2 Multiagent Systems and Societies of Agents

Michael N. Huhns and Larry M. Stephens

2.1 Introduction

Agents operate and exist in some environment, which typically is both computational and physical. The environment might be open or closed, and it might or might not contain other agents. Although there are situations where an agent can operate usefully by itself, the increasing interconnection and networking of computers is making such situations rare, and in the usual state of affairs the agent interacts with other agents. Whereas the previous chapter defined the structure and characteristics of an individual agent, the focus of this chapter is on systems with multiple agents. At times, the number of agents may be too numerous to deal with them individually, and it is then more convenient to deal with them collectively, as a society of agents.

In this chapter, we will learn how to analyze, describe, and design environments in which agents can operate effectively and interact with each other productively. The environments will provide a computational infrastructure for such interactions to take place. The infrastructure will include protocols for agents to communicate and protocols for agents to interact.

Communication protocols enable agents to exchange and understand messages. Interaction protocols enable agents to have conversations, which for our purposes are structured exchanges of messages. As a concrete example of these, a communication protocol might specify that the following types of messages can be exchanged between two agents:

- Propose a course of action
- Accept a course of action
- Reject a course of action
- Retract a course of action
- Disagree with a proposed course of action
- Counterpropose a course of action

Based on these message types, the following conversation—an instance of an interaction protocol for negotiation—can occur between Agent1 and Agent2:

- Agent1 proposes a course of action to Agent2
  Agent2 evaluates the proposal and
- sends acceptance to Agent1
  or
- sends counterproposal to Agent1
  or
- sends disagreement to Agent1
  or
- sends rejection to Agent1

This chapter describes several protocols for communication and interaction among both large and small groups of agents.

2.1.1 Motivations

But why should we be interested in distributed systems of agents? Indeed, centralized solutions are generally more efficient: anything that can be computed in a distributed system can be moved to a single computer and optimized to be at least as efficient. However, distributed computations are sometimes easier to understand and easier to develop, especially when the problem being solved is itself distributed. Distribution can lead to computational algorithms that might not have been discovered with a centralized approach. There are also times when a centralized approach is impossible, because the systems and data belong to independent organizations that want to keep their information private and secure for competitive reasons.

The information involved is necessarily distributed, and it resides in information systems that are large and complex in several senses: (1) they can be geographically distributed, (2) they can have many components, (3) they can have a huge content, both in the number of concepts and in the amount of data about each concept, and (4) they can have a broad scope, i.e., coverage of a major portion of a significant domain. Also, the components of the systems are typically distributed and heterogeneous. The topology of these systems is dynamic and their content is changing so rapidly that it is difficult for a user or an application program to obtain correct information, or for the enterprise to maintain consistent information.

There are four major techniques for dealing with the size and complexity of these enterprise information systems: modularity, distribution, abstraction, and intelligence, i.e., being smarter about how you seek and modify information. The use of intelligent, distributed modules combines all four of these techniques, yielding a distributed artificial intelligence (DAI) approach [25, 18].

In accord with this approach, computational agents need to be distributed and embedded throughout the enterprise. The agents could function as intelligent application programs, active information resources, "wrappers" that surround and buffer conventional components, and on-line network services. The agents would be knowledgeable about information resources that are local to them, and cooperate to provide global access to, and better management of, the information. For the practical reason that the systems are too large and dynamic (i.e., open) for global solutions to be formulated and implemented, the agents need to execute autonomously and be developed independently.

The rationale for interconnecting computational agents and expert systems is to enable them to cooperate in solving problems, to share expertise, to work in parallel on common problems, to be developed and implemented modularly, to be fault tolerant through redundancy, to represent multiple viewpoints and the knowledge of multiple experts, and to be reusable.

The possibility of an agent interacting with other agents in the future, in unanticipated ways, causes its developer to think about and construct it differently. For example, the developer might consider "What exactly does my agent know?" and "How can another agent access and use the knowledge my agent has?" This might lead to an agent's knowledge being represented declaratively, rather than being buried in procedural code.

Multiagent systems are the best way to characterize or design distributed computing systems. Information processing is ubiquitous. There are computer processors seemingly everywhere, embedded in all aspects of our environment. Your kitchen likely has many, in such places as the microwave oven, toaster, and coffee maker, and this number does not consider the electrical power system, which probably uses hundreds in getting electricity to the kitchen. The large number of processors and the myriad ways in which they interact makes distributed computing systems the dominant computational paradigm today.

When the processors in the kitchen are intelligent enough to be considered agents, then it becomes convenient to think of them in anthropomorphic terms. For example, "the toaster *knows* when the toast is done," and "the coffee pot *knows* when the coffee is ready." When these systems are interconnected so they can interact, then they should also *know* that the coffee and toast should be ready at approximately the same time. In these terms, your kitchen becomes more than just a collection of processors—a distributed computing system—it becomes a *multiagent system.*

Much of traditional AI has been concerned with how an agent can be constructed to function intelligently, with a single locus of internal reasoning and control implemented in a Von Neumann architecture. But intelligent systems do not function in isolation—they are at the very least a part of the environment in which they operate, and the environment typically contains other such intelligent systems. Thus, it makes sense to view such systems in societal terms.

2.1.2 Characteristics of Multiagent Environments

1. Multiagent environments provide an infrastructure specifying communication and interaction protocols.

2. Multiagent environments are typically open and have no centralized designer.
3. Multiagent environments contain agents that are autonomous and distributed, and may be self-interested or cooperative.

A multiagent execution environment includes a number of concerns, which are enumerated as possible characteristics in Table 2.1.

Table 2.1 Characteristics of Multiagent Environments

Property	**Range of values**
Design Autonomy	Platform/Interaction Protocol /Language/Internal Architecture
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?
Security Services	Timestamps/Authentication
Remittance Services	Billing/Currency
Operations Support	Archiving/Redundancy /Restoration/Accounting

Table 2.2 Environment-Agent 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 2.2 lists some key properties of an environment with respect to a specific agent that inhabits it. These generalize the presentation in [38].

2.2 Agent Communications

We first provide a basic definition for an agent, which we need in order to describe the languages and protocols needed by multiagent systems. Fundamentally, an agent is an active object with the ability to perceive, reason, and act. We assume that an agent has explicitly represented knowledge and a mechanism for operating on or drawing inferences from its knowledge. We also assume that an agent has the ability to communicate. This ability is part perception (the receiving of messages) and part action (the sending of messages). In a purely computer-based agent, these may be the agent's only perceptual and acting abilities.

2.2.1 Coordination

Agents communicate in order to achieve better the goals of themselves or of the society/system in which they exist. Note that the goals might or might not be known to the agents explicitly, depending on whether or not the agents are goal-based. Communication can enable the agents to coordinate their actions and behavior, resulting in systems that are more coherent.

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, while negotiation is coordination among competitive or simply self-interested 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.

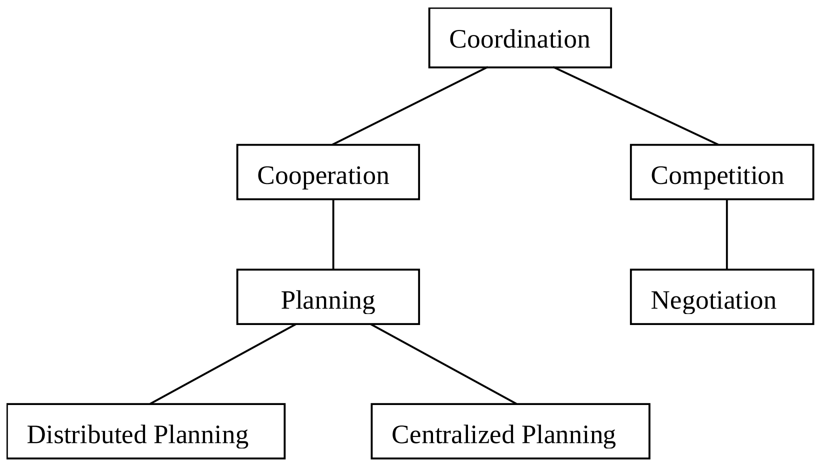

Figure 2.1 A taxonomy of some of the different ways in which agents can coordinate their behavior and activities

Coherence is how well a system behaves as a unit. A problem for a multiagent system is how it can maintain global coherence without explicit global control. In this case, the agents must be able on their own to determine goals they share with other agents, determine common tasks, avoid unnecessary conflicts, and pool knowledge and evidence. It is helpful if there is some form of organization among the agents. Also, social commitments can be a means to achieving coherence, which is addressed in Section 2.4.

Section 2.3.7 discusses another means, based on economic principles of markets. In this regard, Simon [40] 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. It is believed that coherence and optimality are intimately related.

2.2.2 Dimensions of Meaning

There are three aspects to the formal study of communication: syntax (how the symbols of comunication are structured), semantics (what the symbols denote), and pragmatics (how the symbols are interpreted). Meaning is a combination of semantics and pragmatics. Agents communicate in order to understand and be understood, so it is important to consider the different dimensions of meaning that are associated with communication [42].

Descriptive vs. Prescriptive. Some messages describe phenomena, while others prescribe behavior. Descriptions are important for human comprehension, but are difficult for agents to mimic. Appropriately, then, most agent communication languages are designed for the exchange of information about activities and behavior.

Personal vs. Conventional Meaning. An agent might have its own meaning for a message, but this might differ from the meaning conventionally accepted by the other agents with which the agent communicates. To the greatest extent possible, multiagent systems should opt for conventional meanings, especially since these systems are typically open environments in which new agents might be introduced at any time.

Subjective vs. Objective Meaning Similar to conventional meaning, where meaning is determined external to an agent, a message often has an explicit effect on the environment, which can be perceived objectively. The effect might be different than that understood internally, i.e., subjectively, by the sender or receiver of the message.

Speaker's vs. Hearer's vs. Society's Perspective Independent of the conventional or objective meaning of a message, the message can be expressed according to the viewpoint of the speaker or hearer or other observers.

Semantics vs. Pragmatics The pragmatics of a communication are concerned with how the communicators use the communication. This includes considerations of the mental states of the communicators and the environment in which they exist, considerations that are external to the syntax and semantics of the communication.

Contextuality Messages cannot be understood in isolation, but must be interpreted in terms of the mental states of the agents, the present state of the environment, and the environment's history: how it arrived at its present state. Interpretations are directly affected by previous messages and actions of the agents.

Coverage Smaller languages are more manageable, but they must be large enough so that an agent can convey the meanings it intends.

Identity When a communication occurs among agents, its meaning is dependent on the identities and roles of the agents involved, and on how the involved agents are specified. A message might be sent to a particular agent, or to just any agent satisfying a specified criterion.

Cardinality A message sent privately to one agent would be understood differently than the same message broadcast publicly.

2.2.3 Message Types

It is important for agents of different capabilities to be able to communicate. Communication must therefore be defined at several levels, with communication at the lowest level used for communication with the least capable agent. In order to be of interest to each other, the agents must be able to participate in a dialogue. Their role in this dialogue may be either active, passive, or both, allowing them to function as a master, slave, or peer, respectively. In keeping with the above definition for and assumptions about an agent, we assume that an agent can send and receive messages through a communication network. The messages can be of several types, as defined next.

There are two basic message types: assertions and queries. Every agent, whether active or passive, must have the ability to accept information. In its simplest form, this information is communicated to the agent from an external source by means of an assertion. In order to assume a passive role in a dialog, an agent must additionally be able to answer questions, i.e., it must be able to 1) accept a query from an external source and 2) send a reply to the source by making an assertion. Note that from the standpoint of the communication network, there is no distinction between an unsolicited assertion and an assertion made in reply to a query.

In order to assume an active role in a dialog, an agent must be able to issue queries and make assertions. With these capabilities, the agent then can potentially control another agent by causing it to respond to the query or to accept the information asserted. This means of control can be extended to the control of subagents, such as neural networks and databases.

An agent functioning as a peer with another agent can assume both active and passive roles in a dialog. It must be able to make and accept both assertions and queries. A summary of the capabilities needed by different classes of agents is shown in Table 2.3.

Table 2.3 Agent Capabilities

	Basic Agent	Passive Agent	Active Agent	Peer Agent
Receives assertions	•	•	•	•
Receives queries		•		•
Sends assertions		•	•	•
Sends queries			•	•

Table 2.4 Interagent Message Types

Communicative Action	Illocutionary Force	Expected Result
Assertion	Inform	Acceptance
Query	Question	Reply
Reply	Inform	Acceptance
Request	Request	
Explanation	Inform	Agreement
Command	Request	
Permission	Inform	Acceptance
Refusal	Inform	Acceptance
Offer/Bid	Inform	Acceptance
Acceptance		
Agreement		
Proposal	Inform	Offer/Bid
Confirmation		
Retraction		
Denial		

Other types of messages, derived from work on speech-act theory [43], are listed in Table 2.4.

2.2.4 Communication Levels

Communication protocols are typically specified at several levels. The lowest level of the protocol specifies the method of interconnection; the middle level specifies the format, or syntax, of the information being transfered; the top level specifies the meaning, or semantics, of the information. The semantics refers not only to the substance of the message, but also to the type of the message.

There are both binary and n-ary communication protocols. A binary protocol involves a single sender and a single receiver, whereas an n-ary protocol involves a single sender and multiple receivers (sometimes called broadcast or multicast). A protocol is specified by a data structure with the following five fields:

1. sender
2. receiver(s)

3. language in the protocol
4. encoding and decoding functions
5. actions to be taken by the receiver(s).

2.2.5 Speech Acts

Spoken human communication is used as the model for communication among computational agents. A popular basis for analyzing human communication is *speech act theory* [1, 39]. Speech act theory views human natural language as *actions*, such as requests, suggestions, commitments, and replies. For example, when you request something, you are not simply making a statement, but creating the request itself. When a jury declares a defendant guilty, there is an action taken: the defendant's social status is changed.

A speech act has three aspects:

1. Locution, the physical utterance by the speaker
2. Illocution, the intended meaning of the utterance by the speaker
3. Perlocution, the action that results from the locution.

For example, John might say to Mary, "Please close the window." This act consists of the physical sounds generated by John (or the character sequences typed by John), John's intent for the message as a request or a command, and if all goes well, the window being shut.

In communication among humans, the intent of the message is not always easily identified. For example, "I am cold," can be viewed as an assertion, a request for a sweater, or a demand for an increase in room temperature. However, for communication among agents, we want to insure that there is no doubt about the type of message.

Speech act theory uses the term *performative* to identify the illocutionary force of this special class of utterance. Example performative verbs include *promise, report, convince, insist, tell, request*, and *demand.* Illocutionary force can be broadly classified as assertives (statements of fact), directives (commands in a master-slave structure), commissives (commitments), declaratives (statements of fact), and expressives (expressions of emotion).

Performatives are usually represented in the stylized syntatic form "I hereby tell..." or "I hereby request..." Because performatives have the special property that "saying it makes it so," not all verbs are performatives. For example, stating that "I hereby solve this problem" does not create the solution. Although the term speech is used in this discussion, speech acts have to do with communication in forms other than the spoken word.

In summary, speech act theory helps define the type of message by using the concept of the illocutionary force, which constrains the semantics of the communication act itself. The sender's intended communication act is clearly defined, and the receiver has no doubt as to the type of message sent. This constraint simplifies the design of our software agents.

The message contained *within* the protocol may be ambiguous, may have no simple response, or may require decomposition and the assistance of other agents; however, the communication protocol itself should clearly identify the type of message being sent.

2.2.6 KQML

A fundamental decision for the interaction of agents is to separate the semantics of the communication protocol (which must be domain independent) from the semantics of the enclosed message (which may depend on the domain). The communication protocol must be universally shared by all agents. It should be concise and have only a limited number of primitive communication acts.

The knowledge query and manipulation language (KQML) is a protocol for exchanging information and knowledge, as illustrated in Figure 2.2. The elegance of KQML is that all information for understanding the content of the message is included in the communication itself. The basic protocol is defined by the following structure:

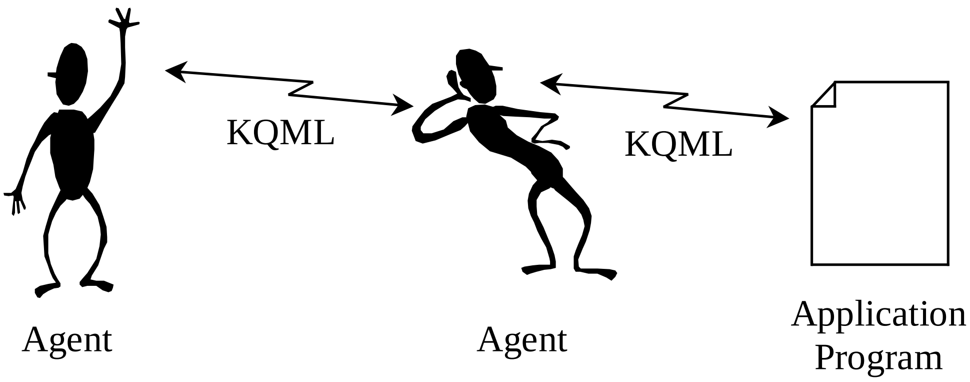

Figure 2.2 KQML is a protocol for communications among both agents and application programs

```
(KQML-performative
          :sender      <word>
          :receiver    <word>
          :language    <word>
          :ontology    <word>
          :content     <expression>
          ...)
```

The syntax is Lisp-like; however, the arguments—identified by keywords preceded by a colon—may be given in any order. The KQML-performatives are modeled on speech act performatives. Thus, the semantics of KQML performatives is domain independent, while the semanatics of the message is defined by the fields `:content` (the message itself), `:language` (the langauge in which the message is expressed), and `:ontology` (the vocabulary of the "words" in the message). In effect, KQML "wraps" a message in a *structure* that can be understood by any agent. (To understand the message itself, the recipient must understand the language and have access to the ontology.)

The terms `:content`, `:language`, and `:ontology` delineate the semantics of the message. Other arguments, including `:sender`, `:receiver`, `:reply-with`, and `:in-reply-to`, are parameters of the message passing. KQML assumes asynchronous communications; the fields `:reply-with` from a sender and `:in-reply-to` from a responding agent link an outgoing message with an expected response.

KQML is part of a broad research effort to develop a methodology for distributing information among different systems [35]. One part of the effort involves defining the Knowledge Interchange Format (KIF), a formal syntax for representing knowledge. Described in the next section, KIF is largely based on first-order predicate calculus. Another part of the effort is defining ontologies that define the common concepts, attributes, and relationships for different subsets of world knowledge. The definitions of the ontology terms give meaning to expressions represented in KIF. For example, in a Blocks-World ontology, if the concept of a wooden block of a given size is represented by the unary predicate Block, then the fact that block A is on top of block B could be communicated as follows:

```
(tell
        :sender      Agent1
        :receiver    Agent2
        :language:   KIF
        :ontology:   Blocks-World
        :content     (AND (Block A) (Block B) (On A B))
```

The language in a KQML message is not restricted to KIF; other languages such as PROLOG, LISP, SQL, or any other defined agent communication language can be used.

KQML-speaking agents appear to each other as clients and servers. Their communications can be either synchronous or asynchronous, as illustrated in Figure 2.3. For a synchronous communication, a sending agent waits for a reply. For an asynchronous communication, the sending agent continues with its reasoning or acting, which would then be interrupted when replies arrive at a later time.

Interestingly, KQML messages can be "nested" in that the content of a KQML message may be another KQML message, which is self contained. For example, if Agent1 cannot communicate directly with Agent2 (but can communicate with Agent3), Agent1 might ask Agent3 to forward a message to Agent2:

In a forwarded KQML message, the value of the :from field becomes the value

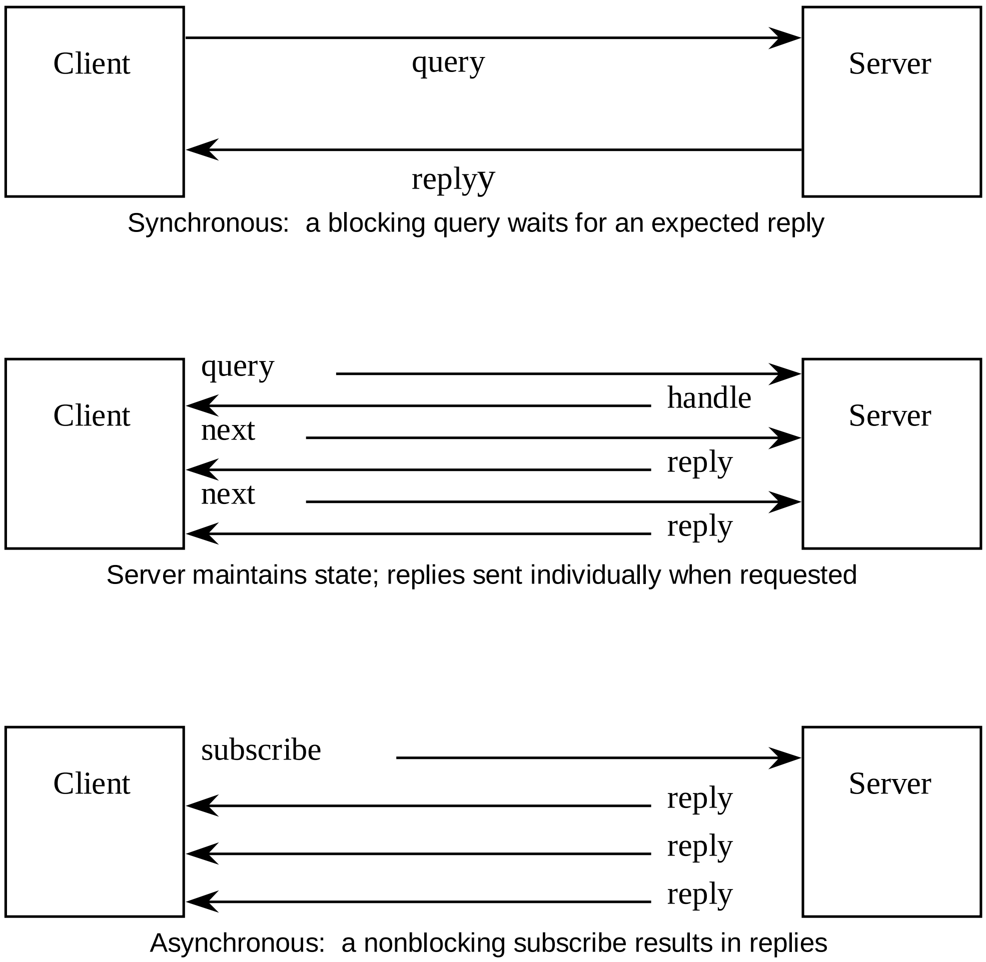

Figure 2.3 Synchronous and asynchronous communications among agents that understand KQML

```
(forward
        :from        Agent1
        :to          Agent2
        :sender      Agent1
        :receiver    Agent3
        :language    KQML
        :ontology    kqml-ontology
        :content     (tell
                         :sender      Agent1
                         :receiver    Agent2
                         :language    KIF
                         :ontology:   Blocks-World
                         :content     (On (Block A) (Block B))))
```

in the :sender field of the `:content` message, and the value of the `:to` field in the forward becomes the value of the `:receiver` field.

The KQML performatives may be organized into seven basic categories:

- Basic query performatives (evaluate, ask-one, ask-all, ...)
- Multiresponse query performatives (stream-in, stream-all, ...)
- Response performatives (reply, sorry, ...)
- Generic informational performatives (tell, achieve, cancel, untell, unachieve, ...)
- Generator performatives (standby, ready, next, rest, ...)
- Capability-definition performatives (advertise, subscribe, monitor, ...)
- Networking performatives (register, unregister, forward, broadcast, ...)

The advertise performative is used by a :sender agent to inform a `:receiver` about the `:sender`'s capabilities:

```
(advertise
        :sender      Agent2
        :receiver    Agent1
        :language    KQML
        :ontology    kqml-ontology
        :content     (ask-all
                         :sender        Agent1
                         :receiver      Agent2
                         :in-reply-to   id1
                         :language      Prolog
                         :ontology:     Blocks-World
                         :content       "on(X,Y)"))
```

Now Agent1 may query Agent2:

```
(ask-all
        :sender         Agent1
        :receiver       Agent2
        :in-reply-to    id1
        :reply-with     id2
        :language:      Prolog
        :ontology:      Blocks-World
        :content        "on(X,Y)"
```

Agent2 could respond with matching assertions from its knowledge base:

```
(tell
     :sender         Agent2
     :receiver       Agent1
     :in-reply-to    id2
     :language:      Prolog
     :ontology:      Blocks-World
     :content        "[on(a,b),on(c,d)]"
```

Issues:

The sender and receiver must understand the agent communication language being used; the ontology must be created and be accesssible to the agents who are communicating.

KQML must operate within a communication infrastructure that allows agents to locate each other. The infrastructure is not part of the KQML specification, and implemented systems use custom-made utility programs called *routers* or *facilators* to perform this function. In the advertise example above, if Agent2 sent the message to a facilator agent, then other agents could query the facilitator to find out about Agent2's capabilities.

KQML is still a work in progress and its semantics have not been completely defined. Labrou and Finin [31] have recently proposed a new KQML specification that refines the original draft [15]. However, there is yet no offical KQML specification that agent builders can rely on.

2.2.7 Knowledge Interchange Format

Agents need descriptions of real-world things. The descriptions could be expressed in natural languages, such as English and Japanese, which are capable of describing a wide variety of things and situations. However, the meaning of a natural language statement is often subject to different interpretations.

Symbolic logic is a general mathematical tool for describing things. Rather simple logics (e.g., the first order predicate calculus) have been found to be capable of describing almost anything of interest or utility to people and other intelligent agents. These things include simple concrete facts, definitions, abstractions, inference rules, constraints, and even metaknowledge (knowledge about knowledge).

KIF, a particular logic language, has been proposed as a standard to use to describe things within expert systems, databases, intelligent agents, etc. It is readable by both computer systems and people. Moreover, it was specifically designed to serve as an "interlingua," or mediator in the translation of other languages. For example, there is a translation program that can map a STEP/PDES expression about products into an equivalent KIF expression and vice versa. If there were a translation program for mapping between the healthcare language HL7 and KIF, then there would be a way to translate between STEP/PDES and HL7 (to exchange information about healthcare products) using KIF as an intermediate representation.

KIF is a prefix version of first order predicate calculus with extensions to support nonmonotonic reasoning and definitions. The language description includes both a specification for its syntax and one for its semantics. KIF provides for the expression of simple data. For example, the sentences shown below encode 3 tuples in a personnel database (arguments stand for employee ID number, department assignment, and salary, respectively):

```
(salary 015-46-3946 widgets 72000)
(salary 026-40-9152 grommets 36000)
(salary 415-32-4707 fidgets 42000)
```

More complicated information can be expressed through the use of complex terms. For example, the following sentence states that one chip is larger than another:

```
(> (* (width chip1) (length chip1))
   (* (width chip2) (length chip2)))
```

KIF includes a variety of logical operators to assist in the encoding of logical information, such as negation, disjunction, rules, and quantified formulas. The expression shown below is an example of a complex sentence in KIF. It asserts that the number obtained by raising any real-number ?x to an even power ?n is positive:

```
(=> (and (real-number ?x)
         (even-number ?n))
    (> (expt ?x ?n) 0))
```

KIF provides for the encoding of knowledge about knowledge, using the backquote (‘) and comma (,) operators and related vocabulary. For example, the following sentence asserts that agent Joe is interested in receiving triples in the salary relation. The use of commas signals that the variables should not be taken literally.

Without the commas, this sentence

```
(interested joe '(salary ,?x ,?y ,?z))
```

would say that agent joe is interested in the sentence `(salary ?x ?y ?z)` instead of its instances.

KIF can also be used to describe procedures, i.e., to write programs or scripts for agents to follow. Given the prefix syntax of KIF, such programs resemble Lisp or Scheme. The following is an example of a three-step procedure written in KIF. The first step ensures that there is a fresh line on the standard output stream; the second step prints "Hello!" to the standard output stream; the final step adds a carriage return to the output.

```
(progn (fresh-line t)
       (print "Hello!")
       (fresh-line t))
```

The semantics of the KIF core (KIF without rules and definitions) is similar to that of first-order logic. There is an extension to handle nonstandard operators (like backquote and comma), and there is a restriction that models must satisfy various axiom schemata (to give meaning to the basic vocabulary in the format). Despite these extensions and restrictions, the core language retains the fundamental characteristics of first-order logic, including compactness and the semi-decidability of logical entailment.

2.2.8 Ontologies

An ontology is a specification of the objects, concepts, and relationships in an area of interest. In the Blocks-World example above, the term Block represents a concept and the term On represents a relationship. Concepts can be represented in first-order logic as unary predicates; higher-arity predicates represent relationships. To express the idea that a block is a physical object, we might use the first-order expression

```
∀x (Block x) ⇒ (PhysicalObject x)
```

There are other, more general representations. Instead of `(Block A)`, the expression `(instanceOf A Block)` could be used. Both `A` and `Block` are now objects in the universe of discourse, and new relationships `instanceOf` and `subclassOf` are introduced:

```
(class Block)
(class PhysicalObject)
(subclassOf Block PhysicalObject)
∀x,y,z (instanceOf x y) ∧ (subclassOf y z) ⇒ (instanceOf x z)
```

The last sentence is a rule that expresses the notion of a type hierarchy.

An ontology is more than a taxonomy of classes (or types); the ontology must describe the relationships. The classes and relationships must be represented in the ontology; the instances of classes need not be represented. For example, there is no need to represent `A` in the ontology for either `(Block A)` or `(instanceOf A Block)`. An ontology is analogous to a database schema, not the contents of a database itself.

Implicit in this discussion is that an agent must represent its knowledge in the vocabulary of a specified ontology. Since agents are constructed by people, the effect is that the agent's creator must use a specified ontology to represent the agent's knowledge. All agents that share the same ontology for knowledge representation have an understanding of the "words" in the agent communication language.

Many agents have knowledge bases in which relationships are defined in more detail than just a character string. For example, the domain and range of a binary relationship can be specified;

```
(domain On PhysicalObject)
(range On PhysicalObject)
```

These restrictions limit the values allowed in *using* a relationship. `(On A B)` is permitted since both `A` and `B` are instances of `PhysicalObject` via transitive closure of `subclassOf`; `(On A Dream1)` would be prohibited assuming that `Dream1` is not of type `PhysicalObject`.

Ontology editors, such as those developed at Stanford [14] and the University of South Carolina [32], are typically frame-based knowledge-representation systems that allow users to define ontologies and their components: classes, instances, relationships, and functions. Figure 2.4 shows an example of such an ontology. Ontology editors offer a variety of features, such as the ability to translate ontologies into several representation languages or the ability for distributed groups to develop ontologies jointly over the Internet.

2.2.9 Other Communication Protocols

The above protocols for interagent communication in no way preclude *other* means by which computational agents can interact, communicate, and be interconnected. For example, one agent may be able to view a second agent with a camera, and use the resulting images to coordinate its own actions with those of the second agent.

Once communication protocols are defined and agreed upon by a set of agents, higher level protocols can be readily implemented. The next section describes some of these.

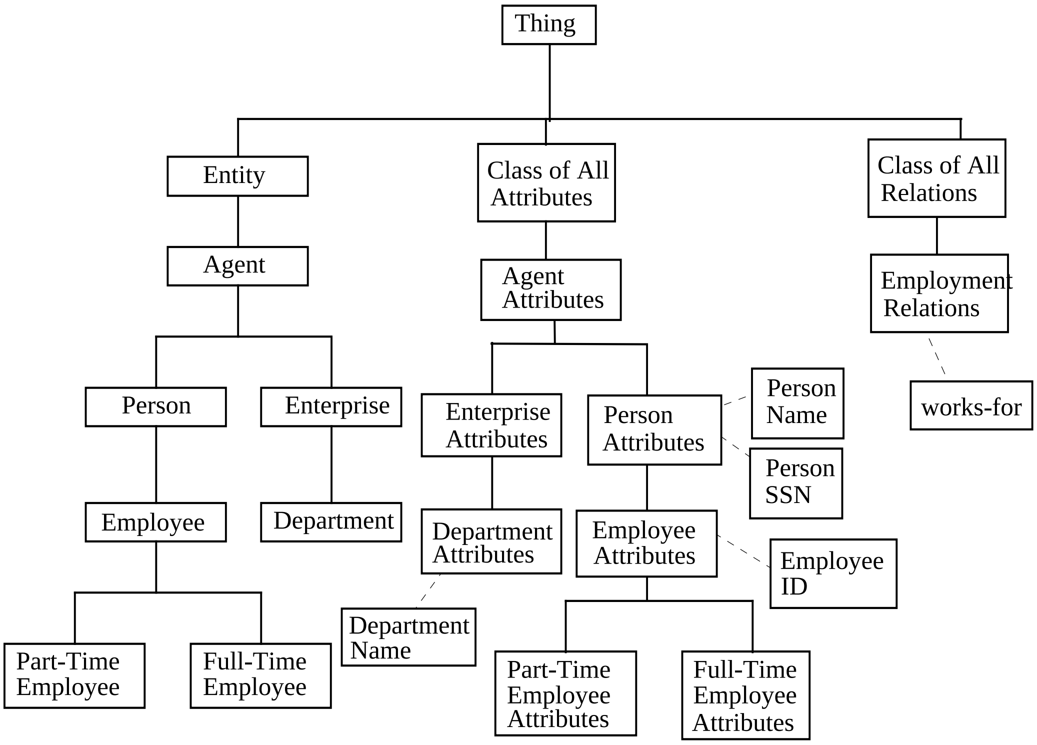

Figure 2.4 Example ontology for a simple business, showing classes and their subclasses, relationships, and instances (indicated by a dashed line)

2.3 Agent Interaction Protocols

The previous section describes mechanisms for agents to communicate *single* messages. Interaction protocols govern the exchange of a *series* of messages among agents—a conversation. 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 [37]. 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 [11]. For the latter cases, important aspects include how to

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

2.3.1 Coordination Protocols

In an environment with limited resources, agents must coordinate their activities with each other to further their own interests or satisfy group goals. The actions of multiple agents need to be coordinated because there are dependencies between agents' actions, there is a need to meet global constraints, and no one agent has sufficient competence, resources or information to achieve system goals. Examples of coordination include supplying timely information to other agents, ensuring the actions of agents are synchronized, and avoiding redundant problem solving.

To produce coordinated systems, most DAI research has concentrated on techniques for distributing both control and data. Distributed control means that agents have a degree of autonomy in generating new actions and in deciding which goals to pursue next. The disadvantage of distributing control and data is that knowledge of the system's overall state is dispersed throughout the system and each agent has only a partial and imprecise perspective. There is an increased degree of uncertainty about each agent's actions, so it is more difficult to attain coherent global behavior.

The actions of agents in solving goals can be expressed as search through a classical AND/OR goal graph. The goal graph includes a representation of the dependencies between the goals and the resources needed to solve the primitive goals (leaf nodes of the graph). Indirect dependencies can exist between goals through shared resources.

Formulating a multiagent system in this manner allows the activities requiring coordination to be clearly identified. Such activities include: (1) defining the goal graph, including identification and classification of dependencies; (2) assigning particular regions of the graph to appropriate agents; (3) controlling decisions about which areas of the graph to explore; (4) traversing the graph; and (5) ensuring that successful traversal is reported. Some of the activities may be collaborative, while some may be carried out by an agent acting in isolation. Determining the approach for each of the phases is a matter of system design.

While the distributed goal search formalism has been used frequently to characterize both global and local problems, the key agent structures are commitment and convention [29]. *Commitments* are viewed as pledges to undertake a specified course of action, while *conventions* provide a means of managing commitments in changing circumstances. Commitments provide a degree of predictability so that agents can take the future activities of others into consideration when dealing with interagent dependencies, global constraints, or resource utilization conflicts. As situations change, agents must evaluate whether existing commitments are still valid. Conventions constrain the conditions under which commitments should be reassessed and specify the associated actions that should then be undertaken: either retain, rectify or abandon the commitments.

If its circumstances do not change, an agent will endeavor to honor its commitments. This obligation constrains the agent's subsequent decisions about making new commitments, since it knows that sufficient resources must be reserved to honor its existing ones. For this reason, an agent's commitments should be both internally consistent and consistent with its beliefs.

Conventions help an agent manage its commitments, but they do not specify how the agent should behave towards others if it alters or modifies its commitments. However for goals that are dependent, it is essential that the relevant agents be informed of any substantial change that affects them. A convention of this type is a social one. If communication resources are limited, the following social convention might be appropriate:

```
LIMITED-BANDWIDTH SOCIAL CONVENTION

INVOKE WHEN
        Local commitment dropped
        Local commitment satisfied

ACTIONS
RULE1:  IF Local commitment satisfied
        THEN inform all related commitments

Rule2:  IF local commitments dropped because unattainable or
           motivation not present
        THEN inform all strongly related commitments

Rule3:  IF local commitments dropped because unattainable or
           motivation not present
        AND communication resources not overburdened
        THEN inform all weakly related commitments
```

When agents decide to pursue a joint action, they jointly commit themselves to a common goal, which they expect will bring about the desired state of affairs. The minimum information that a team of cooperating agents should share is (1) the status of their commitment to the shared objective, and (2) the status of their commitment to the given team framework. If an agent's beliefs about either of these issues change, then the semantics of joint commitments requires that all team members be informed. As many joint actions depend upon the participation of an entire team, a change of commitment by one participant can jeopardize the team's efforts. Hence, if an agent comes to believe that a team member is no longer jointly committed, it also needs to reassess its own position with respect to the joint action. These three basic assumptions are encoded in a convention that represents the minimum requirement for joint commitments, as shown below.

```
BASIC JOINT-ACTION CONVENTION

INVOKE WHEN
        Status of commitment to joint action changes
        Status of commitment to attaining joint action in present
```

```
            team context changes
         Status of joint commitment of a team member changes

ACTIONS
Rule1:   IF Status of commitment to joint action changes
           OR
         IF Status of commitment to present team
               context changes
         THEN inform all other team member of these changes

Rule2:   IF Status of joint commitment of a team member changes
         THEN Determine whether joint commitment still viable
```

Commitments and conventions are the cornerstones of coordination: commitments provide the necessary structure for predictable interactions, and social conventions provide the necessary degree of mutual support.

2.3.2 Cooperation Protocols

A basic strategy shared by many of the protocols for cooperation 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 [13]:

- 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.

Figure 2.5 illustrates two of the methods of task distribution. Details of additional methods are described in the sections that follow.

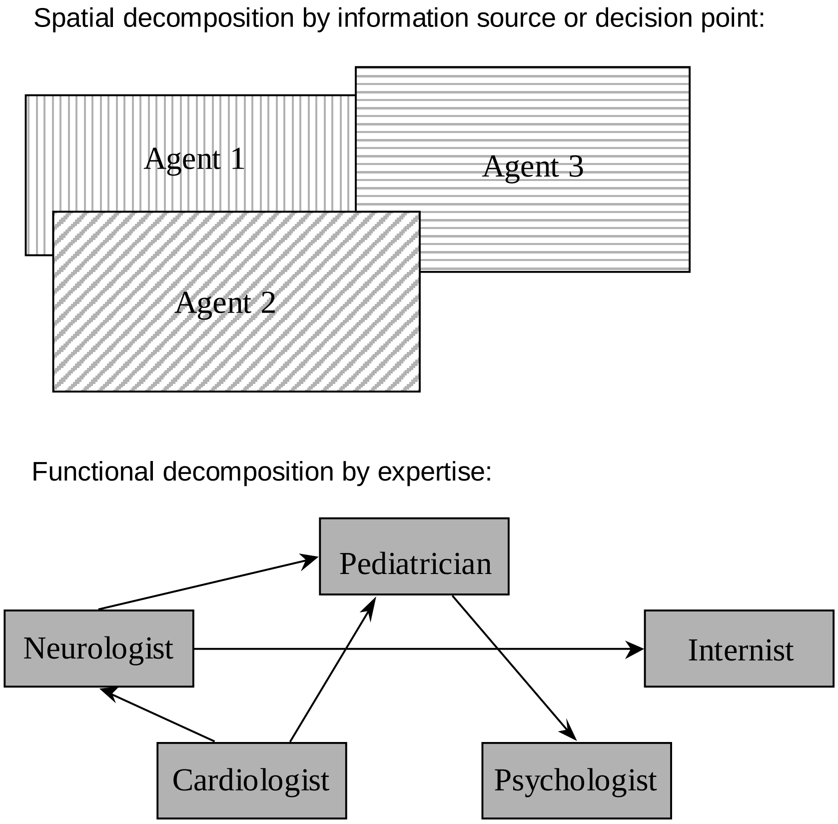

Figure 2.5 Two commonly used methods for distributing tasks among cooperative agents

2.3.3 Contract Net

Of the above mechanisms, the best known and most widely applied is the contract net protocol [44, 9]. The contract net protocol is an interaction protocol for cooperative problem solving among agents. It is modeled on the contracting mechanism used by businesses to govern the exchange of goods and services. The contract net provides a solution for the so-called *connection problem*: finding an appropriate agent to work on a given task. Figure 2.6 illustrates the basic steps in this protocol.

An agent wanting a task solved is called the *manager*; agents that might be able to solve the task are called potential *contractors*. From a manager's perspective, the process is

- Announce a task that needs to be performed
- Receive and evaluate bids from potential contractors
- Award a contract to a suitable contractor
- Receive and synthesize results.

From a contractor's perspective, the process is

- Receive task announcements
- Evaluate my capability to respond
- Respond (decline, bid)
- Perform the task if my bid is accepted
- Report my results.

The roles of agents are not specified in advance. Any agent can act as a manager by making task announcements; any agent can act as a contractor by responding to task announcements. This flexibility allows for further task decomposition: a contractor for a specific task may act as a manager by soliciting the help of other agents in solving parts of that task. The resulting manager-contractor links form a control hierarchy for task sharing and result synthesis.

The contract net offers the advantages of graceful performance degradation. If a contractor is unable to provide a satisfactory solution, the manager can seek other potential contractors for the task.

The structure of a task announcement includes slots for *addressee*, *eligibility specification*, *task abstraction*, *bid specification*, and *expiration time*. The tasks may be addressed to one or more potential contractors who must meet the criteria of the eligibility specification. The task abstraction, a brief description of the task, is used by contractors to rank tasks from several task announcements. The bid specification tells potential contractors what information must be provided with the bid; returned bid specifications give the manager a basis for comparing bids from different potential contractors. The expiration time is a deadline for receiving bids.

Each potential contractor evaluates unexpired task announcements to determine if it is eligible to offer a bid. The contractor then chooses the most attractive task (based on some criteria) and offers a bid to the corresponding manager.

A manager receives and evaluates bids for each task announcement. Any bid deemed satisfactory may be accepted before the expiration time of the task announcement. The manager notifies the contractor of bid acceptance with an *an-*

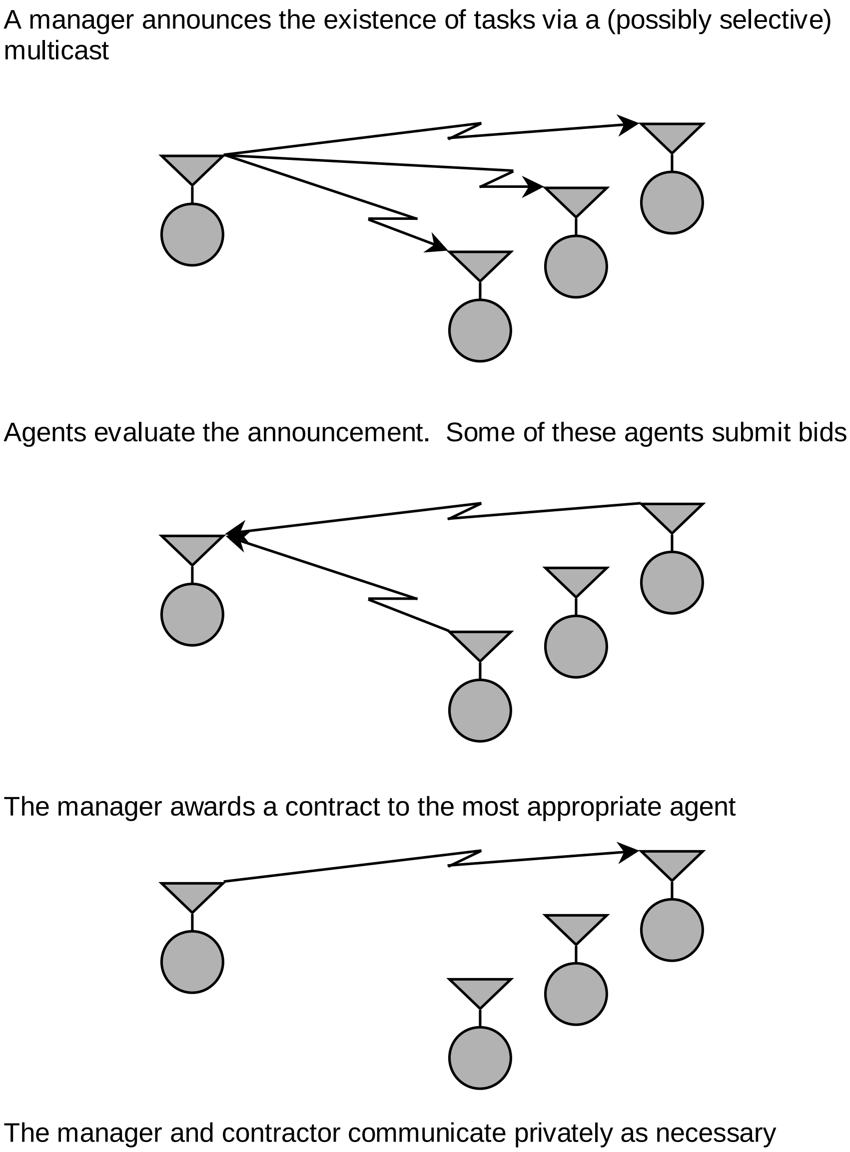

Figure 2.6 The basic steps in the contract net, an important generic protocol for interactions among cooperative agents

nounced award message. (A limitation of the contract net protocol is that a task might be awarded to a contractor with limited capability if a better qualified contractor is busy at award time. Another limitation is that a manager is under no obligation to inform potential contractors that an award has already been made.)

A manager may not receive bids for several reasons: (1) all potential contractors are busy with other tasks, (2) a potential contractor is idle but ranks the proposed task below other tasks under consideration, (3) no contractors, even if idle, are capable of working on the task. To handle these cases, a manager may request immediate response bids to which contractors respond with messages such as *eligible but busy*, *ineligible*, or *uninterested* (task ranked too low for contractor to bid). The manager can then make adjustments in its task plan. For example, the manager can wait until a busy potential contractor is free.

The contract net provides for *directed contracts* to be issued without negotiation. The selected contractor responds with an *acceptance* or *refusal*. This capability can simplify the protocol and improve effiency for certain tasks.

2.3.4 Blackboard Systems

Blackboard-based problem solving is often presented using the following metaphor:

"Imagine a group of human or agent specialists seated next to a large blackboard. The specialists are working cooperatively to solve a problem, using the blackboard as the workplace for developing the solution. Problem solving begins when the problem and initial data are written onto the blackboard. The specialists watch the blackboard, looking for an opportunity to apply their expertise to the developing solution. When a specialist finds sufficient information to make a contribution, he records the contribution on the blackboard. This additional information may enable other specialists to apply their expertise. This process of adding contributions to the blackboard continues until the problem has been solved."

This metaphor captures a number of the important characteristics of blackboard systems, each of which is described below.

Independence of expertise. The specialists (called knowledges sources or KSs) are not trained to work solely with that specific group of specialists. Each is an expert on some aspects of the problem and can contribute to the solution independently of the particular mix of other specialists in the room.

Diversity in problem-solving techniques. In blackboard systems, the internal representation and inferencing machinery used by each KS are hidden from direct view.

Flexible representation of blackboard information. The blackboard model does not place any prior restrictions on what information can be placed on the blackboard.

Common interaction language. KSs in blackboard systems must be able to correctly interpret the information recorded on the blackboard by other KSs. In practice, there is a tradeoff between the representational expressiveness of a specialized representation shared by only a few KSs and a fully general representation understood by all KSs.

Event-based activation. KSs in blackboard systems are triggered in response to blackboard and external events. Blackboard events include the addition of new information to the blackboard, a change in existing information, or the removal of existing information. Rather than having each KS scan the blackboard, each KS informs the blackboard system about the kind of events in which it is interested. The blackboard system records this information and directly considers the KS for activation whenever that kind of event occurs.

Need for control. A control component that is separate from the individual KSs is responsible for managing the course of problem solving. The control component can be viewed as a specialist in directing problem solving, by considering the overall benefit of the contributions that would be made by triggered KSs. When the currently executing KS activation completes, the control component selects the most appropriate pending KS activation for execution.
When a KS is triggered, the KS uses its expertise to evaluate the quality and importance of its contribution. Each triggered KS informs the control component of the quality and costs associated with its contribution, without actually performing the work to compute the contribution. The control component uses these estimates to decide how to proceed.

Incremental solution generation. KSs contribute to the solution as appropriate, sometimes refining, sometimes contradicting, and sometimes initiating a new line of reasoning.

Figure 2.7 shows the architecture of a basic blackboard system.

2.3.5 Negotiation

A frequent form of interaction that occurs among agents with different goals is termed negotiation. *Negotiation* is a process by which a joint decision is reached by two or more agents, each trying to reach an individual goal or objective. The agents first communicate their positions, which might conflict, and then try to move towards agreement by making concessions or searching for alternatives.

The major features of negotiation are (1) the language used by the participating agents, (2) the protocol followed by the agents as they negotiate, and (3) the decision process that each agent uses to determine its positions, concessions, and criteria for agreement.

Many groups have developed systems and techniques for negotiation. These can be either environment-centered or agent-centered. Developers of environment-centered techniques focus on the following problem: "How can the rules of the environment be designed so that the agents in it, regardless of their origin, capabilities, or intentions, will interact productively and fairly?" The resultant negotiation mechanism should ideally have the following attributes:

Efficiency: the agents should not waste resources in coming to an agreement.

Stability: no agent should have an incentive to deviate from agreed-upon strategies.

Simplicity: the negotiation mechanism should impose low computational and bandwidth demands on the agents.

Distribution: the mechanism should not require a central decision maker.

Symmetry: the mechanism should not be biased against any agent for arbitrary or inappropriate reasons.

An articulate and entertaining treatment of these concepts is found in [36]. In particular, three types of environments have been identified: worth-oriented domains, state-oriented domains, and task-oriented domains.

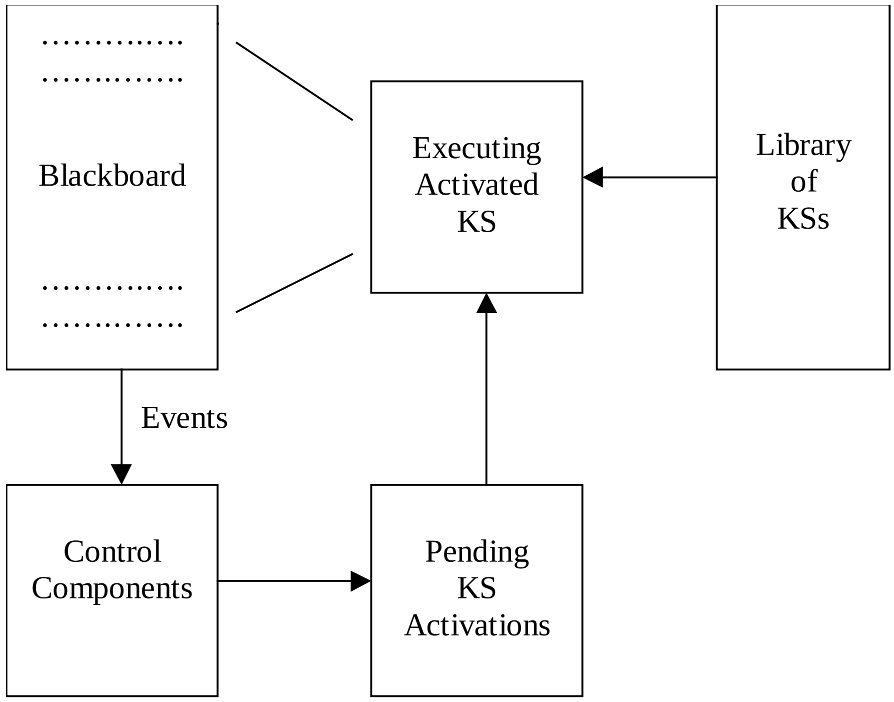

Figure 2.7 The architecture of a basic blackboard system, showing the blackboard, knowledge sources or agents, and control components

A task-oriented domain is one where agents have a set of tasks to achieve, all resources needed to achieve the tasks are available, and the agents can achieve the tasks without help or interference from each other. However, the agents can benefit by sharing some of the tasks. An example is the "Internet downloading domain," where each agent is given a list of documents that it must access over the Internet. There is a cost associated with downloading, which each agent would like to minimize. If a document is common to several agents, then they can save downloading cost by accessing the document once and then sharing it.

The environment might provide the following simple negotiation mechanism and constraints: (1) each agent declares the documents it wants, (2) documents found to be common to two or more agents are assigned to agents based on the toss of a coin, (3) agents pay for the documents they download, and (4) agents are granted access to the documents they download, as well as any in their common sets. This mechanism is simple, symmetric, distributed, and efficient (no document is downloaded twice). To determine stability, the agents' strategies must be considered.

An optimal strategy is for an agent to declare the true set of documents that it needs, regardless of what strategy the other agents adopt or the documents they need. Because there is no incentive for an agent to diverge from this strategy, it is stable.

Developers of agent-centered negotiation mechanisms focus on the following problem: "Given an environment in which my agent must operate, what is the best strategy for it to follow?" Most such negotiation strategies have been developed for specific problems, so few general principles of negotiation have emerged. However, there are two general approaches, each based on an assumption about the particular type of agents involved.

For the first approach, speech-act classifiers together with a possible world semantics are used to formalize negotiation protocols and their components. This clarifies the conditions of satisfaction for different kinds of messages. To provide a flavor of this approach, we show in the following example how the commitments that an agent might make as part of a negotiation are formalized [21]:

$$
\begin{aligned}
&\forall x (x \neq y) \wedge \\
&\neg(Precommit_a\ y\ x\ \phi) \wedge (Goal\ y\ Eventually(Achieves\ y\ \phi)) \wedge (Willing\ y\ \phi) \\
&\Longleftrightarrow (Intend\ y\ Eventually(Achieves\ y\ \phi))
\end{aligned}
$$

This rule states that an agent forms and maintains its commitment to achieve ϕ individually iff (1) it has not precommitted itself to another agent to adopt and achieve ϕ, (2) it has a goal to achieve ϕ individually, and (3) it is willing to achieve ϕ individually. The chapter on "Formal Methods in DAI" provides more information on such descriptions.

The second approach is based on an assumption that the agents are economically rational. Further, the set of agents must be small, they must have a common language and common problem abstraction, and they must reach a common solution. Under these assumptions, Rosenschein and Zlotkin [37] developed a unified negotiation protocol. Agents that follow this protocol create a *deal*, that is, a joint plan between the agents that would satisfy all 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. Each agent wants to maximize its own utility. The agents discuss a *negotiation set*, which is the set of all deals that have a positive utility for every agent.

In formal terms, a task-oriented domain under this approach becomes a tuple

$$< T, \; A, \; c >$$

where T is the set of tasks, A is the set of agents, and $c(X)$ is a monotonic function for the cost of executing the tasks X. A deal is a redistribution of tasks. The utility of deal d for agent k is

$$U_k(d) = c(T_k) - c(d_k)$$

The conflict deal D occurs when the agents cannot reach a deal. A deal d is individually rational if $d > D$. Deal d is pareto optimal if there is no deal $d' > d$. The set of all deals that are individually rational and pareto optimal is the negotiation set, NS. There are three possible situations:

1. conflict: the negotiation set is empty
2. compromise: agents prefer to be alone, but since they are not, they will agree to a negotiated deal
3. cooperative: all deals in the negotiation set are preferred by both agents over achieving their goals alone.

When there is a conflict, then the agents will not benefit by negotiating—they are better off acting alone. Alternatively, they can "flip a coin" to decide which agent gets to satisfy its goals. Negotiation is the best alternative in the other two cases.

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 constrained. Another aspect of the research problem 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 yet been fully worked out.

2.3.6 Multiagent Belief Maintenance

A multiagent truth-maintenance system can serve as a detailed example of a high-level interaction among agents. A truth-maintenance system (TMS) [10] is designed to ensure the integrity of an agent's knowledge, which should be stable, well-founded, and logically consistent. Depending on how beliefs, justifications, and data are represented, a *stable* state of a knowledge base is one in which 1) each datum that has a valid justification is believed, and 2) each datum that lacks a valid justification is disbelieved. A *well-founded* knowledge base permits no set of its beliefs to be mutually dependent. A *logically consistent* knowledge base is one that is stable at the time that consistency is determined and in which no logical contradiction exists. A consistent knowledge base is one in which no datum is both believed and disbelieved (or neither), or in which no datum and its negation are both believed. Other desirable properties for a knowledge base are that it be complete, concise, accurate, and efficient.

A single-agent TMS attempts to maintain well-founded stable states of a knowledge base by adjusting which data are believed and which are disbelieved. However, it is important for a group of agents to be able to assess and maintain the integrity of communicated information, as well as of their own knowledge. A multiagent TMS can provide this integrity [27].

We consider a modified *justification-based* TMS, in which every datum has a set of justifications and an associated status of `INTERNAL` (believed, because of a valid local justification), `EXTERNAL` (believed, because another agent asserted it), or `OUT` (disbelieved). Consider a network of many agents, each with a partially-independent system of beliefs. The agents interact by communicating data, either unsolicited or in response to a query. For well-foundedness, a communicated datum must be `INTERNAL` to at least one of the agents that believes it and either `INTERNAL` or `EXTERNAL` to the rest.

The support status of a communicated datum is jointly maintained by several agents. Hence, a single agent is generally not free to change the status on its own accord. It must coordinate with the other agents so that they are all consistent on the status of the datum.

The multiagent TMS is invoked by the addition or removal of a justification, and obeys the following principles:

- Belief changes should be resolved with as few agents as possible.
- Belief changes should be resolved by changing as few beliefs as possible.

When invoked, it does the following three things:

1. Unlabels some data, including the newly justified datum and, presumably, its consequences. This unlabeled data set might be confined to a single agent or it might span several agents. If a communicated datum is unlabeled in some agent, it must be unlabeled in all the agents that share it.
2. Chooses labelings for all the unlabeled shared data, as defined above.
3. Initiates labeling by each of the affected agents with respect to the requirements imposed by the shared data. If any of the affected agents fails to label successfully, it then backtracks. It either chooses different labelings for the shared data (step 2), or unlabels a different set of data (step 1).

Consider the justification network in Figure 2.8. There are two agents, Agent 1 and Agent 2, and they share the communicated datum T. Assume that the initial labeling shown in the diagram is perturbed by the addition of a new justification for Q. Agent 1 initially unlabels just the changed datum and private data downstream, P and Q, but there is no consistent relabeling. Hence, Agent 1 unlabels all shared data downstream of P and Q, and all private data downstream from there: P, Q, both Ts, and U. Again labeling fails. Since there is no further shared data downstream, Agent 1 and Agent 2 unlabel upstream and privately downstream

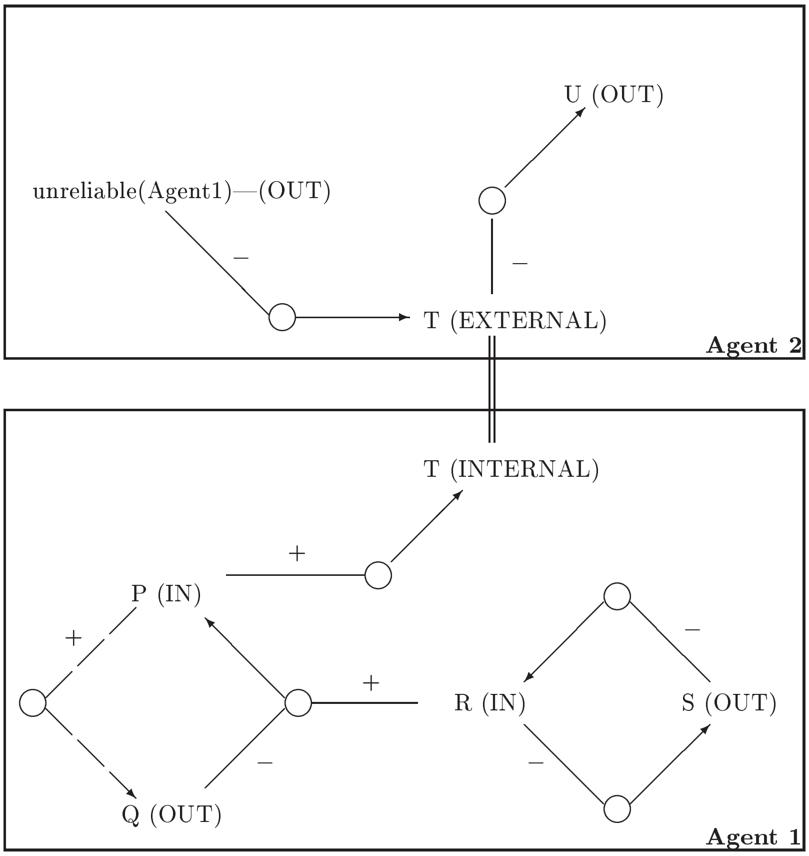

Figure 2.8 A multiagent TMS network before a new justification for datum Q (shown dashed) is added; this invokes the multiagent TMS algorithm and results in a relabeling of the network

from there: P, Q, Ts, U, R, and S. Now labeling succeeds, with S and U `IN` and everything else `OUT`, as shown in Figure 2.9. Had labeling failed, **unlabel** would not be able to unlabel more data, and would report that the network is inconsistent.

2.3.7 Market Mechanisms

Most of the protocols and mechanisms described earlier in this chapter require agents to communicate with each other directly, so are appropriate for small numbers of agents only. Other mechanisms for coordination are needed when there are a large or unknown number of agents. One mechanism is based on voting, where agents choose from a set of alternatives, and then adopt the alternative receiving the most votes. This mechanism is simple, equitable, and distributed, but it requires significant amounts of communication and organization, and is most useful when there are just a few well defined issues to be decided.

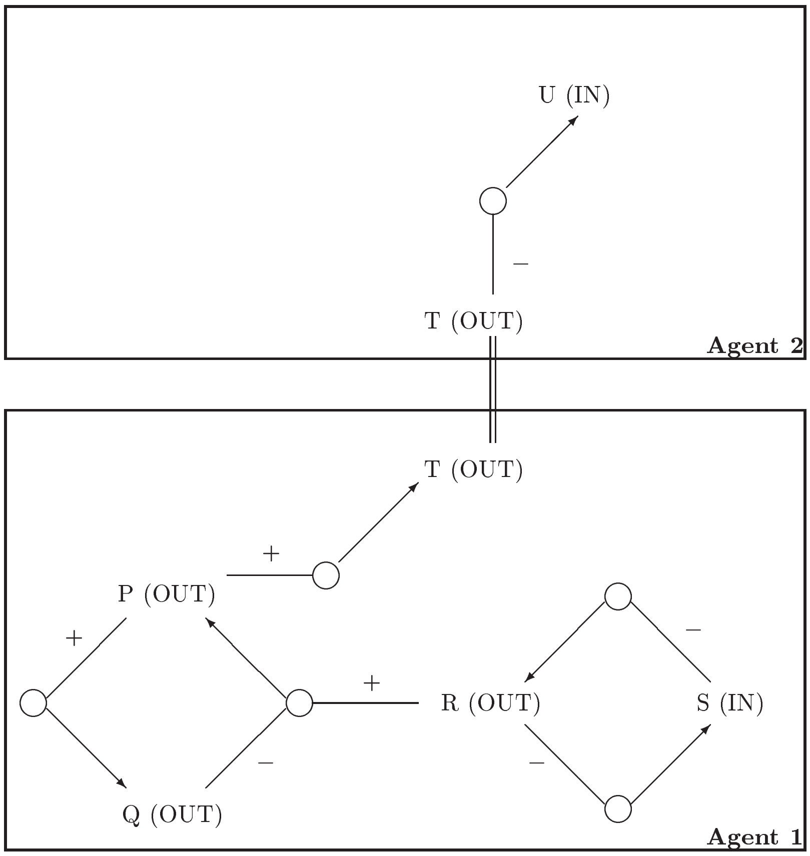

Figure 2.9 The resultant stable labeling of the justification network that is produced by the multiagent TMS algorithm

Computational economies, based on market mechanisms, are another approach [47]. 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 specific 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. There are two types of agents, *consumers*, who exchange goods, and *producers*, who transform some goods into other goods. Agents bid for goods at various prices, but all exchanges occur at current market prices. All agents bid so as to maximize either their profits or their utility.

To cast a problem in terms of a computational market, one needs to specify

- the goods being traded
- the consumer agents that are trading the goods
- the producer agents, with their technologies for transforming some goods into others
- the bidding and trading behaviors of the agents.

Since the markets for goods are interconnected, the price of one good will affect the supply and demand of others. The market will reach a competitive equilibrium such that (1) consumers bid to maximize their utility, subject to their budget constraints, (2) producers bid to maximize their profits, subject to their technological capability, and (3) net demand is zero for all goods.

The important property is that an equilibrium corresponds—in some 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.

In an open market, agents are free to choose their own strategy, and they do not have to behave rationally. Economic rationality assumes that the agent's preferences are given along with knowledge of the effects of the agent's actions. From these, the rational action for an agent is the one that maximizes its 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 [24, 22].

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. The next section describes social commitments more fully.

2.4 Societies of Agents

Much of traditional AI has been concerned with how an agent can be constructed to function intelligently, with a single locus of internal reasoning and control implemented in a Von Neumann architecture. But intelligent systems do not function in isolation—they are at the very least a part of the environment in which they operate, and the environment typically contains other such intelligent systems. Thus, it makes sense to view such systems in societal terms.

There are promising opportunities engendered by the combination of increasingly large information environments, such as the national information infrastructure and the intelligent vehicle highway system, and recent advances in multiagent systems. Planned information environments are too large, complex, dynamic, and open to be managed centrally or via predefined techniques—the only feasible alternative is for computational intelligence to be embedded at many and sundry places in such environments to provide distributed control. Each locus of embedded intelligence is best thought of as an autonomous agent that finds, conveys, or manages information. Because of the nature of the environments, the agents must be long-lived (they should be able to execute unattended for long periods of time), adaptive (they should be able to explore and learn about their environment, including each other), and social (they should interact and coordinate to achieve their own goals, and the goals of their society; they should rely on other agents to know things so they do not have to know everything).

Techniques for managing societies of autonomous computational agents are useful not only for large open information environments, but also for large open physical environments. For example, such techniques can yield new efficiencies in defense logistics: by considering each item of materiel to be an intelligent entity whose goal is to reach a destination, a distribution system could manage more complicated schedules and surmount unforeseen difficulties.

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.

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 payoffs.

Although mental primitives, such as beliefs, desires, and intentions, 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. [7] 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. Social commitments have been studied by a number of researchers, including [17, 28]. There are a number of definitions in the literature, which add components such as witnesses [5] or contexts [41]. Social commitments are a flexible means through which the behavior of autonomous agents is constrained. An important concept is that of social dependence, defined as

$$\begin{aligned}(\mathit{SocialDependence}\ x\ y\ a\ p) \equiv\ & (\mathit{Goal}\ x\ p) \wedge \\ & \neg(\mathit{CanDo}\ x\ a) \wedge \\ & (\mathit{CanDo}\ y\ a) \wedge \\ & ((\mathit{DoneBy}\ y\ a) \Longrightarrow \mathit{Eventually}\ p)\end{aligned}$$

that is, agent x depends on agent y with regard to act a for realizing state p, when p is a goal of x and x is unable to realize p while y is able to do so.

Social dependence can be voluntary when the agents adopt the roles that bind them to certain commitments. However, it is an objective relationship, in that it holds independently of the agents' awareness of it. Of course, there may be consequences that occur when the agents become aware of it, such as x might try to influence y to pursue p.

Social dependencies may be compound. For example, mutual dependence occurs when x and y depend on each other for realizing a common goal p, which can be achieved by a plan including at least two different actions, such that x depends on y doing a_y and y depends on x doing a_x, as

$$\exists p((\mathit{SocialDependence}\ x\ y\ a_y\ p) \wedge (\mathit{SocialDependence}\ y\ x\ a_x\ p))$$

Cooperation is a form of such mutual dependence.

Reciprocal dependence occurs when x and y depend on each other for realizing different goals, p_x for x and p_y for y, as

$$\exists p_x \exists p_y ((SocialDependence\ x\ y\ a_y\ p_x) \wedge (SocialDependence\ y\ x\ a_x\ p_y))$$

Social exchange is a form of such reciprocal dependence.

With this as a basis, a group of agents form a cooperative team when

- All agents share a common goal.
- Each agent is required to do its share to achieve the common goal by the group itself or a subgroup.
- Each agent adopts a request to do its share.

Beyond social dependencies, social laws may govern the behaviors of large numbers of agents in a society. See [34] for a treatment of this concept.

2.5 Conclusions

This chapter described elements of a computational environment that are needed for the interaction of multiple software agents. The elements enable agents to communicate, cooperate, and negotiate while they act in the interests of themselves or their society.

Further research is needed to develop the basis and techniques for societies of autonomous computational agents that execute in open environments for indefinite periods. This research will rely on the ability of agents to acquire and use representations of each other. This is what is needed for negotiation, cooperation, coordination, and multiagent learning. What should be the contents of these representations? Subsequent chapters of this textbook provide the answers.

2.6 Exercises

1. *[Level 1]* What are some of the advantages and disadvantages of synchronous versus asynchronous communications among agents?
2. *[Level 1]* Imagine that two agents are negotiating a contract. In the course of the negotiation, they engage in the following speech acts: *propose, counter-propose, accept, reject, retract, explain, ask-for-clarification, agree, disagree.* Draw a state diagram for the negotiation protocol followed by each agent.
3. *[Level 3]* Consider an environment having one broker agent with which many information agents can advertise. When an information agent advertises, it provides the broker with a list of predicate calculus expressions summarizing its knowledge. To find information agents who are knowledgeable about certain topics, a query agent supplies predicate calculus expressions to the broker and asks for pointers to the relevant information agents. The broker then returns

a list of all relevant information agents.

(a) List the KQML message that would be sent when query agent Q1 asks broker agent B1 for pointers to information agents knowledgeable about the predicate calculus expression `weight(Automobile ?x)`. Hint: the following is an example KQML message for an information agent advertising with a broker:

```
(advertise
:content weight(Automobile ?z)
:language Predicate-Calculus
:ontology Transportation-Domain
:sender info-agent-3
:receiver broker-1)
```

(b) The `Transportation-Domain` ontology is common to all agents. Draw a state transition diagram for each agent. Be sure that every speech act sent and received serves as a "condition" for a state transition. State any simplifying assumptions used.

4. *[Level 1]* What is the difference between the concepts coherence and coordination?

5. *[Level 1]* Give an advantage and disadvantage of the use of the contract net protocol.

6. *[Level 2]* Formalize the following protocol for the contract net in KQML. Clearly state which parts must be in the `:content` part of the communications. "One agent, the Manager, has a task that it wants to be solved. The Manager announces the task by broadcasting the task description in a task-announcement message to the other agents, the potential contractors. When contractors receives a task announcement, they evaluate it and some of them respond with a bid message, containing an estimate of their ability and a cost. The manager evaluates the bids, chooses the best one, and sends an award message to the winning contractor."

7. *[Level 2]* List the sequence of KQML performatives that must the generated by agents A, B, and C in solving the following problem: "Agent A wants to find out the cost of football tickets. Agent A does not know the cost, but Agent A knows that Agent B exists. Agent B does not know the cost either, but Agent B knows that Agent C exists. Agent C knows the cost." Assume that the agents are cooperative and truthful.

8. *[Level 2]* Describe how three agents might negotiate to find a common telephone line for a conference call. Assume that Agent A has telephones lines 1, 2, 3; Agent B, 1, 3; and Agent C, 2, 3.
The negotiation proceeds pair-wise: two agents at a time. The agents negotiate in order: A, B, C, A, B, C, A,... Also, alternate lines are chosen in the order specified above for each agent.
Initially,

Agent A proposes line 1 to Agent B, and Agent B accepts it.
Agent B proposes line 1 to Agent C, but Agent C rejects it.
Complete the process until all agents have picked a common line.

9. *[Level 3]* "Multiagent Truth Maintenance:" A single agent who knows P and P $\Rightarrow$ Q would have its knowledge labeled as follows:

fact1: P

status:	(IN)
shared with:	(NIL)
justification:	(PREMISE)

rule1: P $\Rightarrow$ Q

status:	(IN)
shared with:	(NIL)
justification:	(PREMISE)

fact2: Q

status:	(IN)
shared with:	(NIL)
justification:	(fact1, rule1)

If the agent shares fact1 with another agent, fact1's status changes to INTERNAL, and the agent receiving the knowledge labels its new fact as having status EXTERNAL.

Now consider the following situation in which the knowledge is initially local to each agent:

Agent A	Agent B	Agent C
fact1: P	rule1: P $\Rightarrow$ Q	fact1: R
rule1: S $\Rightarrow$ V	rule2: R $\Rightarrow$ Q	
	rule3: R $\Rightarrow$ S	
	rule4: Q $\Rightarrow$ W	

(a) Suppose that Agent A shares fact1 with Agent B, who uses forward chaining to make all possible conclusions from his knowledge. Show the effect of Agent A sharing fact1 on the `status`, `shared with`, and `justification` fields for all data in each agent.

(b) Now suppose Agent C shares fact1 with Agent B. Show the effect of sharing this knowledge on the `status`, `shared with`, and `justification` fields for all data in each agent.

(c) Now suppose that Agent A retracts fact1 by making fact1 have status OUT. Show the changes that would occur to the `status`, `shared with`, and `justification` fields for all data in each agent.

10. *[Level 1]* In the discussion of the unified negotiation protocol, it is stated that the agents might decide to "flip a coin" when the negotiation set is empty. Under what conditions might this be beneficial to the agents.

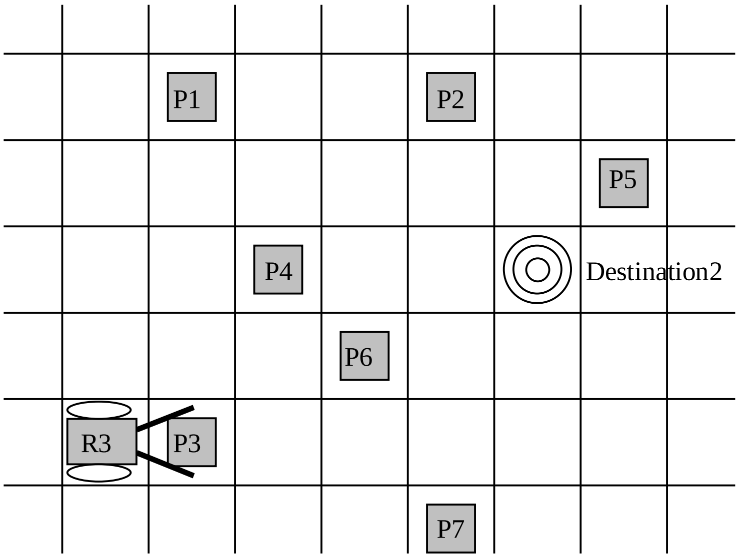

Figure 2.10 A domain where robots must move packages to their destinations

11. *[Level 4]* Imagine a two-dimensional domain consisting of packages and destinations. In this domain, robots must move the packages to the correct destinations. Robots can carry only one package at a time, and they are not allowed to travel through a package—they must maneuver around it. There is a cost associated with moving a package, but not with picking it up or setting it down. If a robot encounters a package when it is already carrying another, it can either move the package out of the way, or it can go around it. Moving it has a higher cost, but it might be beneficial to itself or other robots in the future to have the package out of the way. Assume that a robot is rewarded according to the amount that it moves a package closer to its destination. Develop a computer simulation of this domain, and try to establish answers to the following questions:

    (a) Will the robots develop any social conventions regarding which direction they move packages that are obstacles?

    (b) Under what conditions will "roadways" (paths without obstacles) form for the robots to travel on?

    (c) Destination points will likely become congested with robots attempting to drop off their packages. Gridlock might even occur. Will the robots become specialized in their operation, where some robots bring packages near the destinations and other robots move them from the drop-off points to the final destinations?

    (d) If the robots communicate information about their intentions regarding the packages they are moving, will other robots be able to take advantage of the information?

Suggestions: choose a grid of size NxN containing P packages, R robots, and D destinations, where initial values for the parameters are N=100, P=50, R=8, and D=3. Assume that a robot and a package each take up one square of the grid. Assume that a robot can move to any of its 8 adjoining squares, or stay where it is, in each time interval.

12. *[Level 1]* The initial state in a Block's World is On(B,C), On(D,A), Table(A), and Table(C). The desired goal state is On(A,B), On(B,C), Table(C), and Table(D). Agent1 can manipulate only blocks A and B; Agent2 can manipulate only blocks C and D. In solving this problem, the action MoveToTable(agent, block) can be used to place block D on the table. Express the movement of block D to the table in terms of the social dependence formula in this chapter.

2.7 References

1. John L. Austin. *How to do Things with Words.* Clarendon, Oxford, UK, 1962.
2. Will Briggs and Diane Cook. Flexible Social Laws. In *Proc. 14th IJCAI*, 1995.
3. Birgit Burmeister, Afsaneh Haddadi, and Kurt Sundermeyer. Generic Configurable Cooperation Protocols for Multi-Agent Systems. In *Proceedings of the 3rd European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, 1993.
4. Stefan Bussman and Jurgen Muller. A Communication Architecture for Cooperating Agents. *Computers and Artificial Intelligence*, Vol. 12, No. 1, pages 37–53, 1993.
5. Cristiano Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the International Conference on Multiagent Systems*, pages 41–48, 1995.
6. Man Kit Chang. *SANP: A Communication Level Protocol for Supporting Machine-to-Machine Negotiation in Organization.* MS Thesis, U. of British Columbia, Vancouver, B.C., Canada, 1991.
7. Rosaria Conte and Cristiano Castelfranchi. *Cognitive and Social Action.* UCL Press, London, 1995.
8. Daniel D. Corkill, Kevin Q. Gallagher, and Kelly E. Murray. GBB: A Generic Blackboard Development System. In *Proc. AAAI-86*, Philadelphia, PA, pages 1008–1014, 1986.
9. Randall Davis and Reid G. Smith. Negotiation as a Metaphor for Distributed Problem solving. *Artificial Intelligence*, Vol. 20, No. 1, pages 63–109, January 1983.
10. Jon Doyle. A Truth Maintenance System. *Artificial Intelligence*, Vol. 12, No. 3, pages 231–272, 1979.
11. Edmund H. Durfee. *Coordination of Distributed Problem Solvers.* Kluwer, 1988.
12. Edmund H. Durfee and Thomas A. Montgomery. A Hierarchical Protocol for Coordinating Multiagent Behaviors. In *Proc. AAAI-90*, 1990.
13. Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, C-36(11):1275–1291, 1987.

14. Adam Farquhar, Richard Fikes, and James Rice. The Ontolingua Server: A tool for Collaborative Ontology Construction. Technical Report KSL-96-26, Knowledge Systems Laboratory, Stanford University, September 1996.
15. Tim Finin, Don McKay, and Rich Fritzson. An Overview of KQML: A knowledge query and mnaipulation language. Technical Report, U. of Maryland CS Department, 1992.
16. S. Fiske and S. E. Taylor. *Social Cognition.* Addison Wesley, New York, 1984.
17. Les Gasser. Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence*, 47:107–138, 1991.
18. Les Gasser and Michael N. Huhns, editors. *Distributed Artificial Intelligence, Volume II.* Pitman Publishing, London, 1989.
19. N. Gilbert and J. E. Doran, editors. Simulating Societies: The Computer Simulation of Social Phenomena. In *Proceedings of the Symposium on Simulating Societies.* University College Press, London, 1994.
20. N.Gilbert and R.Conte, editors. *Artificial Societies: Computer Simulation of Social Life.* University College Press, London, 1995.
21. Afsaneh Haddadi. Towards a Pragmatic Theory of Interactions. In *Proc. International Conference on MultiAgent Systems (ICMAS)*, San Francisco, 1995.
22. Peter Haddawy. Believing change and changing belief. *IEEE Transactions on Systems, Man, and Cybernetics Special Issue on Higher-Order Uncertainty*, 26(5), 1996.
23. Carl Hewitt. Open Information Systems Semantics for Distributed Artificial Intelligence. *Artificial Intelligence*, Vol. 47, pages 79-106, 1991.
24. Eric Horvitz and Geoffrey Rutledge. Time-dependent utility and action under uncertainty. In *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence*, pages 151–158, 1991.
25. M. N. Huhns, U. Mukhopadhyay, L. M. Stephens, and R. D. Bonnell. DAI for Document Retrieval: The MINDS Project. In M. N. Huhns, editor, *Distributed Artificial Intelligence.* Pittman, London, 1987.
26. Michael N. Huhns and Munindar P. Singh. A Mediated Approach to Open, Large-Scale Information Management. In *Proc. IEEE Int. Phoenix Conf. on Computers and Communications*, 1995.
27. Michael N. Huhns and David M. Bridgeland. Multiagent Truth Maintenance. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 6, pages 1437–1445, December 1991.
28. N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 2(3):223–250, 1993.
29. Nick R. Jennings. Coordination Techniques for distributed Artificial Intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. John Wiley & Sons, Inc., New York, 1996.
30. R. Kakehi and M. Tokoro. A Negotiation Protocol for Conflict Resolution in Multi-Agent Environments. In *Proc. ICICIS*, pages 185–196, 1993.
31. Yannis Labrou and Tim Finin. A Semantics approach for KQML—a general purpose communication language for software agents. In *Proc. Int. Conf on Information and Knowledge Management*, 1994.
32. Kuha Mahalingam and Michael N. Huhns. An Ontology Tool for Distributed Information Environments. *IEEE Computer*, 30(6):80–83, June 1997.

33. Robin Milner. Elements of Interaction. *CACM*, Vol. 36, No. 1, pages 78–89, 1993.
34. Yoram Moses and Moshe Tenenholtz. On Computational Aspects of Artificial Social Systems. In *Proc. 11th DAI Workshop*, Glen Arbor, MI, 1992.
35. R. Neches, R. Fikes, T. Finin, R. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. AI Magazine, 12(3):36–56, Fall 1991.
36. Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter.* The MIT Press, Cambridge, MA, 1994.
37. Jeffrey S. Rosenschein and Gilad Zlotkin. Designing conventions for automated negotiation. *AI Magazine*, pages 29–46, 1994.
38. Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, NJ, 1995.
39. John R. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge U. Press, 1970.
40. Herbert Simon. *The Sciences of the Artificial.* MIT Press, Cambridge, MA, third edition, 1996.
41. Munindar P. Singh. 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)*, 1997.
42. Munindar P. Singh. "Considerations on Agent Communication," presented at FIPA Workshop, 1997.
43. Munindar P. Singh. A Semantics for Speech Acts. *Annals of Mathematics and AI*, Vol.8, No.I-II, pages 47–71, 1993.
44. Reid G. Smith. The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, Vol. C-29, No. 12, pages 1104–1113, December 1980.
45. Reid G. Smith and Randall Davis. Frameworks for Cooperation in Distributed Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 1, pages 61–70, January 1981.
46. Katia Sycara. Resolving Goal Conflicts via Negotiation. In *Proc. AAAI-88*, pages 245–250, 1988.
47. Michael P. Wellman. A computational market model for distributed configuration design. *AI EDAM*, 9:125–133, 1995.
48. Eric Werner. Cooperating Agents: A unified theory of communication and social structure. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence*, Volume II, pages 3–36. Pittman, London, 1989.
49. Gio Wiederhold. Mediators in the Architecture of Future Information Systems, *IEEE Computer*, Vol. 25, No. 3, pages 38–49, March 1992.
50. Carson Woo and Frederick H. Lochovsky. Knowledge Communication in Intelligent Information Systems. *International Journal of Intelligent and Cooperative Information Systems*, Vol. 1, No. 1, pages 203–228, 1992.