Agents and Multiagent Systems: Themes, Approaches, and Challenges

Michael N. Huhns[*]
Dept of Electrical & Computer Engineering
University of South Carolina
Columbia, SC 29208, USA
huhns@sc.edu

Munindar P. Singh[†]
Dept of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
singh@ncsu.edu

Abstract

We present some of the key themes in agent research, approaches for building agent systems and tools, and the challenges that remain.

1 Introduction

Due to the proliferation of computing and networking, the desires of almost everyone to be interconnected, and the needs to make data accessible at any time and any place, modern information environments have become large, open, and heterogeneous. They are composed of distributed, largely autonomous, often legacy components. Recent approaches introduce software *agents* into such environments to deal with these characteristics. The agents represent the components in interactions, where they mediate differences and provide a syntactically uniform and semantically consistent middleware. Their greatest difficulty in achieving uniformity and consistency is the dynamism that open environments introduce. At the same time, the complexity and dynamism of the information environments has led to a pressing need for user interfaces that are active and adaptive personal assistants—in other words, *agents*.

Applications such as information access, information filtering, electronic commerce, workflow management, intelligent manufacturing, education, and entertainment are becoming ever more prevalent. What these applications have in common is a need for mechanisms for advertising, finding, fusing, using, presenting, managing, and updating information. Since the underlying environment is open—in that the sources of information are autonomous and heterogeneous, and may be added or removed dynamically—the associated mechanisms must be extensible and flexible. Increasingly, people are coming to the conclusion that agents are an integral part of such mechanisms. The charm of agents is that they provide a natural means for performing the above tasks over uncontrollable environments. Further, agents are inherently modular, and can be constructed locally for each resource, provided they satisfy some high-level protocol of interaction.

1.1 Agents

So, what exactly is an agent? Must it be intelligent? Adaptive? Itinerant? There are almost as many opinions on this as there are agents themselves, leading to frequent debates flaring up on several Internet forums. There is a common core of concepts among these opinions, however, which we have synthesized into our own definition: *Agents are active, persistent (software) components that perceive, reason, act, and communicate.*

Some researchers add further properties, such as being autonomous, goal-directed, reactive, or declaratively programmed. Others limit agents to the role of representing a user or database.

There are two extreme views of agents. One longstanding tradition takes agents as essentially conscious, cognitive entities that have feelings, perceptions, and emotions just like humans. Under this view, all of the computational work on agents is inherently inadequate. An alternative view is that agents are merely automata, and behave exactly as they are designed or programmed. This view admits a large variety of computations, including computational agents. A concern is that it might be too permissive!

Which of these views is correct? The truth lies somewhere in between. Here, we shall desist from attempting a precise definition, and will instead discuss the kinds of applications, architectures, and models that are typically associated with agents. A key issue is that of complexity. The pure automaton view of agents fails to involve the kinds of abstractions that are often essential in describing, analyzing, understanding, or explaining the behavior of agents. The traditional view has many of the abstractions, which work well for people, but does not allow their usage for artificial systems. However, in the study of agents, it is often helpful to use the folk abstractions that were meant for people, but give them specific technical meanings that are applicable for computational agents. We discuss some of these in section 4.

In any case, as a practical matter, one should always ask the question: *what is special about an entity that it may be called an agent?* or *what does calling it an agent*

[*]Michael Huhns is supported by NIST, DARPA, and EPA.

[†]Munindar Singh is supported by the National Science Foundation under grants IRI-9529179 and IRI-9624425, IBM Corporation, and the NCSU College of Engineering.

buy us? The answer would not be the same in each case, but it should be nonempty for the notion of agency to be nonvacuously applied.

1.2 Multiagent Systems

All too often, agents are best developed not in isolation but as parts of a multiagent system. The motivation for this claim is entirely practical. Consider the case of the web. The computational architecture that seems to be evolving out of an informationally chaotic web consists of numerous agents representing users, services, and data resources. A typical paradigm of usage is as follows. The resource agents advertise to the services; the user agents use the services to find the resource agents, and then query them for the information they need.

A web agent can do its job well only if it can take advantage not only of all the information resources in the web, but also of all the other agents that might be operating there. Agents representing different users might collaborate in finding and fusing information, but they might compete for goods and resources. Similarly, the service agents may collaborate or compete with user, resource, and other service agents.

Whether they are collaborators or competitors, the agents will be interacting with each other. They will be interacting not unwittingly, but purposefully. Most purposeful interactions—whether to inform, query, or deceive—require that the agents talk to one another, be aware of each other, and reason about each other. In other words, the agents would be developed to perform as members of a multiagent system.

1.3 Key Characteristics

A taxonomy of agents requires identifying the key characteristics of agent systems, including the characteristics of the agents, the multiagent systems they participate in, the frameworks they are developed in, the roles they play, and the environments they inhabit. We discuss these next. In the interest of brevity, we do not elaborate many of the characteristics, which can be understood from their short definitions or admissible ranges of values. However, some of the characteristics, especially, autonomy, rationality, construction, sociability, and mobility, are discussed in greater detail.

1.3.1 Agents

The characteristics of agents are fundamentally tied to not only their intrinsic properties—which are defined for an agent by itself—but also their extrinsic properties—which are defined for an agent in the context of other agents. Tables 1 and 2 enumerate the intrinsic and extrinsic characteristics of agents, respectively.

1.3.2 System

Table 3 enumerates the key attributes of multiagent systems, independent of the agents that constitute them.

1.3.3 Frameworks

An agent execution environment or framework includes a number of concerns, which are enumerated as possible characteristics in Table 4.

1.3.4 Environments

Table 5 lists some key properties of an environment with respect to a specific agent that inhabits it. These generalize the presentation in [Russell & Norvig, 1995].

1.3.5 Autonomy

Autonomy is often cited as an important property of agents (and other systems), yet it is difficult to define. We find it useful to distinguish different varieties of autonomy, which serve different purposes in the study and design of agents.

It is important to identify the intended unit of autonomy. This could be the computational agent plus its human owner, or an agent plus some associated information resource. An agent that appears autonomous to other agents may in fact be a slave for the human or information resource to which it is attached.

Absolute Autonomy. Autonomy is intuitively related to predictability—the less predictable an agent is the more autonomous it appears. This view is autonomy as arbitrariness; thus an absolutely autonomous agent may do anything it pleases. An *autistic* agent—one that is simply unaware of the other agents inhabiting its environment or indeed of the environment itself—may be observed to be performing actions absolutely autonomously. It wouldn't heed to requests of others because it would not even recognize them.

Absolute autonomy is rarely useful in the design of an agent system, because agents typically must serve some purpose, which constrains them. However, a multiagent system may be able to tolerate some absolutely autonomous agents: we discuss that feature under execution autonomy.

Social Autonomy. A more interesting case is social autonomy, where an agent is aware of its colleagues and is sociable, but nevertheless exercises its autonomy in certain circumstances. Autonomy is in natural tension with coordination or with higher-level notions such as commitments (section 4.3). To be coordinated with other agents or to keep its commitments, an agent must lose some of its autonomy. However, an agent that is sociable and responsible can still be autonomous. It would attempt to coordinate with others where appropriate and to keep its commitments as much as possible. However, it would exercise its autonomy in entering into those commitments in the first place.

Consequently, social is not an intrinsic property, but an extrinsic property, and is included in Table 2.

Interface Autonomy. In most practical systems, where absolute autonomy is not feasible, the most autonomy we can hope for is autonomy with respect to the internal design, provided some application programming interface (API) is maintained. The API defines what is exposed by the agent, but doesn't guarantee that the agent will always do as requested—that is a matter of

Property	**Range of values**
Lifespan	Transient to Long-lived
Level of Cognition	Reactive to Deliberative
Construction	Declarative to Procedural
Mobility	Stationary to Itinerant
Adaptability	Fixed to Teachable to Autodidactic
Modeling	Of environment, themselves, or other agents

Table 1: Agent Characteristics: Intrinsic

Property	**Range of values**
Locality	Local to Remote
Social Autonomy	Independent to Controlled
Sociability	Autistic, Aware, Responsible, Team Player
Friendliness	Cooperative to Competitive to Antagonistic
Interactions	*Logistics:* Direct or via facilitators, mediators, or nonagents *Style/Quality/Nature:* With agents/world/both *Semantic Level:* Declarative or procedural communications

Table 2: Agent Characteristics: Extrinsic

Property	**Range of values**
Uniqueness	Homogeneous to Heterogeneous
Granularity	Fine-grained to Coarse-grained
Control Structure	Hierarchy to Democracy
Interface Autonomy	*Communication:* specify vocabulary, language, protocol *Intellect:* specify goals, beliefs, ontologies *Skills:* specify procedures, behaviors
Execution Autonomy	Independent to Controlled

Table 3: System Characteristics

Property	**Range of values**
Design Autonomy	Platform/Language/Internal Architecture/Interaction Protocol
Communication Infrastructure	Shared memory (blackboard) or Message-based Connected or Connection-less (email) Point-to-Point, Multicast, or Broadcast Push or Pull Synchronous or Asynchronous
Directory Service	White pages, Yellow pages
Message Protocol	KQML HTTP and HTML OLE, CORBA, DSOM
Mediation Services	Ontology-based? Transactions (updates)?
Security Services	Timestamps/Authentication
Remittance Services	Billing/Currency
Operations Support	Archiving/Redundancy/Restoration/Accounting

Table 4: Framework Characteristics

Property	Definition
Knowable	To what extent is the environment known to the agent
Predictable	To what extent can it be predicted by the agent
Controllable	To what extent can the agent modify the environment
Historical	Do future states depend on the entire history, or only the current state
Teleological	Are parts of it purposeful, i.e., are there other agents
Real-time	Can the environment change while the agent is deliberating

Table 5: Environment-Agent Characteristics

execution autonomy. The interface requirements are imposed by a system on the agents who might join it. For this reason, this property appears as a system characteristic in Table 3.

Execution Autonomy. Execution autonomy corresponds to the freedom an agent has while executing in an environment. For example, personal assistants can be constrained to behave in a helpful manner—they must always speak the truth, and believe and help the user. However, since agents can represent different interests, some execution autonomy is generally required, even when interface autonomy is constrained. Execution autonomy is crucial to the openness of a system. Restrictions on execution autonomy can be imposed by specific systems, for specific roles. Thus, this is a system characteristic in Table 3.

Design Autonomy. Design autonomy deals with how the agents are constructed—it is the autonomy of the designers. Design autonomy is crucial if we want a system to be truly open, in that people should be able to contribute their agents to the system while having to satisfy as few requirements as possible. Thus, design autonomy corresponds to heterogeneity of the agents. It is orthogonal to execution autonomy, because agents of a fixed design might choose their own actions, while agents of different designs might be controlled externally.

Different agent frameworks come with different requirements that impinge upon design autonomy. Some require that all agents be built using a specific language. Some approaches, e.g., for agent communication, require that the agents must necessarily represent beliefs; they be must able to perform logical inferences and plan; and they be must rational. Design autonomy is listed as a property of frameworks in Table 4.

1.3.6 Intelligence and Rationality

Intelligence is notoriously difficult to define precisely. However, we know when a system is behaving intelligently. One simplistic, but sometimes useful, definition is that an intelligent system is one that performs a task that if performed by a human would earn the human the attribute intelligent. However, many tasks that are performed routinely by humans are difficult to realize in a computational system. By contrast, some tasks that are difficult for a human, such as various forms of arithmetic, are easy for a machine.

The concept of rationality is intimately related to intelligence, but can be formalized more easily. Most formalizations require that the agent have preferences about states of the world and choose actions that maximize the preferences. Thus, rationality depends on the following attributes [Russell & Norvig, 1995]:

- The performance measure for success
- What the agent has perceived so far
- What the agent knows about the environment
- The actions the agent can perform.

An ideal rational agent is defined as follows: for each possible percept sequence, it acts to maximize its expected utility, on the basis of its knowledge and the evidence from the percept sequence.

Logical rationality considers qualitative notions such as the consistency of beliefs or the consistency of beliefs, intentions, desires with chosen actions (these terms are revisited in section 4). Economic rationality assumes that the agent has utility functions to guide its choice of actions.

Although rationality is a compelling notion for several purposes, it imposes certain artificial constraints on how one may understand the world and the agents that inhabit it. Agents—artificial and human—are limited in their computational powers and their knowledge of their environment, and may not be quite as rational as one might like. The study of *bounded rationality* remains an important, yet difficult, subject [Simon, 1996].

1.3.7 Procedural vs. Declarative Construction

Procedural approaches to programming specify exactly how a computation should proceed; declarative approaches specify what it should do without giving too much detail of how. There is a longstanding debate in computer science about the relative merits of procedural and declarative approaches. For example, the seventies saw the famous *procedural-declarative controversy* [Winograd, 1975].

Given the way computing systems develop, procedural approaches are the first ones to come about before the necessary generalizations have been made to yield declarative approaches. Also, in a narrow sense, procedural approaches can be more efficient. However, when the flexibility of solutions and the productivity of programmers are taken into consideration, declarative approaches usually pay off. Declarative approaches offer advantages in

- Modularity: requirements can be captured independently from each other.
- Incremental change: it is much easier to add or remove components from a declarative specification than to rewrite procedural programs.
- Semantics: declarative notations can be given a formal semantics directly, whereas procedural languages must first be mapped to declarative structures. Formal semantics is crucial for validating tools for building agents and their interaction protocols. It assures predictable behavior, and enables efficiencies in implementation without jeopardizing soundness.
- User interfaces: declarative specifications are easier to generate than procedural code, leading to greater productivity for interface developers and, coupled with clean semantics, greater predictability for users.
- Inspectability: being explicit, declarative specifications can be examined to determine (a) the current constraints on an agent and its interactions, (b) how far the constraints have been satisfied, and (c) the rationales for different actions.
- Learnability: declarative specifications are easier to learn, enabling an agent to discover how other agents behave, and how to participate in an ongoing "discussion" among agents.

For these reasons, time and again throughout the history of computing, higher level declarative techniques have won out. Examples include high-level programming languages vs. assembly languages, SQL vs. navigational queries, conceptual vs. physical data models, and formal grammars and compiler generators vs. hard-coded compilers. However, declarative approaches can be significantly inefficient for some problems, and can require a heavy infrastructure to interpret declarative specifications.

1.4 Historical Remarks

Agents and agency have been the object of study for centuries. They were first considered in the philosophy of action and ethics. In this century, with the rise of psychology as a discipline, human agency has been studied intensively.

Within the five decades of artificial intelligence (AI), computational agents have been an active topic of exploration. The AI work in its earliest stages investigated agents explicitly, albeit with simple models. However, the optimism of building truly intelligent systems with the theories and technologies of the fifties and sixties faded. In the seventies and eighties, the trend of AI research turned toward narrower studies of specific techniques. To some extent this was justifiable, because researchers had come to realize that there were a number of technical problems to be solved before any progress could be made on a general notion of intelligent agency. There was much work in knowledge representation, search techniques, game playing, expert systems, machine learning, and natural language. However, the specialization had the unfortunate effect of fragmenting the field while reducing the payoffs from pursuing further research.

From the late seventies onward, while mainstream AI was turning toward specific capabilities, a research community was forming under the rubric of *distributed AI (DAI)* [Bond & Gasser, 1988; Huhns, 1987; Gasser & Huhns, 1989]. The DAI community concerned itself with agents as computational entities that interacted with each other to solve various kinds of distributed problems. Although there are specific techniques in DAI, just as in the rest of AI, large systems were more important to this community. To this end, whereas AI at large borrowed abstractions, such as beliefs and intentions, from psychology, the DAI community borrowed abstractions and insights from sociology, organizational theory, economics, and the philosophies of language and linguistics. These abstractions complement rather than oppose the psychological abstractions, but being about groups of agents, are fundamentally better suited to large distributed systems.

The term *agent* also came to be used in the eighties and nineties by the traditional computing communities of database, operating systems, and networking. An agent was essentially a proxy for a computation or a site—a transaction, a process, or a network router—which could be used to poll the state of the underlying entity. Agents were thus a formal interface to an arbitrary system. For example, the Simple Network Management Protocol (SNMP), defines certain kinds of agents upon which a network management application can be designed. This application can in principle execute in a multivendor environment, as long as the agents implemented by the different vendors yield the same functionality, while hiding the internal details of the proprietary components, such as network routers.

With the expansion of the Internet and the web in the nineties, we witnessed the emergence of software agents, geared to open information environments. These agents perform tasks on behalf of a user, or serve as nodes—brokers or information sources—in the global information system. Although software agents of this variety do not involve specially innovative techniques, it is their synthesis of existing techniques and their suitability to their application that makes them powerful and popular. Thus, much of the attention they have received is well-deserved. However, we believe that their fundamental shortcoming, which will require significant changes in their design, is their centralization. The enhancement of conventional software agents through distribution will be one of the major areas of expansion in agent technology.

The past few years have also seen the development of user interface agents or personal assistants. These agents help a human user deal with a complex environment. Increasingly, they are called upon to be natural interlocutors with personalities and emotions of their own. This is another promising direction in agent technology.

1.5 Organization

The literature on agents poses some special challenges to classification. Since agent technology is a new area, applications and architectures are usually developed hand in hand. Similarly, architectures and theories or models are also intimately related, because the models inherently make architectural assumptions, and architectures often assume a model of agency.

However, in keeping with our present goals, we categorize the literature into applications, architectures and infrastructure, and models, which we discuss in the following sections, followed by some speculations on future trends and challenges.

2 Applications

There are numerous applications for agents. Many involve varieties of personal assistants, and others are specialized for information-rich environments. Still others involve topics such as art, drama, and design—well beyond the traditional applications of computing, but increasingly important.

2.1 Information-Rich Environments

Information-rich or open environments are broadly defined as environments consisting of a large number and variety of distributed and heterogeneous information sources. The associated applications are varied. They involve the purely informational ones, such as database access, information malls, workflow management, electronic commerce, and virtual enterprises. They also include the information component of physical ones, such as sensor arrays, manufacturing, transportation, energy distribution, and telecommunications. By way of distinction, open environments

- Span enterprise boundaries
- Have components that are heterogeneous in the underlying database management systems used, or the semantics associated with the information stored or manipulated
- Comprise information resources that can be added or removed in a loosely structured manner
- Lack global control of the content of those resources, or how that content may be updated
- Incorporate intricate interdependencies among their components.

Thus, open information environments are partially knowable, predictable, and controllable. They are often teleological, but mostly not historical or real-time.

Cooperative Information Systems (CISs) are multiagent systems with organizational and database abstractions geared to open environments. Each component of the environment, as well as the human user(s), is modeled as associated with an agent. The agents capture and enforce the requirements of their associated parties. They interact with one another appropriately, and help achieve the necessary robustness and flexibility.

2.1.1 Enterprises

The enterprise applications of interest always involve heterogeneity and often the updating of information. Updates are qualitatively more complex than retrievals, because they can potentially introduce inconsistencies. This is especially the case when several databases are involved, and there are subtle interdependencies among them. This is the province of a *workflow*, a composite activity to solve some business need that accesses different resources and may involve human interaction.

Traditional databases transactions satisfy the so-called ACID properties [Gray & Reuter, 1993]. A transaction happens entirely or not at all, does not violate consistency, does not expose any partial results, and if successful has permanent results. Transactions are effective in homogeneous and centralized databases, but cannot model workflows, as shown in Table 6. For example, atomicity requires the component databases to expose their internal control states and operations, violating their execution autonomy and often also their interface autonomy. Isolation requires locking data items even when it is essential to let a collaborator access them.

In many cases, multiple workflows can arise and interact with each other. For example, in a telecommunications setting, a channel assignment workflow must wait until enough channels have been created by another workflow. Some of these interactions can be pernicious in that one workflow may cause the failure of another workflow. Some of the interactions, however, are useful. The challenge is to identify the (potential) interactions and to control them appropriately.

Agents can manage individual workflows, especially through potential exception conditions. They can also model and manage their interactions with one another.

2.1.2 Network Information Access

Network information access involves accessing information from intranets or the Internet. It involves the tasks of

- resource discovery—finding the proper source to query or search
- information retrieval—querying unstructured or semistructured data
- database querying—querying structured data of different formats
- information filtering—obtaining information from a source that is producing a stream of data
- information fusion—merging results in a meaningful manner.

The size of the information space, and the variety of the sources in it, make network information access a daunting task. Unstructured data prove especially challenging, because the absence of a clear semantics makes it difficult to reliably relate results from a query to the original information needs of the user [Croft, 1993]. Thus, the user must actively participate in the querying process,

Property	**Meaning**	**Undesirable when**
Atomicity	all or nothing	legacy systems or nonterminating processes are involved
Consistency	integrity preserving	integrity conditions cannot be defined or data values expire
Isolation	hidden partial results	collaboration is desired
Durability	permanent committed results	backing out is necessary

Table 6: The ACID Properties

making information access a prime candidate for personal assistants (section 2.2).

As a specific example of the use of agents in these applications, consider Warren, which is a system of intelligent agents for helping someone manage a financial portfolio [Sycara & Zeng, 1996]. It coalesces market data, financial report data, technical models, analysts' reports, and breaking news with current prices from a stock ticker. For example, while one agent finds and plots the current price of a company's stock, another will monitor the newswire for anything that mentions the company. Later, it might be seen that a sharp drop in the price occurred shortly after a news release from a brokerage that downgraded the stock. All of the information is already available in some form on the web—Warren simply integrates it by having a specialized agent responsible for each resource, and then presenting it to, or alerting, a user. The agents operate on behalf of users for months, whether they are logged on or not.

Several other applications include aspects of personal assistance:

- Decision support from Firefly, AgentSoft, Verity, and Amulet: information agents that learn and adapt to users' information needs and then proactively retrieve and organize targeted information; Firefly uses collaborative filtering
- Information filtering from Intel: a Smart News Reader sorts and ranks newsgroup articles based on learned user preferences.

2.2 Personal Assistants

User interfaces are now widely recognized as one of the booming areas of computing. Interface technologies such as graphics, animation, virtual reality, and wearable or mobile interfaces are becoming ever more common.

Whereas traditional user interfaces were rigid and not helpful, the decreasing cost of computers and physical interface devices coupled with the spread of computing to the lay population, has supported an increasing trend toward intelligent, cooperative interfaces. Also, whereas traditional user interfaces were for applications such as database access, modern applications involve the control of software and hardware tools in general. These can be as varied as tools for scheduling meetings, finding people with interests similar to one's own, carrying out statistical reasoning, designing artificial environments, animating entities in virtual worlds, and education.

Traditional AI sought to develop automated tools for solving problems that required some intelligence—not necessarily using human mechanisms of intelligence, which are not well understood, but at least addressing problems of sufficient complexity that a human solving them would be considered intelligent. Recent successes of computing, AI included, have taken a radically different approach. Rather than attempting to automate the reasoning process fully, the current trend is to develop tools that assist humans in carrying out the reasoning. There are a number of good motivations for the trend:

- Many interesting problems are too complex to have tractable solutions that are fully automated.
- In many settings, for issues of ethics and responsibility, computers cannot be trusted to perform critical actions unilaterally. In such settings, it is crucial to keep a human in the decision loop.
- Some applications inherently require the active participation of a human, because the problem cannot be specified in a form that will admit to automatic solution. An important case is information retrieval, where users typically do not have a precise query that can be processed automatically. Instead, users need to ask some leading queries to understand the information space they are searching and to formulate a precise query only gradually (section 2.1.2). Another example is education: it would be inappropriate with current technology to eliminate the human user from an educational system!

As a consequence of the above trend, modern user interfaces are playing an increasing role in complex systems. The trend has shifted from passive interfaces to active interfaces—those that have a life of their own, i.e., agents! Such interfaces are *dialogue-based* to some extent—not because they must carry out spoken or textual dialogues, but because they are aware of users and interact with them dynamically. In this respect, they typically support *mixed-initiative* interactions in which neither the computer nor the human are master or slave, but equal partners—either party can take initiative in the dialogue as appropriate.

In many applications, it is important to have interfaces that not only get the job done, but also exude a *persona*. In such interfaces, the agent has an explicit presence, e.g., as an on-screen animated figure. Such agents are called *believable agents* and are suited for applications involving instruction through the means of an animated tutor figure or role in a game or dramatic play.

Thus, personal assistants can be characterized as

- multimodal: support interactions in different input and output modalities, such as voice or typing

- dialogue-based: carry out a conversation, not necessarily spoken, with the user
- mixed-initiative: if dialogue-based, do they dynamically let the user control the dialogue or make unexpected requests
- anthropoid: endowed with a personality; typically emotional
- cooperative: assist the user in defining the user's real needs—this typically requires some ability to model the user and the task the user is engaged in
- adaptive: learn from past interactions with the user.

A major challenge for agent-based interfaces is the development of toolkits and methodologies through which they can be engineered to have as many of the above characteristics as desired in an application.

Property	**User**	**Backend system**
Knowable	Yes	Yes
Predictable	Partially	No
Controllable	No	Partially
Historical	Yes	No
Teleological	Yes	Maybe
Real-time	Yes	Possibly

Table 7: A Personal Assistant's Environment

The environment of a personal assistant has two main components: the human user and the backend information system. These have different properties, which place interesting requirements on the designs of the assistants. Table 7 lists the key properties of users and backend systems using the terms introduced in Table 5. We assume that an assistant can find all that is relevant about the user and system. It can partially predict the user's behavior, which motivates having adaptive user modeling. Since the user is historical and teleological, a dialogue functionality is required along with some task modeling. Since the assistant cannot control the user, and the user can change his mind in real-time, it must allow interrupts, i.e., be mixed-initiative.

3 Architectures and Infrastructure

Table 8 reviews the development of software architectures leading to agent-based ones. In traditional terms, an agent might be a client and do everything itself or might tell a server how to do something. Correspondingly, an agent may be a server and do nothing or do exactly as told. In peer-to-peer settings with smarter agents, the client need only tell a server what it would like to have done, and the server may satisfy high-level requests. The client and server autonomously preserve their own interests. The traditional case has a mobile variant in which the messages are procedural scripts, rather than declarative invocations of services.

3.1 Agent Architectures

Agents are constructed and operate in environments whose characteristics we described in section 1. The environments impose constraints on the behavior of the agents, and provide services and facilities that can be used by the agents. A number of agent architectures have been proposed. These are typically layered in some way with components for perception and action at the bottom and reasoning at the top. The perception feeds the reasoning subsystem, which governs the actions, including deciding what to perceive next.

A challenge that agent architectures must surmount is arranging for the reasoning and perception to both be engaged and disengaged as appropriate. For example, it is undesirable for the agent to keep reasoning about some future while there is a crisis in its immediate environment; conversely, if its perceptions continually interrupt the reasoning, it would not be able to perform any complex chains of reasoning. Agents in which perception or reasoning dominates are called reactive or deliberative, respectively. There are no general, domain-independent approaches that will do the right thing in all circumstances. Some useful papers are [Bonasso *et al.*, 1996; Norman & Long, 1996; Fischer *et al.*, 1996]. We revisit related issues in section 4.

3.2 Agent System Architectures

Agent architectures look at agents in the small. It is often more important to consider agents in the large. This is where agent system architectures come in. These are meant to facilitate agents' actions in obeying the constraints of their environments, while taking advantage of available services and facilities. Additionally, the system architectures provide for different types of specialized agents to operate and interact with the environments and each other.

One type of agent manages protocols on behalf of applications and resources. In this capacity, the agents produce a layer of homogeneity among the heterogeneous components of an environment. The layer might be at a low communication level, such as produced by Aides [Singh & Huhns, 1994], heads [Lux & Steiner, 1995], and front-end processors [Zhang & Bell, 1991], at a semantic level, such as produced by knowledge handlers [Wong, 1993], ontology agents [Bayardo *et al.*, 1997], type brokers, and wrappers [Gray, 1991], or at an information management level, such as produced by mediators [Wiederhold, 1992], routers [Sayre & Gray, 1993], intelligent information agents [Papazoglou *et al.*, 1992], and facilitators [Neches *et al.*, 1991].

Type brokers provide a means to manage the structure and semantics of information and query languages. They define standard types by which computations can communicate. Most of this work pertains to lower level issues, which typically involve a set of such type brokers and a way to distribute type information. An application uses the broker to find a service, and then communicates directly with the desired service. Type brokers

Role Architecture	*Entity 1*	*Entity 2*	*Communication*
Client-Server	Master: Full Control	Slave: No Control	RPC
	Tells how	Does as told	
Distributed	Peer: Assigns tasks	Peer: Satisfies requests	Asynchronous declarative messages
	Self-interested	Autonomous	
Agent-Based	Peer: Creates or invokes commitments	Peer: Keeps commitments	Speech act
	Self-interested	Autonomous	

Table 8: Roles of Computations

give slightly more semantics than directories by including the type signature of methods, not just their names. With more sophisticated notions of service semantics, these could be more useful.

Other types of agents access information from heterogeneous sources on behalf of users or other agents. One of the best examples of these is *mediators* [Wiederhold, 1992]. A mediator is a simplified agent that acts on behalf of a set of information resources or applications. Mediators come in a wide range of capabilities, from database and protocol converters, to intelligent modules that capture the semantics of the domain and learn from the data. The basic idea is that a mediator is responsible for mapping the resources or applications to the rest of the world. Mediators thus shield the different components of a system from each other. To construct mediators effectively requires some common representation of the meanings of the resources and applications they connect, which are discussed in section 3.4.1.

3.3 Agent Frameworks

A number of research groups have made tools available for constructing agents of the types described above:

- ABE, an "Agent Building Environment" from IBM, is written in C++ and Java; agents in ABE have rule-based reasoning and interfaces to the web (http), news groups (nntp), and email (smtp)
- JAT, the "Java Agent Template," and JAT-Lite from Stanford, enable simple Java agents to communicate over a LAN via KQML
- JESS, the "Java Expert System Shell," is basically CLIPS in Java; it enables solitary reasoning agents to be constructed
- Voyager, from ObjectSpace Inc., provides an Object Request Broker for Java agents
- Open Agent Architecture, from SRI, enables the construction of agents that are based on a logic-based declarative InterAgent Communication Language and run on top of CORBA. One of the agents, a facilitator, distributes tasks to other agents, with results sent to user agents.

Table 9 shows how these tools can be categorized according to the features described in Table 4. One can see that the feature space has not nearly been explored by these tools.

3.4 Agent Infrastructures

The above tools and architectures presume or provide an infrastructure within which the agents operate and interact. The infrastructure might supply the means for an agent to communicate, to be understood, and to move. A key aspect of the infrastructure is the facilities for communications and knowledge sharing it offers.

3.4.1 Common Ontologies

A major challenge to agents' understanding each other is that they may have different systems of belief, i.e., different terms for the same concept, the same term for different concepts, different class systems or schemas, or differences in depth and breadth of coverage. Intuitively, a shared representation is essential to successful communication and coordination. For humans, this is provided by the physical, biological, and social world. For computational agents, this is provided by a *common ontology*, which is a representation of knowledge—typically taxonomic—of some domain of discourse and which is made available to all the agents and other components in an information system [Neches *et al.*, 1991].

For the present purpose, it is convenient to think of the agents—or, more properly, the information sources that underlie the agents—as databases, although they may in fact be files or sensors without all of the database functionalities [Elmasri & Navathe, 1994]. In heterogeneous environments, it is possible, and indeed common, that when different databases store information on related topics, each provides a unique model of it. The databases might use different terms, e.g., *employee* or *staff*, to refer to the same concept. Worse still, they might use the same term to have different meanings. For example, one database may use *employee* to mean anyone currently on the payroll, whereas another may use *employee* to mean anyone currently receiving benefits. The former will include assigned contractors; the latter will include retirees. Consequently, merging information

Property	**ABE**	**JAT**	**JESS**	**Voyager**	**OAA**
Design Autonomy	Platform Language Internal Arch.	Platform Language Internal Arch.	Platform Language Internal Arch.	Platform Language Internal Arch.	Platform Language Internal Arch.
Communication Infrastructure	Message-based Connection-less Multicast Push Asynchronous	Message-based Connected Point-to-point Push Synchronous	None	Message-based Connection-less Point-to-point Push Asynchronous	Message-based Connected Point-to-point Push Synchronous
Directory Service	None	Name Server	None	ORB	ORB
Message Protocol	http, nntp, smtp	KQML	None	IIOP	IIOP
Mediation Services	None	None	None	None	Facilitator
Security Services	None	None	None	None	None
Remittance Services	None	None	None	None	None
Operations Support	None	None	None	None	None

Table 9: Characteristics of Agent-Building Tools

meaningfully is nontrivial. The problem is exacerbated by competitive pressures to use advances in communications infrastructure: different companies or divisions of a large company, which previously proceeded independently of one another, are now expected to have some linkage with each other.

The linkages can be thought of as semantic mappings between the application (which consumes or produces information), and the various databases. If the application somehow knows that *employee* from a database has one meaning, it can insert appropriate tests to eliminate spurious records. Clearly, this approach would be a nightmare to maintain. The slightest changes in a database would require modifying all the applications that access its contents! This would be a fundamental step backward from the very idea of the database architecture [Elmasri & Navathe, 1994, ch. 1], which sought to separate and shield applications from the storage of data.

As we hinted above, common ontologies can be used to mediate among the semantic representations of different databases or agents. They thus promise to simplify the creation and maintenance of semantic mappings. There are two big challenges in using ontologies for this purpose: (a) to build them, and (b) to link to them. There are several large-scale efforts underway to build ontologies. These include Cyc [Lenat & Guha, 1989], DARPA ontology sharing project [Patil *et al.*, 1992], Ontology Base (ISI) [Knight & Luk, 1994], and WordNet (Princeton) [Miller, 1995]. Tools for creating semantic mappings are also being built, e.g., [Woelk *et al.*, 1996], but greater sophistication would be welcome. We revisit this point in section 5.3.

3.4.2 Communication Protocols

For interoperability, agents should be able to communicate with agents supplied by different implementors or vendors. The obvious solution is a lingua franca—whereby all the agents who implement the (same) lingua franca can communicate. To approach this ideal, an agent communication language (ACL) must be standardized, so that different parties can build their agents to interoperate. Further, it must have a formal semantics, so that different implementations preserve the essential features of the language. By specifying an ACL, we effectively codify the atoms of the interactions that can take place among autonomously developed agents. This is what makes ACLs and their standards and semantics worth studying.

KQML. The Knowledge Query and Manipulation Language, KQML [Finin *et al.*, 1994], was defined under the DARPA-sponsored Knowledge Sharing Effort. KQML assumes a layered architecture. It assumes, at the bottom, functionality for message transport or communication. It leaves, at the top, the content to be specified by the applications, typically in some formal language such as Knowledge Interchange Format (KIF) [Genesereth, 1991] or Structured Query Language (SQL) [Elmasri & Navathe, 1994]. It provides, in the middle, the primitives with which agents can exchange meaningful messages. In other words, KQML provides a way to structure the messages, but lets the agent designers decide what is in them.

KQML provides a large set of primitives through which agents may `tell` facts to or `evaluate` other agents, or `subscribe` to services. The primitives are based on speech acts (section 4.4.1). The KQML primitives fall into a few major classes. One class, which includes `tell`, `evaluate`, and `subscribe`, is geared toward the communication of content. Another class includes primitives to control the flow of information by, for example, using the performative `next` to request answers one at a time. Yet another class of primitives allows for `recruiting` agents and performing other brokering and facilitator functions.

KQML assumes the message transport is reliable and preserves the order of messages, but may not guarantee delivery times. For this reason, the underlying paradigm of communication is asynchronous; at the application-level, the effect of synchronous communication is achieved for primitives such as query and reply. KQML allows tagging messages to relate messages, e.g., responses to corresponding queries. In this way, KQML supports some elementary protocols, although more sophisticated protocols must be defined externally.

The KQML semantics is given informally. KQML agents are assumed to have a virtual knowledge base (VKB) containing beliefs and goals. They can communicate about the virtual knowledge base of themselves and others. Thus a `tell` directs the recipient to change its VKB; a `evaluate` directs the recipient to produce a response based on its VKB.

Arcol and FIPA. Arcol is another ACL based on speech acts [Sadek, 1991; Breiter & Sadek, 1996]. Arcol was the basis for the first version of the proposed standard of the Foundation for Intelligent Physical Agents (FIPA) standard, and many of its components survive in the second version as well. Agents conforming to the FIPA specification can deal explicitly with actions. They make requests, and they can nest the speech acts. The FIPA specification has a formal semantics.

Comparison. KQML suffers from, as yet, poorly defined semantics. As a result, of the many implementations of KQML, each seems unique. This makes communication difficult, and a KQML agent might not be understood. Security has not been a major issue in the KQML work. The FIPA specification, by contrast, attempts to formalize the semantics and a security model. However, in view of its recency, it has not been widely tested or adopted. We evaluate these approaches conceptually in section 4.4.

3.4.3 Interaction with the Infrastructure

An agent communication language (ACL) must inevitably interact with the infrastructure on which the agents are created and managed, and over which the communications are effected. Although a number of the transport issues can be separated from the ACL—and have been traditionally—certain issues merit further attention. These include provisions in the infrastructure for multicasting, making the identity of message receivers known, and covering important locutions and conversational structures.

Coverage of Locutions and Conversational Structures. An ACL should include locutions for dealing appropriately with the underlying information system, e.g., in terms of initiating and maintaining sessions, authorizing actions, committing to actions, rolling back to a previous state in the dialogue in case of corruption or error. Additional locutions are required for human-computer dialogue, where the dialogue is mixed-initiative and prolonged, and the participants have limited memory or attention spans. A classification of conversational locutions along these lines is being developed [Singh *et al.*, 1997]; the sessional, commissive, and authorizational locutions, in particular, are applicable to ACLs.

Identity of Receivers. At some point in the course of sending a message, the identity of the receiver must be determined so the message can be delivered. How late in the flow of control can this binding be established? In some applications, e.g., workflow management, the receiver is only known by its role until a specific message needs to be sent. The management of roles is an intrinsic part of designing and implementing multiagent systems. We believe there should be functionality for role management in the agent infrastructure with corresponding primitives in the ACL component.

Knowing the identity of the receivers bears on the ACL semantics. When a message is to be sent to a potential receiver whose identity is unknown, the sender can make no assumptions about the receivers' beliefs or intentions, because the lower (transport) layer cannot validate them. Consequently, a semantics which requires the sender to reason about the receiver leads to an ACL in which known agents are preferred.

Cardinality of Receivers. Although early approaches generally considered exactly one receiver for each message, many applications require multicasting. Multicasting is typically worked in through some additional mechanism, e.g., a special forwarding or broadcast agent, which is not a part of the ACL. However, distributed systems are emerging that support multicast, and an ACL should exploit these distributed systems where available. The challenge as before is to ensure that the reasoning required about the beliefs and intentions of the potential receivers is minimized.

3.4.4 Interaction Protocols

Several interaction protocols have been devised for systems of agents. In cases where the agents have conflicting goals or are simply self-interested, the objective of the protocols is to maximize the payoffs (utilities) of the agents [Rosenschein & Zlotkin, 1994]. In cases where the agents have similar goals or common problems, as in distributed problem solving (DPS), the objective of the protocols is to maintain globally coherent performance of the agents without violating autonomy, i.e., without explicit global control [Durfee, 1988]. For the latter cases, important aspects include how to

- determine shared goals
- determine common tasks
- avoid unnecessary conflicts
- pool knowledge and evidence.

A basic strategy shared by many of the protocols for DPS is to decompose and then distribute tasks. Such a divide-and-conquer approach can reduce the complexity of a task: smaller subtasks require less capable agents and fewer resources. However, the system must decide among alternative decompositions, if available, and the decomposition process must consider the resources and capabilities of the agents. Also, there might be interactions among the subtasks and conflicts among the agents.

Task decomposition can be done by the system designer, whereby decomposition is programmed during implementation, or by the agents using hierarchical planning, or it might be inherent in the representation of the problem, as in an AND-OR graph. Task decomposition might be done spatially, based on the layout of information sources or decision points, or functionally, according to the expertise of available agents.

Once tasks are decomposed, they can be distributed according to the following criteria [Durfee *et al.*, 1987]:

- Avoid overloading critical resources
- Assign tasks to agents with matching capabilities
- Make an agent with a wide view assign tasks to other agents
- Assign overlapping responsibilities to agents to achieve coherence
- Assign highly interdependent tasks to agents in spatial or semantic proximity. This minimizes communication and synchronization costs
- Reassign tasks if necessary for completing urgent tasks.

The following mechanisms are commonly used to distribute tasks:

- Market mechanisms: tasks are matched to agents by generalized agreement or mutual selection (analogous to pricing commodities)
- Contract net: announce, bid, and award cycles
- Multiagent planning: planning agents have the responsibility for task assignment
- Organizational structure: agents have fixed responsibilities for particular tasks.

Of these, the best known and most widely applied is the Contract Net protocol [Davis & Smith, 1983]. This generic protocol repeats the following steps:

- A manager announces the existence of tasks via a (possibly selective) multicast. The task announcement describes the tasks and the criteria and format for bidding.
- Agents evaluate the announcement, and some of these agents submit bids. A bid specifies the agent's relevant capabilities for performing the task.
- The manager evaluates the bids and awards a contract to the most appropriate agent. The award consists of a complete specification of the task.
- The manager and contractor then communicate privately as necessary.

The Contract Net is best used when the application has a well-defined hierarchy of tasks, the problem has a coarse-grained decomposition, and the subtasks minimally interact with each other, but cooperate when they do. The result is not only a high-level communication protocol for distributing tasks, but also a means of self-organization for a group of agents.

3.4.5 Mobility

While most agents are static, in that they exist as a single process or thread on one host computer, others can pick up and move their code and data to a new host in the web, where they then resume executing.

Are such agents mobile, itinerant, dynamic, wandering, roaming, or migrant? And, are they sent, beamed, teleported, transported, moved, relocated, or RPC'd? These are some of the questions swirling around web discussion groups these days. However, since anything that can be done with mobile agents can be done with conventional software techniques, the key questions are really

- Are mobile agents a useful part of a distributed computing system?
- Are there applications that are easier to develop using mobile agents?
- Under what circumstances is it useful for an agent to be mobile?

We find that there are very few such circumstances, in spite of all the effort being spent on developing techniques for mobility. And, there is a fundamental reason, the dichotomy between procedural and declarative constructions (section 1.3.7), why this is so. Nevertheless, there are several interesting uses for mobile agents.

Distributed Software Updates. In general, the best applications for mobility might be those that involve the dynamic installation of code to update software or extend the functionality of an existing system. This would address a potential limitation of current static systems, which are not easily enhanced. However, new functionality can be installed without requiring a full-blown mobile agent, by using the standard message type `install(function_name, version, argument_types, code)`. The receiving agent can autonomously decide—based among other things on its level of trust in the sender—whether to install the corresponding code; if it does, new functionality becomes available. And, state information never needs to be shipped around.

Disconnected Operation. A major consideration for personal digital assistants (PDAs) is battery capacity and, therefore, connect time. Because of this, PDAs are forced to spend most of their existence off-line. Now, suppose you have constructed an agent that knows your preferences and interests, and can filter information sent to you from multiple sources. Further, suppose your agent can provide real-time feedback to the sources that would enable them to improve the precision of their information. This agent can run on your PDA, where you can interact with it and instruct it. However, you do not want your agent to stop functioning when you turn off your PDA—when this happens, your agent should move to a host that is on-line.

Multihop Applications. Mobile agents appear suited to testing distributed network hardware. For example, a distributed telecommunication switch is built out of thousands of different cards, with different test rules. The code for thorough testing can be quite large, and can improve over time. A traditional approach is to load the board-level testing code directly into the boards and have these boards self-test periodically, sending their results to a main testing controller. The system level tests do not fit into the boards and consume too much network bandwidth, so they are loaded remotely when the system is inactive.

A mobile agent approach is to launch testing agents into the active network. These agents can roam between boards, performing tests stochastically. This allows boards to accommodate many testing strategies with a small amount of memory, since the agents can come and go over time. Testing agents carry with them both their previous testing history and the means to perform the test. Testing agents can make local decisions, repeating tests or testing neighboring boards without having to report back to a central controller.

Information Commerce. There are times when an information consumer would like to apply proprietary algorithms from one company to proprietary data from another company. A solution would be to find a trusted third party to whom both the data and the algorithms, encoded in a mobile agent, could be sent.

Customized Searches on Servers. A frequently proposed use for mobile agents is to send them to execute on servers, particularly when the servers have more information than can be reasonably communicated back to a client for processing there and lack the necessary procedures to perform the desired processing themselves.

Procedural vs. Declarative. We view the mobility of agents as primarily an issue of infrastructure—a matter of how agent functionality might be realized. A client seeking information from a server can either send a procedure to execute on the server and find the desired information, or send a message requesting the server to find the information using its own procedure. Our criticism of mobile agents is that they are a low-level procedural means to achieve what communication techniques can support at a higher declarative level.

For example, to customize searches on a server, a declarative approach would implement a protocol of search primitives that could be invoked via messages. This approach would mitigate the security worries that the mobile agent would run amok, intentionally or otherwise. It would also offer efficiency advantages. When a mobile agent runs remotely, the server gives up control of disk, memory, and processor resources to the agent. Instead, if the server accepted a sequence of declarative search primitives, it could schedule and carry them out in a way that is optimized to its current state. For example, a modern DBMS could use its own optimized techniques to compute a join much more efficiently than a remote user could program an agent.

Mobility includes procedural encodings in two distinct respects. One, the behavior of a mobile agent is procedurally coded. This might be reasonable for some static agents as well. Two, the interactions of a mobile agent are implicit in the code that constitutes it. This is unnecessary when the agent is static. A static agent's interactions can be explicitly specified in terms of protocols involving its communications. Static agents can then be supplied by different vendors and programmed in different languages as long as they communicate properly with each other.

Ultimately, there is no difference between a complex request language and a simple programming language. There is, in fact, a continuum of approaches.

Mobile Agent Frameworks. There are a number of active efforts underway to develop systems, protocols, and frameworks for both the construction and use of mobile agents. Most of the following frameworks allow agents to be started, stopped, and moved, and a few allow them to be monitored.

There is an effort by the Object Management Group (OMG) to establish industry standards for mobile agent technology and interoperability among agent systems, such as Odyssey, Aglets, and MOA. The OMG intends to define a Mobile Agent Facility (MAF) for CORBA.

Security Concerns. There are two main aspects to security involving mobile agents. The first, and most commonly considered, is protection of the server against intentionally or accidentally malicious agents. The second is protection of a mobile agent against malicious servers. The former aspect has been dealt with extensively in the context of operating systems, which establish and maintain protection levels for process execution. Security in the latter aspect cannot be guaranteed, because in order for the mobile agent's code to be executed, the agent has to expose both its code and data to the server. A detection, but not prevention mechanism, is to have the agent return itself with its data, to verify that it has not been altered.

A prevention mechanism might hinge on a determination of legal responsibility: are you liable or not for your agent's deeds? And, who pays if, e.g., a malicious server causes you to buy something on another server?

Authentication, integrity, confidentiality, and nonrepudiation are other important aspects of security. Authentication validates the identity of the person or agent with whom you are interacting. Integrity ensures that what you see has not been tampered with, confidentiality ensures that what you intend to be private remains so, and nonrepudiation means you are liable and cannot change your mind.

Challenges for Mobile Agents. We believe that mobility has a useful, albeit limited, role to play in agent computing. For it to be successful in that role, agent languages are needed that

- can express useful remote computations
- do not violate the security of the sender or receiver
- are portable and extensible.

Mobility can improve the survivability of an agent—it can move if its execution on a host is threatened—but it can also result in the agent continuing to exist long after its usefulness has ended. Once mobile agents are launched, it is nontrivial to monitor and manage them. The problem is compounded when agents are given the ability to replicate. Consequently, techniques for managing distributed computations are needed that can

- disseminate extensions to the programming language interpreter
- ensure security
- control agent lifetimes
- prevent the flooding of communication or storage resources.

4 Models of Agency

The study of agents inevitably includes some model of agency. There is an immense body of literature on philosophies and theories of agents, which we could not hope to review here. The interested reader may consult [Barwise & Perry, 1983; Brand, 1984; Bratman, 1987; Davidson, 1980; Dennett, 1987; Goldman, 1970; Miller *et al.*, 1960; Pylyshyn, 1984; Ryle, 1949; Simon, 1996; Stalnaker, 1984].

Simply put, there are three major classes of models or philosophies of agents and their environments.

- *Behaviorism* or *positivism* is the doctrine that considers only the direct behaviors of agents and their environments. In its most rigid form, this doctrine takes the view that the universe involves plain sequences of events; any patterns are in the eye of the beholder. Therefore, even causality might not be a first-class concept. The agents do not have any mental states such as intentions or beliefs; multiagent systems have no social states such as commitments.
- *Subjectivism*, by contrast, takes the view that there are intentions and commitments, but they are as represented in an agent. Similarly, the universe can have causation, but only as represented in an agent.
- *Realism* takes the middle ground. The universe has causes; the agents have intentions and beliefs; multiagent systems exist and involve the agents' commitments. These abstract notions are somehow grounded in the universe.

Of the above, behaviorism essentially throws away the abstractions that are crucial for modeling a complex world, and for designing agents to operate in it. Subjectivism admits the abstractions, but puts few bounds on them, thus leaving it susceptible to charges of being *ad hoc*. Realism takes a pragmatic position in admitting the necessary abstractions, but asking that they be related to the model of the world independent of the views of a specific agent. This position is attractive in many respects, but is technically more challenging, because it is often nontrivial to relate high-level abstractions to the "real" world.

The models are related to the perspective a given theory of agency might take. These perspectives can be the agents', the designers', or the evaluators'. Table 10 shows how the model and the perspective interact. Under subjectivism, the external evaluators' perspective is impossible, but the agents' perspective is natural, as in traditional AI. Similarly, in behaviorism, the agents' perspective is meaningless, but the evaluators' perspective is natural, as in traditional distributed computing (DC). The designers' perspective, which relates the agents' and the evaluators' perspectives, is unnatural and can be carried only with additional assumptions. Under realism, the agents' perspective guides their action (as in subjectivism), the evaluators' perspective enables testing compliance (as in behaviorism), and the designers' perspective enables relating the two. However, as remarked above, it is nontrivial to carry this out.

4.1 Rational Agency: Logical

Logical rationality includes logical, typically qualitative concepts, such as consistency of beliefs or the suitability of actions given beliefs and intentions.

4.1.1 Consistency Maintenance

A somewhat limited, but widely employed, class of logical models involves the single abstraction of belief. The models capture relationships among the beliefs of an agent, which enables maintaining consistency of the beliefs. There are two main views of the consistency. One is *well-foundedness*, which states that all beliefs, except premises, should be justified by other beliefs, and these justifications should be acyclic. The other view is *coherence*, which states that the beliefs should hold together as a coherent body, even if they lack external justification. Under well-foundedness, an agent cannot hold unsupported beliefs, but under coherence it can. It is recognized that human behavior is often closer to coherence than well-foundedness, although traditional logic favors the latter. Both approaches are used successfully for agents through tools called truth maintenance system (TMSs).

The well-founded view is implemented as justification-based TMSs (JTMSs) [Doyle, 1979] and the coherent view as assumption-based TMSs (ATMSs) [de Kleer, 1979]. In general, a TMS performs some form of propositional reasoning to update beliefs incrementally when data are added or removed. Using a TMS imposes a special architecture on an agent, in which the problem-solving or reasoning component is separated from the TMS. The former represents domain knowledge, e.g., as rules or procedures, and chooses what to focus on next. The TMS keeps track of the current state of the actions of the problem-solver. It uses constraint satisfaction to maintain consistency in the inferences made by the problem solver. Table 11 shows how the integrity of knowl-

Perspective Dimension	*Agents'*	*Designers'*	*Evaluators'*
Subjectivism	Traditional AI	With caveats	Impossible
Realism	Guide action	Implement	Test compliance
Behaviorism	Impossible	With caveats	Traditional DC

Table 10: Dimensions of Models Related to Perspective

edge should be maintained by an agent.

Property	**Meaning**
Stability	believe everything justified validly; disbelieve everything justified invalidly
Well-Foundedness	beliefs are not circular
Consistency	no contradictions
Completeness	find a consistent state, if any

Table 11: Knowledge Integrity

Single-agent TMSs meet all the requirements of Table 11. However, additional problems arise when knowledge is distributed, and different agents must achieve consistency. The kinds of inconsistency include

- Both a fact and its negation are believed
- A fact is both believed and disbelieved
- An object is believed to be of two incompatible types, i.e., two terms are used for the same object
- Two different objects are believed to be the same
- A single-valued fact is given multiple values.

Degree	**Meaning**
Inconsistency	one or more agents are inconsistent
Local Consistency	agents are locally consistent
Local-and-Shared Consistency	agents are locally consistent and all agents are consistent about shared data
Global Consistency	agents are globally consistent

Table 12: Degrees of Logical Consistency

In light of the above, different degrees of logical consistency in a multiagent system may be defined [Huhns & Bridgeland, 1991]. Local consistency leaves open the possibility that the different agents may be in serious disagreement. However, global consistency is typically not tractable or even essential. In many cases, it is enough that the agents are in agreement about the data that they share. For this reason, the distributed JTMS (DTMS) maintains local-and-shared consistency and well foundedness [Huhns & Bridgeland, 1991]. In this approach, each agent has a justification-based TMS, but the justifications can be external, i.e., based on what another agent said. Agents keep track of what they told to whom, so they can suggest updates when their original assertions are no longer supported.

The Distributed ATMS was introduced in [Mason & Johnson, 1989]. In this approach, the agents are locally, but not globally, consistent, based on a local ATMS. The agents believe only results they can substantiate locally, and communicate by explicitly contradicting their previous assertions that have been invalidated.

4.1.2 Mental Agency

The consistency maintenance view is useful and attractive. However, it fails to tell the whole story of agency. What goes in the reasoner component of an agent? How might it affect the so-called beliefs of that agent? What other abstractions would we need to characterize the agent's actions and purposeful behavior?

Some powerful abstractions are based on the mental concepts of folk psychology. Roughly, *beliefs* characterize what an agent imagines the state of the world to be, i.e., the state of the world as "represented" by the agent; *know-how* characterizes what the agent can really control in its environment; *goals* describe what states the agent would prefer; *desires* describe the agent's preferences and may sometimes have a motivational aspect; *intentions* characterize the goals or desires that the agent has selected to work on. Table 13 puts these concepts in broad categories, which resemble the characteristics of environments of Table 5, but with respect to the agent. The above concepts have been extensively studied, e.g., see [Belnap & Perloff, 1988; McDermott, 1982; Singh, 1992] and the citations below.

There are a large number of variations on the formalizations of beliefs and intentions, but they fall into two main categories. One class of definitions includes what we term the "implicit" versions of these concepts, e.g., [Moore, 1984; Cohen & Levesque, 1990; Rao & Georgeff, 1991; Singh, 1994]. These approaches are technically more tractable. However, they support the inference that if an agent believes or intends a proposition, he believes or intends all propositions that are logically equivalent. This is, of course, patently false when applied to the actual beliefs or intentions of limited agents. However, it can be justified for modeling agents from a designer or evaluator's perspective, in which only the relevant beliefs and intentions need to be explicitly considered. The second class of definition takes a "representationalist" stance toward beliefs and intentions [Konolige, 1986; Fagin & Halpern, 1988; Konolige & Pollack, 1989; Singh & Asher, 1993]. These are more accurate from the agents' perspective, because

Abstraction	**Rough Meaning**
Belief, Knowledge	Information about Environment
Ability, Know-How, Seeing-to-it-that	Control of Environment
Goals, Desires, Intentions	Purposeful Behavior

Table 13: Mental Abstractions for Agents

they do not support the above inference. However, they tend to support few inferences in general, which makes them less tractable technically.

```
{
  acl.send(Bo, "inform(raining)")
}
```

Figure 1: Al, A Very Simple Agent: Does it Comply?

The philosophical positions of the *intentional stance* [Dennett, 1987; McCarthy, 1979] or the *knowledge level* [Newell, 1982] are that any system can in principle be described using "intentional" terms, such as beliefs and intentions. This is indeed a compelling view for the purposes of understanding or designing intelligent systems. However, this view does not guarantee that there is a unique representation of a complex system in terms of beliefs and intentions. In other words, it does not solve the practical problem of how the beliefs and intentions are mapped from the design of an arbitrary agent. For example, how can we say by just looking at the source code whether agent Al of Figure 1 believes that it is raining or not? A narrow definition is that the agent should have the string "raining" in a data structure labeled `beliefs`. But this violates design autonomy. Moreover, two agents with the same `beliefs` data structure could act differently enough that it wouldn't be clear whether they *really* had the same beliefs.

The mental states of agents are important only because they are related to the agents' actions, through which the agents attempt to control their environment and satisfy their ends. There is a long tradition in AI for reasoning about actions and planning, which we could not hope to summarize here [Allen *et al.*, 1990]. This work, which considers models of actions and reasons about their preconditions and effects, is intimately related to studies of know-how and intentions [Singh, 1994; Lespérance *et al.*, 1996].

In addition to defining the above concepts, we must also characterize their evolution: how do the beliefs and intentions of an agent change in response to fresh evidence or in response to additional reasoning. This remains an unsolved problem in general, although good progress has been made in some areas, especially belief revision [Gärdenfors, 1988; Haddawy, 1996]. There has also been some preliminary work on intention adoption, maintenance, and revision [Georgeff & Rao, 1995; Wobcke, 1995; Singh, 1997b].

Many current theories include the notion of *commitments*, which are understood as being internal, i.e., applying only to the given agent's "attachment" to their beliefs or intentions. An agent who is committed to his beliefs may keep holding them even when the justification is lost [Harman, 1986; Gärdenfors, 1988]—this is closely related to the coherence view of TMSs discussed above. Similarly, an agent may be committed to his intentions and would not give them up easily [Brand, 1984; Bratman, 1987]. These might appear to be patently irrational behaviors, but they usually are not, because the agents are computationally limited. Reconsidering one's beliefs and intentions presupposes a computationally expensive process of deliberation. There is a trade-off between (a) doggedly holding beliefs and intentions longer than warranted and (b) paying the price of reasoning about them. Option (a) can be seen as a form of *satisficing* in which an agent merely seeks a "good enough" or acceptable solution, not the optimal solution [Simon, 1996]. [Singh, 1997b] suggests a way to reconcile commitments with rationality, but the problem is still wide open.

4.2 Rational Agency: Economic

Economic approaches assume that the agent's preferences are given along with knowledge of the effects of the agent's actions. From these, the rational action is the one which maximizes preferences.

Economic rationality has the charm of being a simple, "least common denominator" approach—if you can reduce everything to money, you can talk about maximizing it. But to apply it well requires a careful selection of the target problem.

One of the oldest applications of economic rationality is in decision-theoretic planning, which models the costs and effects of actions quantitatively and probabilistically. For many applications, where the probabilities can be estimated reliably, this leads to highly effective plans of actions [Horvitz & Rutledge, 1991; Haddawy, 1996].

The need to maximize preferences essentially requires that there be a scalar representation for all the true preferences of an agent. In other words, all of the preferences must be reduced to a single scalar that can be compared effectively with other scalars. This is often difficult unless one can carefully circumscribe the application domain. Otherwise, one ends up essentially recreating all of the other concepts under a veneer of rationality. For example, if we would like an agent to be governed by its past commitments, not just the most attractive choice at the present time, then we can develop a utility function that gives additional weight to past commitments. This approach may work in principle, but, in practice, it only serves to hide the structure of commitments in the utility function that one chooses.

Principles of Negotiation. One of the important applications of economic rationality is in certain kinds of negotiation. These involve a small set of agents with a common language, problem abstraction, and solution. In the unified negotiation protocol of [Rosenschein & Zlotkin, 1994], a *deal* is a joint plan between two agents that would satisfy both of their goals. The utility of a deal for an agent is the amount he is willing to pay minus the cost of the deal. The negotiation set is the set of all deals that have a positive utility for every agent. There are three possible situations:

- conflict: the negotiation set is empty
- compromise: agents prefer to be alone, but will agree to a negotiated deal
- cooperative: all deals in the negotiation set are preferred by both agents over achieving their goals alone.

Since the agents have some execution autonomy, they can in principle deceive or mislead each other. Therefore, an interesting research problem is to develop protocols or societies in which the effects of deception and misinformation can be contained. Their goal is to develop protocols under which it is rational for agents to be honest with each other.

The connections of the economic approaches with human-oriented negotiation and argumentation have not been fully worked out.

Market-Oriented Programming. Computational markets are an approach to distributed computation based on market mechanisms [Wellman, 1995]. These are effective for coordinating the activities of many agents with minimal direct communication among the agents. The research challenge is to build computational economies to solve problems of distributed resource allocation.

Everything of interest to an agent is described by current prices—the preferences or abilities of others are irrelevant except insofar as they (automatically) affect the prices. Agents offer to exchange goods at various prices. There are two types of agents, *consumers*, who exchange goods, and *producers*, who transform some goods into other goods. All agents bid so as to maximize profits or utility.

The important property is that an equilibrium corresponds—in a sense, optimally—to an allocation of resources and dictates the activities and consumptions of the agents. In general, equilibria need not exist or be unique, but under certain conditions, such as when the effect of an individual on the market is assumed negligible, they can be guaranteed to exist uniquely.

4.3 Social Agency

Social agency involves abstractions from sociology and organizational theory to model societies of agents. Since agents are often best studied as members of multiagent systems, this view of agency is important and gaining recognition. Sociability is essential to cooperation, which itself is essential for moving beyond the somewhat rigid client-server paradigm of today to a true peer-to-peer distributed and flexible paradigm that modern applications call for, and where agent technology finds its greatest pay offs.

Although the mental primitives are appropriate for a number of applications and situations, they are not suitable in themselves for understanding all aspects of social interactions. Further, economic models of agency, although quite general in principle, are typically limited in practice. This is because the value functions that are tractable essentially reduce an agent to a selfish agent. [Conte & Castelfranchi, 1995] argue that a self-interested agent need not be selfish, because it may have other interests than its immediate personal gain. This is certainly true in many cases when describing humans, and is likely to be a richer assumption for modeling artificial agents in settings that are appropriately complex.

Social commitments are the commitments of an agent to another agent. These must be carefully distinguished from internal commitments, as discussed in section 4.1. Social commitments have been studied by a number of researchers, including [Gasser, 1991; Jennings, 1993]. There are a number of definitions in the literature, which add components such as witnesses [Castelfranchi, 1995] or contexts [Singh, 1997a]. Social commitments are a flexible means through which the behavior of autonomous agents is constrained. This can be voluntary when the agents adopt the roles that bind them to certain commitments.

Coordination and Coherence. Coordination is a property of a system of agents performing some activity in a shared environment. The degree of coordination is the extent to which they avoid extraneous activity by reducing resource contention, avoiding livelock and deadlock, and maintaining applicable safety conditions. Cooperation is coordination among nonantagonistic agents. Typically, to cooperate successfully, each agent must maintain a model of the other agents, and also develop a model of future interactions. This presupposes sociability.

Coherence is how well a system behaves as a unit. It requires some form of organization. Social commitments are a means to achieve coherence. [Simon, 1996] argues eloquently that although markets are excellent for *clearing* all goods, i.e., finding a price at which everything is sold, they are less effective in computing optimal allocations of resources. Organizational structures are essential for that purpose. We believe that coherence and optimality are intimately related.

4.4 Interactive Agency

Interactions occur when agents exist and act in close proximity. Interactions can be of various kinds, but can be classified into two main categories: intended or otherwise. The best example of unintended interactions is resource contention, e.g., when agents accidentally bump into each other or inadvertently delete files the other needs to access. Intended interactions are primarily communications—these may occur through shared resources, which can then be viewed as a kind of shared memory. For example, an agent may delete a file to indicate that the other agent is prohibited from proceeding. Typically, for such communication to take place, some shared conventions must be in place, just as for communications based on language.

Communication is generally more expensive and less reliable than computation:

- Recomputing is often faster than requesting information over a communication channel
- Communication can lead to prolonged negotiation
- Chains of belief and goal updates caused by communication might not terminate.

However, communication is qualitatively superior, because information cannot always be reconstructed locally, and communication can be avoided only when the agents are set up to share all necessary knowledge a priori. This is a highly limiting assumption, of course.

4.4.1 Speech Acts

Speech acts are used to motivate message types in agent systems, although speech act theory was invented in the fifties and sixties to help understand human language. The idea is that language is not only for making statements, but also for *performing actions* [Austin, 1962]. For example, when you request something, you do not just report on a request, but you actually cause the request; when a justice of the peace declares a couple man and wife, she is not reporting on their marital status, but changing it. The stylized syntactic form for speech acts that begins "I hereby request ..." or "I hereby declare ..." is called a performative. With a performative, literally, saying it makes it so! [Austin, 1962, p. 7]. Interestingly, verbs that cannot be put in this form are not speech acts. For example, "solve" is not a performative, because "I hereby solve this problem" just doesn't work out—or students would be a much happier lot!

In natural language, it is not always obvious what speech act is being performed. For example, if Mike says "It's cold here", he might be telling you about the temperature, or he may be requesting you to turn up the heat. Artificial languages can avoid this problem.

Several hundred verbs in English have performative forms. This obviously calls for classifications, and many have been given. For most computing purposes, speech acts are classified into assertives (informing), directives (requesting or querying), commissives (promising), prohibitives, declaratives (causing events in themselves, e.g., what the justice of the peace does in a marriage ceremony), and expressives (expressing emotions).

The speech acts have a social component, which relies upon commitments. For example, a commissive creates a commitment from the sender to the receiver to do as promised. An assertion might create a commitment from the sender about the veracity of the asserted proposition. A directive might presuppose some commitment on the part of the receiver to do as told.

4.4.2 Compliance

When different vendors build their products to a standard, there must be a normative means to verify that the products are indeed compliant. If an interaction breaks down, the different developers should be able to determine which of the components were not complying with the standard, so they can be fixed. In our view, a "standard" is meaningless if it does not yield some tests of compliance.

One can be sure that agents who are communicating are interacting with other agents, and are therefore—by that fact alone—social. To preserve design autonomy, we cannot require that they are mentalistic. For example, consider the agent Al as given in Figure 1. Does Al comply? In fact, in one approach, if Al's designer simply wanted it to comply it does!

4.4.3 Dimensions of Meaning

We use the terms *meaning* and *understanding* informally, and the terms *semantics* and *pragmatics* technically. Any agent communication language (ACL) must—explicitly or implicitly—take specific positions along the *dimensions of meaning*. We describe these next.

One design issue is whether the given ACL semantics is designed to mirror human languages or not. Traditional theories of computational linguistics and discourse build upon work in linguistics and philosophy. However, the needs of human communication are potentially greatly different from the needs of communication for artificial agents. This makes traditional discourse approaches unsuitable for ACLs in general, although they apply well to personal assistants [Breiter & Sadek, 1996].

Personal vs. Conventional Meaning. Personal meaning is based on the "intent" or interpretation assigned by one of the participants. It is a generalization over Grice's notion of utterers' meaning [Grice, 1969]. The personal meaning doctrine bases the meaning of the speech acts on the intentions of the agent who performs the given act, or even the intentions of the agent(s) toward whom the act is directed.

Conventional meaning is based on the conventions of usage. The very idea of an agent lingua franca is founded on there being a well-defined conventional meaning. Indeed, having different explicit acts is a way of streamlining the conventional meaning. Inherently, no formal specification can avoid the conventions of having different labels for the different speech acts.

However, traditional approaches indicate a bias toward personal (utterer's) meaning, e.g., [Sadek, 1991]. If an agent says *inform* but lacks the corresponding beliefs and intentions, then under traditional theories an *inform* does not occur. Conversely, if an agent says *request* but has beliefs and intentions corresponding to an *inform*, then an *inform* does occur.

There is effectively no conventional meaning, just idiolects of different agents! Idiolects cannot be standardized!

Perspective. There are potentially three different perspectives on each communicative act—of the sender, the receiver, or the society or other observers. Whereas the personal vs. conventional dichotomy discussed above has to do with the meaning of specific communicative acts, the perspective has to do with whose viewpoint that meaning is expressed in. Traditional approaches are concerned only with the sender. This is not a popular view, even in the recent discourse literature, where the sender and receiver are treated as equal participants in the discourse. However, a recent proposal looks at the postconditions of communications, which also take the receiver's perspective [Labrou & Finin, 1997]. The social perspective should not be ignored in specifying the meaning of an ACL, because testing compliance presupposes a public perspective.

Semantics vs. Pragmatics. There are classically three aspects to the formal study of language: *syntax*, *semantics*, and *pragmatics* [Morris, 1938]. Syntax deals with how the symbols are structured, semantics with what they denote, and pragmatics with how they are interpreted (or used). Intuitively, the meaning of a communication is a combination of its semantics and pragmatics. Pragmatics inherently includes considerations, such as the environment and mental states of the communicating agents, that are external to the language proper. The Gricean maxims of communication are a prime example of pragmatics [Grice, 1975].

- *Quantity:* make your contribution (to the conversation) as informative as necessary, but no more informative (to reduce confusion)
- *Quality:* try to make only true contributions—don't say what you believe is false or for which you lack adequate evidence
- *Relation:* be relevant
- *Manner:* avoid obscurity and ambiguity; be brief and orderly.

The Gricean maxims are a cornerstone of discourse processing. They are used as (a) requirements for cooperation, and (b) bases for performing additional inference (implicature) when they are (apparently) violated. Interestingly, the performance conditions of Arcol essentially follow the discourse literature, but embody primarily the maxim of quality. Although the Gricean maxims are important, their suitability for artificial languages is not established. In any case, the maxims are pragmatic, and it is not conceptually clean to include them in the semantics of an ACL.

Contextuality. In general, it is impossible to understand communications on purely conventional terms without regard to context. There are a number of components to context, several of which involve the particular physical (or simulated) environment in which the agents are situated. For example, in understanding definite descriptions, such as "the user," we need to know who the given (user interface) agent is dealing with. This form of context is crucial in computer-human applications, and where artificial agents collaborate in a shared environment. By and large, this form of context applies to the content language, although it could apply in the ACL itself in terms of identifying the appropriate agent. Another component of context that we must deal with is social context. It suffices to state that social context determines the agents' expectations of one another—in terms of expected ranges of responses, sincerity, and so on.

Importantly, context can play a role in the meaning of a language only if the semantics is not over-specified. For example, consider the simple communicative act of informing. Arcol requires that the informing agent believe that the proposition being asserted is true, that the informed agent does not already believe it, and that the informer intends that the informed agent come to believe it. This is an instance of over-specification, because some agents, e.g., politicians, may say only what they believe their interlocutors agree with [Huhns & Singh, 1997]. The ACL cannot prevent that behavior without violating execution autonomy.

Summary of Meaning Dimensions. Addressing the above dimensions, we can talk of compliance more reasonably. In a fundamental sense, whether Al from Figure 1 complies or not depends on Al's actions viewed from the public perspective, contextualized by the protocol or society in which Al participates. Table 14 summarizes our views of what an appropriate ACL should be like, and contrasts them with Arcol and KQML. This table only considers the dominant position of each view on each axis.

Dimension	Desired	Arcol	KQML
Conventional?	Y	N	N
Pragmatic?	N	Y	Y
Perspective:	Social	Sender's	Sender's
Contextual?	Y	N	N

Table 14: Dimensions of Meaning

4.5 Adaptive Agency

One of the important properties of agents, and one that lay users expect, is that agents are adaptive. This typically presupposes that the agents are persistent and can learn. Much of the learning that one encounters in connection with agents has to do with learning values for some parameters, e.g., to personalize a user interface by modeling the user. This and some other work essentially applies ideas from traditional machine learning to agents, which is indeed valuable. The challenge is to cast agent learning in terms of existing approaches. For example, learning from an environment is appropriate for agents that behave autonomously to some extent. A good introduction to this literature is [Shen, 1993].

5 Future Directions and Challenges

The successes that agent technology has enjoyed in distributed sensing, information retrieval, and collaborative filtering has led to many other efforts to address new application areas and to research additional capabilities. Three of the most important directions are (1) using agents as a basis for software and its development, (2) formulating principles for the design and description of agent-based systems, and (3) adding a learning capability to agents.

5.1 Agent-Based Software Engineering

The field of software engineering has exhibited almost glacial progress over the last twenty years. Programmers still produce approximately the same number of lines of tested and debugged code per day as they did in 1975, in spite of several "silver bullets," such as structured programming, declarative specifications, object-oriented programming, formal methods, and visual languages. This should not be surprising, for three reasons:

1. Software systems are the most complicated artifacts people have ever attempted to construct
2. Software systems are (supposedly) guaranteed to work correctly only when all errors have been detected and removed, which is infeasible in light of the above complexity
3. The effect of an error is unrelated to its size, i.e., a single misplaced character out of millions can render a system useless or, worse, harmful.

Software engineering attempts to deal with these problems by considering both the process of producing software and the software that is produced. The major goal for the software is that it be correct, and the major goal for the process is that it be conducted efficiently. One fundamental approach to meeting these goals is to exploit modularity and reuse of code. The expectations are that small modules are easier to test, debug, and verify, and therefore more likely to be correct, that small modules will be more likely to be reused, and that reusing debugged modules is more efficient than coding them afresh. This is the major result of the series of programming paradigms that have evolved from the machine language of the 1950's:

1. Imperative: procedure-oriented programming
2. Declarative: functional and logic programming
3. Interactive: object-based and distributed programming.

However, software has not kept pace with the increased rate of performance for processors, communication infrastructure, and the computing industry in general [Lewis, 1996]. Whereas processor performance has been increasing at a 48% annual rate and network capacity at a 78% annual rate, software productivity has been growing at a 4.6% annual rate and the power of programming languages and tools has been growing at an 11% annual rate. CASE tools, meant to formalize and promote reuse, have not been widely adopted [Iivari, 1996]. In addition to these sluggish rates for software, there is a legacy of approximately 50 billion (10^9) lines of Cobol, representing roughly 80% of all software written since 1960. It is unlikely that we can replace it anytime soon, even though maintaining it is a $3 billion annual expense.

The current "hot" computing paradigm is based on Java, and the ability it provides for users to download the specific functionality they want when they request it. This is leading to the rise of a software-component industry, which will produce and then distribute on demand the components that have the users' unique desired functionality [Yourdon, 1995]. However, because of this uniqueness, how can component providers be confident that their components will behave properly? This is a problem that can be solved by agent-based components that actively cooperate with other components to realize the user's goals.

5.1.1 Requirements for New Applications

Evolving applications have characteristics that lead naturally to an agent-based approach to their development [Woelk *et al.*, 1995]:

- They solve a specific business problem by providing a user with seamless interaction with remote information, application, and human resources.
- The identities of the resources to be used are mostly unknown at the time the application is developed.
- The pattern of interaction (workflow) among the resources is a critical part of the application, but the pattern might be unknown at the time the application is developed and might vary over time.

The development of these new applications requires improved programming languages and improved system services. These are not alternatives to such capabilities as OMG CORBA and Microsoft DCOM, but rather advanced features implemented at a higher level of abstraction and useful across multiple heterogeneous distributed computing environments.

Because each application executes as a set of geographically distributed parts, a distributed active-object architecture is required. It is likely that the objects taking part in the application were developed in various languages and execute on various hardware platforms. A simple, powerful paradigm is needed for communications among these heterogeneous objects. Due to the distributed nature of the application, an object might not always be available when it is needed. For example, an object executing on a PDA might be out of physical communications with the rest of the application.

Because the identities of the resources are not known when the application is developed, there must be an infrastructure to enable the discovery of pertinent objects. Once an object has been discovered, the infrastructure must facilitate the establishment of constructive communication between the new object and existing objects in the application.

Fundamentally, most business software modules are intended to be models of some real object within the business. A problem is that these modules are passive, unlike most of the objects they represent.

5.1.2 Multiagent-Based Development

Therefore, it is appropriate to consider a completely different approach to software systems. We propose one based on the (intentionally provocative) recognition that

- errors will always be a part of complex systems
- error-free code can at times be a disadvantage
- where systems interact with the complexities of the physical world, there is a concomitant power that can be exploited.

We suggest an open architecture consisting of multiple, redundant, agent-based modules interacting via a verified kernel. The appropriate analogy is that of a large, robust, natural system.

Multiagent-based interoperation is a new paradigm distinguished by the features that requests are specified in terms of "what" and not "how;" agents can take an active role, monitoring conditions in their environment and reacting accordingly; and agents may be seen as holding beliefs about the world.

5.2 Interaction-Oriented Programming

Although multiagent systems are inherently superior to single agent systems, they are difficult to construct. The designer of a multiagent system must handle not only the application-specific aspects of the various agents, but also their interactions with one another. However, constructing multiagent systems manually can lead to unnecessarily rigid or suboptimal designs, wasted development effort, and violations to the autonomy of the agents.

We propose *interaction-oriented programming (IOP)* as a class of languages, techniques, and tools to develop multiagent systems. Briefly, IOP focuses on what is between, rather than within, agents. Interactions may conveniently be classified into three layers (lower to upper):

- *coordination*, which enables the agents to operate in a shared environment [Hewitt, 1977]
- *commitment*, which adds coherence to the agents' actions [Castelfranchi, 1995; Singh, 1997a]
- *collaboration*, which includes knowledge-level protocols on commitments and communications [Grosz & Kraus, 1996; Singh, 1994].

Informal interaction concepts, such as competition, may be classified into different layers: auctions require only coordination, whereas commerce involves commitments, and negotiation involves sophisticated protocols.

The key tenets of IOP are as follows (we describe and defend these in some of our other work):

1. Correctness or data integrity may be difficult to characterize, but *coherence* is still crucial.
2. Complex interactions greatly exacerbate the difficulties in developing robust multiagent systems.
3. *Customizable* approaches can yield productivity gains that far outweigh any performance penalties.

Interaction has been studied, albeit fragmentarily, in distributed computing (DC) [Agha *et al.*, 1993; Milner, 1993], databases (DB) [Gray & Reuter, 1993], and DAI. The DB and DC work focuses on narrower problems, and eschews high-level concepts. Thus it is less flexible, but more robust, than the DAI work. The challenge is in achieving both rigor and flexibility in the abstractions for building multiagent systems.

5.2.1 Commitments and Societies

Social commitment is a key abstraction for supporting coherent interactions, while preserving autonomy. In order to formalize commitments, we observe that

1. Agents can be structured, and are recursively composed of heterogeneous individuals or groups of agents [Singh, 1991]
2. Agents are autonomous, but constrained by commitments—or we would have chaos!
3. Social commitments cannot be reduced to internal commitments, which apply within an agent
4. Commitments are, in general, revocable; the clauses for revoking them are no less important than the conditions for satisfying them!
5. Commitments arise, exist, are satisfied or revoked, all in a social context; commitments not only rely on the social structure of the groups in which they exist, but also help create that structure.

The above motivate a view of commitments as relating a *debtor*, a *creditor*, and a *context*. A number of natural operations can be defined on commitments, along with social policies that agents acquire when they (autonomously in some cases) adopt a role.

A group of agents can form a small society in which they play different roles. The group defines the roles, and the roles define the commitments associated with them. When an agent joins a group, he joins in one or more roles, and acquires the commitments of that role. Agents join a group autonomously, but are then constrained by the commitments for the roles they adopt. The groups define the *social context* in which the agents interact.

5.2.2 Protocols

Different protocols are suited for different applications and usage scenarios. For example, there would be protocols for electronic commerce, travel applications, student registration, and so on. Different vendors can supply agents to play different roles in these protocols. Each vendor's agent must comply with the protocols in which it participates. The designers who agree on standard protocols can assume a lot more about each others' agents, but not in general. However, the implementation of the agent is known only to the vendor.

Protocols are defined and publicly known. By adopting them, agents are committed to playing specific roles, and guaranteeing properties such as sincerity, and producing appropriate responses. The commitments associated with each role are metacommitments, because the base-level commitments are created during execution from these metacommitments. For example, an agent may inform another agent of a value, but become committed to proactively notifying the other of any changes. This is a metacommitment; when the value changes, the metacommitment leads to a base-level commitment to actually send the specific notification.

The protocols can be specified using some formal notation, such as temporal logic, finite state machines, Petri Nets, or constraint languages. These have all been used in specifying protocols in distributed systems. For the more rigidly defined protocols, the first three approaches are practically equivalent; for more flexible protocols, a rule-based notation is likely to be the most effective.

The above framework presupposes a richer infrastructure for agent management that we term *society management*. This infrastructure has support for defining commitments, roles, and groups. It supports operations for agents to join a society in some roles, to change roles, and to exit the society. These operations must of course respect the definitions of the given groups.

The challenge is to develop formal semantics, methodologies, and tools to sustain the above vision.

5.3 Adaptive Agents

We restrict our attention to adaptivity applied in open information-rich environments. To build systems that work effectively within such environments requires balancing ease of construction and robustness with flexibility. This presents a number of technical difficulties. Foremost among these is the need to handle the unpredictability in the environment as new components appear, and old ones disappear or change. Since the information components in the environment cannot easily be altered, the agents that represent them must be able to learn and adapt. Table 15 summarizes the associated challenges for machine learning.

5.3.1 Learning about Passive Components

Agents need to learn about the databases and knowledge bases within their environment, and be able to form relationships among what they learn.

Relationship Formation. The major problem with ontology-based approaches is the effort required to build an ontology, and to relate different resources and applications to it. As agents extend their world model, they need to be able to acquire and integrate ontologies autonomously. They also should learn the ontologies of other agents. In other cases, tools that assist a human designer are needed. These tools must have a strong machine learning component, to be able to not only relate concepts across databases, but also help identify relationships within an ontology. Such relationships, e.g., generalization or containment, are necessary for CIS query processing approaches, e.g., [Arens *et al.*, 1996; Huhns *et al.*, 1994]. For example, *port* is a generalization of *airport* and can be used to answer queries about airports only if additional restrictions are added.

Further, different domains often have a rich variety of relationships that compose elegantly with each other [Huhns & Stephens, 1989]. To give a simple, albeit somewhat contrived, example, if a person owns a car, and the car contains a wheel, then the person also owns the wheel. These relationships form part of the common sense knowledge that is essential in relating information from different databases: the two tables *car-ownership* and *car-parts* in one database may correspond to a single table *auto-part-ownership* in another database.

Concept Acquisition. How may we identify the concepts in a remote knowledge base? This is potentially useful, but extremely difficult, when dealing with a previously unknown source. It remains useful, and becomes more tractable, when the structure of the database is known, but the structure does not faithfully reflect the meaning of the content. In other words, the concepts are hidden inside the data values. A challenge is to discover the rules for partitioning the mixed up concepts into the correct categories. Some progress is already being made, mostly under the rubric of *data mining*, e.g., [Fayyad, 1996]. An issue that has not drawn much attention is collaborative and context-dependent learning of the concepts. This can be important, because different uses of the data might treat the implicit concepts differently.

5.3.2 Learning about Active Components

The active components of an environment are workflows and agents, and their interactions. The challenge is to learn the potential coordination constraints of the workflows, to infer the activities or plans of other agents, and to learn from repeated interactions with them. Related problems arise when the information environment is truly open and new agents are added dynamically, or the agents involved do not repeat interactions. In such cases, an agent still needs to learn how to collaborate with classes of agents, and to categorize them appropriately. For example, an agent may infer that agents who request a price quote for valves will often also want a price quote on matching hoses.

Source of Uncertainty	Traditional	Agent-Based
Environment	Agent learns about a predictable, but not teleological environment	Agent learns about an unpredictable, teleological environment
Environment and Agent's Sensors/Inputs	Agent might have imprecise sensors that yield inaccurate data	Agent might be misled deliberately

Table 15: Challenges for Machine Learning in Agent Systems

The foregoing can be generalized still further to learning about the agents' dispositions to one another. For example, it is important to learn to what extent other agents will cooperate with the given agent. Indeed, if the agents form a team or coalition, they will be able to assist each other and prevent mishaps [Shehory & Kraus, 1996]. It is also useful to have models of the learning abilities of the other agents.

5.3.3 Themes

Many applications of agents have the unifying themes that (a) the effect of logical homogeneity and centralization must be attained despite physical distribution and heterogeneity, and (b) logical openness must be supported. Openness translates into a number of interesting systemic challenges, relating to how a system may initialize and stabilize when some agents meet, are added, or leave. These lead to the following challenges:

- learning about each other
- learning about society and the environment
- learning from repeat interactions with changing agent instances
- learning biased by social structure
- forgetting by a group about its former members.

The further development of agents that will learn will undoubtedly lead to significant new applications for agents.

References

[Agha *et al.*, 1993] Agha, Gul; Frølund, Svend; Kim, WooYoung; Panwar, Rajendra; Patterson, Anna; and Sturman, Daniel; 1993. Abstraction and modularity mechanisms for concurrent computing. *IEEE Parallel and Distributed Technology* 3–14.

[Allen *et al.*, 1990] Allen, James; Hendler, James; and Tate, Austin, editors. *Readings in Planning.* Morgan Kaufmann.

[Arens *et al.*, 1996] Arens, Yigal; Hsu, Chun-Nan; and Knoblock, Craig A.; 1996. Query processing in the SIMS information mediator. In *Proceedings of the ARPA/Rome Laboratory Knowledge-Based Planning and Scheduling Initiative Workshop.* Morgan Kaufmann.

[Austin, 1962] Austin, John L.; 1962. *How to Do Things with Words.* Clarendon, Oxford.

[Barwise & Perry, 1983] Barwise, Jon and Perry, John; 1983. *Situations and Attitudes.* MIT Press, Cambridge, MA.

[Bayardo *et al.*, 1997] Bayardo, R.; Bohrer, W.; Brice, R.; Chichocki, A.; Fowler, G.; Helal, A.; Kashyap, V.; Ksiezyk, T.; Martin, G.; Nodine, M.; Rashid, M.; Rusinkiewicz, M.; Shea, R.; Unnikrishnan, C.; Unruh, A.; and Woelk, D.; 1997. InfoSleuth: Semantic integration of information in open and dynamic environments. In *Proceedings of the ACM SIGMOD Conference.*

[Belnap & Perloff, 1988] Belnap, Nuel and Perloff, Michael; 1988. Seeing to it that: A canonical form for agentives. *Theoria* 54(3):175–199.

[Bonasso *et al.*, 1996] Bonasso, R. Peter; Kortenkamp, David; Miller, David P.; and Slack, Marc; 1996. Experiences with an architecture for intelligent, reactive agents. In *Intelligent Agents II: Agent Theories, Architectures, and Languages.* 187–202.

[Bond & Gasser, 1988] Bond, Alan and Gasser, Les, editors. *Readings in Artificial Intelligence.* Morgan Kaufmann, San Francisco.

[Brachman & Levesque, 1985] Brachman, Ronald and Levesque, Hector, editors. *Readings in Knowledge Representation.* Morgan Kaufmann.

[Brand, 1984] Brand, Myles; 1984. *Intending and Acting.* MIT Press, Cambridge, MA.

[Bratman, 1987] Bratman, Michael E.; 1987. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA.

[Breiter & Sadek, 1996] Breiter, P. and Sadek, M. D.; 1996. A rational agent as a kernel of a cooperative dialogue system: Implementing a logical theory of interaction. In *ECAI-96 Workshop on Agent Theories, Architectures, and Languages*, Heidelberg, Germany. Springer Verlag. 261–276.

[Bukhres & Elmagarmid, 1996] Bukhres, Omran A. and Elmagarmid, Ahmed K., editors. *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications.* Prentice Hall.

[Castelfranchi, 1995] Castelfranchi, Cristiano; 1995. Commitments: From individual intentions to groups and organizations. In *Proceedings of the International Conference on Multiagent Systems.* 41–48.

[Cohen & Levesque, 1990] Cohen, Philip R. and Levesque, Hector J.; 1990. Intention is choice with commitment. *Artificial Intelligence* 42:213–261.

[Conte & Castelfranchi, 1995] Conte, Rosaria and Castelfranchi, Cristiano; 1995. *Cognitive and Social Action.* UCL Press, London.

[Croft, 1993] Croft, W. Bruce; 1993. Knowledge-based and statistical approaches to text retrieval. *IEEE Expert* 8(2):8–12.

[Davidson, 1980] Davidson, Donald; 1980. *Essays on Actions and Events.* Oxford University Press, Oxford.

[Davis & Smith, 1983] Davis, Randall and Smith, Reid G.; 1983. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence* 20:63–109. Reprinted in [Bond & Gasser, 1988].

[de Kleer, 1979] de Kleer, Johan; 1979. An assumption-based truth maintenance system. *Artificial Intelligence* 28(2):127–162.

[Dennett, 1987] Dennett, Daniel C.; 1987. *The Intentional Stance.* MIT Press, Cambridge, MA.

[Doyle, 1979] Doyle, Jon; 1979. A truth maintenance system. *Artificial Intelligence* 12(3):231–272.

[Durfee *et al.*, 1987] Durfee, Edmund H.; Lesser, Victor R.; and Corkill, Daniel D.; 1987. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers* C-36(11):1275–1291.

[Durfee, 1988] Durfee, Edmund H.; 1988. *Coordination of Distributed Problem Solvers.* Kluwer.

[Elmasri & Navathe, 1994] Elmasri, Ramez and Navathe, Shamkant; 1994. *Fundamental of Database Systems.* Benjamin Cummings, Redwood City, CA, second edition.

[Fagin & Halpern, 1988] Fagin, Ronald and Halpern, Joseph Y.; 1988. Belief, awareness, and limited reasoning. *Artificial Intelligence* 34:39–76.

[Fayyad, 1996] Fayyad, Usama, editor. *Special Issue on Data Mining*, volume 39(11) of *Communications of the ACM.* ACM.

[Finin *et al.*, 1994] Finin, Tim; Fritzson, Richard; McKay, Don; and McEntire, Robin; 1994. KQML as an agent communication language. In *Proceedings of the International Conference on Information and Knowledge Management.* ACM Press.

[Fischer *et al.*, 1996] Fischer, Klaus; Müller, Jörg; and Pischel, Markus; 1996. A pragmatic BDI architecture. In *Intelligent Agents II: Agent Theories, Architectures, and Languages.* 203–218.

[Gärdenfors, 1988] Gärdenfors, Peter; 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* MIT Press, Cambridge, MA.

[Gasser & Huhns, 1989] Gasser, Les and Huhns, Michael N., editors. *Distributed Artificial Intelligence, Volume II.* Pitman/Morgan Kaufmann.

[Gasser, 1991] Gasser, Les; 1991. Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence* 47:107–138.

[Genesereth, 1991] Genesereth, Michael R.; 1991. Knowledge interchange format. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning.* 599–600.

[Georgeff & Rao, 1995] Georgeff, Michael P. and Rao, Anand S.; 1995. The semantics of intention maintenance for rational agents. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).* 704–710.

[Goldman, 1970] Goldman, Alvin I.; 1970. *A Theory of Human Action.* Prentice-Hall, Englewood Cliffs, NJ.

[Gray & Reuter, 1993] Gray, Jim and Reuter, Andreas; 1993. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann.

[Gray, 1991] Gray, Michael A.; 1991. On integrating expert system programs into cooperative systems. In *Proceedings of the IEEE/ACM International Conference on Developing and Managing Expert System Programs.* 90–97.

[Grice, 1969] Grice, Paul; 1969. Utterer's meaning and intentions. *Philosophical Review.* Reprinted in [Martinich, 1985].

[Grice, 1975] Grice, H. P.; 1975. Logic and conversation. In Cole, P. and Morgan, J. L., editors, *Syntax and Semantics, Volume 3.* Academic Press, New York. Reprinted in [Martinich, 1985].

[Grosz & Kraus, 1996] Grosz, Barbara J. and Kraus, Sarit; 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(2):269–357.

[Haddawy, 1996] Haddawy, Peter; 1996. Believing change and changing belief. *IEEE Transactions on Systems, Man, and Cybernetics Special Issue on Higher-Order Uncertainty* 26(5).

[Harman, 1986] Harman, Gilbert; 1986. *Change in View.* MIT Press, Cambridge, MA.

[Hewitt, 1977] Hewitt, Carl; 1977. Viewing control structures as patterns of passing messages. *Artificial Intelligence* 8(3):323–364.

[Horvitz & Rutledge, 1991] Horvitz, Eric and Rutledge, Geoffrey; 1991. Time-dependent utility and action under uncertainty. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence.* 151–158.

[Huhns & Bridgeland, 1991] Huhns, Michael N. and Bridgeland, David M.; 1991. Multiagent truth maintenance. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6):1437–1445.

[Huhns & Singh, 1997] Huhns, Michael N. and Singh, Munindar P.; 1997. Conversational agents. *IEEE Internet Computing* 1(2):73–75. Instance of the column *Agents on the Web.*

[Huhns & Stephens, 1989] Huhns, Michael N. and Stephens, Larry M.; 1989. Plausible inferencing using extended composition. In *Proceedings of the International Joint Conference on Artificial Intelligence.* 1420–1425.

[Huhns *et al.*, 1994] Huhns, Michael N.; Singh, Munindar P.; and Ksiezyk, Tomasz; 1994. Global information management via local autonomous agents. In *Proceedings of the ICOT International Symposium on Fifth Generation Computer Systems: Workshop on Heterogeneous Cooperative Knowledge Bases.*

[Huhns, 1987] Huhns, Michael N., editor. *Distributed Artificial Intelligence.* Pitman/Morgan Kaufmann.

[Iivari, 1996] Iivari, Juhani; 1996. Why are CASE tools not used? *Communications of the ACM* 39(10):94–103.

[Jennings, 1993] Jennings, N. R.; 1993. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review* 2(3):223–250.

[Knight & Luk, 1994] Knight, Kevin and Luk, Steve K.; 1994. Building a large knowledge base for machine translation. In *Proceedings of the National Conference on Artificial Intelligence.*

[Konolige & Pollack, 1989] Konolige, Kurt G. and Pollack, Martha E.; 1989. A representationalist theory of intentions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*

[Konolige, 1986] Konolige, Kurt; 1986. *A Deduction Model of Belief.* Morgan Kaufmann.

[Labrou & Finin, 1997] Labrou, Yannis and Finin, Tim; 1997. Semantics and conversations for an agent communication language. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

[Lenat & Guha, 1989] Lenat, Douglas B. and Guha, R. V.; 1989. *Building Large Knowledge Base Systems.* Addison Wesley, Reading, MA.

[Lespérance *et al.*, 1996] Lespérance, Yves; Levesque, Hector J.; Lin, Fangzhen; Marcu, Daniel; Reiter, Raymond; and Scherl, Richard B.; 1996. Foundations of a logical approach to agent programming. In *Intelligent Agents II: Agent Theories, Architectures, and Languages.* 331–346.

[Lewis, 1996] Lewis, Ted; 1996. The next $10,000_2$ years: Part II. *IEEE Computer* 30(5):78–86.

[Lux & Steiner, 1995] Lux, Andreas and Steiner, Donald; 1995. Understanding cooperation: An agent's perspective. In *Proceedings of the International Conference on Multiagent Systems.* 261–268.

[Martinich, 1985] Martinich, Aloysius P., editor. *The Philosophy of Language.* Oxford University Press, New York.

[Mason & Johnson, 1989] Mason, Cindy L. and Johnson, Rowland R.; 1989. DATMS: A Framework for Distributed Assumption-Based Reasoning. In *[Gasser & Huhns, 1989].* 293–318.

[McCarthy, 1979] McCarthy, John; 1979. Ascribing mental qualities to machines. In Ringle, Martin, editor, *Philosophical Perspectives in Artificial Intelligence.* Harvester Press.

[McDermott, 1982] McDermott, Drew; 1982. A temporal logic for reasoning about processes and plans. *Cognitive Science* 6(2):101–155.

[Miller *et al.*, 1960] Miller, George A.; Galanter, Eugene; and Pribram, Karl; 1960. *Plans and the Structure of Behavior.* Henry Holt, New York.

[Miller, 1995] Miller, George A.; 1995. WordNet: A lexical database for English. *Communications of the ACM* 38(11):39–41.

[Milner, 1993] Milner, Robin; 1993. Elements of interaction. *Communications of the ACM* 36(1):78–89. Turing Award Lecture.

[Moore, 1984] Moore, Robert C.; 1984. A formal theory of knowledge and action. In Hobbs, Jerry R. and Moore, Robert C., editors, *Formal Theories of the Commonsense World.* Ablex Publishing Company, Norwood, NJ. 319–358.

[Morris, 1938] Morris, Charles, editor. *Foundations of the Theory of Signs.* University of Chicago Press, Chicago and London.

[Neches *et al.*, 1991] Neches, Robert; Fikes, Richard; Finin, Tim; Gruber, Tom; Patil, Ramesh; Senator, Ted; and Swartout, William R.; 1991. Enabling technology for knowledge sharing. *AI Magazine* 12(3):36–56.

[Newell, 1982] Newell, Allen; 1982. The knowledge level. *Artificial Intelligence* 18(1):87–127.

[Norman & Long, 1996] Norman, Timothy J. and Long, Derek; 1996. Alarms: An implementation of motivated agency. In *Intelligent Agents II: Agent Theories, Architectures, and Languages.* 219–234.

[Papazoglou *et al.*, 1992] Papazoglou, Mike P.; Laufmann, Steven C.; and Sellis, Timothy K.; 1992. An organizational framework for cooperating intelligent information systems. *International Journal of Intelligent and Cooperative Information Systems* 1(1):169–202.

[Patil *et al.*, 1992] Patil, Ramesh S.; Fikes, Richard E.; Patel-Schneider, Peter F.; McKay, Don; Finin, Tim; Gruber, Thomas; and Neches, Robert; 1992. The DARPA knowledge sharing effort: Progress report. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning.*

[Pylyshyn, 1984] Pylyshyn, Zenon; 1984. *Computation and Cognition.* MIT Press, Cambridge, MA.

[Rao & Georgeff, 1991] Rao, Anand S. and Georgeff, Michael P.; 1991. Modeling rational agents within a BDI-architecture. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning.* 473–484.

[Rosenschein & Zlotkin, 1994] Rosenschein, Jeffrey S. and Zlotkin, Gilad; 1994. Designing conventions for automated negotiation. *AI Magazine* 29–46.

[Russell & Norvig, 1995] Russell, Stuart J. and Norvig, Peter; 1995. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, NJ.

[Ryle, 1949] Ryle, Gilbert; 1949. *The Concept of Mind.* Oxford University Press, Oxford.

[Sadek, 1991] Sadek, M. D.; 1991. Dialogues acts are rational plans. In *Proceedings of the ESCA/ETRW Workshop on the Structure of Multimodal Dialogue*, Maratea, Italy. 1–29.

[Sayre & Gray, 1993] Sayre, Kirk and Gray, Michael A.; 1993. Backtalk: A generalized dynamic communication system for DAI. *Software—Practice and Experience* 23(9):1043–1058.

[Shehory & Kraus, 1996] Shehory, Onn and Kraus, Sarit; 1996. Formation of overlapping coalitions for precedence-ordered task execution among autonomous agents. In *Proceedings of the International Conference on Multiagent Systems.* 330–337.

[Shen, 1993] Shen, Wei-Min; 1993. *Autonomous Learning from the Environment.* Computer Science Press and W. H. Freeman Publishers, New York.

[Simon, 1996] Simon, Herbert; 1996. *The Sciences of the Artificial.* MIT Press, Cambridge, MA, third edition.

[Singh & Asher, 1993] Singh, Munindar P. and Asher, Nicholas M.; 1993. A logic of intentions and beliefs. *Journal of Philosophical Logic* 22:513–544.

[Singh & Huhns, 1994] Singh, Munindar P. and Huhns, Michael N.; 1994. Automating workflows for service provisioning: Integrating AI and database technologies. *IEEE Expert* 9(5).

[Singh *et al.*, 1997]
Singh, Mona; Barnett, Jim; and Singh, Munindar; 1997. Enhancing conversational moves for portable dialogue systems. www.csc.ncsu.edu/ faculty/ ms/ interfaces/ c-moves.ps.

[Singh, 1991] Singh, Munindar P.; 1991. Group ability and structure. In Demazeau, Y. and Müller, J.-P., editors, *Decentralized Artificial Intelligence, Volume 2.* Elsevier/North-Holland, Amsterdam. 127–145.

[Singh, 1992] Singh, Munindar P.; 1992. A critical examination of the Cohen-Levesque theory of intentions. In *Proceedings of the 10th European Conference on Artificial Intelligence.*

[Singh, 1994] Singh, Munindar P.; 1994. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications.* Springer Verlag, Heidelberg, Germany.

[Singh, 1997a] Singh, Munindar P.; 1997a. Commitments among autonomous agents in information-rich environments. In *Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW).*

[Singh, 1997b] Singh, Munindar P.; 1997b. Commitments in the architecture of a limited, rational agent. In *Proceedings of the Workshop on Theoretical and Practical Foundations of Intelligent Agents.* Springer Verlag. 72–87. Invited paper.

[Stalnaker, 1984] Stalnaker, Robert C.; 1984. *Inquiry.* MIT Press, Cambridge, MA.

[Sycara & Zeng, 1996] Sycara, Katia and Zeng, Dajun; 1996. Coordination of multiple intelligent software agents. *International Journal of Cooperative Information Systems* 5:181–212.

[Wellman, 1995] Wellman, Michael P.; 1995. A computational market model for distributed configuration design. *AI EDAM* 9:125–133.

[Wiederhold, 1992] Wiederhold, Gio; 1992. Mediators in the architecture of future information systems. *IEEE Computer* 25(3):38–49.

[Winograd, 1975] Winograd, Terry; 1975. Frame representations and the declarative/procedural controversy. In Bobrow, D. G. and Collins, A. M., editors, *Representation and Understanding.* Academic Press, New York. Reprinted as [Brachman & Levesque, 1985, chapter 20, pages 358–370].

[Wobcke, 1995] Wobcke, Wayne; 1995. Plans and the revision of intentions. In *Proceedings of the Australian Workshop on Distributed Artificial Intelligence, LNAI 1087*, Heidelberg, Germany. Springer Verlag. 100–114.

[Woelk *et al.*, 1995] Woelk, Darrell; Huhns, Michael; and Tomlinson, Christine; 1995. Uncovering the next generation of active objects. *Object Magazine* 33–40.

[Woelk *et al.*, 1996] Woelk, Darrell; Cannata, Philip; Huhns, Michael; Jacobs, Nigel; Ksiezyk, Tomasz; Lavender, Greg; Meredith, Greg; Ong, Kayliang; Shen, Wei-Min; Singh, Munindar; and Tomlinson, Christine; 1996. Carnot prototype. In *[Bukhres & Elmagarmid, 1996].* Chapter 18.

[Wong, 1993] Wong, Stephen T.C.; 1993. COSMO: A communication scheme for cooperative knowledge-based systems. *IEEE Transactions on Systems, Man, and Cybernetics* 23(3):809–824.

[Yourdon, 1995] Yourdon, Edward; 1995. Java, the web, and software development. *IEEE Computer* 29(8):25–30.

[Zhang & Bell, 1991] Zhang, C. and Bell, D. A.; 1991. HECODES: A framework for heterogeneous cooperative distributed expert systems. *Data & Knowledge Engineering* 6:251–273.

Contents