updates to their Web pages, this requirement for filtering information will not go away. Unless the broadcast sites are able to send out very personalized streams of information, the user will still have to separate the valuable information from the useless noise.

While the Internet and Web have captured the public's attention, businesses are quickly adapting the way they use information technology through their internal networks or intranets. Companies use intranets for internal and external e-mail, to post information, and to handle routine administrative tasks. Intranets allow a wide variety of client computers to connect to centralized servers, without the cost and complexity of developing

client/server applications. Intranets serve the same purpose and have the same advantages for companies that the Internet has for individuals. A standardized client application, the Web browser, running on standard personal computers or low-cost network computers, can provide a single point of access to a collection of corporate-wide network-based applications.

The emergence of e-business has dramatically changed the Internet and expanded its use by businesses and consumers. The Business to Consumer (B2C) model, in which companies "go direct" to their customers, has lowered the cost of doing business and endangered middlemen in the supply chain. Companies such as Dell Computer and Amazon.com have transformed retailing and forced more traditional bricks-and-mortar companies to become net-savvy clicks-and-bricks businesses with a substantial on-line presence. The Business to Business (B2B) opportunities have spawned hundreds of vertical electronic marketplaces where buyers and suppliers in industries such as chemicals, metals, and electronics come together to do business. Both B2C and B2B require intelligent software to provide personalized information, to perform automated negotiations, and to perform planning and scheduling functions.

## Intelligent Agents

As is often the case when a technical field provokes commercial interest, there has been a large movement and change of focus in the AI research community to apply the basic AI techniques to distributed computer systems, company-wide Intranets, the Internet, and the Web. Initially, the focus was limited to word searches, information retrieval, and filtering tasks. But as more and more commercial transactions are performed on networks, there is more interest in having smart agents that can perform specific actions. By taking a step back and looking at what the Internet has become, many researchers who had been looking at how intelligent agents could cooperate to achieve tasks on distributed computer systems have realized that there is finally a problem in search of a technology (as opposed to the other way around). Intelligent agents can provide real value to businesses and users in this new, interconnected world.

Up to this point, we have discussed AI and its evolution into software agents at an abstract level. In the following sections, we explore some of the technical facets of intelligent agents, how they work, and how we can classify them based on their abilities and underlying technologies.

## Events, Conditions, and Actions

Suppose we have an intelligent agent, running autonomously, primed with knowledge about the tasks we require of it and ready to move out onto the network when the opportunity arises. Now what? How does the agent know that we want it to do something for us, or that it should respond to someone who is trying to contact us? This is where we have to deal with events, recognize conditions, and take actions.

## Are They Agents or Distributed Objects?

**For the skeptics in the audience, you may be asking, "What's the big deal with agents?" They're just distributed objects with a different name. CORBA has been around for years, Microsoft's (OLE, ActiveX, COM, DCOM, etc.) objects have been around for years. What's the difference?**

**To address this question, let's examine a distributed object application and a distributed multiagent system application. A distributed object application is defined by the objects, the data, and the behavior (methods) required to implement the functions needed. The interactions between objects are explicitly defined; the sequence of method calls is spelled out in gory detail. An object is a software entity that encapsulates state (data) and behavior (function) and exposes a set of methods or procedures to manipulate that state. Objects are *used* by other objects to perform actions. Objects don't initiate actions of their own volition.**

**In a multiagent system, we have a collection of software entities that autonomously perform actions. Each agent has a more complex internal state than an object, but even more importantly, they have internal goals. Agents decide what to do and when to do it. Agents can say "no" when requested to perform an action. Objects have fixed roles. Agents can change roles dynamically as the application runs.**

**In a distributed object application, every object is contributing a small piece of function in achieving a single application goal. In an agent system, a collection of goal-oriented software entities cooperate to achieve a single application goal. Objects invoke methods. Agents have conversations. Objects are data packets with buttons waiting to be pushed. Agents are actively deciding what buttons to push.**

In the context of intelligent agents, an event is anything that happens to change the environment or anything of which the agent should be aware. For example, an event could be the arrival of a new piece of mail. Or it could be a change to a Web page. Or it could be a timer going off at midnight—time to start sending out the faxes that are queued up. Short of having our agent constantly running and checking or polling all the devices and computer systems we want it to monitor, having events signal important occurrences is the next best thing. Actually, it may be the best thing, because our agent can sleep, think about what has happened during the day, do housekeeping tasks, or do anything else useful while it is waiting for the next event to occur.

When an event does occur, the agent has to recognize and evaluate what the event means and then respond to it. This second step, determining what the condition or state of the world is, could be simple or extremely complex, depending on the situation. If mail has arrived, then the event will be self-describing: a new piece of mail has arrived.

The agent may then have to query the mail system to find out who sent the mail and what the topic is, or even scan the mail text to find keywords. All of this is part of the *recognize* component of the cycle. The initial event may wake up the agent, but the agent then has to figure out what the significance of the event is in term of its duties. In the mail example, suppose the agent recognizes that the mail is from your boss, and that the message is classified as URGENT. This brings us to the next and perhaps most useful aspect of intelligent agents: actions.

If intelligent agents are going to make our lives easier (or at least more interesting), they must be able to take action, to do things for us. Having computers do things for us is not a new idea. Computers were developed to help people do work. However, having the computer initiate an action on our behalf is something totally different from entering a command on the command line and pressing *Enter* to run the command. While the results of our typing the command and pressing ENTER may not always be exactly what we had in mind when we typed it in (it always seems that we realize what we should have typed after we press ENTER), we know that whatever happens, it is our doing. Having an agent (intelligent or not, human or computerized) take an action for us requires a certain leap of faith or at least some level of trust. We must trust that our intelligent agent is going to behave rationally and in our best interest. Like all situations in which we delegate responsibility to a third party, we have to weigh the risks and the rewards. The risk is that the agent will mess things up, and we will have to do even more work to set things right. The reward is that we are freed from having to worry about the details of getting that piece of work done.

So, events-conditions-actions define the workings of our agent. Some researchers feel that an agent must also be proactive. It must not only react to events, it must be able to plan and initiate actions on its own. We agree. However, in our view, this action (signaling some event, or calling some application interface) is the result of some earlier event that caused our agent to go into planning mode. We want our intelligent agents to be able to initiate transactions with other agents on our behalf, using all of the intelligence and domain knowledge they can bring to bear. But this is just an extension of the event-condition-action paradigm.

## Taxonomies of Agents

While intelligent agents are still somewhat new in commercial computing environments, they have been the focus of researchers for years. During that time, many different ways of classifying or categorizing agents have been proposed. One way is to place the agent in the context of intelligence, agency, and mobility. Another approach is to focus on the primary processing strategy of the agent. A third is to categorize the agent by the function it performs. In the following sections we explore all three perspectives on viewing agent capabilities.

### Agency, Intelligence, and Mobility

When we talk about software agents, there are three dimensions or axes that we use to measure their capabilities: agency, intelligence, and mobility [IBM 1996]. Agency deals

with the degree of autonomy the software agent has in representing the user to other agents, applications, and computer systems. An agent represents the user, helps the user, guides the user, and in some cases, takes unilateral actions on the user's behalf. This progression from simple helper to full-fledged assistant takes us from agents that can be hard-coded, to those which, out of simple necessity, must contain more advanced intelligence techniques.

Intelligence refers to the ability of the agent to capture and apply application domain-specific knowledge and processing to solve problems. Thus our agents can be relatively dumb, using simple coded logic, or they can be relatively sophisticated, using complex AI-based methods such as inferencing and learning.

An agent is mobile if it can move between systems in a network. Mobility introduces additional complexity to an intelligent agent, because it raises concerns about security (the agent's and the target system's) and cost. Intranets are a particularly ripe environment for mobile intelligent agents to roam because they require less security than in the wide-open Internet.

## Processing Strategies

One of the simplest types of agents is *reactive* or *reflex* agents, which respond in the event-condition-action mode. Reflex agents do not have internal models of the world. They respond solely to external stimuli and the information available from their sensing of the environment [Brooks 1986]. Like neural networks, reactive agents exhibit *emergent behavior*, which is the result of the interactions of these simple individual agents. When reactive agents interact, they share low-level data, not high-level symbolic knowledge. One of the fundamental tenets of reactive agents is that they are grounded in physical sensor data and are not operating in the artificial symbol space. Applications of these agents have been limited to robots, which use sensors to perceive the world.

*Deliberative* or *goal-directed* agents have domain knowledge and the planning capability necessary to take a sequence of actions in the hope of reaching or achieving a specific goal. Deliberative agents may proactively cooperate with other agents to achieve a task. They may use any and all of the symbolic AI reasoning techniques that have been developed over the past forty years.

*Collaborative* agents work together to solve problems. Communication between agents is an important element, and while each individual agent is autonomous, it is the synergy resulting from their cooperation that makes collaborative agents interesting and useful. Collaborative agents can solve large problems that are beyond the scope of any single agent and they allow a modular approach based on specialization of agent functions or domain knowledge. For example, collaborative agents may work as design assistants on large, complex engineering projects. Individual agents may be called upon to verify different aspects of the design, but their joint expertise is applied to ensure that the overall design is consistent. In a collaborative agent system, the agents must be able to exchange information about beliefs, desires, and intentions, and possibly even share their knowledge.

Agents with beliefs, desires, and intentions are known as BDI model agents in the AI research community [Bratman 1987]. There is some contention in the field as to whether BDI agents, which are quite heavyweight, are really necessary to build multi-agent systems. The beliefs represent knowledge about the state of the world. What the agent believes to be true about the world is its reality. It is the basis for all of its reasoning, planning, and subsequent actions. Beliefs are essential for an agent because it cannot ever have complete knowledge of the outside world. Things change. Some things can't be seen or known. Because the agent's beliefs can't be perfect, they must be able to model uncertainty and imprecision in the facts it knows. Desires are assignments of goodness to states of the world, from the agent's perspective. Desires turn into goals when the agent is reasoning about how it wants the world to be. When the agent reasons about the state of the world (beliefs) and its desires (goals) it must decide what course of action to take. These committed plans are called intentions. So, BDI agents have quite lofty requirements from an AI reasoning perspective. BDI agents are autonomous AI software components.

As mentioned earlier, a *mobile* agent is a software process (a running program's code and its state) that can travel across computer systems in a network doing work for its owner. An advantage of mobile agents is that the communications between the home system and the remote systems are reduced. By allowing the agent to go to the remote system and access data locally on that system, we enable a whole new class of applications. For example, mobile agents could provide an easy way to do load balancing in distributed systems. "Oh, the processor is heavily loaded here, better hop on over to System X for a while."

Danny Lange, who led the IBM Aglets development team, lists the following seven reasons for using mobile agents [Lange 1998]:

- They can reduce network load, because agents move to a system and do their work there rather than take up network bandwidth sending messages back and forth.
- They can overcome network latency because they are resident on the machine rather than remote.
- They can encapsulate protocols as they move around the network talking to other mobile agents.
- They can operate autonomously so that they keep working even when network connections go down.
- They can dynamically adapt to changes in system loading.
- They are heterogeneous.
- They are fault-tolerant because they can move from a system that is having difficulty or about to fail.

As Nwana (1996) says, "mobility is neither a necessary nor sufficient condition for agenthood." Sometimes it makes sense to have your agent go out on the network; other times it does not. For some people, the idea of sending a mobile agent off to work is comfortable and natural. For others, the lack of a familiar computing model (it is neither server-based nor client/server) makes mobile agents hard to fathom. For example,

if your agent contains some exclusive domain knowledge or algorithms, then sending that intellectual property out on the network to reside on foreign hosts, may not be a good idea. Better to keep that agent at home in a safe, secure system, and send out messenger collaborative agents to do the traveling. Perhaps the biggest inhibitor to widespread use of mobile agents is security. We have a name for software that comes unbidden onto our systems and starts executing. We call them *viruses*. How do we make sure that only "good" mobile agents can run on our system, but not "bad" agents? And how can we tell the difference?

## Processing Functions

Perhaps the most natural way of thinking about the different types of agents is based on the function they perform. Thus, we have user interface agents, which try to "do what you mean" rather than what you say when interacting with a piece of application or system software. We have search agents, which go out on the Internet and find documents for us. We have filter agents, which process incoming mail or news postings, ferreting out the stuff of interest from the more mundane or uninteresting rubbish. We have domain-specific assistants, which can book a business trip, schedule a meeting, or verify that our design does not violate any constraints. In short, any combination of intelligent agent attributes can be combined and applied to a specific domain to create a new, function-specific intelligent agent. All we need are the software tools and computing infrastructure to be able to add the domain knowledge and reasoning or learning capabilities to our agents, and the comfort level that is required to trust the agent to do our bidding.

*Interface* agents work as personal assistants to help a user accomplish tasks. Interface agents usually employ learning to adapt themselves to the work habits and preferences of the user. Patti Maes (1994) at MIT identifies four ways that learning can occur. First, an agent can learn by watching over the user's shoulder, observing what the user does and imitating the user. Second, the agent can offer advice or take actions on the user's behalf and then learn by receiving feedback or reinforcement from the user. Third, the agent can get explicit instructions from the user (if this happens, then do that). Finally, by asking other agents for advice, an agent can learn from their experiences. Note that interface agents collaborate primarily with the user, not with other agents (asking advice is the one exception). Using various learning mechanisms, interface agents offer the promise of customizing the user interface of a computer system or set of applications for a particular user and her unique working style. If interface agents can collaborate and share their knowledge about how to do a task, then when one person in a workgroup figures out how to do something, that skill can be transferred to all other users in that workgroup through their interface agents. The productivity gains could be enormous.

Another generic class of agents is *information* agents. In some ways, information agents are the Dr. Jekyll/Mr. Hyde of software programs. They seem harmless enough, providing information to you, but can quickly transform into the monster of *information overload*. Some information agents go out on the Internet or the Web and seek out information of interest to the user. Others filter streams of information coming in e-mail correspondence and newsgroup postings. But either way, information agents try to help with the core problem of getting the right information at the right time. The question is not whether there is too much information or too little, but making sure that you see the

right information. Of course, what is "right" depends on the context in which you are working. This information overload problem is one of the prime factors in the emergence of intelligent software agents as viable commercial products. Whether routing particularly important e-mail messages, or actively constructing a personal newspaper (consisting only of interesting articles and advertisements), information agents promise relief from the overwhelming amount of data we are exposed to each day. Information agents called Spiders are already being used to index the Web. In general, information agents can be static, sitting on one system using search engines and Web indexes to gather information for the user, or they can be mobile, going out and actively searching for information. Either way, their function remains the same: to deliver useful information to the user.

In the preceding sections, we have described three of the major taxonomies for classifying agents, but there are others. Some feel that collaboration with other agents is a major distinction. Others feel that whether the agent interacts directly with a human user or not is important. Still others feel that learning ability should be a primary basis for classifying agents. Whether the agent works across a network or only locally on a PC or workstation is another distinguishing attribute. Because intelligent agents exist in a multidimensional space, characterizing agents by two or three of those dimensions is somewhat risky [Nwana 1996]. However, we feel that the loss of some precision is more than offset by the clarity of the two- or three-dimensional approach. People find it hard to think in six- or seven-dimensional space. In our opinion, agency and intelligence are the fundamental underlying capabilities on which agents should be classified. Mobility or some other characteristic could be used as the third axis, as required.

To be sure, there are software agents that are autonomous, but not intelligent. These agents often act as simple machine performance monitors in distributed system management applications. Simple Network Management Protocol (SNMP) agents are an example of this kind. On the other hand, there are programs or applications which use AI techniques such as learning and reasoning, but which have relatively little autonomy. Classic expert systems are an example. They are not intelligent agents in the sense used in this book. We are interested in programs at the intersection of two domains, intelligence and agency. All other characteristics are secondary and may or may not be present for us to use the term *intelligent agent.*

Intelligent agents are software programs, nothing more and nothing less. Sometimes this has a negative impact when someone new to the topic comes to the realization that there is no magic here, just programming. However, intelligent agents, at least as we define and refer to them, are software programs with an attitude. They exist to help users get their work done. You may say that that is what application software is supposed to do. This is true. However, many applications today assume a level of familiarity and sophistication in the users that many users are incapable of or unwilling to achieve. People just want to get their job done. Most don't care if the font is TrueType or Adobe, if the component model is COM, CORBA, or JavaBeans, or whether the code is client/server or Web-based. Toward this end, intelligent agent software is practical software. It just gets the job done. If intelligent agent software introduces another level of complexity that the user has to deal with, then it will be a failure. Intelligent agents must be enabling and automating, not frustrating or intrusive.

# Agents and Human-Computer Interfaces

One of the perennial complaints of users and one of the sticking points between humans and computers has been the interface between them. A QWERTY keyboard and a mouse pointing device have become accepted as the way people interact with computers, but they are far from natural. The most natural way for people to interact is through language, facial expressions, and body language. After 30 years of research into speech processing, we are finally getting usable software that will allow us to talk to our computers. But recognition is not the same as understanding. Although a computer can display the words you just spoke, they can't necessarily figure out the reference to *him* or *her*, detect sarcasm, or catch literary references. But things are changing, fast.

The Blue Eyes project at IBM Almaden Research Center features a camera that can figure out where a user is looking on the screen (gaze identification) to determine what article they are reading. Gesture recognition software allows computers to respond to waves of the hand, and even understand facial expressions. And, no surprise here, intelligent software forms the basis for these types of applications.

The COLLAGEN project at Lotus Research and Mistubishi Reseach develops agents that can watch a user interact with an application and figure out the task that the user is trying to perform and give assistance [Rich and Sidner 1997]. The OpenSesame application on Macintosh watches a user, learns their behavior, and offers to automate repetitive tasks [Caglayan, et al. 1997].

Avatars, or on-line graphic persona, are becoming more important. Chat rooms and other on-line forums provide the ability for people to mix and exchange information. Avatars allow people to show (or hide) their true personality. Microsoft Agents provide a basic capability to develop avatars for applications running on Windows operating systems. For example, Microsoft Office has these little "Assistants" that appear on the screen to help you. These animated creatures change expression as you interact with the computer, blink their eyes, and straighten up at attention when you mouse over to them. They provide context-sensitive help. Some people love them, but others find that having movement on the periphery of the screen is quite annoying. But, like them or not, these Microsoft Agents are most likely the first wave of the future of human-computer interface agents (see http://msdn.microsoft.com/).

# Using Java for Intelligent Agents

In this section, we talk about the Java language and the specific features of Java that support intelligent agent applications. These features will be explored in more detail in the remainder of the book.

## Overview of the Java Language

Java is an object-oriented programming language. It was originally designed for programming real-time embedded software for consumer electronics, particularly set-top boxes

that interface between cable providers or broadcasters, and televisions or television-like appliances. However, the effort was redirected to the Internet when the market for set-top boxes did not develop quickly enough, while the Internet exploded in popularity.

Originally the developers of Java intended to use C++ for their software development. But they needed a language that could execute on different sets of computer chips to accommodate the ever-changing consumer electronics market. So they decided to design their own language that would be independent of the underlying hardware.

It is this "architecture-neutral" aspect of Java that makes it ideal for programming on the Internet. It allows a user to receive software from a remote system and execute it on a local system, regardless of the underlying hardware or operating system. An interpreter and runtime called the Java Virtual Machine (JVM) insulates the software from the underlying hardware.

Unlike more traditional languages, Java source code does not get translated into the machine instructions for a particular computer platform. Instead, Java source code (.java) is compiled into an intermediate form called bytecodes which are stored in a *class* file. These bytecodes can be executed on any computer system that implements a JVM. This portability is perhaps one of the most compelling features of the Java language, from a commercial perspective. In the current era of cross-platform application development, any tool that allows programmers to write code once and execute it on many platforms is going to get attention.

The portable, interpreted nature of Java impacts its performance. While the performance of interpreted Java code is better than scripting languages and fast enough for interactive applications, it is slower than traditional languages whose source code is compiled directly into the machine code for a particular machine. To improve performance, Just-In-Time compilers (JITs) have been developed. A JIT compiler runs concurrently with the Java Virtual Machine and determines what pieces of Java code are called most often. These are compiled into machine instructions on-the-fly so that they do not need to be interpreted each time they are encountered within a program. Static compilers have also been developed to compile the Java source code into machine code that can be executed without interpretation (but with loss of portability).

The bytecode portability is what enables Java to be transported across a network and executed on any target computer system. Java applets are small Java programs designed to be included in an HTML (HyperText Markup Language) Web document. HTML tags specify the name of the Java applet and its Uniform Resource Locator (URL). The URL is the location on the Internet where the applet bytecodes reside. When a Java-enabled Web browser displays an HTML document containing an applet tag, the Java bytecodes are downloaded from the specified location and the Java Virtual Machine interprets or executes the bytecodes. Java applets enable Web pages to contain animated graphics and interactive content.

Because Java applets can be downloaded from any system, security mechanisms exist within the Java Virtual Machine to protect against malicious or errant applets. The Java runtime system verifies the bytecodes as they are downloaded from the network to ensure that they are valid bytecodes and checks that the code does not violate any of the inherent restrictions placed on applets. Java applets are restricted from communicating

with any server other than the originating host, the one from which they were downloaded. They cannot run a local executable program or access local files. The restrictions are in place to prevent a Java applet from gaining access to the underlying operating system or data on the system. These restrictions can be eased, however, through the use of digital signatures and alternate **SecurityManager** implementations.

But Java can be used for more than programming applets to run within a browser. Java is a full-function programming language that can be used to write standalone applications. These applications are not placed under the same security restrictions as applets and therefore can access data and underlying operating system functions.

As an object-oriented programming language, Java borrows heavily from Smalltalk, Objective C, and C++. It is characterized by many as a better, safer C++. Java uses C/C++ syntax and is readily accessible to the large existing C++ development community. Java, however, does not drag along the legacy of C. It does not allow global variables, functions, or procedures. With the exception of a few primitive data types like integers and floating-point numbers, everything in Java is an object. Object references are not pointers, and pointer manipulation is not allowed. This contributes to the general robustness of Java programs since pointer operations tend to be particularly nasty and bug-prone. Java also manages memory itself, thereby avoiding problems with allocation and deallocation of objects. It does not allow multiple inheritance like C++ does, but supports another type of reuse through the use of formal *interface* definitions.

Java is similar enough to C and C++ that it already feels familiar to most of the existing programming community. But it is different enough in important ways (memory management and cross-platform portability) that it is worth it for programmers to switch to a new language.

## Autonomy

For a software program to be autonomous, it must be a separate process or thread. Java applications are separate processes and as such can be long-running and autonomous. A Java application can communicate with other programs using sockets. In an application, an agent can be a separate thread of control. Java supports threaded applications and provides support for autonomy using both techniques.

Earlier, we described intelligent agents as autonomous programs or processes. As such, they are always waiting, ready to respond to a user request or a change in the environment. One question that comes to mind is "How does the agent know when something changes?" In our model, as with many others, the agent is informed by sending it an event. From an object-oriented design perspective, an event is nothing more than a method call or message, with information passed along on the method call that defines what happened or what action we want the agent to perform, as well as data required to process the event.

In Java there is an event-processing mechanism that is used in the Abstract Windowing Toolkit (AWT) to pass user-defined events such as mouse movements and menu selections to the underlying window components. Depending on the type of agent we are building, we may need to process these low-level events, such as when a GUI control

gets focus, or a window is resized or moved, or the user presses a key. The *awt.event* package also supports higher-level semantic events such as user actions.

## Intelligence

The intelligence in intelligent agents can range from hard-coded procedural or object-oriented logic to sophisticated reasoning and learning capabilities. While Prolog and Lisp are the two languages usually associated with AI programming, in recent years, much of the commercial AI work has been coded in C and C++. As a general-purpose, object-oriented programming language, Java provides all of the base functions needed to support these behaviors. In the past few years, Java has clearly become the favorite language for AI implementations.

There are two major aspects to AI applications, knowledge representation and algorithms that manipulate those representations. All knowledge representations are based on the use of slots or attributes that hold information regarding some entity, and links or references to other entities. Java objects can be used to encode this data and behavior as well as the relationships between objects. Standard AI knowledge representation such as frames, semantic nets, and if-then rules can all be easily and naturally implemented using Java.

## Mobility

There are several different aspects to mobility in the context of intelligent agents and intelligent applications. Java's portable bytecodes and Java archive (JAR) files allow groups of compiled Java classes to be sent over a network and then executed on the target machine. Java applets provide a mechanism for running Java code remotely via a Web browser. Other environments, such as the IBM Aglets, allow Java processes to be started, suspended, and moved.

One of the prime requirements for mobile programs is the ability to save the state of the running process, ship it off, and then resume where the process left off, only now it is running on a different system. Computer science researchers have explored this topic in great detail in relation to load balancing on distributed computer systems such as networks of workstations. Having homogeneous machines was a crucial part of making this work. Once again, the Java Virtual Machine comes to the rescue. By providing a standard computing environment for a Java process to run in, the JVM provides a homogeneous virtual machine that allows Java agents to move between heterogeneous hardware systems (from a Palm device, to a PC or Macintosh computer, to a Sun workstation, to an IBM eServer iSeries 400) without losing a beat.

Other aspects of Java also enable mobility. The JavaBean delegation event model allows dynamic registration of **EventSources** and **EventListeners**. Thus a mobile Java agent could "plug in" to an already-running server environment when it arrives, and then "unplug" itself, when it is time to move on. The *java.net* package provides network communications capability that allows mobile agents or a mobile agent infrastructure to talk to other servers and send serialized Java code and process state data over sockets.

The Java remote method invocation (java.rmi) package allows Java objects to call methods on other objects across a network, creating distributed object applications. The *rmiregistry* can be used to locate remote objects by name, and RMI will even dynamically load the class bytecodes when they are needed. By implementing the *java.rmi.Remote* interface and by throwing *java.rmi.RemoteExceptions*, methods in any Java class can be extended to work in distributed applications. Agents using RMI could easily be spread out, and even move across a network, while communicating with each other using remote method calls.

## Summary

In this chapter, we presented an introduction to artificial intelligence and intelligent agents. The major points include the following:

- The field of AI is approximately 40 years old. During that time, AI has evolved from trying to build general problem solvers using search, to building expert systems with deep but narrow domain knowledge, to combining knowledge, reasoning, and learning to solve difficult real-world problems.
- A software agent exhibits *intelligence* if it acts rationally. It uses knowledge, information, and reasoning to take reasonable actions in pursuit of a goal or goals.
- Most AI techniques are based on *symbol processing*. A set of tokens or symbols is used to represent knowledge and the state of the world, and algorithms manipulate those symbols to mimic high-level cognitive processing functions.
- *Neural networks* are another school of AI, based on a brain-like adaptive parallel computing model that does not use explicit symbols. Neural networks learn or adapt when exposed to data and correspond to cognitive functions such as low-level sensory processing and pattern recognition.
- The Internet, intranets, and the Web have caused an explosion in the amount of information available to people. The easy-to-use Web browsers have also attracted many new computer users to the on-line world. These factors have contributed to create a huge opportunity for *intelligent agents*, software that helps users do complex computing tasks.
- Intelligent agents must be able to recognize events, determine the meaning of those events, and then take actions on behalf of a user.
- Agents can be categorized by placing them in a three-dimensional space, where the axes are *agency*, the amount of autonomy an agent has; *intelligence*, the knowledge, reasoning, and learning capabilities of the agent; and *mobility*, the ability to move between systems in a network.
- *Reactive agents* are relatively simple agents that sense their world, and respond reflexively to external stimuli. *Deliberative agents* are more complex, using knowledge, reasoning, and even learning to plan and achieve their goals. *Collaborative agents* work together as a team, combining their knowledge and specialized skills to solve large problems. *BDI agents* have beliefs, desires, and intentions, with complex internal mental states and reasoning capabilities.

- Intelligent agents can also be classified by the functions they perform. *Interface agents* work as personal assistants to help users accomplish tasks. *Information agents* work to prevent information overload. They can either actively search out desired information on the Web, or filter out uninteresting or unwanted data as it arrives. *Domain-specific agents* can do on-line shopping, make travel arrangements, suggest a good book or CD, or help schedule a meeting.
- The Java language has many attributes that make it ideal for implementing agents and multiagent systems. These include its object-oriented style, support for threads, distributed objects, network-centric design and network access, and code portability.
- In the end, the success or failure of intelligent agents will depend on how much value they provide to their users.