Chapter 1

An Introduction to Software Agents

Jeffrey M. Bradshaw

Since the beginning of recorded history, people have been fascinated with the idea of non-human agencies.[1] Popular notions about androids, humanoids, robots, cyborgs, and science fiction creatures permeate our culture, forming the unconscious backdrop against which software agents are perceived. The word "robot," derived from the Czech word for drudgery, became popular following Karel Capek's 1921 play *RUR: Rossum Universal Robots.* While Capek's robots were factory workers, the public has also at times embraced the romantic dream of robots as "digital butlers" who, like the mechanical maid in the animated feature "The Jetsons," would someday putter about the living room performing mundane household tasks. Despite such innocuous beginnings, the dominant public image of artificially intelligent embodied creatures often has been more a nightmare than a dream. Would the awesome power of robots reverse the master-slave relationship with humans? Everyday experiences of computer users with the mysteries of ordinary software, riddled with annoying bugs, incomprehensible features, and dangerous viruses reinforce the fear that the software powering autonomous creatures would pose even more problems. The more intelligent the robot, the more capable of pursuing its own self-interest rather than its master's. The more humanlike the robot, the more likely to exhibit human frailties and eccentricities. Such latent concerns cannot be ignored in the design of software agents—indeed, there is more than a grain of truth in each of them!

Though automata of various sorts have existed for centuries, it is only with the development of computers and control theory since World War II that anything resembling autonomous agents has begun to appear. Norman (1997) observes that perhaps "the most relevant predecessors to today's intelligent agents are servomechanisms and other control devices, including factory control and the automated takeoff, landing, and flight control of aircraft." However, the agents now being contemplated differ in important ways from earlier concepts.

Significantly, for the moment, the momentum seems to have shifted from hardware to software, from the atoms that comprise a mechanical robot to the bits that make up a digital agent (Negroponte 1997).[2]

Alan Kay, a longtime proponent of agent technology, provides a thumbnail sketch tracing the more recent roots of software agents:

> "The idea of an agent originated with John McCarthy in the mid-1950's, and the term was coined by Oliver G. Selfridge a few years later, when they were both at the Massachusetts Institute of Technology. They had in view a system that, when given a goal, could carry out the details of the appropriate computer operations and could ask for and receive advice, offered in human terms, when it was stuck. An agent would be a 'soft robot' living and doing its business within the computer's world." (Kay 1984).

Nwana (1996) splits agent research into two main strands: the first beginning about 1977, and the second around 1990. Strand 1, whose roots are mainly in distributed artificial intelligence (DAI), "has concentrated mainly on deliberative-type agents with symbolic internal models." Such work has contributed to an understanding of "*macro* issues such as the interaction and communication between agents, the decomposition and distribution of tasks, coordination and cooperation, conflict resolution via negotiation, etc." Strand 2, in contrast, is a recent, rapidly growing movement to study a much broader range of agent types, from the moronic to the moderately smart. The emphasis has subtly shifted from *deliberation* to *doing*; from *reasoning* to *remote action.* The very diversity of applications and approaches is a key sign that software agents are becoming mainstream.

The gauntlet thrown down by early researchers has been variously taken up by new ones in distributed artificial intelligence, robotics, artificial life, distributed object computing, human-computer interaction, intelligent and adaptive interfaces, intelligent search and filtering, information retrieval, knowledge acquisition, end-user programming, programming-by-demonstration, and a growing list of other fields. As "agents" of many varieties have proliferated, there has been an explosion in the use of the term without a corresponding consensus on what it means. Some programs are called agents simply because they can be scheduled in advance to perform tasks on a remote machine (not unlike batch jobs on a mainframe); some because they accomplish low-level computing tasks while being instructed in a higher-level of programming language or script (Apple Computer 1993); some because they abstract out or encapsulate the details of differences between information sources or computing services (Knoblock and Ambite 1997); some because they implement a primitive or aggregate "cognitive function" (Minsky 1986, Minsky and Riecken 1994); some because they manifest characteristics of distributed intelligence (Moulin and Chaib-draa 1996); some because they serve a mediating role among people and programs (Coutaz 1990; Wiederhold 1989; Wiederhold 1992); some because they perform the role of an "intelligent assistant" (Boy 1991, Maes 1997) some because they can migrate in a self-directed way from computer to computer

(White 1996); some because they present themselves to users as believable characters (Ball et al. 1996, Bates 1994, Hayes-Roth, Brownston, and Gent 1995); some because they speak an agent communication language (Genesereth 1997, Finin et al. 1997) and some because they are viewed by users as manifesting intentionality and other aspects of "mental state" (Shoham 1997).

Out of this confusion, two distinct but related approaches to the definition of agent have been attempted: one based on the notion of agenthood as an *ascription* made by some person, the other based on a *description* of the attributes that software agents are designed to possess. These complementary perspectives are summarized in the section "What Is a Software Agent." The subsequent section discusses the "why" of software agents as they relate to two practical concerns: 1) simplifying the complexities of distributed computing and 2) overcoming the limitations of current user interface approaches. The final section provides a chapter by chapter overview of the remainder of the book.

What Is a Software Agent?

This section summarizes the two definitions of an agent that have been attempted: agent as an ascription, and agent as a description.

'Agent' as an Ascription

As previously noted, one of the most striking things about recent research and development in software agents is how little commonality there is between different approaches. Yet there is something that we intuitively recognize as a "family resemblance" among them. Since this resemblance cannot have to do with similarity in the details of implementation, architecture, or theory, it must be to a great degree a function of the eye of the beholder.[3] "Agent is that agent does"[4] is a slogan that captures, albeit simplistically, the essence of the insight that agency cannot ultimately be characterized by listing a collection of *attributes* but rather consists fundamentally as an *attribution* on the part of some person (Van de Velde 1995).[5]

This insight helps us understand why coming up with a once-and-for-all definition of agenthood is so difficult: one person's "intelligent agent" is another person's "smart object"; and today's "smart object" is tomorrow's "dumb program." The key distinction is in our expectations and our point of view. The claim of many agent proponents is that just as some algorithms can be more easily expressed and understood in an object-oriented representation than in a procedural one (Kaehler and Patterson 1986), so it sometimes may be easier for developers and users to interpret the behavior of their programs in terms of agents rather than as more run-of-the-mill sorts of objects (Dennett 1987).[6]

The *American Heritage Dictionary* defines an agent as "one that acts or has

the power or authority to act… or represent another" or the "means by which something is done or caused; instrument." The term derives from the present participle of the Latin verb *agere:* to drive, lead, act, or do.

As in the everyday sense, we expect a software agent to act on behalf of someone to carry out a particular task which has been delegated to it.[7] But since it is tedious to have to spell out every detail, we would like our agents to be able to infer what we mean from what we tell it. Agents can only do this if they "know" something about the context of the request. The best agents, then, would not only need to exercise a particular form of expertise, but also take into account the peculiarities of the user and situation.[8] In this sense an agent fills the role of what Negroponte calls a "digital sister-in-law:"

> "When I want to go out to the movies, rather than read reviews, I ask my sister-in-law. We all have an equivalent who is both an expert on movies and an expert on us. What we need to build is a digital sister-in-law.
>
> In fact, the concept of "agent" embodied in humans helping humans is often one where expertise is indeed mixed with knowledge of you. A good travel agent blends knowledge about hotels and restaurants with knowledge about you… A real estate agent builds a model of you from a succession of houses that fit your taste with varying degrees of success. Now imagine a telephone-answering agent, a news agent, or an electronic-mail-managing agent. What they all have in common is the ability to model you." (Negroponte 1997).

While the above description would at least seem to rule out someone claiming that a typical payroll system could be regarded as an agent, there is still plenty of room for disagreement (Franklin and Graesser 1996). Recently, for example, a surprising number of developers have re-christened existing components of their software as agents, despite the fact that there is very little that seems "agent-like" about them. As Foner (1993) observes:

> "… I find little justification for most of the commercial offerings that call themselves agents. Most of them tend to excessively anthropomorphize the software, and then conclude that it must be an agent because of that very anthropomorphization, while simultaneously failing to provide any sort of discourse or "social contract" between the user and the agent. Most are barely autonomous, unless a regularly-scheduled batch job counts. Many do not degrade gracefully, and therefore do not inspire enough trust to justify more than trivial delegation and its concomitant risks."[9]

Shoham provides a practical example illustrating the point that although anything *could* be described as an agent, it is not always advantageous to do so:

> "It is perfectly coherent to treat a light switch as a (very cooperative) agent with the capability of transmitting current at will, who invariably transmits current when it believes that we want it transmitted and not otherwise; flicking the switch is simply our way of communicating our desires. However, while this is a coherent view, it does not buy us anything, since we essentially understand the mechanism sufficiently to have a simpler, mechanistic description of its behavior." (Shoham 1993).[10]

Physical Stance	Predict based on physical characteristics and laws
Design Stance	Predic t based on what it is designed to do
Intentional Stance	Precit based on assumption of rational agency

Table 1. Dennett's three predictive stances (from Sharp 1992, 1993).

Dennett (1987) describes three predictive stances that people can take toward systems (table 1). People will choose whatever gives the most simple, yet reliable explanation of behavior. For natural systems (e.g., collisions of billiard balls), it is practical for people to predict behavior according to physical characteristics and laws. If we understand enough about a *designed* system (e.g., an automobile), we can conveniently predict its behavior based on its functions, i.e., what it is designed to do. However as John McCarthy observed in his work on "advice-takers" in the mid-1950's, "at some point the complexity of the system becomes such that the best you can do is give advice" (Ryan 1991). For example, to predict the behavior of people, animals, robots, or agents, it may be more appropriate to take a stance based on the assumption of rational agency than one based on our limited understanding of their underlying blueprints.[11]

Singh (1994) lists several pragmatic and technical reasons for the appeal of viewing agents as intentional systems:

> "They (i) are natural to us, as designers and analyzers; (ii) provide succinct descriptions of, and help understand and explain, the behaviour of complex systems; (iii) make available certain regularities and patterns of action that are independent of the exact physical implementation of the agent in the system; and (iv) may be used by the agents themselves in reasoning about each other."

'Agent' As a Description

A more specific definition of "software agent" that many agent researchers might find acceptable is: a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes (Shoham 1997). The requirement for continuity and autonomy derives from our desire that an agent be able to carry out activities in a flexible and intelligent manner that is responsive to changes in the environment without requiring constant human guidance or intervention. Ideally, an agent that functions continuously in an environment over a long period of time would be able to learn from its experience. In addition, we expect an agent that inhabits an environment with other agents and processes to be able to communicate and cooperate with them, and perhaps move from place to place in doing so.

All this being said, most software agents today are fairly fragile and special-purpose beasts, no one of which can do very much of what is outlined above in a generic fashion. Hence the term "software agent" might best be viewed as an umbrella term that covers a range of other more specific and limited agent types (Nwana 1996). Though as individuals the capabilities of the agents may be rather restricted, in their aggregate they attempt to simulate the functions of a primitive "digital sister-in-law," as particular ones intimately familiar with the user and situation exchange knowledge with others who handle the details of how to obtain needed information and services. Consistent with the requirements of a particular problem, each agent might possess to a greater or lesser degree attributes like the ones enumerated in Etzioni and Weld (1995) and Franklin and Graesser (1996):

- *Reactivity*: the ability to selectively sense and act
- *Autonomy*: goal-directedness, proactive and self-starting behavior
- *Collaborative behavior*: can work in concert with other agents to achieve a common goal
- *"Knowledge-level" (Newell 1982) communication ability:* the ability to communicate with persons and other agents with language more resembling human-like "speech acts" than typical symbol-level program-to-program protocols
- *Inferential capability*: can act on abstract task specification using prior knowledge of general goals and preferred methods to achieve flexibility; goes beyond the information given, and may have explicit models of self, user, situation, and/or other agents.
- *Temporal continuity:* persistence of identity and state over long periods of time[12]
- *Personality*: the capability of manifesting the attributes of a "believable" character such as emotion
- *Adaptivity*: being able to learn and improve with experience
- *Mobility*: being able to migrate in a self-directed way from one host platform to another.

To provide a simpler way of characterizing the space of agent types than would result if one tried to describe every combination of possible attributes, several in the agent research community have proposed various classification schemes and taxonomies.

For instance, AI researchers often distinguish between *weak* and *strong* notions of agency: agents of the latter variety are designed to possess explicit mentalistic or emotional qualities (Shoham 1997; Wooldridge and Jennings 1995). From the DAI community, Moulin and Chaib-draa have characterized agents by degree of problem-solving capability:

> "A *reactive* agent reacts to changes in its environment or to messages from other agents.… An *intentional* agent is able to reason on its intentions and beliefs, to create plans of actions, and to execute those plans.… In addition to intentional agent

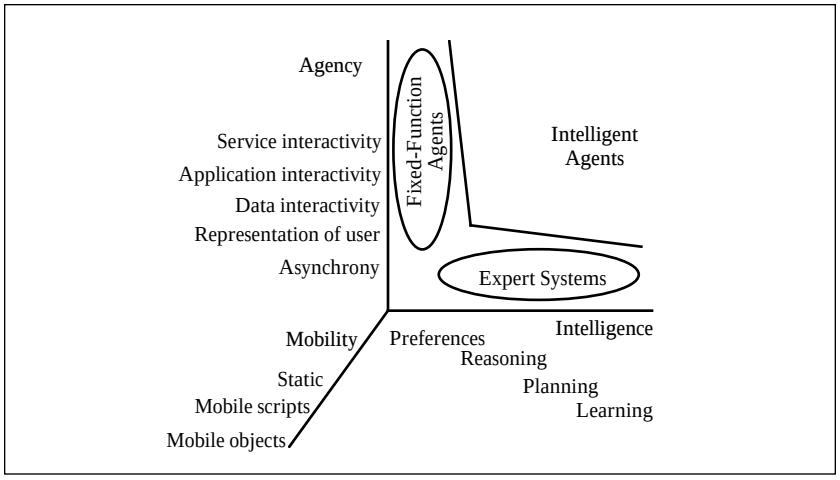

Figure 1. Scope of intelligent agents (Adapted from Gilbert et al. 1995).

capabilities, a *social* agent possesses explicit models of other agents." (Moulin and Chaib-draa 1996, pp. 8-9).

An influential white paper from IBM (Gilbert et al. 1995) described intelligent agents in terms of a space defined by the three dimensions of *agency, intelligence,* and *mobility* (figure 1):

"*Agency* is the degree of autonomy and authority vested in the agent, and can be measured at least qualitatively by the nature of the interaction between the agent and other entities in the system. At a minimum, an agent must run asynchronously. The degree of agency is enhanced if an agent represents a user in some way… A more advanced agent can interact with… data, applications,… services… [or] other agents.

Intelligence is the degree of reasoning and learned behavior: the agent's ability to accept the user's statement of goals and carry out the task delegated to it. At a minimum, there can be some statement of preferences… Higher levels of intelligence include a user model… and reasoning.… Further out on the intelligence scale are systems that *learn* and *adapt* to their environment, both in terms of the user's objectives, and in terms of the resources available to the agent…

Mobility is the degree to which agents themselves travel through the network… *Mobile scripts* may be composed on one machine and shipped to another for execution… *[Mobile objects* are] transported from machine to machine in the middle of execution, and carrying accumulated state data with them."

Nwana (1996) proposes a typology of agents that identifies other dimensions of classification. Agents may thus be classified according to:

- Mobility, as *static* or *mobile*
- Presence of a symbolic reasoning model, as *deliberative* or *reactive*

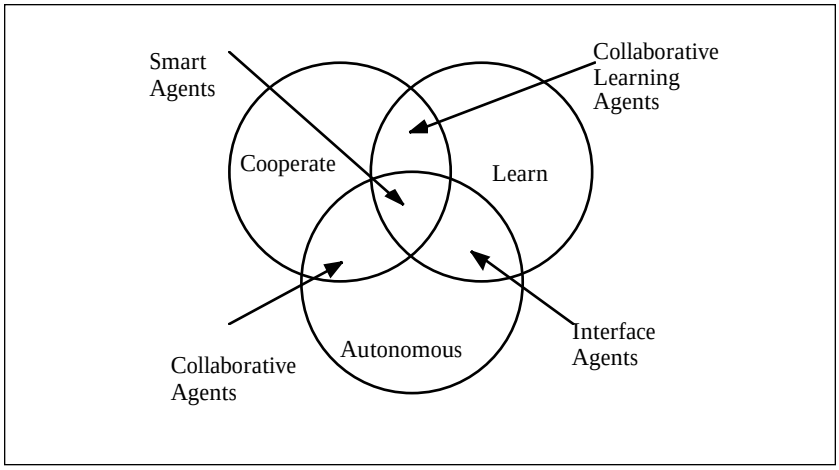

Figure 2. Typology based on Nwana's (Nwana 1996) primary attribute dimension.

- Exhibition of ideal and primary attributes, such as *autonomy, cooperation, learning.* From these characteristics, Nwana derives four agent types: *collaborative, collaborative learning, interface,* and *smart* (see figure 2).
- Roles, as *information* or *Internet*
- Hybrid philosophies, which combine two or more approaches in a single agent
- Secondary attributes, such as versatility, benevolence, veracity, trustworthiness, temporal continuity, ability to fail gracefully, and mentalistic and emotional qualities.

After developing this typology, Nwana goes on to describe ongoing research in seven categories: *collaborative agents, interface agents, mobile agents, information/Internet agents, reactive agents, hybrid agents,* and *smart agents.*

After listing several definitions given by others, Franklin and Graesser (1996) give their own: "an autonomous agent is a system situated within and part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." Observing that by this definition even a thermostat could qualify as an agent, they discuss various properties of agents and offer the taxonomy in figure 3 as one that covers most of the examples found in the literature. Below this initial classification, they suggest that agents can be categorized by control structures, environments (e.g., database, file system, network, Internet), language in which they are written, and applications.

Finally, Petrie (1996) discusses the various attempts of researchers to distinguish agents from other types of software. He first notes the difficulties in satisfactorily defining intelligence and autonomy. Then he shows how most of the

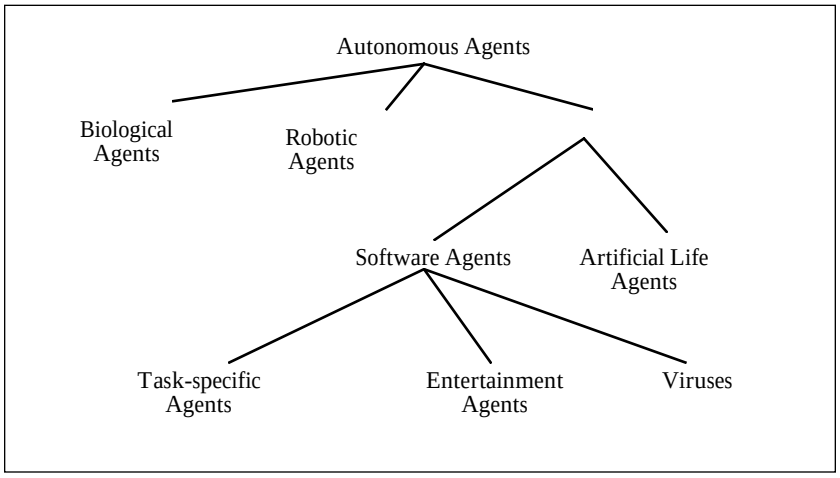

Figure 3. Franklin and Graesser's (1996) agent taxonomy.

current web-based searching and filtering "agents," though useful, "are essentially one-time query answering mechanisms" that are adequately described by the less glamorous computer science term "server." Similarly, "mobile process" would be a less confusing term than "mobile agent" for those Java applets whose only "agent-like" function is to allow processes to run securely on foreign machines. In contrast to these previous attempts to describe a set of unambiguous defining characteristics for agents in general, Petrie argues the case for one specific class: *typed-message agents.* Typed-message agents are distinguished from other types of software by virtue of their ability to communicate as a community using a shared message protocol such as KQML. In the shared message protocol, at least some of the message semantics "are typed and independent of the applications. And semantics of the message protocol necessitate that the transport protocol not be only client/server but rather a peer-to-peer protocol. An individual software module is not an agent at all if it can communicate with the other candidate agents only with a client/server protocol without degradation of the collective task performance."

Time and experience will ultimately determine both the meaning and the longevity of the term "agent." Like many other computing terms in common usage such as "desktop," "mouse," and "broker," it began with a metaphor but will end up denoting concrete software artifacts. As public exposure to useful and technically viable implementations of agent software increases, the term will either come to mean something that everyone understands because they have seen many examples of it, or it will fall into disuse because it describes a concept that is no longer appropriate. What is *unlikely* to disappear are the motivations that have incited the development of agent-based software. These are described in the following section.

Why Software Agents?

While the original work on agents was instigated by researchers intent on studying computational models of distributed intelligence, a new wave of interest has been fueled by two additional concerns of a practical nature: 1) simplifying the complexities of distributed computing and 2) overcoming the limitations of current user interface approaches.[13] Both of these can essentially be seen as a continuation of the trend toward greater abstraction of interfaces to computing services. On the one hand, there is a desire to further abstract the details of hardware, software, and communication patterns by replacing today's program-to-program interfaces with more powerful, general, and uniform agent-to-agent interfaces; on the other hand there is a desire to further abstract the details of the human-to-program interface by delegating to agents the details of specifying and carrying out complex tasks. Grosof (Harrison, Chess, and Kershenbaum 1995) argues that while it is true that point solutions not requiring agents could be devised to address many if not all of the issues raised by such problems, the aggregate advantage of agent technology is that it can address all of them at once.

In the following two subsections, I discuss how agents could be used to address the two main concerns I have mentioned. Following this, I sketch a vision of how "agent-enabled" system architectures of the future could provide an unprecedented level of functionality to people.

Simplifying Distributed Computing

Barriers to Intelligent Interoperability. Over the past several years, Brodie (1989) has frequently discussed the need for *intelligent interoperability* in software systems. He defines the term to mean intelligent cooperation among systems to optimally achieve specified goals. While there is little disagreement that future computing environments will consist of distributed software systems running on multiple heterogeneous platforms, many of today's most common configurations are, for all intents and purposes, disjoint: they do not really communicate or cooperate except in very basic ways (e.g., file transfer, print servers, database queries) (figure 4). The current ubiquity of the Web makes it easy to forget that until the last few years, computer systems that *could* communicate typically relied on proprietary or *ad hoc* interfaces for their particular connection. The current growth in popularity of object-oriented approaches and the development of a few important agreed-upon standards (e.g., TCP/IP, HTTP, IIOP, ODBC) has brought a basic level of encapsulated connectivity to many systems and services. Increasingly, these connections are made asynchronously through message passing, in situations where the advantages of loose coupling in complex cooperating systems can be realized (Mellor 1994; Perrow 1984; Shaw 1996).

We are now in the midst of a shift from the network operating system to In-

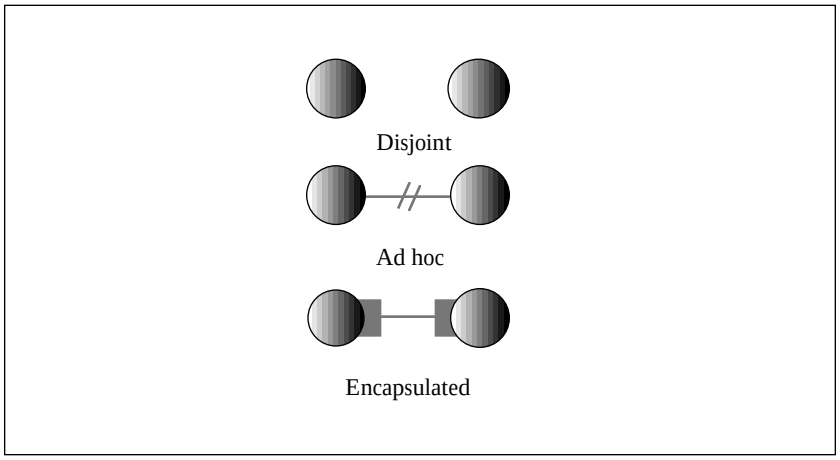

Figure 4. Evolution of system connectivity (Adapted from Brodie 1989).

ternet and intranet-based network computing (Lewis 1996). As this transition takes place, we are seeing the proliferation of operating system-independent interoperable network services such as naming, directory, and security. These, rather than the underlying operating systems, are defining the network, reducing the operating systems to commodities. Lewis (1996) asserts that Netscape is the best example of a vendor focused exclusively on such a goal. Federations of such vendors are defining standards-based operating system-independent services (directory, security, transactions, Web, and so forth), truly universal server-independent clients (Web browsers), and network-based application development support (Java, JavaScript, ActiveX). In such approaches, both the client and server operating systems become little more than a collection of device drivers.

Incorporating Agents as Resource Managers

A higher level of interoperability would require knowledge of the capabilities of each system, so that secure task planning, resource allocation, execution, monitoring, and, possibly, intervention between the systems could take place. To accomplish this, an intelligent agent could function as a global resource manager (figure 5).

Unfortunately, while a single agent might be workable for small networks of systems, such a scheme quickly becomes impractical as the number of cooperating systems grows. The activity of the single agent becomes a bottleneck for the (otherwise distributed) system. A further step toward intelligent interoperability is to embed one or more peer agents within each cooperating system (figure 6). Applications request services through these agents at a higher level corresponding more to user *intentions* than to specific *implementations*, thus providing

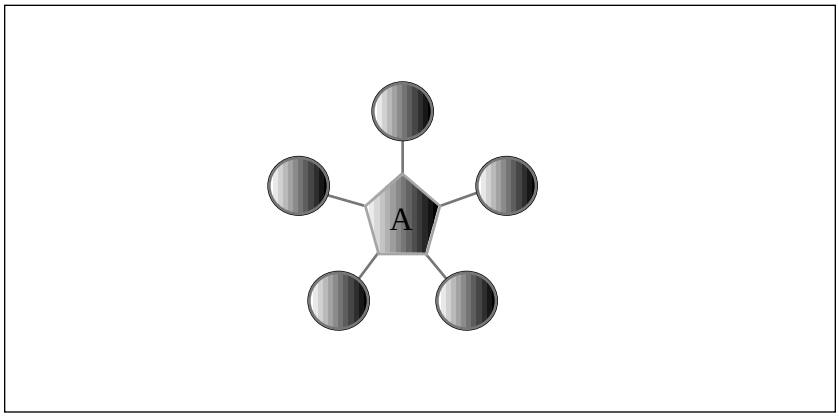

Figure 5. Cooperating systems with single agent as a global planner. Connections represent agent-to-application communication (Adapted from Brodie 1989).

a level of encapsulation at the planning level, analogous to the encapsulation provided at the lower level of basic communications protocols. As agents increasingly evolve from stationary entities to mobile ones, we will see an even more radical redefinition of distributed object computing within corporate networks and on the World Wide Web (Chang and Lange 1996). These scenarios presume, of course, timely agreement on basic standards ensuring agent interoperability (Gardner 1996; Lange 1996; Virdhagriswaran, Osisek, and O'Connor 1995; White 1997).

Overcoming User Interface Problems

Limitations of Direct Manipulation Interface. A distinct but complementary motivation for software agents is in overcoming problems with the current generation of user interface approaches. In the past several years, *direct manipulation* interfaces (Hutchins, Hollan, and Norman 1986; Shneiderman 1983; Shneiderman 1984; Smith, et al. 1982) have become the standard. For many of the most common user tasks, they are a distinct improvement over command-line interfaces. Since direct manipulation requires software objects to be visible, users are constantly informed about the kinds of things they can act upon. If, in addition, the objects have a natural correspondence to real-world or metaphorical counterparts, users can apply previously acquired experience to more quickly learn what the objects can do and how to do it. Many advantages of direct manipulation begin to fade, however, as tasks grow in scale or complexity. For example, anyone who has had much experience with iconic desktop interfaces knows that there are times when sequences of actions would be better automated than directly performed by the user in simple, tedious steps.[14] Several researchers have analyzed the limitations of passive artifact metaphors for com-

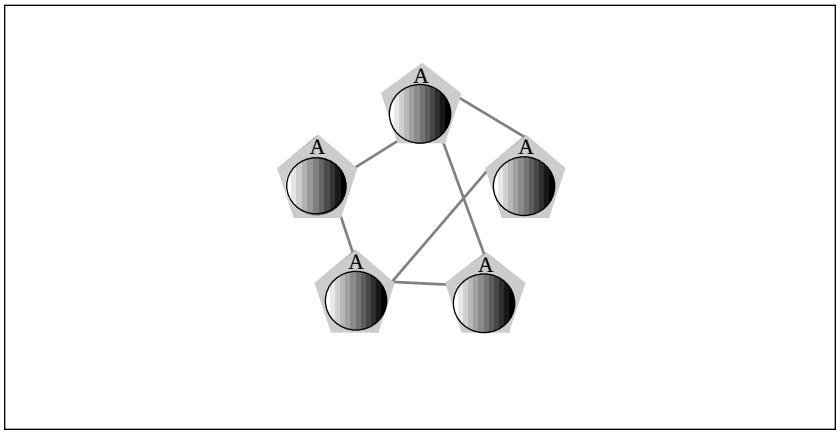

Figure 6. Cooperating systems with distributed agents. Connecting lines represent ongoing agent-to-agent communication (Adapted from Brodie 1989).

plex tasks (diSessa 1986; Erickson 1996; Kay 1990; Whittaker 1990). Among others, people are likely to encounter the following problems:

- *Large search space:* In large distributed systems it is difficult to find what we need through browsing or the use of traditional indexing methods. What is practical and possible for a few hundred items becomes unwieldy and impossible for several thousand.
- *Actions in response to immediate user interaction only:* Sometimes instead of executing an action immediately, we want to schedule it for a specific time in the future. Or, we may want to have software automatically react to system-generated events when we are away from the machine.
- *No composition:* With most direct manipulation interfaces, we cannot easily compose basic actions and objects into higher-level ones.
- *Rigidity:* The consistency that makes passive artifact interfaces predictable and easy-to-learn for simple tasks makes them brittle and untrustworthy for complex ones.
- *Function orientation:* Software is typically organized according to generic software functions rather than the context of the person's task and situation.
- *No improvement of behavior:* Traditional software does not notice or learn from repetitive actions in order to respond with better default behavior.

Indirect Management Using Agents

Researchers and developers are attempting to address these problems by combining the expression of user intention through direct manipulation with the notion of an *indirect management* style of interaction (Kay 1990). In such an ap-

proach, users would no longer be obliged to spell out each action for the computer explicitly; instead, the flexibility and intelligence of software agents would allow them to give general guidelines and forget about the details.

Many of the actions now performed by users could be delegated to various software agents. Thus, in a glimpse of the future, Tesler (1991) imagines the following directives being given by a person to a software agent:

- On what date in February did I record a phone conversation with Sam?
- Make me an appointment at a tire shop that is on my way home and is open after 6 PM.
- Distribute this draft to the rest of the group and let me know when they've read it.
- Whenever a paper is published on fullerene molecules, order a copy for my library.

Later on in the day, Tesler imagines the agent catching up to the person with these follow-up messages:

- You asked me when you last recorded a phone conversation with Sam. It was on February 27. Shall I play the recording?
- You scribbled a note last week that your tires were low. I could get you an appointment for tonight.
- Laszlo has discarded the last four drafts you sent him without reading any of them.
- You have requested papers on fullerene research. Shall I order papers on other organic microclusters as well?

Direct manipulation and indirect management approaches are not mutually exclusive. Interface agent researchers are not out to completely do away with computing as we know it, but more modestly hope that complementing see-and-point interfaces with ask-and-delegate extensions will help reduce required knowledge and simplify necessary actions while maintaining a sufficient level of predictability. Specifically, the use of software agents will eventually help overcome the limitations of passive artifact interfaces in the following ways (table 2):

- *Scalability:* Agents can be equipped with search and filtering capabilities that run in the background to help people explore vast sources of information.
- *Scheduled or event-driven actions:* Agents can be instructed to execute tasks at specific times or automatically "wake up" and react in response to system-generated events.
- *Abstraction and delegation:* Agents can be made extensible and composable in ways that common iconic interface objects cannot. Because we can "communicate" with them, they can share our goals, rather than simply process our commands. They can show us how to do things and tell us what went wrong (Miller and Neches 1987).
- *Flexibility and opportunism:* Because they can be instructed at the level of

Typical Limitations of Direct Manipulation Interfaces	Advantages of Agent-Oriented Approach
Large search space	Scalability
Actions in response to immediate user interaction only	Scheduled or event-driven actions
No composition	Abstraction and delegation
Rigidity	Flexibility and opportunism
Function orientation	Task orientation
No improvement of behavior	Adaptivity

Table 2. Typical limitations of direct manipulation interfaces and advantages of agent-oriented approach.

goals and strategies, agents can find ways to "work around" unforeseen problems and exploit new opportunities as they help solve problems.

- *Task orientation:* Agents can be designed to take the context of the person's tasks and situation into account as they present information and take action.
- *Adaptivity:* Agents can use learning algorithms to continually improve their behavior by noticing recurrent patterns of actions and events.

Toward Agent-Enabled System Architectures

In the future, assistant agents at the user interface and resource-managing agents behind the scenes will increasingly pair up to provide an unprecedented level of functionality to people. A key enabler is the packaging of data and software into components that can provide comprehensive information about themselves at a fine-grain level to the agents that act upon them.

Over time, large undifferentiated data sets will be restructured into smaller elements that are well-described by rich metadata, and complex monolithic applications will be transformed into a dynamic collection of simpler parts with self-describing programming interfaces. Ultimately, all data will reside in a "knowledge soup," where agents assemble and present small bits of information from a variety of data sources on the fly as appropriate to a given context (figure 7) (Neches et al. 1991; Sowa 1990). In such an environment, individuals and groups would no longer be forced to manage a passive collection of disparate documents to get something done. Instead, they would interact with active *knowledge media* (Barrett 1992; Bradshaw et al. 1993b; Brown and Duguid 1996; Glicksman, Weber, and Gruber 1992; Gruber, Tenenbaum, and Weber 1992) that integrate needed resources and actively collaborate with them on their tasks.

Figure 7 illustrates the various roles agents could play in an agent-enabled system architecture. Some could act in the role of intelligent user interface managers,

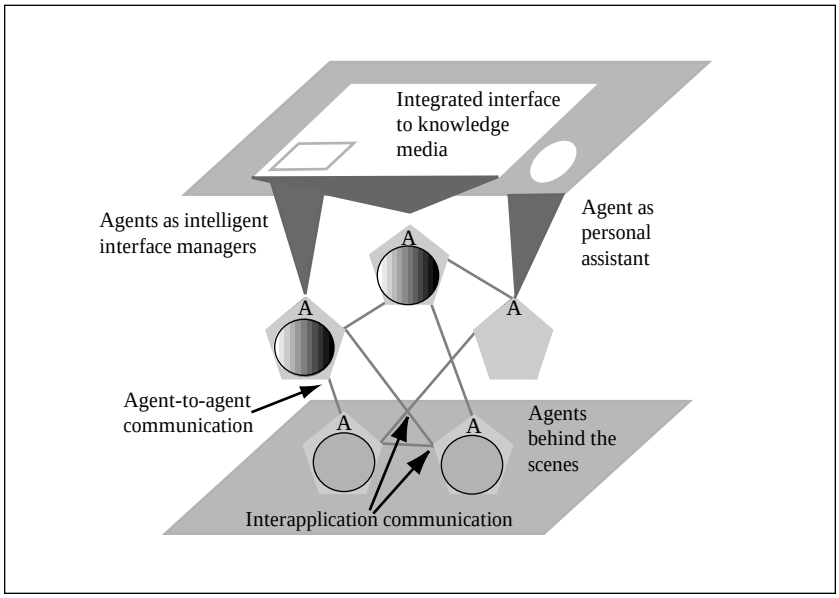

Figure 7 An agent-enabled system architecture.

drawing on the resources of other agents working behind the scenes (Arens et al. 1991; Browne, Totterdell, and Norman 1990; Kay 1990; Neal and Shapiro 1994; Sullivan and Tyler 1991). Such agents would work in concert to help coordinate the selection of the appropriate display modes and representations for the relevant data (Bradshaw and Boose 1992; Johnson et al. 1994), incorporating semantic representations of the knowledge in the documents to enhance navigation and information retrieval (Boy 1992; Bradshaw and Boy 1993; Gruber, Tenenbaum, and Weber 1992; Lethbridge and Skuce 1992; Mathé and Chen 1994). Because the layout and content of the views would be driven by context and configuration models rather than by hand-crafted user-interface code, significant economies could be realized as the data and software components are reused and semi-automatically reconfigured for different settings and purposes. Some agents might be represented explicitly to the user as various types of personal assistants (Maes 1997). Ideally, each software component would be "agent-enabled," however for practical reasons components may at times still rely on traditional interapplication communication mechanisms rather than agent-to-agent protocols.

Overview of the Book

The first subsection summarizes the first set of chapters under the heading of "Agents and the User Experience," which contain introductory pieces authored by proponents (and a critic) of agent technology. The next set, "Agents for

Learning and Intelligent Assistance," describes how agents have been used to enhance learning and provide intelligent assistance to users in situations where direct manipulation interfaces alone are insufficient. The final set, "Agent Communication, Collaboration, and Mobility," details various approaches to agent-oriented programming, agent-to-agent communication, and agent mobility, as well as the use of agents to provide intelligent interoperability between loosely-coupled components of distributed systems.

Agents and the User Experience

How Might People Interact with Agents? Norman's (1997) introductory chapter sets the stage for the first section of the book. "Agents occupy a strange place in the realm of technology," he opens, "leading to much fear, fiction, and extravagant claims." Because the new crop of intelligent agents differ so significantly in their computational power from their predecessors, we need to take into account the social issues no less than the technical ones if our designs are to be acceptable to people:

> "The technical aspect is to devise a computational structure that guarantees that from the technical standpoint, all is under control. This is not an easy task.
>
> The social part of acceptability is to provide reassurance that all is working according to plan… This is [also] a non-trivial task."

The reassurance that all is working according to plan is provided by an understandable and controllable level of feedback about the agent's intentions and actions. We must also think about how to accurately convey the agent's capabilities and limitations so that people are not misled in their expectations. Part of the problem is the natural overenthusiasm of agent researchers; part of the problem is people's tendency to falsely anthropomorphize.[15] Although designers can carefully describe agent capabilities and limitations within accompanying instructional manuals, it is even more important to find clever ways to weave this information naturally and effectively into the agent interface itself.

Safety and privacy are additional concerns. "How does one guard against error, maliciousness (as in the spread of computer viruses), and deliberate intent to pry and probe within one's personal records?" Legal policies to address these issues must be formulatedimmediately, at local, national, and global levels.

A final concern is how to design the appropriate form of interaction between agents and people. For example, how do ordinary people program the agent to do what they want? While programming-by-demonstration or simplified visual or scripting languages have been suggested, none of them seem adequate to specify the kinds of complex tasks envisioned for future intelligent agents.[16]

Since "agents are here to stay," we must learn how to cope with the dangers along with the positive contributions. "None of these negative aspects of agents are inevitable. All can be eliminated or minimized, but only if we

consider these aspects in the design of our intelligent systems."

Agents: From Direct Manipulation to Delegation. In his chapter, Nicholas Negroponte (1997), a longtime champion of agent technology (Negroponte 1970, 1995), extols the virtues of delegation in intelligent interfaces:

> "The best metaphor I can conceive of for a human-computer interface is that of a well-trained English butler. The "agent" answers the phone, recognizes the callers, disturbs you when appropriate, and may even tell a white lie on your behalf. The same agent is well trained in timing… and respectful of idiosyncrasies. People who know the butler enjoy considerable advantage over a total stranger. That is just fine." (Negroponte 1997.)

What will such digital butlers do? They will filter, extract, and present the relevant information from bodies of information larger than we could ordinarily digest on their own. They will act as "digital sisters-in-law," combining their knowledge of information and computing services with intimate knowledge about the person on whose behalf they are acting.

To create agents that are intelligent enough to perform these tasks to our level of satisfaction, we will need to re-open profound and basic questions of intelligence and learning that past AI research has left largely untouched. An understanding of decentralized approaches to intelligence is key: coherence can emerge from the activity of independent agents who coordinate their actions indirectly through shared external influences in their common environment.[17]

User interface design will also be decentralized. Instead of being an effort by professionals to produce the best interface for the masses, it will become an individual affair, driven more by the personalized intelligence of one's local agents than the blanket application of general human-factors knowledge. The agent's long-time familiarity with a person, gained by numerous shared experiences, will be critical to the new beyond-the-desktop metaphor. Though direct manipulation has its place, Negroponte believes that most people would prefer to run their home and office life with a gaggle of well-trained butlers.

Interface Agents: Metaphors with Character. The thesis of Laurel's (1997) chapter is that unabashed anthropomorphism in the design of interface agents is both natural and appropriate:

> "First, this form of representation makes optimal use of our ability to make accurate inferences about how a character is likely to think, decide, and act on the basis of its external traits. This marvelous cognitive shorthand is what makes plays and movies work… Second, the agent as character (whether humanoid, canine, cartoonish, or cybernetic) invites conversational interaction… [without necessarily requiring] elaborate natural language processing… Third, the metaphor of *character* successfully draws our attention to just those qualities that form the essential nature of an agent: responsiveness, competence, accessibility, and the capacity to perform actions on our behalf."

Recognizing the considerable resistance many will have to this idea, she responds

to some of the most common criticisms. First, is the objection to having to face "whining, chatting little irritants" each time you turn on the machine. Laurel notes that the problem is not "agents *per se*, but rather the traits they are assumed to possess." To address this problem, we must allow the traits of agents to be fully user-configurable. Another criticism is the indirection implied by the presence of an agent, "Why should I have to negotiate with some little dip in a bowtie when I know exactly what I want to do?" The answer is that if you know what you want to do and if you want to do it yourself, the agent should quickly get out of your way. Agent-based assistance should be reserved for tedious or complex tasks that you don't want to do yourself, and that you are comfortable entrusting to a software entity. Will people's acquired habit of bossing agents around lead them to treating real people the same way? Laurel argues that this is a real issue, but should not be handled by repression of the dramatic form—rather it should be addressed as an ethical problem for agent designers and the culture at large. Finally, there is the oft-heard criticism that "AI doesn't work." Laurel counters with examples of successful use of AI techniques in well-defined domains. Moreove, she asserts that most agents do not need a full-blown "artificial personality," but can be implemented much more simply.

Laurel concludes with a discussion of key characteristics of interface agents (agency, responsiveness, competence, and accessibility) and of an R&D agenda that includes an appreciation of the contribution of studies of story generation and dramatic character.[18]

Designing Agents as if People Mattered. Erickson (1997) explores the many difficulties surrounding adaptive functionality and the agent metaphor. With respect to adaptive functionality, he describes three design issues raised in a study of users of the DowQuest information retrieval system. In brief, people need to *understand* what happened and why when a system alters its response; they need to be able to *control* the actions of a system, even when it does not always wait for the user's input before it makes a move; and they need to *predict* what will happen, even though the system will change its responses over time. Several approaches to these problems have been suggested, including: providing users with a more accurate model of what is going on, managing overblown expectations of users at the beginning so they are willing to persist long enough to benefit from the system's incremental learning, and constructing a plausible 'fictional' model of what is going on.

Given the potential of the agent metaphor as a possible fiction for portraying system functionality, Erickson examines three strands of research that shed some light on how well this approach might work. Designed to encourage students to explore an interactive encyclopedia, the Guides project allowed researchers to observe the kinds of attributions and the level of emotional engagement people had with stereotypic characters that assisted in navigation. Erickson also reviews the extensive research that Nass and his colleagues have

performed on the tendency of people to use their knowledge of people and social rules to make judgments about computers. Finally, he discusses recent research on the reaction of people to extremely realistic portrayals of agents.

In the final section of the chapter, Erickson contrasts the desktop object and agent conceptual models, and argues that they can be used together in the same interface so long as they are clearly distinguished from one another. Specific computing functionality can be portrayed either as an object or an agent, depending on what is most natural. The desktop metaphor takes advantage of users' previous knowledge that office artifacts are visible, are passive, have locations, and may contain things. "Objects stay where they are: nice, safe predictable things that just sit there and hold things." Ontological knowledge of a different sort comes into play when the agent metaphor is employed. Our common sense knowledge of what agents can do tells us that, unlike typical desktop objects, they can notice things, carry out actions, know and learn things, and go places.[19] "Agents become the repositories for adaptive functionality." The overall conclusion is that research "which focuses on the portrayal of adaptive functionality, rather than on the functionality itself, is a crucial need if we wish to design agents that interact gracefully with their users."

Direct Manipulation Versus Agents: Paths to Predictable, Controllable, and Comprehensible Interfaces. Breaking with the tone of cautious optimism expressed in the preceding chapters, Shneiderman, a longtime advocate of direct manipulation, is troubled by the concept of intelligent interfaces in general:

> "First, such a classification limits the imagination. We should have much greater ambition than to make a computer behave like an intelligent butler or other human agent…
>
> Second, the quality of predictability and control are desirable. If machines are intelligent or adaptive, they may have less of these qualities…
>
> [Third,] I am concerned that if designers are successful in convincing the users that computers are intelligent, then the users will have a reduced sense of responsibility for failures…
>
> Finally,… [m]achines are not people… [and if] you confuse the way you treat machines with the way you treat people… you may end up treating people like machines."[20]

Shneiderman backs up his general concerns with lessons from past disappointments in natural language systems, speech I/O, intelligent computer-assisted instruction, and intelligent talking robots.

Shneiderman observes that agent proponents have not come up with good definitions of what is and is not an agent. "Is a compiler an agent? How about an optimizing compiler? Is a database query an agent? Is the print monitor an agent? Is e-mail delivered by an agent? Is a VCR scheduler an agent?" His examination of the literature reveals six major elements of the agent approach: anthropomorphic presentation, adaptive behavior, acceptance of vague goal specification, gives you what you need, works while you don't, and works

where you aren't. The first three, on closer examination, seem counterproductive, while the last three are good ideas that could be achieved by other means.

The alternative to a vision of computers as intelligent machines is that of predictable and controllable user interfaces, based on direct manipulation of representations of familiar objects. Shneiderman concludes with a description of two examples from his own lab (tree maps and dynamic queries) that show the power of visual, animated interfaces "built on promising strategies like informative and continuous feedback, meaningful control panels, appropriate preference boxes, user-selectable toolbars, rapid menu selection, easy-to-create macros, and comprehensible shortcuts." These, he argues, rather than vague visions of intelligent machines, will allow users to specify computer actions rapidly, accurately, and confidently.

Agents for Learning and Intelligent Assistance

Agents for Information Sharing and Coordination: A History and Some Reflections. The chapter by Malone, Grant, and Lai (1997) reviews more than ten years of seminal work on a series of programs which were intended to allow unsophisticated computer users to create their own cooperative work applications using a set of simple, but powerful, building blocks. The work is based on two key design principles, which each imply a particular kind of humility that should be required of agent designers:

> "1. Don't build computational agents that try to solve complex problems all by themselves. Instead, build systems where the boundary between what the agents do and what the humans do is a flexible one. We call this the principle of *semiformal systems*…
>
> 2. Don't build agents that try to figure out for themselves things that humans could easily tell them. Instead, try to build systems that make it as easy as possible for humans to see and modify the same information and reasoning processes their agents are using. We call this the principle of *radical tailorability*…"

Information Lens, the first program in the series, was a system for intelligent sorting and processing of electronic mail messages. *Object Lens* and *Oval* were successor programs providing much more general and tailorable environments that extended beyond the domain of electronic mail filtering.

The name "Oval" is an acronym for the four key components of the system: objects, views, agents, and links. "By defining and modifying templates for various semi-structured *objects*, users can represent information about people, tasks, products, messages, and many other kinds of information in a form that can be processed intelligently by both people and their computers. By collecting these objects in customizable folders, users can create their own *views* which summarize selected information from the objects. By creating semi-autonomous *agents*, users can specify rules for automatically processing this information in different ways at different times. Finally, *links*, are used for connecting and relating different objects" (Lai and Malone 1992).

The authors describe several different applications that were created to show the power and generality of the Oval approach.[21] All these demonstrate the surprising power that semiformal information processing can provide to people, and lend credence to the claim that people without formal programming skills can be enabled to create agent-based computing environments that suit their individual needs.

Agents that Reduce Work and Information Overload. While the developers of Oval have explored ways to simplify agent authoring, Pattie Maes and her colleagues at MIT have pursued an approach that allows personal assistants to *learn* appropriate behavior from user feedback (Maes 1997). The personal assistant starts out with very little knowledge and over time becomes more experienced, gradually building up a relationship of understanding and trust with the user:

> "[We] believe that the learning approach has several advantages over [end-user programming and knowledge-based approaches]… First, it requires less work from the end-user and application developer. Second, the agent can more easily adapt to the user over time and become customized to individual and organizational preferences and habits. Finally, the approach helps in transferring information, habits and know-how among the different users of a community."

A learning agent acquires its competence from four different sources. First, it can "look over the shoulder" of users as they perform actions. Second, it can learn through direct and indirect feedback from the user. Indirect feedback is provided when the user ignores the agent's suggested action. Third, the agent can learn from user-supplied examples. Finally, the agent can ask advice from other users' agents that have may have more experience with the same task.
Two assumptions determine whether the learning approach is appropriate for a given application:

1. *The application should involve a significant amount of repetitive behavior.* Otherwise, there would be no consistent situation-action patterns for the agent to learn.
2. *Repetitive behavior should be different for different users.* Otherwise, the behavior could be more efficiently hard-coded once and for all in a program, rather than implemented using learning agents.

Maes describes four agent-based applications built using the learning approach: electronic mail handling *(Maxims)*, meeting scheduling,[22] Usenet Netnews filtering *(Newt)*, and recommending books, music or other forms of entertainment *(Ringo)*[23] Through clever user feedback mechanisms and tailoring options, this approach provides a great deal of functionality from the combination of relatively simple mechanisms.

KidSim: Programming Agents without a Programming Language. Like Malone and his colleagues, Smith, Cypher, and Spohrer (1997) have focused their attention on the problem of agent authoring. What is unique to their application, however, is that they are trying to create a general and powerful tool for use by

children. To make this possible, they have adopted a "languageless" approach:

> "We decided that the question is not: what language can we invent that will be easier for people to use? The question is: should we be using a language at all?… We've come to the conclusion that since all previous languages have been unsuccessful…, *language itself is the problem.*.
>
> … [The] graphical user interface eliminated command lines by introducing visual representations for concepts and allowing people to directly manipulate those representations… Today all successful editors on personal computers follow this approach. But most programming environments do not. This is the reason most people have an easier time *editing* than *programming."*

KidSim[24] (now called "Cocoa") is a tool kit where children can build worlds populated by agents that they program themselves by demonstration and direct manipulation. Existing agents (simulation objects) can be modified, and new ones can be defined from scratch. Although agents cannot inherit from one another, they can share elements such as rules. In keeping with the design philosophy of direct manipulation, all elements of the simulation are visible in the interface.

"Languageless" programming is accomplished by combining two ideas: *graphical rewrite rules*, and *programming-by-demonstration.*[25] Graphical rewrite rules define transformations of a region of the game board from one state to another. Programming-by-demonstration is accomplished by letting the child put the system into "record mode" to capture all actions and replay them. A major strength of KidSim is that the results of recording user actions can be shown graphically, rather than as a difficult-to-understand script, as in most other such systems.

The authors describe a series of classroom studies in which children from ages eight to fourteen have used development versions of KidSim. The studies have led to a refinement of many of the concepts in KidSim, and have in turn provided convincing evidence that reasonably complex simulations of this sort can be constructed by very young children. No doubt there are many agent applications for adults that could take advantage of similar principles to make programming accessible.

Lifelike Computer Characters: The Persona Project at Microsoft. While many software agent researchers are content to make agents that are merely "useful," others seek the more ambitious goal of making "complete" agents that are highly visible in the user interface, project the illusion of being aware and intentioned, and are capable of emotions and significant social interaction. The *Persona* project (Ball et al. 1997) was formed to prototype possible interfaces to future computer-based assistants, in this case a "conversational, anthropomorphic computer character that will interact with the user to accept task assignments and report results." To be successful, such assistants will need to support interactive give and take including task negotiation and clarifying questions, understand how and when it is appropriate to interrupt the user with a report or request for

input, and acknowledge the social and emotional impacts of interaction.

The creation of lifelike computer characters requires a wide variety of technologies and skills, including speech recognition, natural language understanding, animation, and speech synthesis. A sophisticated understanding of subtle dialogue mechanisms and social psychology is also essential. To add to this challenge, all the necessary computational machinery to support these technologies must ultimately be able to run with acceptable performance on garden variety computing platforms.

The application chosen for the project described in this chapter involves an animated character (Personal Digital Parrot One, PDP1, or Peedy for short) that acts as a knowledgeable compact disc changer: "The assistant can be queried as to what CDs are available by artist, title or genre, additional information can be obtained about the CDs, and a playlist can be generated." Peedy responds verbally and by doing things to spoken commands in a restricted subset of natural language.

The application relies on two major technology components: reactive animation and natural language. *ReActor* represents a visual scene and accompanying entities such as cameras and lights hierarchically. The most interesting aspect of the animation is its reactivity, i.e., the fact that complex behaviors, including "emotion," can be triggered by user input. The natural language capability relies on the *Whisper* speech recognition module and on a broad-coverage natural language parser.

The creation of such prototypes has allowed Ball and his colleagues to discover and explore many little-understood aspects of human-computer interaction and to point the way toward the creation of increasingly sophisticated lifelike conversational assistants in the future.

Software Agents for Cooperative Learning. Boy's chapter (1997) examines the role of agents in learning technology. He briefly reviews significant trends of the past, including computer-based training, intelligent tutoring systems, interactive learning systems, and cooperative learning systems. Computer-supported cooperative learning (CSCL) builds on the lessons learned from these past approaches to provide an environment where knowledge is exchanged via active electronic documents.

Four requirements guide the design of active documents: 1. providing the *appropriate illusion* that is useful and natural for the user to understand its content; 2. providing *appropriate indexing and linking mechanisms* to connect the document with other relevant documents; 3. providing *adaptivity* so that over time the document becomes increasingly tailored to the information requirements of particular users; and 4. including *dynamic simulations* to enable people to understand aspects of complex situations that cannot be adequately represented using a static medium such as paper.

Software agents for cooperative learning are designed to transform standard electronic documents into active ones. Drawing on extensive past research expe-

rience with the *Situation Recognition and Analytical Reasoning (SRAR)* model and the *knowledge block representation* (Boy 1992; Boy 1991; Boy and Mathé 1993; Mathé and Chen 1994), he defines an agent in the context of this chapter to be "a software entity that can be represented by a knowledge block with an interface metaphor (appearance)."

As an example of an agent-based CSCL system Boy describes ACTIDOC, a prototype environment for active documents that has been applied in the domain of physics instruction. ACTIDOC documents consist of an ordered set of pages containing content and software agents (to make the content active). Each agent contains a name, a context, a set of triggering conditions, a set of internal mechanisms, and a set of interface metaphors. From Schank and Jona's (1991) six learning architectures, Boy derives classes of agents useful in active document design: *case-based learning agent, incidental learning agent, problem-solving agent, video database agent, simulation agent, and suggestive-questions agent.* Additionally he defines the roles of *evaluation, instructor aid,* and *networking agents.* These are illustrated using a physics example that demonstrates one way that agents can be used to make document content come alive.

The M System. Based on Minsky's *Society of Mind* (SOM) theory (Minsky 1986), the M system (Riecken 1997) is designed to provide intelligent assistance in a broad range of tasks through the integration of different reasoning processes *(societies of agents).* The architecture has previously been applied in the domains of music composition and intelligent user interface agents; this paper describes how M assists users of a desktop multimedia conferencing environment to classify and manage metaphorical electronic objects such as documents, ink, images, markers, white boards, copy machines, and staplers.

In the Virtual Meeting Room (VMR) application, participants collaborate using pen-based computers and a telephone:

> "Each user is supported by a personalized assistant, which attempts to recognize and define relationships between domain objects, based on the actions performed by the users and the resulting new states of the world. For example, VMR participants may perform actions on a group of electronic documents such as joining them as a set or annotating them collectively. M attempts to identify all domain objects and classify relationships between various subsets based on their physical properties and relevant user actions."

Within M there are five major reasoning processes, each of which are viewed as individual agents: *spatial, structural, functional, temporal, and causal.* Other more simple agents function as *supporting agents.* Functioning as a set of SOM *memory machines,* these supporting agents represent conceptual knowledge about things like color, shape, and spatial relationships. As an architecture of integrated agents, M dynamically generates, ranks, and modifies simultaneous theories about what is going on in the VMR world. As a faithful implementation of SOM theory, M provides for an I/O system, a spreading activation semantic net-

work (to implement Minsky's K-lines/polynemes), a rule-based system, a scripting system, a blackboard system (to implement Minsky's trans-frames and pronomes), and a history log file system.

To Riecken, an agent is fundamentally a simple, specialized "reasoning" process, whereas an assistant is composed of many "agencies of agents:" "To handle a common sense problem, one would not typically call on an agent—instead, one would want an assistant endowed with the talents of many integrated agents."

Agent Communication, Collaboration, and Mobility

An Overview of Agent-Oriented Programming. *Agent-oriented programming* (AOP) is a term that Shoham (1977) has proposed for the set of activities necessary to create software agents. What he means by 'agent' is "an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments." Agent-oriented programming can be thought of as a specialization of object-oriented programming approach, with constraints on what kinds of state-defining parameters, message types, and methods are appropriate. From this perspective, an agent is essentially "an object with an attitude."

An agent's "mental state" consists of components such as beliefs, decisions, capabilities, and obligations. Shoham formally describes the state in an extension of standard epistemic logics, and defines operators for obligation, decision, and capability. *Agent programs* control the behavior and mental state of agents. These programs are executed by an *agent interpreter.* In the spirit of speech act theory, interagent communication is implemented as speech act primitives of various types, such as inform, request, or refrain.

An agent interpreter assures that each agent will iterate through two steps at regular intervals: 1) read the current messages and update its mental state (including beliefs and commitments), and 2) execute the commitments for the current time, possibly resulting in further belief change. Shoham's original agent interpreter, AGENT-0, implements five language elements: *fact statements* ("Smith is an employee of Acme"), *communicative action statements* (inform, request, refrain), *conditional action statements* ("If, at time *t,* you believe that Smith is an employee of Acme, then inform agent A of the fact"), *variables,* and *commitment rules* ("If you receive message *x* while in the mental state *y,* perform action *z*").

The basic concepts described by Shoham have influenced the direction of many other agent researchers. He and his colleagues have continued their investigations on several fronts including mental states, algorithmic issues, the role of agents in digital libraries, and social laws among agents.

KQML as an Agent Communication Language. While Shoham's definition of an agent is built around a formal description of its mental state, other groups of researchers have taken agent communication as their point of departure.[26] In this

chapter, Finin, Labrou and Mayfield (1997) justify such a rationale as follows:

> "The building block for intelligent interaction is knowledge sharing that includes both mutual understanding of knowledge and the communication of that knowledge. The importance of such communication is emphasized by Genesereth, who goes so far as to suggest that an entity is a software agent if and only if it communicates correctly in an agent communication language (Genesereth and Ketchpel 1994). After all, it is hard to picture cyberspace with entities that exist only in isolation; it would go against our perception of a decentralized, interconnected electronic universe."

After an overview of the work of the Knowledge Sharing Effort (KSE) consortium (Neches et al. 1991) to tackle various issues relating to software agents and interoperability, the authors focus on one particular result of the effort: KQML (Knowledge Query Manipulation Language).

The authors suggest seven categories of requirements for an agent communication language:

- *Form*. It should be declarative, syntactically simple, and easily readable by people and programs.
- *Content*. A distinction should be made between the language that expresses communicative acts ("performatives") and the language that conveys the content of the message.
- *Semantics*. The semantics should exhibit those desirable properties expected of the semantics of any other language.
- *Implementation*. The implementation should be efficient, provide a good fit with existing software, hide the details of lower layers, and allow simple agents to implement subsets of the language.
- *Networking*. It should support all important aspects of modern networking technology, and should be independent of transport mechanism.
- *Environment*. It must cope with heterogeneity and dynamism.
- *Reliability*. It must support reliable and secure agent communication.

After a review of the features of KQML, the authors describe how the features of KQML support each of these requirements. The authors conclude by describing various applications of KQML and by giving a comparison with two related approaches: AOP and Telescript.

An Agent-Based Framework for Interoperability. Genesereth (1997) continues the theme of agent communication with his chapter on the role of agents in enabling interoperability, meaning that software created by different developers and at different times works together in seamless manner. He discusses two limitations of current software interoperability technologies: 1. they lack the ability to communicate definitions, theorems, and assumptions that may be needed for one system to communicate effectively with another, and 2. there is no general way of resolving inconsistencies in the use of syntax and vocabulary.

Like Finin and his colleagues, Genesereth has been a major contributor to

the KSE. His *ACL* (agent communication language) draws on three cornerstones of the KSE approach: *vocabularies*, (ontologies) *KIF (Knowledge Interchange Format)*, and *KQML*. The vocabulary of ACL is represented as a sophisticated open-ended dictionary of terms that can be referenced by the cooperating agents and applications.[27] KIF is a particular syntax for first order predicate calculus that provides for a common internal knowledge representation, an "inner" language for agents (Genesereth and Fikes 1992). It was originally developed by Genesereth's group and is currently being refined as part of an ISO standardization effort. In the ACL approach, KQML is viewed as a linguistic layer on top of KIF that allows information about the context (e.g., sender, receiver, time of message history) to be taken into account as part of agent messages. In short, "an ACL message is a KQML expression in which the 'arguments' are terms or sentences in KIF formed from words in the ACL vocabulary" (Genesereth and Ketchpel 1994).

The concept of a *facilitator* is central to ACL. Agents and facilitators are organized into a *federated system*, in which agents surrender their autonomy in exchange for the facilitator's services. Facilitators coordinate the activities of agents and provide other services such as locating other agents by name (white pages) or by capabilities (yellow pages), direct communication, content-based routing, message translation, problem decomposition, and monitoring. Upon startup, an agent initiates an ACL connection to the local facilitator and provides a description of its capabilities. It then sends the facilitator requests when it cannot supply its own needs, and is expected to act to the best of its ability to satisfy the facilitator's requests.

Genesereth describes several examples of applications and summarizes issues where further work is needed. ACL is an important step toward the ambitious long-range vision where "any system (software or hardware) can interoperate with any other system, without the intervention of human users or… programmers."

Agents for Information Gathering. The chapter by Knoblock and Ambite (1977) provides an in-depth example of the use of agents for an important class of problems: information gathering. The SIMS architecture for intelligent information agents is designed to provide:

1. *modularity* in terms of representing an information agent and information sources,
2. *extensibility* in terms of adding new information agents and information sources,
3. *flexibility* in terms of selecting the most appropriate information sources to answer a query,
4. *efficiency* in terms of minimizing the overall execution time for a given query, and
5. *adaptability* in terms of being able to track semantic discrepancies among models of different agents."

Each SIMS information agent provides expertise on a specific topic by drawing upon other information agents and data repositories. "An existing database or program can be turned into a simple information agent by building the appropriate interface code, called a *wrapper,* that will allow it to conform to the conventions of the [particular agent] organization… [Such an] approach greatly simplifies the individual agents since they need to handle only one underlying language. This arrangement makes it possible to scale the network into many agents with access to many different types of information sources." Agents that answer queries but do not originate them are referred to as *data repositories.*

"Each SIMS agent contains a detailed model of its domain of expertise [(an ontology)] and models of the information sources that are available to it. Given an information request, an agent selects an appropriate set of information sources, generates a plan to retrieve and process the data, uses knowledge about information sources to reformulate the plan for efficiency, and executes the plan." KQML is used as the communication language in which messages are transmitted among agents, while Loom (MacGregor 1990) is used as the content language in which queries and responses are formulated.

A learning capability helps agents improve their overall efficiency and accuracy. Three modes of learning are used: caching data that is frequently retrieved or which may be difficult to retrieve, learning about the contents of information sources so as to minimize the cost of retrieval, and analyzing the contents of information sources so as to refine its domain model.

To date, the authors have built information agents that plan and learn in the logistics planning domain. They are continuing to extend the planning and learning capabilities of these agents.

KAoS: Toward an Industrial-Strength Open Agent Architecture. It is ironic that as various sorts of agents are increasingly used to solve problems of software interoperability, we are now faced with the problem of incompatible competing agent frameworks:

> "The current lack of standards and supporting infrastructure has prevented the thing most users of agents in real-world applications most need: *agent interoperability* (Gardner 1996; Virdhagriswaran, Osisek, and O'Connor 1995). A key characteristic of agents is their ability to serve as universal mediators, tying together loosely-coupled, heterogeneous components—the last thing anyone wants is an agent architecture that can accommodate only a single native language and a limited set of proprietary services to which it alone can provide access."

The long-term objective of the KAoS (Knowledgeable Agent-oriented System) agent architecture (Bradshaw et al. 1997) is to address two major limitations of current agent technology: 1. failure to address infrastructure, scalability, and security issues; and 2. problems with the semantics and lack of principled extensibility of agent communication languages such as KQML. The first problem is addressed by taking advantage of the capabilities of commercial distributed object

products (CORBA, DCOM, Java) as a foundation for agent functionality, and supporting collaborative research and standards-based efforts to resolve agent interoperability issues. The second problem is addressed by providing an open agent communication *meta*-architecture in which any number of agent communication languages with their accompanying semantics could be accommodated.

Each KAoS agent contains a *generic agent instance*, which implements as a minimum the basic infrastructure for agent communication. Specific extensions and capabilities can be added to the basic structure and protocols through ordinary object-oriented programming mechanisms. Unlike most agent communication architectures, KAoS explicitly takes into account not only the individual message, but also the various sequences of messages in which it may occur. Shared knowledge about message sequencing conventions *(conversation policies)* enables agents to coordinate frequently recurring interactions of a routine nature simply and predictably. *Suites* provide convenient groupings of conversation policies that support a set of related services (e.g., the *Matchmaker suite*). A starter set of suites is provided in the architecture but can be extended or replaced as required.

The authors experience with KAoS leads them to be "optimistic about the prospects for agent architectures built on open, extensible object frameworks and [they] look forward to the wider availability of interoperable agent implementations that will surely result from continued collaboration."

Communicative Actions for Artificial Agents. Cohen and Levesque's (1997) chapter identifies major issues in the design of languages for interagent communication, with specific application to KQML:

> "[The] authors of KQML have yet to provide a precise semantics for this language, as is customary with programming languages.[28] Without one, agent designers cannot be certain that the interpretation they are giving to a "performative" is in fact the same as the one some other designer intended it to have. Moreover, the lack of a semantics for communication acts leads to a number of confusions in the set of reserved "performatives" supplied. Lastly, designers are left unconstrained and unguided in any attempt to extend the set of communication actions."

In KQML, communicative actions are considered to belong to a specific class of speech acts called "performatives" which, in natural language, are utterances that succeed simply because speakers say or assert they are doing so (e.g., "I hereby bequeath my inheritance to my daughter"). The authors identify three general difficulties with KQML. First, the definitions of the performatives suffer from *ambiguity and vagueness*. Second, there are *misidentified performatives*, that should instead be classes as directives (e.g., requests) or assertives (e.g., informs). Third, there are *missing performatives*, such as the commissives (e.g., promises).

The authors respond to these difficulties with an outline an analysis of rational action upon which their theory of speech acts rests. They then show how the speech acts of requesting and informing can be defined in terms of the primitives from this theory. The implications for future KQML design decisions are

twofold. First, if developers are allowed to extend the set of KQML performatives, they must provide both correct implementations of the directive force of new actions as well as assure that the new actions enter into old and new conversation patterns correctly. Second, if the communication primitives are to be handled independently of the content of the message, developers must not allow any attitude operators in the content (e.g., not permit an agent who says that it requests an act to also say that it does not want the act done).

The authors provide a comparison with other agent communication languages including AOP, Telescript, and their own Open Agent Architecture (OAA) approach (Cohen and Cheyer 1994). Additional work in joint intention theory (Cohen 1994; Cohen and Levesque 1991; Smith and Cohen 1995) is required to clarify how communicative actions function in the initiation of team behavior, and how they may be able to predict the structure of finite-state models of interagent conversations as used in agent architectures such as KAoS.[29]

Mobile Agents. Telescript is an object-oriented remote programming language that is designed to address the problem of interoperability for network services (White 1997). What PostScript did for cross-platform, device-independent documents, Telescript aims to do for cross-platform, network-independent messaging:

> "In Telescript technology, mobile agents *go* to *places*, where they perform tasks on behalf of a user. Agents and places are completely programmable, but they are managed by security features such as *permits*, *authorities*, and *access controls*. Telescript technology is portable, allowing it to be deployed on any platform, over any transport mechanism, and through assorted media—wireline and wireless. Telescript technology can also handle different content types, including text, graphics, animations, live video, and sounds. Telescript technology turns a network into an open platform.[30] Simplified development, portability, and support for rich message content make the technology applicable to a range of communicating applications, from workflow automation to information services and from network management to electronic markets" (General Magic 1994).

Telescript technology allows developers to bundle data and procedures into an agent that will be sent over the network and executed remotely on the server.[31] The Telescript agent carries its own agenda and may travel to several places in succession in order to perform a task. Security for host systems is of paramount concern. The Telescript runtime engine can be set to prevent agents from examining or modifying the memory, file system, or other resources of the computers on which they execute. Moreover, each agent carries securely formatted and encrypted identification tickets that must be checked by the host before running code. The ticket may also carry information about what kinds of tasks the agent is permitted to perform, and the maximum resources it is allowed to expend.

White provides a motivation for mobile agent technology in terms of several example applications. A comprehensive overview of Telescript technologies and programming model and a brief discussion of related work round out the chapter.

Parting Thoughts

Readers may legitimately complain about the idiosyncratic selection of chapters for this book. Significant research topics and important bodies of work have certainly been neglected[32] although I hope that some of this may be rectified in a subsequent volume. What I have tried to provide is convenient access to an initial collection of exemplars illustrating the diversity of problems being addressed today by software agent technology. Despite the fact that the solutions described here will ultimately be replaced by better ones; regardless of whether the term "software agent" survives the next round of computing buzzword evolution, I believe that the kinds of issues raised and lessons learned from our exploration of software agent technology points the way toward the exciting developments of the next millennium.

Acknowledgments

Heartfelt thanks are due to Kathleen Bradshaw and to Ken Ford and Mike Hamilton of AAAI Press, who nurtured this project from the beginning and patiently sustained it to a successful end. I am grateful to the authors of the individual chapters for allowing their contributions to appear in this volume, and for many stimulating discussions. Peter Clark, Jim Hoard, and Ian Angus provided helpful feedback on an earlier draft of this chapter. Significant support for this effort was provided by Boeing management including Cathy Kitto, Ken Neves, and Al Erisman; and by my colleagues in the agent research group: Bob Carpenter, Rob Cranfill, Renia Jeffers, Luis Poblete, Tom Robinson, and Amy Sun. The writing of this chapter was supported in part by grant R01 HS09407 from the Agency for Health Care Policy and Research to the Fred Hutchison Cancer Research Center.

Notes

1. Works by authors such as Schelde (1993), who have chronicled the development of popular notions about androids, humanoids, robots, and science fiction creatures, are a useful starting point for software agent designers wanting to plumb the cultural context of their creations. The chapter "Information beyond computers" in Lubar (1993) provides a useful grand tour of the subject. See Ford, Glymour, and Hayes (1995) for a delightful collection of essays on android epistemology.

2. This is perhaps an overstatement, since researchers with strong roots in artificial life (a-life) and robotics traditions have continued to make significant contributions to our understanding of autonomous agents (Maes 1993; Steels 1995). Although most researchers in robotics have concerned themselves with agents embodied in hardware, some have also made significant contributions in the area of software agents. See Etzioni (1993) for arguments that software presents a no-less-attractive platform than hardware for the investigation of complete agents in real-world environments. Williams and Nayak (1996) describe a software-hardware hybrid agent concept they call immobile robots *(immobots).*

3. For example, see the operational definition proposed by Shoham: "An agent is an entity whose state is *viewed as* consisting of mental components such as beliefs, capabilities, choices, and commitments."

4. With apologies to Oliver Goldsmith (Bartlett and Beck 1980, p. 369:9).

5. Alan Turing (Turing 1950) proposed what was perhaps the first attempt to operationalize a test for machine intelligence using the criterion of human ascription. Research on believable agents (Bates et al. 1994), lifelike computer characters (Ball 1996), and agent-based computer games (Tackett and Benson 1985) carries on in the same tradition, aiming to produce the most realistic multimedia experience of computer-based agents possible. As discovered by organizers of the Loebner Prize Competitions (Epstein 1992) and the AAAI Robot Competitions (Hinkle, Kortenkamp, and Miller 1996; Simmons 1995), one significant challenge in objectively judging results of competitions based on pure ascription and performance measures is that unless the evaluation criteria are well-thought out, agents or robots relying on cheap programming tricks may consistently outperform those who may be demonstrating some more legitimate kind of machine intelligence.

6. Russell and Norvig (1995, p. 821) discuss the fact that while ascribing beliefs, desires, and intentions to agents (the concept of an *intentional stance*) might help us avoid the paradoxes and clashes of intuition, the fact that it is rooted in a relativistic folk psychology can create other sorts of problems. Resnick and Martin (Martin 1988; Resnick and Martin 1990) describe examples of how, in real life, people quite easily and naturally shift between the different kinds of descriptions of designed artifacts (see footnote 11). Erickson (1997), Laurel (1997), and Shneiderman (1997) offer additional perspectives on the consequences of encouraging users to think in terms of agents.

7. Milewski and Lewis (1994) review the organizational psychology and management science literature regarding delegation. They draw implications for agent design, including delegation cost-benefit tradeoffs, kinds of communication required, determinants of trust, performance controls, and differences in personality and culture.

8. "The difference between an automaton and an agent is a somewhat like the difference between a dog and a butler. If you send your dog to buy a copy of the *New York Times* every morning, it will come back with its mouth empty if the news stand happens to have run out one day. In contrast, the butler will probably take the initiative to buy you a copy of the *Washington Post*, since he knows that sometimes you read it instead" (Le Du 1994), my translation.

9. Newquist (1994) gives a similarly-flavored critique of the overhyping of "intelligence" in various products.

10. Shoham goes on to cite the following statement by John McCarthy, who distinguishes between the "legitimacy" of describing mental qualities to machines and its "usefulness" "To ascribe certain *beliefs*, *free will*, *intentions*, *consciousness*, *abilities* or *wants* to a machine or computer program is *legitimate* when such an ascription expresses the same information about the machine that it expresses about a person. It is *useful* when the ascription helps us understand the structure of the machine, its past or future behavior, or how to repair or improve it. It is perhaps never *logically required* even for humans, but expressing reasonably briefly what is actually known about the state of the machine in a particular situation may require mental qualities or qualities isomorphic to them. Theories of belief, knowledge and wanting can be constructed for machines in a simpler setting than for humans, and later applied to humans. Ascription of mental qualities is *most straightforward* for machines of known structure such as thermostats and computer operating systems, but is *most useful* when applied to entities whose structure is very incompletely known" (McCarthy 1979).

11. Of course, in real life, people quite easily and naturally shift between the different kinds of descriptions. For example, Resnick and Martin report the following about their research with children building LEGO robots: "As students play with artificial creatures,

we are particularly interested in how the students think about the creatures. Do they think of the LEGO creatures as machines, or as animals? In fact, we have found that students (and adults) regard the creatures in many different ways. Sometimes students view their creatures on a *mechanistic* level, examining how one LEGO piece makes another move. At other times, they might shift to the *information* level, exploring how information flows from one electronic brick to another. At still other times, students view the creatures on a *psychological* level, attributing intentionality or personality to the creatures. One creature 'wants' to get to the light. Another creature 'likes' the dark. A third is 'scared' of loud noises.

Sometimes, students will shift rapidly between levels of description. Consider, for example, the comments of Sara, a fifth-grader (Martin 1988). Sara was considering whether her creature would sound a signal when its touch sensor was pushed:

'It depends on whether the machine wants to tell… if we want the machine to tell us… if we tell the machine to tell us.'

Within a span of ten seconds, Sara described the situation in three different ways. First she viewed the machine on a psychological level, focusing on what the machine 'wants.' Then she shifted intentionality to the programmer, and viewed the programmer on a psychological level. Finally, she shifted to a mechanistic explanation, in which the programmer explicitly told the machine what to do.

Which is the correct level? That is a natural, but misleading question. Complex systems can be meaningfully described at many different levels. Which level is 'best' depends on the context: on what you already understand and on what you hope to learn. In certain situations, for certain questions, the mechanistic level is the best. In other situations, for other questions, the psychological level is best. By playing with artificial creatures, students can learn to shift between levels, learning which levels are best for which situations." (Resnick and Martin 1990).

12. Ideally, this would include some notion of episodic memory. Unfortunately, only two major examples of "agents" incorporating episodic memory in the literature easily come to mind: Winograd's (1973) SHRDLU and Vere and Bickmore's (1990) "basic agent." For a thought-provoking look into the consequences of a future where a personal "agent" might become the ultimate cradle-to-grave companion, experiencing and remembering every event of a lifetime, see "The Teddy" chapter in Norman (1992).

13. In his widely cited article "Eye on the Prize" (Nilsson 1995), Nilsson discusses the shift of emphasis in AI from inventing general problem-solving techniques to what he calls *performance programs*, and gives his reasons for believing that there will soon be a reinvigoration of efforts to build programs of "general, humanlike competence."

14. While macro and scripting languages are technically adequate to solve this problem, it seems unlikely that the majority of "end users" will ever want to endure what it takes to become proficient with them: "In the past two decades there have been numerous attempts to develop a language for end users: Basic, Logo, Smalltalk, Pascal, Playground, HyperTalk, Boxer, etc. All have made progress in expanding the number of people who can program. Yet as a percentage of computer users, this number is still abysmally small. Consider children trying to learn programming… We hypothesize that fewer than 10% of there children who are taught programming continue to program after the class ends… Eliot Soloway states, 'My guess is that the number… is less than 1%! Who in their right mind would use those languages—any of them—after a class?'" (Smith, Cypher, and Spohrer 1997). While the software agent perspective does not obviate the need for end-user programming, I believe it has potential as one means of simplifying some of the conceptual barriers that users and developers face in designing and understanding complex systems.

15. Fortunately, people have a lot of experience in judging the limitations of those with whom they communicate: "Sometimes people overstate what the computer can do, but what people are extremely good at is figuring out what they can get away with. Children can size up a substitute teacher in about five minutes" (Kahle 1993). For evidence that developers of intelligent software are no less prone than other people to overestimate the capabilities of their programs, see McDermott (1976).

16. *Automatic programming* is an enterprise with a long history of insatiable requirements and moving expectations. For example, Rich and Waters (1988) remind us that "compared to programming in machine code, assemblers represented a spectacular level of automation. Moreover, FORTRAN was arguably a greater step forward than anything that has happened since. In particular, it dramatically increased the number of scientific end users who could operate computers without having to hire a programmer." Today, no one would call FORTRAN a form of automatic programming, though in 1958 the term was quite appropriate. The intractability of fully-automated, completely-general programming is analogous to the problem of automated knowledge acquisition (Bradshaw et al. 1993a; Ford et al. 1993). As Sowa observes: "Fully automated knowledge acquisition is as difficult as unrestricted natural language understanding. The two problems, in fact, are different aspects of exactly the same problem: the task of building a formal model for some real world system on the basis of informal descriptions in ordinary language. Alan Perlis once made a remark that characterizes that difficulty: *You can't translate informal specifications into formal specifications by any formal algorithm.*" (Sowa 1989).

17. Van de Velde (1995) provides a useful discussion of the three coupling mechanisms which can enable coordination between multiple agents: knowledge-level, symbol-level, and structural. Symbol-level coupling occurs when agents coordinate by exchange of symbol structures (e.g., "messages") and knowledge-level coupling occurs when an agent "rationalizes the behavior of multiple agents by ascribing goals and knowledge to them that, assuming their rational behavior, explains their behavior" (i.e., through taking an intentional stance). Structural coupling, as discussed extensively by Maturana and Varela (1992), occurs when two agents "coordinate without exchange of representation,… by being mutually adapted to the influences that they experience through their common environment… For example, a sidewalk… plays a coordinating role in the behavior of pedestrians and drivers… [and] the coordination of [soccer] players (within and across teams) is mediated primarily by… the ball." Similarly, as Clancey (1993) argues, the usefulness of the blackboard metaphor is that it provides an external representation that regulates the coordination between multiple agents.

18. These latter issues are discussed in more detail in a Laurel's (1991) book *Computers as Theatre*.

19. It is also easy for people to assume less tangible qualities about agents like that they are internally consistent, are rational, act in good faith, can introspect, can cooperate to achieve common goals, and have a persistent mental state.

20. A more blunt criticism of agents is voiced by Jaron Lanier (1996), who writes, "The idea of 'intelligent agents' is both wrong and evil. I also believe that this is an issues of real consequence to the near-term future of culture and society. As the infobahn rears its gargantuan head, the agent question looms as a deciding factor in whether this new beast will be much better than TV, or much worse." See also his extended online debate with Pattie Maes in Lanier and Maes (1996).

21. Several commercial products have subsequently incorporated Ovallike capability, though with less generality and sophistication. These include cooperative work tools and databases for semi-structured information such as Lotus Notes (Greif 1994) and Caere

Pagekeeper, as well as mail readers with rule-based message sorting. Workflow management tools with some of these capabilities have also appeared.

22. For other approaches to defining agents for scheduling and calendar management tasks, see Kautz et al. (1993); Mitchell et al. (1994).

23. Maes has formed Firefly Network, Inc. in order to extend the technology developed in Ringo to the Web. Her *firefly* service uses knowledge about people with similar tastes and interests in music and movies as a means of personalizing its recommendations.

24. A sort of Java for kids.

25. For a similar approach that relies on graphical rewrite rules, see Repenning's (1995) Agentsheets.

26. This is of course a caricature of both approaches: Shoham does not ignore the importance of agent communication in AOP; neither would most agent communication researchers argue that some representation of "mental state" is unnecessary. Davies' (1994) Agent-K language is an attempt to build a hybrid that extends AGENT-0 to use KQML for communication.

27. One well-known tool that has been used to construct such vocabularies is *Ontolingua* (Gruber 1992a, 1992b).

28. Recent efforts to provide a semantic foundation for KQML are described in Labrou (1996) and Labrou and Finin (1994). Another more general approach to agent language semantics is currently under development by Smith and Cohen (1995).

29. Such a strategy parallels the approach of Rosenschein, who designed a compiler that generates finite state machines whose internal states can be proved to correspond to certain logical propositions about the environment (Kaelbling and Rosenschein 1990; Rosenschein 1995).

30. General Magic is working hard to assure that Telescript can take maximum advantage of developments in Internet technology. Its Tabriz AgentWare and Agent Tools products (General Magic 1996) integrate Telescript and Web technologies, and White has proposed a common agent platform intended to enable interoperability between Telescript and other mobile agent technology (White 1996).

31. With respect to the relationship between Telescript, Tabriz, and Java, General Magic writes: "It is important to note that Telescript and Java are complementary, interoperable languages. Telescript agents can set parameters for Java applets and Java applets can call Telescript operations on the server. This interoperability allows developers to create solutions that leverage the power of the two environments: Java can be used to create and manage compelling user experiences, while Tabriz can manage transactions, instructions, events, and processes" (General Magic 1996).

32. Much more, for example, could have been included about the veritable explosion of work on agents and the Internet (Bowman et al. 1994; Cheong 1996; Etzioni and Weld 1995; Weld, Marks, and Bobrow 1995). None of the many applications of agent technology in complex application areas ranging from digital libraries (Paepcke et al. 1996; Wiederhold 1995) to systems management (Benech, Desprats, and Moreau 1996; Rivière, Pell, and Sibilla 1996) to manufacturing (Balasubramanian and Norrie 1995) could be included. We have slighted the whole fields of artificial life (Langton 1995) situated automata (Brooks 1990; Kaelbling and Rosenschein 1991), learning and adaptation (Gaines 1996; Maes 1995; Sen et al. 1996), and large portions of the voluminous literature on DAI and other fields where important related work is taking place.

References

Apple. 1993. *AppleScript Language Guide.* Reading, Mass.: Addison-Wesley.

Arens, Y.; Feiner, S.; Foley, J.; Hovy, E.; John, B.; Neches, R.; Pausch, R.; Schorr, H.; and Swartout, W. 1991. Intelligent User Interfaces, Report ISI/RR-91-288, USC/Information Sciences Institute, Marina del Rey, California.

Balasubramanian, S., and Norrie, D. H. 1995. A Multi-Agent Intelligent Design System Integrating Manufacturing and Shop-Floor Control. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95),* ed. V. Lesser, 3–9. Menlo Park, Calif.: AAAI Press.

Ball, G. 1996. Lifelike Computer Characters (LCC-96) Schedule and Workshop Information. http://www.research.microsoft.com/lcc.htm.

Ball, G.; Ling, D.; Kurlander, D.; Miller, J.; Pugh, D.; Skelly, T.; Stankosky, A.; Thiel, D.; Dantzich, M. V; and Wax, T. 1996. Lifelike Computer Characters: The Persona Project at Microsoft Research. In *Software Agents,* ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Barrett, E. 1992. Sociomedia: An Introduction. In *Sociomedia: Multimedia, Hypermedia, and the Social Construction of Knowledge,* ed. E. Barrett, 1–10. Cambridge, Mass.: MIT Press.

Bartlett, J., and Beck, E. M., eds. 1980. *Familiar Quotations.* Boston, Mass.: Little, Brown.

Bates, J. 1994. The Role of Emotion in Believable Agents. *Communications of the ACM* 37(7): 122–125.

Bates, J., Hayes-Roth, B., Laurel, B., & Nilsson, N. 1994. Papers presented at the AAAI Spring Symposium on Believable Agents, Stanford University, Stanford, Calif.

Benech, D.; Desprats, T.; and Moreau, J.-J. 1996. A Conceptual Approach to the Integration of Agent Technology in System Management. In *Distributed Systems: Operations and Management (DSOM-96).*

Bowman, C. M.; Danzig, P. B.; Manber, U.; and Schwartz, M. F. 1994. Scalable Internet Resource Discovery: Research Problems and Approaches. *Communications of the ACM* 37(8): 98–107, 114.

Boy, G. 1992. Computer-Integrated Documentation. In *Sociomedia: Multimedia, Hypermedia, and the Social Construction of Knowledge,* ed. E. Barrett, 507–531. Cambridge, Mass.: MIT Press.

Boy, G. A. 1991. *Intelligent Assistant Systems.* San Diego, Calif.: Academic Press.

Boy, G. A. 1997. Software Agents for Cooperative Learning. In *Software Agents,* ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Boy, G. A., and Mathé, N. 1993. Operator Assistant Systems: An Experimental Approach Using a Telerobotics Application. In *Knowledge Acquisition as Modeling,* eds. K. M. Ford and J. M. Bradshaw, 271–286. New York: Wiley.

Bradshaw, J. M., and Boose, J. H. 1992. Mediating Representations for Knowledge Acquisition, Boeing Computer Services, Seattle, Washington.

Bradshaw, J. M., and Boy, G. A. 1993. Adaptive Documents, Internal Technical Report, EURISCO.

Bradshaw, J. M., Dutfield, S., Benoit, P., & Woolley, J. D. 1997. KAoS: Toward an industrial-strength generic agent architecture. In *Software Agents,* ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Bradshaw, J. M.; Ford, K. M.; Adams-Webber, J. R.; and Boose, J. H. 1993. Beyond the

Repertory Grid: New Approaches to Constructivist Knowledge-Acquisition Tool Development. In *Knowledge Acquisition as Modeling*, eds. K. M. Ford and J. M. Bradshaw, 287–333. New York: Wiley.

Bradshaw, J. M.; Richards, T.; Fairweather, P.; Buchanan, C.; Guay, R.; Madigan, D.; and Boy, G. A. 1993. New Directions for Computer-Based Training and Performance Support in Aerospace. Paper presented at the Fourth International Conference on Human-Machine Interaction and Artificial Intelligence in Aerospace, 28–30 September, Toulouse, France.

Brodie, M. L. 1989. Future Intelligent Information Systems: AI and Database Technologies Working Together. In *Readings in Artificial Intelligence and Databases*, eds. J. Mylopoulos and M. L. Brodie, 623–642. San Francisco, Calif.: Morgan Kaufmann.

Brooks, R. A. 1990. Elephants Don't Play Chess. *Robotics and Autonomous Systems* 6.

Brown, J. S., and Duguid, P. 1996. The Social Life of Documents. *First Monday* (http://www.firstmonday.dk).

Browne, D.; Totterdell, P.; and Norman, M., eds. 1990. *Adaptive User Interfaces.* San Diego, Calif.: Academic.

Canto, C., and Faliu, O. (n.d.). *The History of the Future: Images of the 21st Century*. Paris: Flammarion.

Chang, D. T., and Lange, D. B. 1996. Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW. In Proceedings of the OOPSLA 96 Workshop "Toward the Integration of WWW and Distributed Object Technology."

Cheong, F.-C. 1996. *Internet Agents: Spiders, Wanderers, Brokers, and Bots.* Indianapolis, Ind.: New Riders.

Clancey, W. J. 1993. The Knowledge Level Reinterpreted: Modeling Socio-Technical Systems. In *Knowledge Acquisition as Modeling*, eds. K. M. Ford and J. M. Bradshaw, 33–50. New York: Wiley.

Cohen, P. R. 1994. Models of Dialogue. In Cognitive Processing for Vision and Voice: Proceedings of the Fourth NEC Research Symposium, ed. T. Ishiguro, 181–203. Philadelphia, Pa.: Society for Industrial and Applied Mathematics.

Cohen, P. R., and Cheyer, A. 1994. An Open Agent Architecture. Paper presented at the AAAI Spring Symposium on Software Agents, 21–23 March, Stanford, California.

Cohen, P. R.; and Levesque, H. 1997. Communicative Actions for Artificial Agents. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Cohen, P. R., and Levesque, H. J. 1991. Teamwork, Technote 504, SRI International, Menlo Park, California.

Coutaz, J. 1990. *Interfaces Homme Ordinateur: Conception et Réalisation.* Paris: Editions Bordas.

Davies, W. H. E. 1994. AGENT-K: An Integration of AOP and KQML. In Proceedings of the CIKM-94 Workshop on Intelligent Agents, eds. T. Finin and Y. Labrou. http://www.csd.abdn.ac.uk/~pedwards/publs/agentk.html.

Dennett, D. C. 1987. *The Intentional Stance*. Cambridge, Mass.: MIT Press.

diSessa, A. A. 1986. Notes on the Future of Programming: Breaking the Utility Barrier. In *User-Centered System Design*, eds. D. A. Norman and S. W. Draper. Hillsdale, N.J.: Lawrence Erlbaum.

Epstein, R. 1992. The Quest for the Thinking Computer. *AI Magazine* 13(2): 81–95.

Erickson, T. 1996. Designing Agents as If People Mattered. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Etzioni, O. 1993. Intelligence without robotics. *AI Magazine*(Winter), 7-14.

Etzioni, O., & Weld, D. S. 1995. Intelligent agents on the Internet: Fact, fiction, and forecast. *IEEE Expert*, 10(4), 44-49.

Finin, T., Labrou, Y., & Mayfield, J. 1997. KQML as an agent communication language. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Etzioni, O. 1993. Intelligence without Robots: A Reply to Brooks. *AI Magazine* 14(4): 7–13.

Etzioni, O., and Weld, D. S. 1995. Intelligent Agents on the Internet: Fact, Fiction, and Forecast. *IEEE Expert* 10(4): 44–49.

Foner, L. 1993. *What's an Agent, Anyway? A Sociological Case Study*, Agents Memo, 93-01, Media Lab, Massachusetts Institute of Technology.

Ford, K. M.; Glymour, C.; and Hayes, P. J., eds. 1995. *Android Epistemology.* Menlo Park, Calif.: AAAI Press.

Ford, K. M.; Bradshaw, J. M.; Adams-Webber, J. R.; and Agnew, N. M. 1993. Knowledge Acquisition as a Constructive Modeling Activity. In *Knowledge Acquisition as Modeling*, eds. K. M. Ford and J. M. Bradshaw, 9–32. New York: Wiley.

Franklin, S., and Graesser, A. 1996. Is It an Agent or Just a Program? A Taxonomy for Autonomous Agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages.* New York: Springer-Verlag.

Gaines, B. R. 1997. *The Emergence of Knowledge through Modeling and Management Processes in Societies of Adaptive Agents,* Knowledge Science Institute, University of Calgary. Forthcoming.

Gardner, E. 1996. Standards Hold Key to Unleashing Agents. *Web Week* 5, 29 April.

General Magic. 1996. Tabriz White Paper: Transforming Passive Networks into an Active, Persistent, and Secure Business Advantage, White Paper (http://www.genmagic.com/Tabriz/Whitepapers/tabrizwp.html), General Magic, Mountain View, California.

General Magic. 1994. Telescript Technologies at Heart of Next-Generation Electronic Services, News Release, 6 January, General Magic, Mountain View, California.

Genesereth, M. R. 1997. An Agent-based Framework for Interoperability. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Genesereth, M. R., and Fikes, R. 1992. Knowledge Interchange Format Version 3.0 Reference Manual, Logic Group Report, Logic-92-1, Department of Computer Science, Stanford University.

Genesereth, M. R., and Ketchpel, S. P. 1994. Software Agents. *Communications of the ACM* 37(7): 48–53, 147.

Gilbert, D.; Aparicio, M.; Atkinson, B.; Brady, S.; Ciccarino, J.; Grosof, B.; O'Connor, P.; Osisek, D.; Pritko, S.; Spagna, R.; and Wilson, L. 1995. IBM Intelligent Agent Strategy, IBM Corporation.

Glicksman, J.; Weber, J. C.; and Gruber, T. R. 1992. The NOTE MAIL Project for Computer-Supported Cooperative Mechanical Design. Paper presented at the AAAI-92 Workshop on Design Rationale Capture and Use, San Jose, California, July.

Greif, I. 1994. Desktop Agents in Group-Enabled Products. *Communications of the ACM* 37(7): 100–105.

Gruber, T. R. 1992a. *ONTOLINGUA: A Mechanism to Support Portable Ontologies, Version 3.0*, Technical Report, KSL 91-66, Knowledge Systems Laboratory, Department of Computer Science, Stanford University.

Gruber, T. R. 1992b. A Translation Approach to Portable Ontology Specifications. Paper presented at the Seventh Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada.

Gruber, T. R.; Tenenbaum, J. M.; and Weber, J. C. 1992. Toward a Knowledge Medium for Collaborative Product Development. In Proceedings of the Second International Conference on Artificial Intelligence in Design, ed. J. S. Gero.

Harrison, C. G.; Chess, D. M.; and Kershenbaum, A. 1995. Mobile Agents: Are They a Good Idea? IBM T. J. Watson Research Center.

Hayes-Roth, B.; Brownston, L.; and Gent, R. V. 1995. Multiagent Collaboration in Directed Improvisation. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, ed. V. Lesser, 148–154. Menlo Park, Calif.: AAAI Press.

Hinkle, D.; Kortenkamp, D.; and Miller, D. 1996. The 1995 Robot Competition and Exhibition. *AI Magazine* 17(1): 31–45.

Hutchins, E. L.; Hollan, J. D.; and Norman, D. A. 1986. Direct Manipulation Interfaces. In *User-Centered System Design*, eds. D. A. Norman and S. W. Draper, 87–124. Hillsdale, N.J.: Lawrence Erlbaum.

Johnson, P.; Feiner, S.; Marks, J.; Maybury, M.; and Moore, J., eds. 1994. Paper presented at the AAAI Spring Symposium on Intelligent Multi-Media Multi-Modal Systems, Stanford, California.

Kaehler, T., and Patterson, D. 1986. A Small Taste of SMALLTALK. *BYTE*, August, 145–159.

Kaelbling, L. P., and Rosenschein, S. J. 1991. Action and Planning in Embedded Agents. In *Designing Autonomous Agents*, eds. P. Maes , 35–48. Cambridge, Mass.: MIT Press.

Kaelbling, L. P., and Rosenschein, S. J. 1990. Action and Planning in Embedded Agents. *Robotics and Autonomous Systems* 6(1–2): 35–48.

Kahle, B. 1993. Interview of Brewster Kahle. *Intertek* 4:15–17.

Kautz, H.; Selman, B.; Coen, M.; Ketchpel, S.; and Ramming, C. 1994. An Experiment in the Design of Software Agents. In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 438–443. Menlo Park, Calif.: American Association for Artificial Intelligence.

Kay, A. 1990. User Interface: A Personal View. In *The Art of Human-Computer Interface Design*, ed. B. Laurel, 191–208. Reading, Mass.: Addison-Wesley.

Kay, A. 1984. Computer Software. *Scientific American* 251(3): 53–59.

Knoblock, C. A., & Ambite, J.-L. 1996. Agents for Information Gathering. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Labrou, Y. 1996. Semantics for an Agent Communication Language. Ph.D. diss., Dept. of Computer Science, University of Maryland at Baltimore County.

Labrou, Y., and Finin, T. 1994. A Semantics Approach for KQML—A General-Purpose Communication Language for Software Agents. In Proceedings of the Third International Conference on Information and Knowledge Management, eds. N. R. Adam, B. K. Bhargava, and Y. Yesha, 447–455. New York: Association of Computing Machinery.

Lai, K.-Y., and Malone, T. W. 1992. Oval Version 1.1 User's Guide, Center for Coordination Science, Massachusetts Institute of Technology.

Lange, D. B. 1996. Agent Transfer Protocol ATP/0.1 Draft 4, Tokyo Research Laboratory, IBM Research.

Langton, C. G., ed. 1995. *Artificial Life: An Overview.* Cambridge, Mass.: MIT Press.

Lanier, J. 1996. Agents of Alienation. http://www.voyagerco.com/misc/jaron.html.

Lanier, J., and Maes, P. 1996. Intelligent Humans = Stupid Humans? *Hot Wired*, 15–24 July. http://www.hotwired.com/braintennis/96/29/index0a.html.

Laurel, B. 1991. *Computers as Theatre*. Reading, Mass.: Addison-Wesley.

Laurel, B. 1997. Interface agents: Metaphors with Character. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Le Du, B. 1994. Issue 1309, 13 mai. Les Agents, des Assistants dotés d'Intelligence. 01 Informatique, p. 13.

Lethbridge, T. C., and Skuce, D. 1992. Beyond Hypertext: Knowledge Management for Technical Documentation. Submitted to SIGDOC '92. Ottawa, Ontario, Canada.

Lewis, J. 1996. NETSCAPE Gets Serious about Infrastructure. The Burton Group.

Lubar, S. 1993. *InfoCulture: The Smithsonian Book of Information and Inventions.* Boston, Mass.: Houghton Mifflin.

McCarthy, J. M. 1979. Ascribing Mental Qualities to Machines, Technical Report, Memo 326, AI Lab, Stanford University.

McDermott, D. 1976. Artificial Intelligence Meets Natural Stupidity. *SIGART Newsletter* 57:4–9.

MacGregor, R. 1990. The Evolving Technology of Classification-Based Knowledge Representation Systems. In *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, ed. J. F. Sowa, 385–400. San Francisco, Calif.: Morgan Kaufmann.

Maes, P. 1997. Agents that Reduce Work and Information Overload. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Maes, P. 1995. Modeling Adaptive Autonomous Agents. In *Artificial Life: An Overview*, ed. C. G. Langton, 135–162. Cambridge, Mass.: MIT Press.

Maes, P., ed. 1993. *Designing Autonomous Agents.* Cambridge, Mass.: MIT Press.

Maes, P., and Kozierok, R. 1993. Learning Interface Agents. In Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), 459–465. Menlo Park, Calif.: American Association for Artificial Intelligence.

Malone, T. W.; Grant, K. R.; and Lai, K.-Y. 1996. Agents for Information Sharing and Coordination: A History and Some Reflections. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Martin, F. 1988. *Children, Cybernetics, and Programmable Turtles.* Masters Thesis, Media Laboratory, Massachusetts Institute of Technology.

Mathé, N., and Chen, J. 1994. A User-Centered Approach to Adaptive Hypertext Based on an Information Relevance Model. Paper presented at the Fourth International Conference on User Modeling (UM '94), Hyannis, Massachusetts.

Maturana, H. R., and Varela, F. J. 1992. *The Tree of Knowledge: The Biological Roots of Human Understanding* (rev. ed.). Boston: Shambala.

Mellor, P. 1994. CAD: Computer-Aided Disaster. *SOFSEM 94.*

Milewski, A. E., and Lewis, S. M. 1994. Design of Intelligent Agent User Interfaces: Delegation Issues. Technical Report, Oct. 20. AT&T Information Technologies Services.

Miller, J. R., and Neches, R. 1987. Tutorial on Intelligent Interfaces Presented at the Sixth National Conference on Artificial Intelligence, 14–16 July, Seattle, Washington.

Minsky, M. 1986. *The Society of Mind.* New York: Simon & Schuster.

Minsky, M., and Riecken, D. 1994. A Conversation with Marvin Minsky about Agents. *Communications of the ACM* 37(7): 23–29.

Mitchell, T.; Caruana, R.; Freitag, D.; McDermott, J.; and Zabowski, D. 1994. Experience with a Learning Personal Assistant. *Communications of the ACM* 37(7): 81–91.

Moulin, B., and Chaib-draa, B. 1996. An Overview of Distributed Artificial Intelligence. In *Foundations of Distributed Artificial Intelligence*, eds. G. M. P. O'Hare and N. R. Jennings, 3–55. New York: Wiley.

Neal, J. G., and Shapiro, S. C. 1994. Knowledge-Based Multimedia Systems. In *Multimedia System*, ed. J. F. K. Buford, 403–438. Reading, Mass.: Addison-Wesley.

Neches, R.; Fikes, R.; Finin, T.; Gruber, T.; Patil, R.; Senator, T.; and Swartout, W. R. 1991. Enabling Technology for Knowledge Sharing. *AI Magazine* 12(3): 36–55.

Negroponte, N. 1997. Agents: From Direct Manipulation to Delegation. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Negroponte, N. 1995. *Being Digital.* New York: Alfred Knopf.

Negroponte, N. 1970. *The Architecture Machine: Towards a More Human Environment*. Cambridge, Mass.: MIT Press.

Newell, A. 1982. The Knowledge Level. *Artificial Intelligence* 18:87–127.

Newquist, H. P. 1994. Intelligence on Demand—Suckers. *AI Expert*, December, 42–43.

Nilsson, N. J. 1995. Eye on the Prize. *AI Magazine* 16(2): 9–17.

Norman, D. A. 1997. How Might People Interact with Agents? In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Norman, D. A. 1992. *Turn Signals Are the Facial Expressions of Automobiles.* Reading, Mass.: Addison-Wesley.

Nwana, H. S. 1996. Software Agents: An Overview. *Knowledge Engineering Review*, 11(3): 205-244.

Paepcke, A.; Cousins, S. B.; Garcia-Molina, H.; Hassan, S. W.; Ketchpel, S. P.; Röscheisen, M.; and Winograd, T. 1996. Using Distributed Objects for Digital Library Interoperability. *IEEE Computer*, May, 61–68.

Perrow, C. 1984. *Normal Accidents: Living with High-Risk Technologies.* New York: Basic.

Petrie, C. J. 1996. Agent-Based Engineering, the Web, and Intelligence. *IEEE Expert*, 11(6): 24-29.

Repenning, A. 1995. Bending the Rules: Steps toward Semantically Enriched Graphical Rewrite Rules. Paper presented at Visual Languages, Darmstadt, Germany.

Resnick, M., and Martin, F. 1990. Children and Artificial Life, E&L Memo, 10, Media Laboratory, Massachusetts Institute of Technology.

Rich, C., and Waters, R. C. 1988. Automatic Programming: Myths and Prospects. *IEEE Computer* 21(8): 40–51.

Riecken, D. 1997. The M System. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Rivière, A.-I.; Pell, A.; and Sibilla, M. 1996. Network Management Information: Integration Solution for Models Interoperability, Technical Report, Hewlett-Packard Laboratories.

Rosenschein, S. J. 1985. Formal Theories of Knowledge in AI and Robotics. *New Generation Computing* 3(4): 345–357.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. New York: Prentice-Hall.

Ryan, B. 1991. DYNABOOK Revisited with Alan Kay. *BYTE*, February, 203–208.

Schank, R. C., and Jona, H. Y. 1991. Empowering the Student: New Perspectives on the Design of Teaching Systems. *The Journal of the Learning Sciences* 1(1).

Schelde, P. 1993. *Androids, Humanoids, and Other Science Fiction Monsters.* New York: New York University Press.

Sen, S.; Hogg, T.; Rosenschein, J.; Grefenstette, J.; Huhns, M.; and Subramanian, D., eds. 1996. Adaptation, Coevolution, and Learning in Multiagent Systems: Papers from the 1996 AAAI Symposium. Technical Report SS-96-01. Menlo Park, Calif.: AAAI Press.

Sharp, M. 1993. Reactive Agents, Technical Report, Apple Computer, Cupertino, Calif.

Sharp, M. 1992. Principles for Situated Actions in Designing Virtual Realities. Master's thesis, Department of Computer Science, University of Calgary.

Shaw, M. 1996. Some Patterns for Software Architectures. In *Pattern Languages of Program Design*, eds. J. O. Coplien and D. C. Schmidt, 453–462. Reading, Mass.: Addison-Wesley.

Shneiderman, B. 1997. Direct manipulation vs. agents: Paths to predictable, controllable, and comprehensible interfaces. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Shneiderman, B. 1987. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Reading, Mass.: Addison-Wesley.

Shneiderman, B. 1983. Direct Manipulation: A Step beyond Programming Languages. *IEEE Computer* 16(8): 57–69.

Shoham, Y. 1997. An Overview of Agent-oriented Programming. In *Software Agents*, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Shoham, Y. 1993. Agent-Oriented Programming. *Artificial Intelligence* 60(1): 51–92.

Simmons, R. 1995. The 1994 AAAI Robot Competition and Exhibition. *AI Magazine* 16(2): 19–30.

Singh, M. P. 1994. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communication*. Berlin: Springer-Verlag.

Smith, D. C., Cypher, A., & Spohrer, J. 1997. KidSim: Programming Agents Without a Programming Language. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Smith, D. C.; Irby, C.; Kimball, R.; Verplank, W.; and Harslem, E. 1982. Designing the STAR User Interface. *BYTE* 4:242–282.

Smith, I. A., and Cohen, P. R. 1995. Toward a Semantics for a Speech Act–Based Agent Communications Language. In Proceedings of the CIKM Workshop on Intelligent Information Agents, eds. T. Finin and J. Mayfield. New York: Association of Computing Machinery.

Sowa, J. F. 1990. Crystallizing Theories out of Knowledge Soup. In *Intelligent Systems: State of the Art and Future Systems*, eds. Z. W. Ras and M. Zemankova. London: Ellis Horwood.

Sowa, J. F. 1989. Knowledge Acquisition by Teachable Systems. In *EPIA 89, Lecture*

Notes in Artificial Intelligence, eds. J. P. Martins and E. M. Morgado, 381–396. Berlin: Springer-Verlag.

Steels, L. 1995. The Artificial Life Roots of Artificial Intelligence. In *Artificial Life: An Overview*, ed. C. G. Langton, 75–110. Cambridge, Mass.: MIT Press.

Sullivan, J. W., and Tyler, S. W., eds. 1991. *Intelligent User Interfaces.* New York: Association of Computing Machinery.

Tackett, W. A., and Benson, S. 1985. Real AI for Real Games: In *Technical Tutorial and Design Practice*, 467–486.

Tesler, L. G. 1991. Networked Computers in the 1990s. *Scientific American*, September, 86–93.

Turing, A. M. 1950. Computing Machinery and Intelligence. *Mind* 59(236): 433–460.

Van de Velde, W. 1995. Cognitive Architectures—From Knowledge Level to Structural Coupling. In *The Biology and Technology of Intelligent Autonomous Agents*, ed. L. Steels, 197–221. Berlin: Springer Verlag.

Vere, S., and Bickmore, T. 1990. A Basic Agent. *Computational Intelligence* 6:41–60.

Virdhagriswaran, S.; Osisek, D.; and O'Connor, P. 1995. Standardizing Agent Technology. *ACM Standards View*. In press.

Weld, D.; Marks, J.; and Bobrow, D. G. 1995. The Role of Intelligent Systems in the National Information Infrastructure. *AI Magazine* 16(3): 45–64.

White, J. 1997. A Common Agent Platform, http://www.genmagic.com/Internet/Cap/w3c-paper.htm, General Magic, Inc., Sunnyvale, California.

White, J. 1997. Mobile Agents. In *Software Agents*, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.

Whittaker, S. 1990. Next-Generation Interfaces. Paper presented at the AAAI Spring Symposium on Knowledge-Based Human-Computer Interaction, Stanford, California, March.

Wiederhold, G. 1995. Digital Libraries, Value, and Productivity, Stanford University.

Wiederhold, G. 1992. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, March, 38–49.

Wiederhold, G. 1989. The Architecture of Future Information Systems, Technical Report, Computer Science Department, Stanford University.

Williams, B. C., and Nayak, P. P. 1996. Immobile Robots: AI in the New Millennium. *AI Magazine* 17(3): 17–35.

Winograd, T. 1973. A Procedural Model of Language Understanding. In *Computer Models of Thought and Language*, eds. R. Schank and K. Colby, 249–266. New York: Freeman.

Wooldridge, M. J., and Jennings, N. R. 1995. Agent Theories, Architectures, and Languages: A Survey. In *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, eds. M. J. Wooldridge and N. R. Jennings, 1–39. Berlin: Springer-Verlag.

Section One

Agents & the User Experience

@inproceedings{Bradshaw1997AnIT, title={An introduction to software agents}, author={J. Bradshaw}, year={1997} }. J. Bradshaw. Published 1997. Engineering. S ince the beginning of recorded history, people have been fascinated with the idea of non-human agencies. 1 Popular notions about androids, hu-manoids, robots, cyborgs, and science fiction creatures permeate our culture , forming the unconscious backdrop against which software agents are perceived. The word " robot, " derived from the Czech word for drudgery, became popular following Karel Capek's 1921 play RUR: Rossum Unive An introduction to software agents 5. (White 1996); some because they present themselves to users as believable char-acters (Ball et al. 1996, Bates 1994, Hayes-Roth, Brownston, and Gent 1995); some because they speak an agent communication language (Genesereth 1997, Finin et al. 1997) and some because they are viewed by users as manifesting in-tentionality and other aspects of â€œmental stateâ€ (Shoham 1997). Out of this confusion, two distinct but related approaches to the denition of agent have been attempted: one based on the notion of agenthood as an ascription made by some person, the other