Chapter 16
Intelligent Agents

Abstract The intelligent agents are being viewed as new theoretical models of computation that more closely reflects current computing reality, aimed as new generation models for complex and distributed systems. An agent system can work as a single agent, or as a multiagent system. The intelligent agents have many applications—they are used in software engineering, in buying and selling—like online sales, bids, trading; the agents are also modeled for decision-making—with preferences and criteria for making decisions. This chapter also presents the classification of agents, agent system architecture, how the agents should coordinate among themselves, and the formation of a coalition between agents. The multiagents communicate with each other using agents' communication languages which are oriented towards performing actions. Other categories of agents are mobile agents—programs which can be moved to any far off place, and can communicate with the environment. The chapter ends with chapter summary, and the set of exercises.

Keywords Intelligent agent · Mobile agent · Multiagents · Agents' coordination · Cooperative agents · Agents' coalition · Software agents

16.1 Introduction

An *artificial agent* or *intelligent agent* is a recent term in computer science, and specifically in artificial intelligence. There is a number of definitions of agents. The agents are viewed as a new theoretical model of computation, that reflects current computing reality in a better (tangible) way than the existing model of Turing Machine. They are being projected as a next-generation model to engineer the complex and distributed systems.

Among many characterizations of agents, the following definition is most common: An agent is an encapsulated computer system, which is situated in some environment and it is flexible and capable of autonomous action in that environment in order to meet its desired goals. There are associated number of questions about this definition that require further explanation, which becomes somewhat clear through the extended definition of agents, in the following:

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_16

1. Agents are entities for problem-solving for clearly identifiable problems with well-defined boundaries and interfaces;
2. They receive inputs related to the state of their environment through sensors, when embedded in an environment, and act on the environment through effectors;
3. An agent is designed to fulfill a specific requirement, and it has a particular goal to be achieved;
4. They have control over both their own behavior and over the internal state, a property called *autonomous*;
5. The agents have flexible problem-solving behavior. They are both *reactive* (able to respond in time to the changes occurring in their environment) and *proactive* (able to act in anticipation of future goals).

Agents are also being used as a framework to bring together various AI's sub-areas to design and build intelligent systems. In spite of the intense interest of research community and progress made in the area of agents, a number of fundamental questions about the nature and the use of the agent-oriented approach remain unanswered, which are as follows:

- What are the fundamental concepts and notions of agent-based computing?
- What makes the agent-based approach a natural and powerful computational model?
- What are the implications of agent-based computing, in the wider perspectives, for AI and computer science in general?

Learning Outcomes of this Chapter

1. List the defining characteristics of an intelligent agent. [Familiarity]
2. Characterize and contrast the standard agent architectures. [Assessment]
3. Describe the applications of agent theory to domains such as software agents, personal assistants, and believable agents. [Familiarity]
4. Describe the primary paradigms used by learning agents. [Familiarity]
5. Demonstrate using appropriate examples how multiagent systems support agent interaction. [Usage]
6. Syntactic structure of agent languages. [Familiarity]

16.2 Classification of Agents

Although there is no universally accepted definition of an agent, however, as per the most commonly used definitions, an agent is a proactive software component that interacts with its environment, as well as it interacts with other agents on behalf of its user, and reacts to the changes in its environment. A component is called agent if it exhibits several of the properties given below.

Autonomous

It is the property of an agent, as per which it proactively initiates the activities as per its goal. An agent has its own thread of control, can act on behalf of its user, and without necessarily depending on the messages from other agents.

Mobile

An agent can move itself from one execution context to another. For this activity, it can move its code, and carry on executing from the current point onward, or it can start afresh. Other modes of execution can be, serialization of its code and state, such that it may continue its execution in a new context, and at the same time, it may retain the same old state and can continue to work.

Adaptable

An agent is adaptable to a new environment; its behavior can change after its deployment through its own learning, downloading new capabilities, and through user customization.

Knowledgeable

A software agent has reasoning capability, due to which it can reason about the acquired information, about the knowledge of other agents, its user, and about its goals.

Collaborative

Some agents are called collaborative agents, which can communicate with other agents and work in a cooperative manner. This collaboration can be formed either in a static manner or a dynamic manner. The collection of such agents is called multiagent system.

Persistent

The infrastructure used by the agents allows them to retain their knowledge and states over an extended period of time. The agents have property of robustness, i.e., they work correctly on the face of some failures at run time.

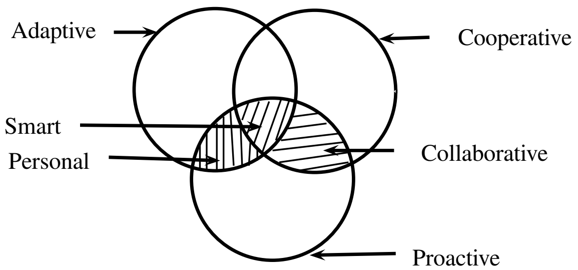
Many characteristics of Intelligent agents are result of capabilities like *adaptability*, *cooperation*, and *proactivity*. see Fig. 16.1 shows agent taxonomy. The circles correspond to general agent capabilities, and intersection corresponds to the agents having either two or three capabilities. For example, an agent having the capability of proactive and cooperative is called a collaborative agent.

Depending on the functions performed, agents can be classified in one of several major categories.

Adaptive agents

They can learn from their previous experience, and can change how they should behave in a given situation, and can also behave differently in given situations.

Fig. 16.1 Agent taxonomy



Cooperative agents

They can communicate with other agents, and their action can be according to the results of the communication performed.

Proactive agents

These agents can initiate proactive actions, i.e., without any prompting from the user or other agents.

Personal agents

They are proactive and can interact directly with a user. While interacting with the user, they present some personality or character, can do monitoring and adapting to the user's activities, can learn the user's style and preferences. They can automate or simplify certain rote tasks. Many software tool-kits, e.g., Microsoft Agent, offer software services set that support the presentation of software agents as interactive personalities and includes natural language and animation capabilities.

Collaborative agents

These agents are proactive and cooperate with other agents. They communicate and interact in groups, many times on behalf of a number of users or organizations, or services. Multiple agents exchange messages to negotiate or share information. Some of their applications are: online auctions, planning, negotiation, logistics, supply-chain management, and telecommunication services.

Smart agents

The smart agents exhibit a combination of all capabilities, i.e., they are adaptive, cooperate with other agents, and are proactive.

Mobile agents

These agents are sent to remote sites to collect the information, and forward it to the central or any other location. Before sending the results to a specified location, these agents can aggregate and analyze data or perform some local control. They are typically implemented in any of the following languages: Java, Java-based component technologies, VBScript, Perl, TCL, or Python. The data-intensive processing is usually performed at the source, as this avoids the shipment of bandwidth consuming raw data. Examples of such applications are network management agents, Internet spiders, and NASA's mobile agents for human planetary exploration, etc.

16.3 Multiagent Systems

The multiagent systems are used by organizations or people with different /conflicting goals, and having proprietary information. In such systems, multiagent system is required to handle their interactions. As an example, in a manufacturing scenario where company *A* produces launching-pad of missiles but subcontracts to company *B* to produce the missiles. To build the whole system comprising missiles launcher and missile, the internals of both companies must be modeled. However, none of these companies are ready to share the details with the other company. Possibly, the two companies may reach to some agreement, or when not ready to share any details due to protocols imposed by the government, multiagent system (MAS) can be created, with one agent for each company, that represents the goals and interest of each company [10].

As another example, consider a teaching time-table system for a college. This domain requires, different agents to represent the interest of different people in the college. Faculty wants their classes should be evenly distributed throughout the week, with possibly all classes in as few rooms as possible, management wants that all the resources be used fully, students want that no more than two/three theory classes be held each day. Similarly, the technicians will have their own requirements. In such a scenario, a multiagent system, where different constraints are handled by agents separately, can create time table static/dynamic to best meet all the constraints.

The multiagent system creates parallelism by assigning different tasks to different agents, hence making overall a fast response system. In addition, many agent's systems will have redundant agents, this helps in building robustness in the system. This is possible because, the control and responsibilities are shared among the agents, hence the system can tolerate failures of some of the agents, and still working correctly and efficiently. The areas of applications that requires graceful degradation at the time of failure, instead of sudden failure, are suitable domains where multiagent systems' use is welcomed. However, if single entity or processor or agent controls everything, the then entire system may crash if there is a single failure.

The multiagent systems have the benefit of scalability. Because they are inherently modular, it is easier to add new systems to a multiagent system than to add new capabilities to monolithic systems. Due to the flexibility available, it is easier to program a multiagent system.

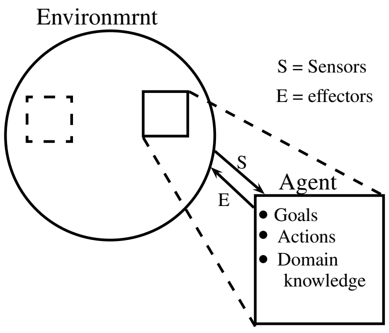


Fig. 16.2 A general single-agent framework

16.3.1 Single-Agent Framework

Though it might appear that a single-agent system might be simpler in concern to dealing with a fixed complex task, however, the opposite is often true. In fact, when control is distributed among number of agents, an individual agent can be simpler. A general agent is a single-agent system, together with the environment, and the interaction between agent and environment. An agent is itself part of the environment, but generally, the agents are considered to have extra-environmental components, which are independent entities, having their own goals, knowledge, and actions. In a single-agent system, no other entities are recognized by the system (see Fig. 16.2).

16.3.2 Multiagent Framework

Following are the taxonomies of multiagent systems:

1. Agent granularity, which can be course or fine.
2. Heterogeneity of agent's knowledge, can be redundant or specialized.
3. Methods used for distributing control can be: benevolent or competitive, team or hierarchical, static or shifting roles.
4. The agents can communicate among other agents, in blackboard or through messages, it can be low-level or high-level communication.

In a multiagent system, there are several agents which are capable of modeling each other's goals and actions. In a general multiagent system, there may be direct interactions among the agents. The inter-agent communication is viewed separate from communication with the environment. A major difference with single-agent system is that, in multiagent systems, the environment dynamics can be determined by the other agents also, which can affect the environment in an unpredictable way. Thus, all multiagent systems can be treated as having dynamic environments.

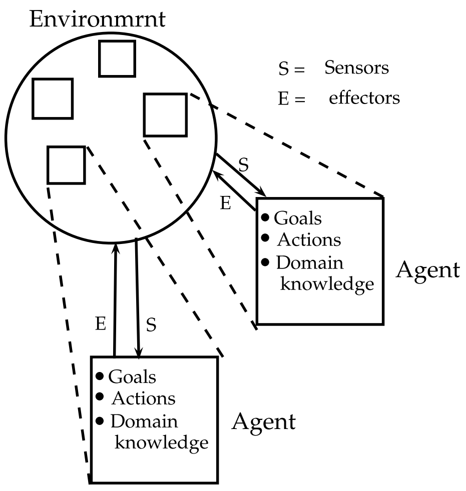


Fig. 16.3 A fully general multiagent framework

The Fig. 16.3 shows multiagent environment, where each agent is part of the environment, as well as can be modeled as a separate entity. There may be any number of agents with different degrees of heterogeneity and with or without the ability to communicate directly [10].

16.3.3 Multiagent Interactions

In agent-oriented view of the world, it has been found that most problems require the participation of multiple agents to represent the distributed nature of the problems, with multiple locations of control, and multiple competing interests. In addition, the agents need to interact with each other to manage dependencies resulting from their existence in a common environment, and to achieve their individual objectives. Such interactions may vary greatly—from a simple information exchanges to, in a more complex form—to request for particular action for coordination/cooperation, or negotiation, or arranging interdependent activities.

Agent interactions are differing on two characteristics with respect to computational models, like networking and shared computing: 1. Agent-oriented interactions are conceptualized as taking place at knowledge level—realized in terms of what goals should be followed, by whom, and at what time. 2. The agents are flexible problem solvers, operating in an environment that is partially observable, and agents have partial control over it. Therefore, the interactions need to be also handled in a similar and flexible manner.

The agents make use computational models to make run-time decisions about the type and scope of their interactions, and also to initiate and respond to interactions that were not anticipated at the time of design of the system [5].

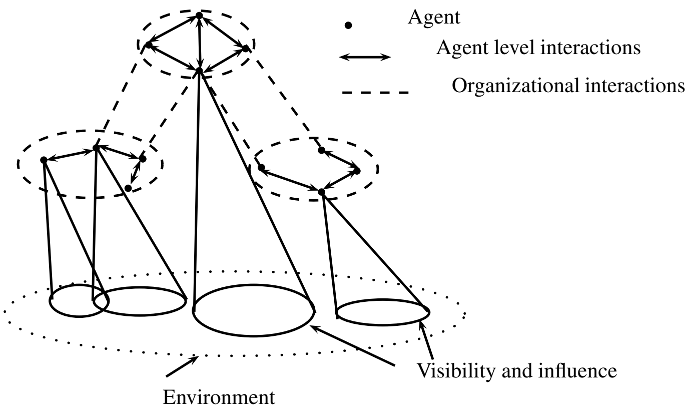


Fig. 16.4 Canonical view of an agent-based system

Agents usually act to achieve objectives as individuals, or as a group for some larger problem-solving initiative. Hence, when agents interact with the environment, there is a hidden organizational context in them, that defines the nature of the relationship between agents—as peers working together in a team or one may be the manager of the other agents, hence may influence their subordinates' behaviors.

Since agents are required to make decisions about the various types of interactions at run time, there is a need for an explicit representation of organizational relationships of agents. Often, these relationships are subject to frequent changes, for example, the agents working on the computation of social interaction must take care of existing relationships in social networks, and should also support the evolution of these relations. The evolution is due to the creation of new relations, and due to the exit of members from these social networks. Looking at these examples, we understand that the life span of these relationships can vary from just long enough to deliver a particular service once to a permanent bond.

To cope with the dynamic scenario of variety and dynamics of relationships in agents, their protocols needs to be devised to support organizational groups to be formed and dismantled, there is need of specified mechanisms to ensure that these groups act together in a coherent way, and also there is need of structures to characterize macro behavior in a collective way.

The Fig. 16.4 shows the essential concepts of agent-based interactions.

The agents capable of having the features presented above are the *Intelligent agents* (also called software agents). These agents are autonomous components, have their own goals and beliefs. They are designed with the capability to reason about their behavior: both present and future, and offer abundant scope for fast, and incremental development of Web-based enterprise applications. The developers can use these systems for a variety of complex and dynamic domains, which range from e-commerce to research on planetary exploration systems.

16.4 Basic Architecture of Agent System

Many complex and intelligent agents navigate on the Internet, collect the relevant data and process them, perform various tasks including data analysis and data communication, and make the decisions on behalf of their users. The present generation of intelligent software agents can manage, organize, and communicate huge amounts of data on behalf of their users. For example, the agents in e-commerce applications can dynamically discover and compose e-services and mediate interactions. They can also be used to serve as delegates to handle routine tasks, like, monitoring activities, set up contracts, win the bid, execute business processes, and can find the best services [3].

Agents reside and execute in a conceptual and physical location called an *agency* (see Fig. 16.5), which provides facilities for locating and messaging to mobile agents and also to those that are detached agents. The agency also facilitates for collecting knowledge about agents. The core of the agency is an agent platform, with component model infrastructure, which provides the local services to agents. The core also includes the proxies to access remote services like security, agent management, communication, persistence, and naming. For mobile agents, the agent platform also provides agent transport. Some additional services provided by most agent systems are in the form of specialized agents that reside in some remote agency. There are some standard service agents, like a *broker*, *auctioneer*, or *community maker*, which augment the basic agents infrastructure. The agent platform along with service agents monitor and control message exchange by detecting any violation of rules while they engage in communication. The agent's platform is the system's core, however, in addition to this, a *component model* infrastructure empowers the agents with local services and proxy access to remote services.

An agent system comprises components with simple interfaces. The major part of the system's capability results from its loose coupling, which helps the agents to interact dynamically through the exchange of messages asynchronously. For communication with each other, the agents must follow some common and well-defined

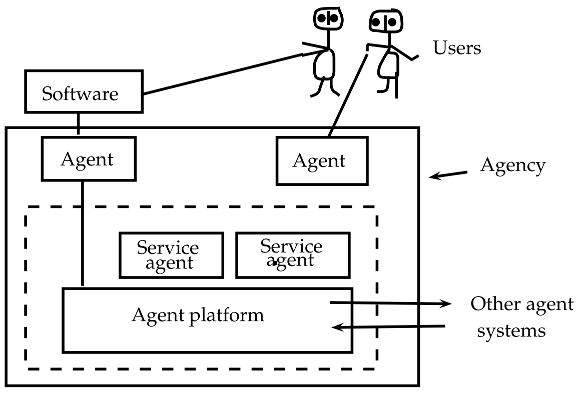


Fig. 16.5 Agent system architecture

protocols. Communication of agents is through a language, called ACL (agents communication language). It is a specialized declarative language, which defines the structure and pattern of interaction between agents. This language is associated with the component model, and partitions the messages into many parts, which are relatively independent of each other. The commonly used message partitions are: message type, addressing, context, content of the message, domain description, and expected conversation patterns. Due to the message protocols and its partitioning of the message, it is easier to dynamically extend the agents to new problem domains, while the system checks conformance to expectations and allows the component model infrastructure to manage messages and agents.

The agents interact among themselves using a set of vocabularies, called ontology, which is designed for the application domain of the agents. The word set in the vocabulary describe the things, their attributes, action performed, various relationships, meanings, and how the agent's system use this vocabulary to structure the interactions, and access the devices.

16.5 Agents' Coordination

For performing complex tasks there is requirements to integrate a group of agents to coordinate the activity. This is possible by a multiagent system, working in either of the two modes, static or dynamic. The agents can coordinate amongst themselves, and also with people. This coordination requires messaging between the agents, the sequence of the messages may have many possible levels of choreography, depending on how loosely or tightly the allowed interactions are controlled by the system. It is common practice that, instead of directly programming as code to handle the messages coordination, some graphical models or high-level declarative rules are used. These models/rules make it easier to visualize how the agents interact. The agent system can use explicit rules or models to monitor or enforce compliance, which makes the programmer's task simple [8].

It is not always the case that coordination would mean cooperation. For example, an effective competitor will coordinate the decisions to maximize his/her advantages against the opponent. This may be seen in the planning of product promotion by a company to undercut a rival.

Various coordination strategies have emerged for computational agents. However, it is not possible to devise a coordination strategy that works equally well in all situations. If any such strategy exists, it can be easily applied for an unlimited number of constructs employed today, such as governments, corporations, markets, teams, committees, professional societies, mailing groups, etc. Whatever strategy is adopted, certain situations can stress it to be a breaking point. Any adopted coordination strategy must now be concerned about how to scale to increasingly complex situations. To map the space of potential coordination strategies, we must find out important dimensions along which they must scale and then evaluate their response to complexities along with those dimensions.

16.5.1 Sharing Among Cooperative Agents

Benefits at a global level of an agent system are bound to improve if all agents cooperate. However, cooperation among the agents is difficult to realize, particularly, in situations when agents are self-interested. For example, if a number of agents are trying to get the same resource, say download a specific file, the download speed is bound to decrease. Instead of this approach, if they work on grouping a social decision that is mutually beneficial, it will be good for all of them. Therefore, designing mechanisms that promote cooperation among self-interested agents is important. In fact, several game theory approaches have been found to be useful for the study of cooperation in agents, e.g., the *Prisoner's Dilemma* (PD) as a theoretical framework (see Chap. 11), which is well known for this purpose. The PD can be useful for understanding the role of local interactions to maintain cooperation among the agents. It is based on the conflict of interest, i.e., between what is the best for the individual (i.e., defection) and what is best for the group (i.e., cooperation). This creates a situation of *social dilemma*. Therefore, specific mechanisms are required to evolve cooperation to help the population to overcome this dilemma.

There are three basic approaches to prevent social dilemmas, and to promote and stabilize cooperation as follows:

Coalition-based mechanism

The coalition-based approach is useful for establishing collaboration among agents, with an individual having properties and objectives. These mechanisms use a *tax model*, due to which agents can achieve cooperation when coalitions are formed around some emerging leaders. To maintain coalitions, the leaders charge some tax from their agents in favor of some benefit (e.g., guaranteed cooperation, protection from cheaters, etc). The concept of coalition has been used in the game theory for a long time, and has been proved useful in real-world economic scenarios. The dynamic coalition formation model considers the grid topology of agents for cooperation between them and makes use of *spatial* prisoner's dilemma.

The coalitions facilitate cooperation between self-interested agents. The first approach is: a leader of coalition is paid by the agents that are in the coalition. The coalition leader also imposes its decision on the agents in the coalition to maximize cooperation. The decision making of each coalition is done in a centralized manner by a single entity, called *leading agent*. The agents' cooperation with their coalition-mates also assumes some restriction in the collaboration.

The coalition-based approach is a clear example of the known trade-off between the benefits versus the costs of collaboration (e.g., taxes). Therefore, this mechanism is called *dynamic coalition formation model*, and also *tax model*.

Partner switching mechanisms

In most real-world situations, the network topology changes frequently. There is an empirical evidence in the games on dynamic topologies that a partner switching leads to cooperative behavior. A variant model of prisoner's dilemma allows agents to either adjust their strategies or switch their defective partners, with the aim that partner switching may help stabilize cooperation.

Self-governing institutions

The resource allocation in case of self-governing Institutions is modeled in a network, based on a formal characterization of socio-economic principles. An agent should autonomously decide how to behave with respect to coalition-mates and agents outside its coalition. Although some mechanisms promote cooperation on different network topologies, these networks are static.

16.5.2 Static Coalition Formation

There are two approaches for static coalition formation: 1. Optimization-based approaches, that focus on finding an optimal coalition, and 2. Game theoretic approaches. The later has applications in many real-world domains, like electronic commerce, auctions, and general resource allocation scenarios. The game theoretic approaches may also involve automated agents [8].

In coalitions with optimization objectives, the challenge in coalition formation is generating coalition structure, which turns out to be an NP-complete problem as a general case, hence the existing algorithms cannot generate solutions in a reasonable time, even with the moderate size of the game (number of agents). Hence, finding an optimal coalition can become intractable because the number of coalition structures grows exponentially with a number of agents.

Among other goals, one goal of coalition formation is to improve cooperation among the agents. The game theory approaches have been widely used to address the issue of cooperation among agents. A class of coalition formation game, called *hedonic games*, is a rich and versatile class for coalition formation, suited for both static and theoretical aspects of coalition formation, and has the property of encapsulation in the stable matching scenarios. The major focus of hedonic games is on critical stability for coalition structures, e.g., Nash stability, individual stability, contractual individual stability, and core stability. These games also characterize conditions under which the set of stable partitions is guaranteed to be non-empty.

16.5.3 Dynamic Coalition Formation

In formation of coalition in a dynamic environment, the agents constantly change the coalition they belong to. In such a scenario, since optimality is possible with a very small number of agents, computing of optimal coalition is either infeasible, or may take a time longer than the lifetime of a coalition for any realistic number of agents. Thus, the time constraint to find an optimal coalition prevents its use in a dynamic multiagent system, where some agents have to decide if it is beneficial to them to join other agents for a small amount of time. This time limitation is due to the fact that for n number of agents, the total number of possible coalition structures to be enumerated are of the order of $O(n^n)$, which is too large unless n is a small number. For large n, the computation cannot be carried out in realistic times. Hence, it is necessary to make use of domain knowledge, with mathematical games and some constraints to solve the problem of coalition formation in an efficient way, for the set of agents of any specified characteristics [8].

To form a dynamic coalition, it is required to have decentralized procedures to allow self-interested agents to negotiate the formation of coalitions, as well as to divide the coalition payoffs. In the real-world scenarios, the agents may turn out to be selfish and may focus on improving their own performance, but if they are cooperative, the performance of the whole system will improve. Hence, the theory of non-cooperative games (i.e., agents are selfish) is suitable to model the formation of coalitions and their dynamics. The prisoners' dilemma is a case of this category, since the prisoners are considered as selfish (defecting is the dominant strategy) in this game. A variant of this, called *Iterated Prisoner's Dilemma* (IPD) game is widely used to model various social and economic phenomena, and the cooperation among agents. In the IPD, where a total number of rounds is random or unknown, sustained cooperation strategies are likely to emerge.

16.5.4 Iterated Prisoner's Dilemma Coalition Model

We present a model, where a graph (or network topology) representing population is iterated, the *nodes* in the network represent agents, and *edges* represent relations between agents. Such agents interact with their peers in the social neighborhood (the agents to which they are linked), and play the game of *Possessors-Traders* (an agent is either a possessor or a trader). Such agents not only cooperate or defect, but have resources using which they can trade. The agents can form coalitions to increase the cooperation level of the multiagent system. In such coalitions, group decisions can result in mutually beneficial cooperation, that holds over some time. The group decisions lead to, and is an indication of social behavior. In addition, the agents' neighborhood is not static, and can change partners through rewiring. Hence, in addition to the trading of resources, each agent decides the following during the game [8].

– To remain independent or to be part of a coalition, depending on which alternative provides more payoffs.
– Whom to rewire with? As agents change their neighborhood, they rewire to improve the benefits.

In addition, all the agents in a coalition behave like a unit, and all together decide how they should behave with those in the coalition (called insiders), and those outside the coalition (called outsiders). Hence, the decision about what is a coalition and what is its behavior, is an important criterion in the dynamics of the coalition system. In their behavior the agents work in cycles: trading strategies, rewiring (changing of coalition) strategies, and coalition strategies. The Algorithm 16.1 shows this cycle, where x represents *Payoffs*.

Algorithm 16.1 One cycle of Agents-dynamics

```
1: x = trade-with-all-neighbors();
2: rewire(x);
3: revise coalition(x);
```

Agents use certain trading strategies to trade among the agents, based on some model, which can be called by a general name *property ownership and trade* (POT) model. The trading model is based on extension of the Iterated prisoner's Dilemma, in which agents can cooperate or defect the actions. The model of POT comprises two types of players: 1. The *Possessors* P, who own the resources, and 2. *Traders* T, who sell and buy resources.

The strategy of any agent $p_i \in P$ models the practice of ownership, but does not trade. The behavior of p_i depends on whether it owns a resource or not. If p_i owns a resource it acts as a defector, but if it does not, then it cooperates. This strategy is shown in Algorithm 16.2, where $owns(p_i, resource)$ indicates that possessor p_i owns some resource, $defects(p_i)$ means p_i defects, and $cooperate(p_i)$ means p_i cooperates.

Algorithm 16.2 Possessor p_i's Strategy

```
1: if $owns(p_i, resource)$ then
2:     $defects(p_i)$
3: else
4:     $cooperates(p_i)$
5: end if
```

An agent $t_j \in T$ is trader, who is willing to sell or buy a resource when dealing with a fellow trader $t_k \in T$. For example, if some t_j has a resource for selling, it will try to get the maximum benefit by selling it. Whenever any pair of traders (t_j, t_k) meet, the trader (say t_j) owning the resource values the resource at a random value y_j (but does not make it open), such that $v < y_j < V$, and $v, V \in \mathbb{R}$ (real numbers). In response, the buyer agent t_k offers a value y_k for the resource, such that $v < y_k < V$. If $y_k > y_j$ then the buyer purchases the resource at some random value y_l, so that $y_j < y_l \leq y_k$. This is called trader T's strategy, with its logic given in Algorithm 16.3. If the trader plays against who is not a trader, then it is the possessor.

Algorithm 16.3 Trader T's Strategy

```
1: if $isTrader(t_j)$ AND $isTrader(t_k)$ then
2:   if $owns(t_j, resource)$ AND $v < y_j < V$ then
3:     Sell for $y_j$
4:   else
5:     if $owns(t_k, resource)$ AND $y_j < y_k$ then
6:       Buy for $y_l$ AND $y_j < y_l \leq y_k$
7:     else
8:       Behave as Possessor
9:     end if
10:  end if
11: else
12:   Behave as Possessor
13: end if
```

16.5.5 Coalition Algorithm

A Algorithm 16.4 shows the basic strategy followed by agents either to join a coalition or leave a coalition (change to a new one). If an agent a_i has the poorest payoff among all its neighboring agents after completion of the previous round of computations (line 1), then a_i makes a new coalition with some agent a_j (line 4), who is free and has the best payoff. If a_j is already in some other coalition, then a_i joins a_j's coalition (line 6). This rule also enables any agent to change from one coalition to another, in case that agent receives poor payoffs in the former coalition [8].

In a dynamic network, agents form coalitions to behave as a unity. An agent can belong to only one coalition at a time. All agents belonging to a coalition are not required to be linked with each other, but behaves as a set to act together to maximize their performance. However, an agent must have at least one link to some agent belonging to its coalition. If that is not the case, it is an isolated agent, hence it must be declared as an independent agent (lines 9–10, Algorithm 16.4). This connection helps an agent to know its coalition information, strategy, share, and divide gains. Again, if an agent changes link, it does not imply that it changes its coalition—it simply rewires to change neighbors.

The agents that are in a coalition, must agree to some specific behavior to play with *insiders* (agents in the coalition) and also with *outsiders* (agents outside the coalition). We assume a flat coalition, i.e., there is no leader or central authority to impose any policy. To decide the coalition behavior in this situation, each agent votes for a strategy of either P or T (for possessors and traders) to play with insiders as well as with outsiders (line 16, Algorithm 16.4).

To decide the vote, each agent uses a Learning Automata (*LA*), trained from its trading history and payoffs (lines 12–15). The LA algorithm keeps two *probability models*: $InProb$, and $OutProb$. The model $[InProbT, InProbP]$ is used to assess the strategy to play against insiders. Here, $InProbT$ is the probability of being inside the coalition as a trader, and $InProbP$ is the probability of being inside the coalition as possessor. In a similar way, $[OutProbT, OutProbP]$ is to assess the strategy to play against outsiders.

Algorithm 16.4 Revise Coalition(Payoffs)

```
1: if $poorestPayoff\_from\_neighbors(a_i)$ then
2:     $a_j = neighborWithBestPayoff()$
3:     if $independent(a_j)$ then
4:         $CreatNewCoalition(a_j, a_i)$
5:     else
6:         $JoinCoalition(a_j, a_i)$
7:     end if
8: end if
9: if $IsolatedAgent(a_i)$ then
10:    $makeIndependent(a_i)$
11: else
12:    $[InProbT, InProbP] = UpdateInsidersLA(Payoffs)$
13:    $[OutProbT, OutProbP] = UpdateOutsidersLA(Payoffs)$
14:    $InAction = InActionChoice(ProbInT, ProbInP)$
15:    $OutAction = OutActionChoice(OutProbT, OutProbP)$
16:    $VoteBest(InAction, OutAcion)$
17: end if
```

16.6 Agent-Based Approach to Software Engineering

In respect of software engineering, we view agents as next-generation components and *agent-oriented software engineering* as an extension of conventional CBSE (case-based software engineering). The developers can integrate different types of agents, like, personal, mobile, and collaborative agents, to build agent-based enterprise systems, covering a wide problem domain area. To patrol the networks to find available resources, special software is used, called Daemons.

Developers often use distributed objects, active objects, and components that can be scripted to implement agents. The agents are often driven by goals and plans instead of procedural code, they encapsulate business or domain knowledge. These agents often differ more from each other by the knowledge they have and the roles they play, than by the differences in their implementing classes and methods. The agents are capable of using different mixes of adaptability, mobility, intelligence, ACL, and even multiagent support. Either AI programming languages or conventional programming languages can be used to implement the agents.

Next, we introduce the techniques for tackling complexity in software [5].

Decomposition

For tackling large problems, the basic technique is to divide the problem into smaller chunks, such that it is better manageable. Each of these chunks is dealt with relative isolation. Since this limits the designer's scope, it helps to tackle the complexity of the issues, because it requires to consider only a small portion of the problem at any given time.

Abstraction

The abstraction is a process of defining a simplified model of any system, such that only the necessary and important details or properties and emphasized, while all unnecessary details are suppressed.

Organization

The process of organization is concerned with the identification and managing the relationships between various problem-solving components. Specifying and implementing organizational relationships are helpful to tackle the complexity due to two reasons: 1. It facilitates the grouping of a number of basic components, which are collectively treated as a unit at a higher level for the purpose of analysis. 2. Due to grouping the components as a unit, as well as to specify the relationships between them, a number of components may work together (cooperate) to provide a particular functionality.

16.7 Agents that Buy and Sell

The Software agents were used much earlier for the applications, like filtering information, match people having similar or identical interests, and automating repetitive behavior.

In the recent past, agents have found the applications in e-commerce to conduct business-to-business, business-to-consumer, and consumer-to-consumer transactions. Consider an example of buying and selling, where a company willing to place an order for procurement of stationery, assigns the tasks to agents to monitor the quantity and usage patterns of paper within the company. It also launches the buying agents when paper inventory is low. The *Buying agents* would typically perform the following tasks, more or less, in order [6].

1. collect the information automatically about vendors and the required products which best fulfills the needs of the company,
2. evaluate the various offers from the vendors,
3. make a decision about merchants and products that require further investigation,
4. negotiate the terms of transactions with these merchants, and finally
5. place orders and make automated payments.

There are several descriptive theories and models that seek to capture buying behaviors, e.g., *Nicosia model*, *Howard-Sheth model*, and *Engel-Blackwell model*. All these share six fundamental stages of the buying process:

1. *Identification*. The buyer can be motivated through product information, hence he/she becomes aware of some unmet needs.

2. *Product brokering*. The buying process comprises as its part, the Information Retrieval to determine what to buy. IR consists of evaluation of product alternatives based on the criteria provided by the buyer, whose result is a set of products, called "consideration set."
3. *Merchant brokering*. This activity combines the "consideration set" with merchant-specific information to help customer to decide as from where to buy the goods. This stage also comprises the evaluation of merchant alternatives based on buyer-provided criteria. The later is typically, the price, warranty, delivery time, availability, and reputation, which are not necessarily be in order, but varies case to case.
4. *Negotiation*. This step considers how to settle on the terms of transition. Negotiations vary in duration and complexity, for price and other attributes.
5. *Purchase and delivery*. This step signals either termination of the negotiation stage or occurs some time afterward.
6. *Product service and evaluation*. This involves the post-purchase product service, customer service, and evaluation of the satisfaction of overall buying experience.

In the present online buying and selling, many of the processes and criteria discussed above are in a common place.

The continuous running personalized autonomous agents are well suited to mediate for consumer behaviors, like, information filtering, IR, personalized evaluations, time-based interactions, and complex coordination. Many agents perform constraint-based, and collaborative filtering. Many websites of online-shopping use rule-based techniques to personalize the products offering for individual customers. Some websites use agents to experiment data-mining techniques to discover patterns in customers' purchase behavior, exploit those behaviors for sales, and use these patterns also to help customers to find other products that meet their true requirements.

The product alternatives are compared at the product brokering stage, whereas the merchant alternatives are compared at the merchant brokering stage.

16.8 Modeling Agents as Decision Maker

For modeling agents as decision makers, it is necessary to have modeling methods that use formal notions of mental state to represent and reason about agents. The mental states may consist of mental attributes such as beliefs, knowledge, and references. In multiagent systems, the success of one's actions and plans are governed by the actions of other agents. Thus, agents can help in constructing plans that are likely to succeed. The mental level models can bring two informal properties: 1. They provide an abstract way of representing agents, which is implementation-independent, and, 2. These models are built using an intuitive approach, and use attributes, such as goals, beliefs, and intentions. The abstract nature of models have the following practical implications [1]:

1. A single formalism can capture different agents, written in different languages, and running on different hardware platforms,
2. There are no implementation details in abstract models, and
3. Fever lower-level details in abstract models result in faster computation.

16.8.1 Issues in Mental Level Modeling

In mental level modeling following are the central questions:

1. *Structure*. The structure holds the designer's initial database of beliefs, goals, intentions, which are manipulated by the agent.
2. *Grounding*. It is the base for the model construction process, which is essential because we cannot directly observe the mental state of another agent.
3. *Existence*. Under what conditions a model will exist? Answer to this question will be helpful in evaluating any proposal for mental level models. Therefore, it is necessary to know, what assumptions are made, or biases we are making when we model agents in this manner.
4. *Choosing a model*. How do we choose between different models that are consistent with our data?

16.8.2 Model Structure

A model's mental level structure consists of three key components: *beliefs*, *references*, and *decision criteria*, which in order corresponds to accounting for the agent's perception about the world, its goals, and method of choosing actions under uncertainty, respectively.

The agents' belief help in establishing about which states of the world it considers, are plausible. As an example, the possible worlds of interest may be about weather conditions: *rainy* and *non-rainy*, and let the agent believes rainy to be plausible. In fact, the agent's preference indicates how much it likes each preference. The agent may have two possible *actions*: take an umbrella along to protect from rain, and do not take an umbrella along. The outcomes of these actions are shown in Table 16.1. The agent's preferences tell us how much significance it gives to these values. We will prefer to use real numbers to describe these values, such that larger numbers indicate better outcomes, as shown in Table 16.2 [1].

An agent would choose its action (take or not take umbrella) by applying its decision criteria to the outcome of different actions in the world. A commonly used decision criterion is *maximin*, where the action for the "best worst-case" outcome is chosen. The best outcome out of (10, −4) and (−1, 8) is 10 and the worst is −4. The best in the worst-case is −1, so the agent chooses the action "Do not take an umbrella". This is because the outcome worst −1 is better than worst −4.

Table 16.1 Decision table for an agent

Action (↓), Worlds (→)	Rainy	Not-rainy
1. Take umbrella	Dry, Heavy	Dry, Heavy, Illogical
2. Do not take umbrella	Wet, Light	Dry, Light

Table 16.2 Table with weighted outcomes

Action (↓), Worlds (→)	Rainy	Not-rainy
1. Take umbrella	10	−4
2. Do not take umbrella	−1	8

Having gone through the above example, we are now in a position for grounding. We can view the problem of describing a mental state of the agent as a CSP (Constraint Satisfaction Problem). The state of the model is such that it should have generated the observed behavior, and it is consistent with the background knowledge.

In the above example, the background knowledge is agents preferences, given in the Table 16.2, and decision criteria is *maximin*. We observe that if the agent goes out without an umbrella, it believes that "no rain will come", for it is had other beliefs it would have taken a different action.

Once an agent's model is constructed, it can be used to predict its future behavior. To give it a formal shape, we consider that an agent $\mathscr{A}$, is described as a state machine, with set of possible local states $L_{\mathscr{A}}$, a set of possible actions $A_{\mathscr{A}}$, and a program, which we call its *protocol* $P_{\mathscr{A}}$. Thus an agent is a tuple,

$$\mathscr{A} = \langle L_{\mathscr{A}}, A_{\mathscr{A}}, P_{\mathscr{A}} \rangle \tag{16.1}$$

where $P_{\mathscr{A}} : L_{\mathscr{A}} \rightarrow A_{\mathscr{A}}$. All the agents function with some environment, so we assume $L_{\mathscr{E}}$ as set of all states in the environment. The environment describes every things external to the agent, which may possibly include other agents also. The combined state of the whole system, i.e., both the agent and the environment are referred to as *global state*, and represent by a pair $(l_{\mathscr{A}} \times l_{\mathscr{E}}) \in L_{\mathscr{A}} \times L_{\mathscr{E}}$. We further assume that environment does not perform actions, and agent's actions are deterministic functions of its state and the environment's state. Thus, the set of possible worlds will be only a subset S of the set of global states $L_{\mathscr{A}} \times L_{\mathscr{E}}$. And, finally, a *transition function*,

$$\tau : (L_{\mathscr{A}} \times L_{\mathscr{E}}) \times A_{\mathscr{A}} \rightarrow (L_{\mathscr{A}} \times L_{\mathscr{E}}) \tag{16.2}$$

maps a global state and an action to a new global state.

Example 16.1 An agent $\mathscr{A}$ to model a an air-conditioner's thermostat control.

The modeling of agent $\mathscr{A}$ is shown in Table 16.3, which shows the results of transition function τ. It should have local states $L_{\mathscr{A}} = \{-, +\}$, where '−' corresponds to

Table 16.3 Transition table for thermostat agent

Worlds: $L_{\mathscr{A}} \times L_{\mathscr{E}} \rightarrow$ Action: $A_{\mathscr{A}} \downarrow$	$(-, cold)$	$(+, cold)$	$(-, ok)$	$(+, ok)$	$(-, hot)$	$(+, hot)$
Turn-on	$(-, ok)$	$(+, ok)$	$(-, hot)$	$(+, hot)$	$(-, hot)$	$(+, hot)$
Turn-off	$(-, cold)$	$(+, cold)$	$(-, ok)$	$(+, ok)$	$(-, ok)$	$(+, ok)$

the state when the thermostat indicates that the temperature is less than the room temperature, and '+' for temperature greater than or equal to desired room temperature. The thermostat's protocol is given below.

State	$-$	$+$
Action	Turn-on	Turn-off

The thermostat's actions are modeled as $A_{\mathscr{A}} = \{turnon, turnoff\}$, and the environment's states are, $L_{\mathscr{E}} = \{cold, hot, ok\}$. For the sake of simplicity, we assume that the possible world is $L_{\mathscr{A}} \times L_{\mathscr{E}}$, which are displayed in the heading row in the Table 16.3.

Given the set of possible worlds W, we can associate with each local state l of the agent (the thermostat), a subset S, $W(l)$, comprising of all worlds in which the local state of the agent is l. □

In the above example, the effects of an action on the environment do not affect the state of the thermostat. In addition, the static one-shot model assumes some simplifications. The first assumption is that the room temperature is affected only by the thermostat and not by external influences. Second assumption is that the thermostat state's actions do not affect its state.

It may be noted that, while the thermostat knows its local state, it knows nothing about the room temperature. Consequently, we made all pairs of $L_{\mathscr{A}} \times L_{\mathscr{E}}$ possible, including the $(-, hot)$ as the possible world, indicating that thermostat's local state is indicating low temperature, while the environment state is *hot*. In one aspect, it simplifies the system by assuming all possible worlds, while in other terms, it is a blessing, as this is taken as a situation, where we assume that there may be a measurement error in the thermostat [1].

Definition 16.1 (*Belief*) A *belief assignment* function B may be defined as $B : L_{\mathscr{A}} \rightarrow (2^S - \phi)$, so that for all $l \in L_{\mathscr{A}}$ we have $B(l) \subseteq W(l)$. The value $B(l)$ is referred as worlds *plausible* at l.

16.8.3 Preferences

The *beliefs* make sense being part of a more detailed description of the agent's mental state, which has more associated aspects. One such aspect is the agent's preference order in the possible worlds, which can be taken as the agent's desire. There are various assumptions about the structure of the agent's preferences, which consider a *total order* on the set of possible worlds W. However, we may need a richer algebraic structure, in some cases, e.g., one in which addition is defined. We use a *value function* to represent the agent's preferences.

Definition 16.2 (*Value function*) A value function is a function $u : S \rightarrow \mathbb{R}$. □

This numeric approach of representation of agent's preferences is most convenient, where a state s_1 is at least as preferred as state s_2, *iff* $u(s_1) \geq u(s_2)$.

Considering the example of the thermostat (as agent), the goal of the agent is to make room temperature *ok*. Thus, the thermostat/agent prefers any global state in which the environment's state is *ok*, over any global state which may be *cold* or *hot*, and is indifferent between *cold* and *hot*. And, it is also indifferent between the states, where the environment's states are identical, i.e., $(+, ok)$ and $(-, ok)$. This preference order over possible worlds can be represented by a value function. The value function assigns zero to global state in that environment when the state is either *hot* or *cold*, and assigns 1 where the environment's state is *ok*. This outcome is represented in Table 16.4, where $*$ stands for either $-$ or $+$.

If the exact state of the world was known to the thermostat, it would have no trouble in selecting proper action based on the value of its outcome. Considering the case of value as *cold*, the *Turn-on* action would lead to the best outcome. However, when there is uncertainty, the thermostat must compare *vectors* of plausible outcomes instead of a single outcome. For example, for the belief assignment $B(l) = \{cold, ok\}$, the plausible outcome of the action *Turn-on* is $(1, 0)$, and of the action *turn-off* it is $(0, 1)$.

Given the transition function τ, the belief assignment B, and an arbitrary, fixed enumeration of elements of $B(l)$, the *plausible outcomes* of a protocol P in l is a tuple whose kth element is the value of the state generated by applying P starting at the state of $B(l)$.

Table 16.4 Global outcome preference for an agent

Worlds ($\rightarrow$) Action ($\downarrow$)	$(*, cold)$	$(*, hot)$	$(*, ok)$
Turn-on	1	0	0
Turn-off	0	1	1

Table 16.5 Table with weighted outcomes

Action (↓), Worlds (→)	Rainy	Not-Rainy
1. Take umbrella	10	−4
2. Do not take umbrella	−1	8

16.8.4 Decision Criteria

The values can be compared easily, however, it is not clear as how to compare the plausible outcomes. Therefore, we choose some protocols. A strategy for making choice under uncertainty is required that depends on the agent's attitude towards risk. The strategy can be represented by decision criteria, which is a function with a set of plausible outcomes and returning a set of most preferred out of these. For this we make use of *maximin* criteria discussed earlier. Hence, we reproduce the same Table as 16.5.

Note that when both the worlds are plausible, the two plausible outcomes are $(10, -4)$ and $(-1, 8)$. On the condition, the *maximin* criteria is used, the first action, corresponding to "take umbrella" is the most preferred one. But, when the *principle of indifference* is used, the plausible outcome "do not take umbrella" is preferred. Accordingly, decision criteria can be defined as follows:

Definition 16.3 Decision criteria.

A decision criteria is a function:

$$\rho : \bigcup_{n \in \mathbb{N}} 2^{\mathbb{R}^n} \rightarrow \bigcup_{n \in \mathbb{N}} 2^{\mathbb{R}^n} - \phi \tag{16.3}$$

that is, from sets of equal length tuples of reals, to sets of equal length tuples of reals, so that $\mathscr{U} \in \bigcup_{n \in \mathbb{N}} 2^{\mathbb{R}^n} - \phi$, we have that $\rho(\mathscr{U}) \subseteq \mathscr{U}$ (i.e., it returns a non-empty subset of the argument set).

Note that the decision criteria can be used to compare tuples. For example, if $\rho\{u, v\} = \{v\}$, then we say that v is more preferred than u.

16.9 Agent Communication Languages

The agents working together, irrespective of whether they are cooperating or competing, is called a multiagent system. These systems provide higher level of abstraction than the traditional distributed computing. The abstractions are closer to the users' expectations, and allow the designers a higher flexibility in determining the behavior. For example, instead of hard-wiring a specific behavior into the agents, multiagent system designers design the agents with the capability to negotiate amongst themselves and find out the best course of action for a given situation. The ACLs (Agent Communication Languages) must be flexible enough to accommodate abstractions such as negotiations. But, the same flexibility makes it harder to succeed in understanding their semantics [9].

Due to this reason, we must examine many elements to arrive at he meaning of a communication, which includes, type of meaning, perspective, basis (semantics or pragmatics), context, and coverage, i.e., number of communication actions included.

The formal study of languages comprises three parts: 1. Syntax, which is concerned with organizing the symbols to create the structure of language sentences, 2. Semantics, which deals with what sense is denoted by the sentences and their parts, and 3. Pragmatics, which is concerned with how the sentences are interpreted and used. The combined meaning of a sentence is obtained due to semantics and pragmatics. The pragmatics includes those considerations that are external to the language, like, state of the agents, and the environment in which the text exists. Therefore, the pragmatics can restrict, as to how the agents can relate to one another and how they process the messages which are sent or received. In a situation when agents are not fully cooperative or they cannot find out the implications, they cannot meet the pragmatic requirements.

Semantics versus pragmatics

A perspective can be combined with a type of meaning, either *personal* or *conventional*. In case of personal, the meaning of communication is based on intention and interpretation of receiver/sender. The action, “purge this file” shall be taken by the receiver as *directive*, whereas “This is an old file”, shall be taken as an assertion. In Fig. 16.6, the `inform` construct is to give the information to the receiving agent, the `request` construct requests for rain, and so on. In *conventional* meaning, the meaning of communication actions is based on usage conventions. A language is nothing but a system of conventions. Violating the idea of conventions, the traditional approaches go against the wisdom of having different labels for communication actions. The language *KQML* (Knowledge Query Management Language) have all acts as variants of `tell`, whereas for communication language *Arcol*, it is `infom`.

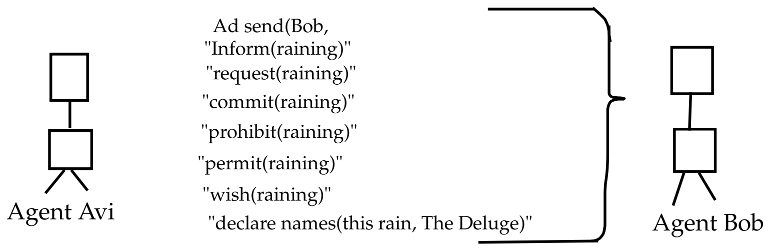


Fig. 16.6 An example of agent language

Context

In general, we do not understand a communication without context. Here, in agents, it is agent's physical or simulated environment, which becomes the context. For agents, the social context is not as subtle as for humans, but they must understand what an agent expects from others.

Coverage of communicative acts

When heterogeneous autonomous agents exchange information, the meaning of the exchange is decided by communicative actions. All these actions fall into one of the following categories:

- *Assertive*. This action is to inform. For example, "The door is shut."
- *Directive*. This is for request, for example, "Shut the door". It can also be used for query, e.g., "Can I shut the door?"
- *commissive*. To promise something, e.g, "I will shut the door."
- *Prohibitive*. It can ban something. For example, "Please do not shut the door"
- *Declarative*. It causes events in themselves. For example, "This information is redundant."
- *Expressive*. To express emotions and evaluations. "I wish that hurricane will stop."

Communication actions can be represented in stylized forms like, "I hereby request …" or "I hereby declare …". The grammatical form emphasize that through the language, you not only make statements but perform actions. The action by speaking becomes the essence of communication. Figure 16.6 shows that all primitives of this agent language are *assertive* or *directive*. In the agent language *Arcol*, one can simulate commissiveness using other acts. All the acts can be reduced to the category of assertive, but these categories have only restricted meanings. For example, a request in *Arcol* language is the same as conveying to the receiver that the sensor intends for it to perform the action.

Considering the code given in Fig. 16.6 for agent Avi, each communication act has a challenge for language, which promotes mental agency. The traditional approaches ignore whether Bob has really the capability to cause rain when it is requested or allowed to do so, or whether it can stop the rain when it is prohibited from causing the rain. Similar is the case for, whether Avi can make it rain when he promises; or whether Avi has the authority to permit or prohibit any of Bob's actions or to name whether conditions.

Finally, the ACL approaches conclude that if Avi's designer wants it to comply with, then it does. This is quite unsatisfactory, because it means that agents do not have any reasoning about their limitations.

16.9.1 Semantics of Agent Programs

A platform that supports the creation and deployment of multiple software agents must have the capability to interoperate with a wide variety of custom-made, as

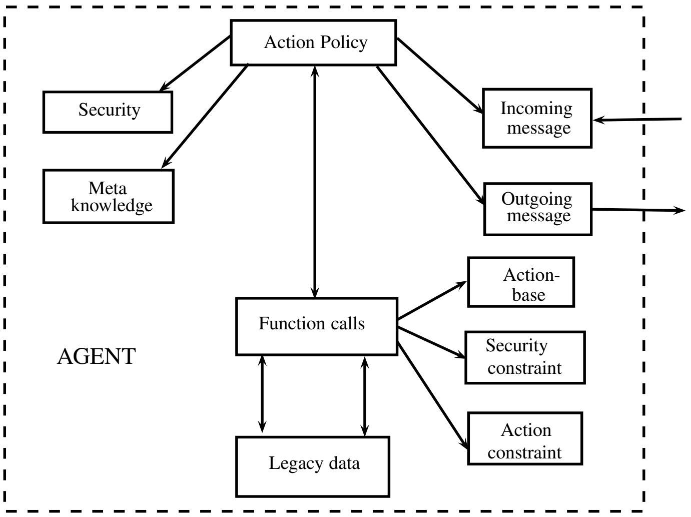


Fig. 16.7 Architecture of IMPACT agent system

well as legacy software sources. What it requires for a software package $\mathscr{S}$ to be considered as an agent program, is that it must come accompanied with tools to augment, modify, and message $\mathscr{S}$ to another agent.

Figure 16.7 shows, the architecture of a system, called *IMPACT*, used for the creation and deployment of multiple interactive intelligent agents. It was a joint research project created by the collaborative work of some Universities. In IMPACT, an agent comprises two parts as described below [2].

Software code

It is a program written in any programming language that supports a well-defined API (application programming interface), which may be part of the code or developed separately to augment the code. The program (S) may be represented as a pair,

$$\mathsf{S} = (\mathsf{T}_S, \mathsf{F}_S) \tag{16.4}$$

where,

- T_S is set of *all data types* manipulated by this program, and the set is closed under all the subtypes, i.e., if τ is subtype of T_S, then $\tau \in \mathsf{T}_S$, and,
- F_S is the set of *all pre-defined functions* of set S that are provided by the package's API.

In other words, S is a collection or hierarchy of objects classes in any standard object data management language.

For example, in Oracle, the database may be viewed as $\mathsf{S} = (\mathsf{T}_S, \mathsf{F}_S)$, where T_S comprises all data types (all attribute domains, tuple of different combinations of domains, and relations on tuples). Whereas, F_S is a set of all functions, i.e., all relational operations: select, project, Cartesian products, join, union, etc.

At any given point of time t, the *state* of an agent will refer to a set $\mathsf{O}_S(t)$ of objects from the type T_S, managed by its internal software code. An agent may change its state by taking *action*, which may be triggered internally or by processing a message received from other agents. But, an agent cannot directly change the state of another agent, but can do so by issuing a request message to that agent.

Semantic wrapper

A *semantic wrapper* contains a large collection of semantic information. Following are the typical contents of this information (see Fig. 16.7):

1. *Service description*. It is represented in some language, with flexibility to modify it.
2. *Message manager*. It manages the data-structure associated with the message box, and specifies and implements the policies.
3. *Action module*. It takes input of a new message consisting of an event. This message is used to trigger zero or more actions. Thus, the action module requires: (1) action base: the actions the agent may take in principle, conditions the agent state must satisfy for the actions to execute, as well as the effects of those actions, (2) Action requirements: conditions under which the agent is allowed or barred from taking the actions, (3) Action policy: What actions to choose out of many?
4. *Meta-knowledge module*. It provides to the agent information about itself, as well as about other existing agents in the *world*. This knowledge may include statistical information on the reliability of other agents, the speed at which other agents can provide the services, financial charges levied for such services. It also provides the self- knowledge, like about its own performance, analysis of various operations performed by itself.

16.9.2 Description Language for Interactive Agents

An Agent's internal mechanism is based on languages, that describe the agent's behavior and its communication protocols. Examples are (1) *Soar*, a general cognitive architecture for developing systems that exhibit intelligent behavior, and (2) Knowledge Query and Manipulation Language (KQML), a language and protocol for developing large-scale sharable and reusable knowledge-bases [4].

There is another language *Q*, which is used for describing interactions between agents. Rather than depending on the internal mechanism, *Q* provides an interface between computing professionals and scenario writers. Due to change in focus, from internal mechanism to interaction, language's syntax and semantics are quite different. For example, agent accepting requests *on* or *off*, which have the standard meanings. However, if agents received a *move* command, it may have different semantics, like move fast, slow, as detailed by the semantics.

Because the language *Q* is suited for interactions, it is used for scenario writing. Example of primitives for interactions are *cue* and *action*. A cue is an event that triggers interaction, while *actions* are requests to an agent which causes the change in the environment. Unlike the programming languages, the language *Q* does not define the semantics of cues and actions. Since different agents execute the cues and actions in different ways, their semantics depend on corresponding agents. The Example 16.2 demonstrates the *cues* (preceded with question mark) and *actions* (preceded by exclamation mark).

Example 16.2 Cues and Actions.

```
(?hear "Hello" : from Tom}
(!walk :from class_room
       :to library)
(!speak "Hello" : to Tom)
(?see  library
       :direction north)
```
□

In the above example, the following cues and actions are there:

1. Agent waits for Tom to say Hello (?`hear`),
2. Tom walks from the class room to library (!`walk`),
3. Agent says hello to Tom (!`speak`), and
4. Agent asks, do you see the library in north (?`see`).

The above are *synchronous* actions, and each one to be followed on completion of the previous.

The *asynchronous* actions allow overlapped execution, like, in the Example 16.2, *walk* can be asynchronous action, we can walk and speak, and so are agents. To represent an action to be executed in asynchronous mode, we precede the action with a double exclamation (!!), e.g., !!*walk*. For this, the agent may say hello to Tom, just after it has begun the walk.

Example 16.3 Guarded commands.

```
(guard
     ((?hear "Hello" :from Tom)
      (!speak "Hello" :to Tom) ... )
     ((?see library
             :direction north)
      (!walk :from class_room
             :to library) ... )
(otherwise
      (!send "I am still waiting"
             :to Dickens) ... )))
```
□

Just like the common programming languages, the interaction language *Q* has commands for conditional branching and recursive calls. Apart from this, its has commands, called *guarded commands* for situations that require to observe multiple cues at the same time. A guarded command combines the cues, actions, and forms. After either of the cues becomes true, the guarded command evaluates the corresponding form. If no cue is satisfied, it evaluates the `otherwise` clause, as shown in Example 16.3.

In the above code, if any cue is encountered, e.g., “agent hears hello from Tom,” then corresponding forms will be performed, i.e., agent says (replies) “hello to Tom.” If no cue is observed by the guard command, it performs the `otherwise` clause, and the agent sends the message “I am still waiting” to Dickens.

A collection of state transitions in the language *Q* constitutes a *scenario*. A scenario defines each state as a guarded command, and it can include the conditions. A program writer can draft scenarios in the form of simple state transitions, which can describe fairly complex tasks. The scenarios can be invoked recursively.

16.10 Mobile Agents

All agents are not of the type, mobile agents. An agent sitting at a far off place can communicate with its environment through older time mechanisms, RPC (remote procedure call) and messaging. Such agents are called *stationary agents*, and executes only on the system on which they begin execution (not moving, but stationary). If such agents need information, which is not available on their systems, or they need interaction with an agent (program) residing on other systems, they usually make use of a communication mechanism, such as RPC or messaging.

However, a *mobile agent* is not bound to a particular system on which it begins execution, but it is free to travel to other hosts in the network. Once created in one execution environment, the mobile agent can transport its state (including data and other information), and its code, to other execution environments in the network, when everything is delivered there, it starts execution. The mobile agent is designed with special ability, due to which it can transport itself from one system to another system in the same network. This ability of the agent allows it to move to the system containing an object with whom this agent wants to interact, then to take the advantage of being in the same host or network, as an *object*. There is a number of benefits of using mobile agents rather than doing the same job by remote procedure calls and messaging. Some of the advantages are as follows.

Reduction in network traffic

Distributed systems need communication, involving interactions with multiple destinations to perform a given task. This results to a large traffic in the entire network. The mobile agents permit us to package a conversation and dispatch it to a destination host where interactions take place locally. This gets rid of the flow of raw data in the network. When a large quantity of data is stored at remote hosts, that data is processed locally by the transported mobile agent rather than transferring over the network. That is, computation is moved to the data center.

Overcome network latency

Critical real-time systems, such as nuclear reactors, and robots in manufacturing processes are required to respond to act to changes in their environments. Controlling such systems through a factory network introduces significant latency, due to many reasons, like network being busy. For critical real-time systems, such latency is not tolerable. Mobile agents offer a solution by moving the required programs and state at the place where it is needed.

Encapsulates protocols

When data is exchanged in a distributed environment, each host owns the code, which implements the protocols. However, when protocols change to add new features, it becomes difficult to upgrade the protocol. Since mobile agents can move to remote hosts to establish "channels", this problem does not occur.

Execute asynchronously and autonomously

The mobile devices often need to rely on expensive and fragile network connections. Tasks that require a continuously active connection between a mobile device and fixed network are not economically, as well as technically feasible. To solve this, tasks are dispatched into the network in the form of mobile agents, which can operate at their ease, can move anywhere, where CPU resources and memory are abundant, and can operate asynchronously and autonomously. The devices can reconnect at a later time to collect back the agent.

Adapt dynamically

The mobile agents have capability to sense their execution environment, and can react autonomously to changes. The mobile multiagent system can distribute the agents geographically among the hosts in the network to perform any required task.

Robust and fault-tolerant

Mobile agents' ability to react dynamically to unfavorable situations is useful to build robust and fault-tolerant distributed systems. For example, if a host is being shutdown, all agents executing on that machine are warned and given time to migrate and continue their operation on another host in the network!

16.11 Social Level View of Multiagents

Since intelligence is mainly a social phenomenon, and it is due to the necessity of social life, there is a need to construct socially intelligent systems to understand it, and we need to build social entities to have intelligent systems. The society has adopted a set of social laws, and each agent will be required to obey these laws, and will assume that all other agents also follow the same. These laws, in one hand, constrain the plans available to agents, and on the other hand, will guarantee certain behaviors on the part of other agents. A social law may include communication protocol which leads to rational deals with multiagents. The protocol may also include rationality constraints for cooperation. The social law also, includes the rules, like those that exist for humans for driving—left-drive or right-drive as it may be prevailing [7].

The idea of traffic rules for mobile robots highlights the important aspects of the artificial social system approach. Rather than having a centralized controller or a robot to continuously negotiate in order to avoid collisions, the better approach is that robot should follow the traffic rules, "keep always to the left of the road".

To consider the applicability of social laws, assume that there is a multi-robot network, and we think of laws for agent mobilization in such systems. In such a network, it is assumed that there is coordination among multiagent (i.e., robots). It is formally defined using the following definition.

Definition 16.4 (*Multi-robot network*) A multi-robot network consists of a graph $G = (V, E)$, and set R of robots, and a strictly positive length function $\lambda = E \rightarrow \mathbb{R}$, such that λ associates with each edge (u, v) of G, a distance which robot needs to travel to go from u to v. We assume that there exists a clock such that a robot is at some node or at some point between the nodes, at each point in the time scale. □

The action of a robot (agent) is direction and velocity. The velocity is a number of distance units it passes in a unit time. The direction and velocity are decided by the robot when it is a node in the graph. Also, a robot can observe another robot. The robots need to meet the goals which arrive at them in a dynamic fashion. The goals shall be met without collision to other robots. A collision may take place if they are at the same node at the same point of time, or at the same step distance on edge at the same time point. Based on these facts, it is possible to define some social laws.

Definition 16.5 (*Social law for robot's movement*) Given a graph $G = (V, E)$, the social law for robot's movement determines a subset $A \subseteq E$ of edges in which robot is allowed to move, and restricts the direction of movement along each edge of A, and also restricts the velocity at which robots are allowed to move along each edge $e \in A$. □

In the above definition, the social law is traffic law, which should guarantee that each robot will be able to achieve its goals (say reaching to a destination node), without any collision with other robots. This irrespective of what the other robots do.

Given a multi-robots system, a useful social law is one that guarantees non-collision system, even if all the robots initially enter the graph G at arbitrary nodes, with offset of at least one unit of time from each other, and they obey the social laws. Such a system also guarantees that all the robots will reach their targets ultimately.

Modeling social actions

Design of social laws can be reduced to the problem of finding a route in a graph. For this purpose consider that there is a simple graph $G = (V, E)$, having no cycles, no parallel edges, no cut-vertex, and has at least three vertices. A graph with no cut-vertex is called *block*.

Let $\mathbb{N}$ be the set positive integers, and $f : V \to \mathbb{N}$ be a labeling of vertices in G. For $U \subseteq V$, let $min\ f(U)$ and $max\ f(U)$, respectively, be the smallest and largest labels of two vertices in U. An *f-minimal* vertex in U is any $u \in U$ for which $f(u) = min\ f(U)$, and similarly for *f-maximal*.

The labeling f as above indicates a directed graph $G_f = (V, A)$ on the same vertex set as G, whose edge set $A \subseteq E$ is obtained from E by removing each edge $(u, v) \in E$ with $f(u) = f(v)$. Alternatively, orienting each edge $(u, v) \in E$ with $f(u) < f(v)$, in the direction vu if v is *f-maximal* and u is *f-minimal* in the block containing (u, v), and in the direction (u, v) otherwise.

Given this scenario, we have the following definition for routing a graph.

Definition 16.6 (*Routing graph*) A routing of a graph $G = (V, E)$ is labeling $f : V \to \mathbb{N}$ of its vertices, for which the induced graph G_f is strongly connected. A routing under which there is unique *f-minimal* vertex $r \in V$, shall be called *root*. Assigning the *root* is called *rooting* process. □

The routing of a graph can serve as a basis for useful social laws in multi-robot systems. For this, the robots are required to enter the graph network from an *f-minimal* vertex with an offset of one or more time unit from another. Additionally, the robots are required to move only along the arcs of the graph G_f induced by the routing f. For this, a velocity function is defined as $v : A \to \mathbb{R}$ as follows: for $e = (u, v) \in A$, put $v(e) = \lambda(e)/D$ if u is *f-maximal*, and v is *f-minimal*, and put $v(e) = \lambda(e)/(f(v) - f(u))$ otherwise. It is mandatory that robots should move at the calculated velocities.

16.12 Summary

We call a component an agent if it exhibits a combination of following characteristics: autonomous, adaptable, knowledgeable, mobile, collaborative, persistent. Accordingly, agents are classified as multiagents, autonomous, adaptable, collaborative, proactive, personal, and mobile agents.

Agents have well-defined boundaries and interfaces, they are autonomous and are capable of flexible, autonomous action in that environment in order to meet its design objectives. An important benefit of multiagents is scalability. Since they are inherently modular, it should be easier to add new agents to a multiagent system than to add new capabilities to a monolithic system. They have flexible problem-solving behavior, and they can be reactive or proactive. While parallelism is achieved using multiagents, robustness and scalability are additional benefits.

In all cases of interactions, there are two major differences of agents when they are compared with networked computing and shared computing: 1. agent-oriented interactions take at knowledge level, and 2. operating in an environment that is partially observable, the apparatus should make run-time decisions.

Software agents navigate on the Internet to collect relevant data, perform tasks, and make decisions autonomously. They can transfer an enormous amount of data on behalf of their users. Some agent systems also include standard service agents, such as broker, auctionerr, or community maker.

Cooperation in multiagents is difficult when agents are self-interested, say, everyone tries to download the same file at the same time, the speed will come down. Thus, they need to communicate and cooperate. The cooperative agents should avoid the situation of a prisoner's dilemma. In cooperative agents coalitions need to be formed. There are two approaches for this: (1) optimization-based, which finds an optimal coalition, and (2) game theoretic approach, which has applications in many real-world domains.

When agents are constantly changing coalitions, there is a need of formation of dynamic coalitions. The total possible coalitions turn out to be of the order of $O(n^n)$ for n number of agents. Hence, the number of agents should be small in number. To study the coalitions phenomena, the agents are represented by nodes of a graph and edges by the links indicating coalitions. The agents which are ready to provide the resources are taken as sellers and those receiving are taken as buyers. This becomes a structure to design a coalition algorithm.

Agents approach can also be applied to *software engineering*, where agents are treated as next-generation components and this software engineering as *case-based software engineering*. The complexity issues in software engineering can be tackled through decomposition, abstraction, and organization. Agents can be assigned the task of buying and selling. Software agents are used for filtering information that matches people-to-people with similar interests, and automate the repetitive behavior.

There are theories to model the buying agents, which share the six fundamental stages of buying processes: need identification, product brokering, merchant brokering negotiations, purchase and delivery, product services and evaluation. For modeling agents as decision makers, formal notions of the mental attributes are used, such as belief, knowledge, and references, accordingly, the modeling is called mental level modeling.

When agents function together in cooperative or competitive mode, the multiagent system must provide the abstractions. Instead of providing specific behavior it is designed flexible and can be coded using agent communication languages. The languages have syntax, semantics, and pragmatics.

Many of the agents are mobile, and can just sit at a far place and communicate with its environment, for example, through remote procedure calls or messaging. A mobile agent has a feature that it can partly execute on one system, and can move to another along with data, and can continue to execute the remaining part. Mobile agents reduce network traffic, overcome network latency, encapsulate protocols, execute asynchronously and autonomously, adapt dynamically, and have features of robustness and fault tolerance.

The society has adopted a set of social laws, and each agent will be required to obey these laws and will assume that all other agents also follow the same. These laws, in one hand, constrain the plans available to agents, and on the other hand, will guarantee certain behaviors on the part of other agents.

Exercises

1. Label the following as an agent or not an agent. Explain your reasoning with justification for each.
    a. There is a program on a website to collect answers for a questionnaire.
    b. Google's web crawler, i.e., Googlebot.
    c. A distributed IR (Information Retrieval) program to helps you locate Web documents, you are interested in.
    d. A program operating for a supermarket to automatically locate and bid for the lowest food prices.
    e. A mail-filtering program that removes SPAM messages in your e-mail received in your account.
    f. An Internet-wide multi-user game playing program.
    g. A "chatterbot" program aimed to send messages to chat-rooms and try to fool the people to make them believe that messages are coming from real human beings.
2. In a multiagent system agent interact with the environment. How you can model a situation where one agent modifies the environment and the other perceive it, as a dynamic system?
3. How the architecture of a computer system is different from agent system? Give the salient differences, and justify their significance.
4. A rat searches for food, and at the same time it has to save itself from its predators, and expecting any such it either runs away or hides. For example, a single-agent system model of a rat succeeds in protecting itself from predators as well as in searching the food.
5. Explain the coordination and coalition functions between agents. How they differ from each other.
6. Write the coalition algorithm in your own language.
7. Give an example of evidence of the prevailing use of agents in online buying from the online stores.
8. Give a brief note of agent communication languages and compare them with other high-level languages.

References

1. Brafman RI, Tennenholtz M (1997) Modeling agents as qualitative decision makers. Artif Intell 94:217–268
2. Eiter T et al (1999) Heterogeneous active agents, I: semantics. Artif Intell 108:179–255
3. Griss ML, Pour G (2001) Accelerating development with agent components. Computer 5:37–43
4. Ishida T (2002) Q: a scenario description language for interactive agents. Computer 11:42–47
5. Jennings NR (2000) On agent-based software engineering. Artif Intell 117:277–296
6. Maes P et al (1999) Agents that buy and sell. Commun ACM 42(3):81–91

7. Onn S, Tennenholtz M (1997) Determination of social laws for multi-agent mobilization. Artif Intell 95:155–167
8. Peleteiro A (2014) Fostering cooperation through dynamic coalition formation and partner switching. ACM Trans Auton Adapt Syst 9:1:1–1:31
9. Singh MP (1998) Agent communication languages: rethinking the principles. Computer 12:40–47
10. Stone P, Veloso M (1997) Multiagent systems: a survey from a machine learning perspective. CMU-CS-193 1-37