Chapter 3
Multi-Agent System

3.1 Introduction

Distributed Artificial Intelligence (DAI) is a sub-field of Artificial Intelligence (AI) concerned with situations in which several computer systems interact in order to solve a common problem (Moulin and Chaib-Draa, 1996). The DAI community started forming in the early 1980s. The concept of DAI originates from the real world where many cases are inherently distributed in space, function, knowledge, expertise or information. DAI systems have received considerable attention for two main reasons (Russell and Norvig, 1995).

- They are applicable in many domains, such as distribution, parallelism, and scalability, which cannot be handled by centralised AI systems.
They are in accordance with the insight gained in disciplines such as AI, psychology and sociology that intelligence is tightly and inevitably coupled with interaction.

Traditionally, researches of systems composed of multiple agents were carried out under the banner of DAI, and have been divided into two main camps: a Distributed Problem Solving (DPS) and a Multi-Agent System (MAS). Whereas initially the emphasis of work on MASs was on behaviour coordination; i.e., several agents coordinate their knowledge and activities and reason about the processes of coordination, the emphasis of work on DPS systems was on task decomposition and solution synthesis; i.e., a particular problem is divided among a number of nodes that divide and share knowledge about the problem and the developing solution.

More recently, the term MAS has extended to have a more general meaning, and is now used to refer to all types of systems composed of multiple autonomous components. Nowadays, the MAS constitute one of the most prominent and attractive technologies in Computer Science and in many diverse domains. Research in the MAS is concerned with study, behaviour, and construction of a collection of possibly pre-existing autonomous agents that interact with each other and their environments. Study of the MAS goes beyond the study of individual intelligence to consider, in addition, problem solving that has social components. The advent of the MAS has brought many disciplines in an effort to build distributed, intelligent, and robust applications. These include information retrieval, user interface design, robotics, electronic commerce, computer mediated collaboration, computer games, education and training, and social simulation. The MAS is not only a very promising technology, it is
emerging as a new way of thinking, a conceptual paradigm for analyzing problems, designing systems, dealing with complexity, distribution and interactivity, and perhaps a new perspective on computing and intelligence.

3.2 Concept of Agent

An agent is usually understood as a system or device acting on behalf of another one. Agents that are capable of behaving in an 'intelligent' way are called intelligent agents. The word 'Intelligent' indicates that the agents pursue their goals and execute their tasks such that they optimize some given performance measures. The capacity of an intelligent agent is limited by its knowledge, its computing resources, and its perspective. To say that agents are intelligent does not mean that they are omniscient or omnipotent, nor does it mean that they never fail. Rather, it means that they operate flexibly and rationally in a variety of environmental circumstances, given the information they have and their perceptual and effectual capabilities. Skolicki and Arciszewski (2003) defined the intelligent agent as “an autonomous system situated within an environment. It senses its environment, maintains some knowledge, and learns upon obtaining new data and, finally, it acts in pursuit of its own agenda to achieve its goals, possibly influencing the environment”. Figure 3.1 shows abstract view of an agent in its environment.
Another definition of the agent was given by Wooldridge and Jennings (1995), as "a computer system situated in some environment that is capable of flexible autonomous action in order to meet its design objective". There are four key concepts in this definition:

- **Autonomy**: agents should operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.

- **Social ability**: agents need to be able to interact with other agents (and possibly humans) via some kind of agent-communication language.

- **Reactivity**: agents should be able to perceive their environment and respond in a timely fashion to changes that occur in it.

- **Proactiveness**: agents should not simply act in response to their environment. Also, they should be able to exhibit goal-directed behaviour by taking the initiative.
Nwana (1996) has taken Wooldridge and Jenning's definition and reduces it to three behavioural attributes which are as follows:

- **Autonomy**: this refers to the principle that agents can operate on their own without the need for human guidance. Agents have individual internal states and goals, and act in such a manner as to meet their goals. A key element of their autonomy is their proactiveness, that is, the ability to 'take the initiative' rather than acting simply in response to their environment.

- **Cooperation**: agents need to possess a social ability to cooperate toward achieving some common goals. Cooperation is the ability to interact with other agents and possibly humans via some communication language.

- **Learning**: agents have to be 'smart', they would have to learn as they react and/or interact with their external environment. Agents must often learn in order to dynamically acquire the knowledge and skills necessary to improve their individual performance, precision, efficiency and scope of solvable problems during run-time.

Nwana's requirements for an agent may be shown as a Venn diagram in Figure 3.2. The diagram neatly provides a framework into which types of software agents can be defined.
Two features make agents an attractive technology for modern design systems:

- **Distribution**: Agents are intrinsically distributed. While their local threads can be supported on a single processor, it is also natural to distribute them across a network, supporting the distribution requirements.

- **Modularity**: Modularity of agents makes it natural to encapsulate humans as peer agents to computer processes using a common language and protocols to integrate people and machines. In nature, this integration requires people to reduce the bandwidth of their communication to a level that computerised agents can handle.
3.3 Multi-Agent System

A multi-agent system (MAS) can be defined as a collection of possibly heterogeneous computational entities, having their own problem solving capabilities and which are able to interact among them in order to reach an overall goal. It may also be the case that the MAS is seen as a system revealing a kind of a synergy that would not be expected from the simple sum of its component agents. This synergy is an emergent property of the system as a whole. Durfee and Lesser (1989) have defined a MAS as "a loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver. These problem solvers, often called agents, are autonomous and can be heterogeneous in nature". The autonomy of agents does not imply that agents can do whatever they want independently of the actions of other agents. In other words, the autonomy of agents in the MAS should be somewhat reduced or adjusted in order for the whole system to proceed correctly by disciplining the abstract trajectories that each agent would follow toward the achievement of the task.

An agent-oriented view of the world becomes apparent that most problems require or involve multiple agents; to represent the decentralized nature of the problem, the multiple threads of control, the multiple perspectives, or the competing interests. Jennings et al. (1998) summarise the characteristics of MAS as follow:

- each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint;
- there is no system global control;
Chapter 3. MAS

- data are decentralized; and

- computation is asynchronous.

MASs offer a way to relax the constraints of centralised, planned, sequential control, although not every MAS takes full advantage of this potential. The motivations for the increasing interest in MAS research include the ability of MASs to do the following:

- Solve problems that are too large for a centralized agent to solve because of resource limitations or the risk of having one centralized system that could be a performance bottleneck or could fail at critical times.

- Allow for the interconnection and interoperation of multiple existing legacy systems. To keep pace with changing business needs, legacy systems must periodically be updated. Completely rewriting such software tends to be prohibitively expensive and is often simply impossible. Therefore, the only way that such legacy systems can remain useful is to incorporate them into a wider cooperating agent community in which they can be exploited by other pieces of software. For example, Genesereth and Ketchpel (1994) build an agent wrapper around the software to enable it to interoperate with other systems.

- Provide solutions to problems that can naturally be regarded as a society of autonomous interacting components-agents. For example, in meeting scheduling a scheduling agent that manages the calendar of its user can be regarded as autonomous and interacting with other similar agents that manage calendars of different users (Garrido and
Sycara 1996). Such agents also can be customized to reflect the preferences and constraint of their users.

- Provide solutions that efficiently use information sources that are spatially distributed. Examples of such domains include sensor networks (Corkill and Lesser, 1983) and information gathering from the internet (Sycara et al., 1996).

- Provide solutions in situations where expertise is distributed. Examples of such problems include concurrent engineering and manufacturing (Lewis and Sycara, 1993).

- Enhance the performance along the dimensions of:
  
  - **Computational efficiency**, concurrency of computation is exploited as long as communication is kept minimal.
  
  - **Reliability**, graceful recovery of component failures is done using agents with redundant capabilities or appropriate inter-agent coordination.
  
  - **Extensibility**, the number and the capabilities of the agents working on a problem can be altered.
  
  - **Robustness**, the system’s ability to tolerate uncertainty, because suitable information is exchanged among agents.
  
  - **Maintainability**, a system composed of multiple components-agents is easier to maintain because of its modularity.
  
  - **Responsiveness**, modularity can handle anomalies locally, not propagate them to the whole system.
• **Flexibility**, agents with different abilities can adaptively organize to solve the current problem.

• **Reuse**, functionally specific agents can be reused in different agent teams to solve different problems.

The MAS is the most powerful paradigm for handling complexity, modularity and abstraction. If a problem domain is particularly complex, large, or unpredictable, then the only way it can reasonably be addressed is to develop a number of functionally specific and modular components (agents) that are specialized at solving a particular problem aspect. This decomposition allows each agent to use the most appropriate paradigm for solving its particular problem. Furthermore, real problems involve distributed, open systems. An open system is one in which the structure of the system itself is capable of dynamically changing. The characteristics of such a system are that its components are not known in advance; can change over time; and can consist of highly heterogeneous agents implemented by different people, at different times, with different software tools and techniques. Perhaps the best-known example of a highly open software environment is the internet. The internet can be viewed as a large, distributed information resource, with nodes on the network designed and implemented by different organizations and individuals. In an open environment, information sources, communication links, and agents could appear and disappear unexpectedly. Currently, agents on the internet mostly perform information retrieval and filtering. The next generation of agent technology will perform information gathering in context and sophisticated reasoning in support of user problem-solving tasks. These capabilities require that agents be able to interoperate and coordinate with each other in peer-to-peer interactions. In addition, these capabilities will
allow agents to increase the problem-solving scope of single agents. Such functions will require techniques based on negotiation or cooperation, which lie in the domain of the MAS technology. It is becoming increasingly clear that to be successful, increased research resources and attention should be given to systems consisting of not one but multiple agents.

Learning in a multi-agent environment is complicated by the fact that the environment effectively changes as other agents learn. Enhancing the decision-making abilities of some of the individuals in the system can either improve or severely degrade overall system performance. The MAS requires two levels of learning:

- **Micro-learning**: All agents try to maximize their own profits.
- **Macro-learning**: The system tries to improve its global objective.

However, micro-learning should lead to improvement of the global system objective as well. The aggregate behaviour of the system is the result of the dense interaction of the relatively simple behaviours of the individual agents. The system should keep the best policy found so far.

### 3.4 Agent Based Programming

The concept of agency is being now broadly used not only as a model for computer programming units displaying a certain kind of characteristics but also in more abstract and general way, as a new metaphor for the analysis, specification and implementation of complex software systems. Object
Chapter 3. MAS

oriented programming (OOP) gave software modules local state as well as local code, but innovation was still determined externally, by sending a message. In the OOP, objects are defined as computational entities that encapsulate some state, are able to perform actions, or methods on this state, and communicate by message passing. The OOP techniques are good in general, but are rather low level for intelligent applications. While there are obvious similarities, there are also significant differences between agents and objects:

- **The degree of autonomy**: Agents embody a stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from other agents. Recall that the defining characteristic of OOP is the principle of encapsulation the idea that objects can have control over their own internal state; e.g., in programming languages like Java, we can declare instance variables (and methods) to be private, meaning they are only accessible from within the object. We can of course also declare them public, meaning that they can be accessed from anywhere, and indeed we must do this for methods so that they can be used by other objects. In this way, an object can be thought of as exhibiting autonomy over its state and has control over it. But an object does not exhibit control over its behaviour. That is, if a method is made available for other objects to invoke, then they can do so whenever they wish. The object has no control over whether or not that method is executed. But, in many types of MASs that contain agents built by different organizations or individuals, it cannot be taken for granted that an agent i will execute an action (method) just because another agent j wants it to and may not be in
the interest of i. If j requests i to perform an action, then i may perform the action or it may not. The locus of control with respect to the decision about whether to execute an action is thus different in agent and object systems. In the object case, the decision lies with the object that invokes the method. In the agent case, the decision lies with the agent that receives the request.

- **Behaviour**: Agents are capable of flexible reactive, pro-active, and social behaviours. But the standard OOP model has nothing whatsoever to say about how to build systems that integrate these types of behaviours.

- **Agents have their own thread of control**: The MAS is inherently multi-threaded in that each agent is assumed to have at least one thread of control. In the standard OOP model, there is a single thread of control in the system. Of course, a lot of work has recently been devoted to concurrency in the OOP. For example, the Java language provides built-in constructs for multi-thread programming.

According to (Parunak et al., 1997), Agents add two things to passive objects: a local thread of control and local initiative (usually expressed as local goals). Together, these enable the agents to monitor and respond to their environment autonomously. The benefit of the agent-based system (ABS) would be a reduction of the semantic gap between analysis on the one hand, and design and implementation on the other, leading to a reduction in the time to design and implement, with the usual trade-off between better expandability and losses in execution efficiency and design specificity. Current methodologies emphasise top-down design, but the ABS adopt a different approach in (Wooldridge, 1997); top-down within
the agent and bottom-up in the agent community. In summary, the ABS research can be regarded as developing a way of looking at problems rather than a technology. Hence, the ABS can use OOP programs, expert systems, and distributed computing technologies to implement applications and tool kits that embody this approach. Indeed, OOP methodologies are not directly applicable to the ABS, because typical agents are significantly more complex in both design and behaviour than objects. More importantly, ABS applications require a cooperative knowledge level.

The Agent-Oriented Programming (AOP) paradigm was introduced by Shoham (1993) in his seminal work. Some tools have been developed in recent years to support the implementation of the MAS, but still none are based on a proper AOP language. Still there are no AOP languages used in practice for developing the MAS.

3.5 Agent Toolkits

An interesting list of agent toolkits is available at the AgentLink Web site (http://www.agentlink.org). Some of these toolkits are given as follows:

3.5.1 JACK

JACK is a commercial agent-oriented development environment built on top of and fully integrated with Java. It includes all components of the Java development environment and also offers specific extensions to implement agent behaviour. JACK provides agent-oriented extensions to the Java
programming language whereby source code is first compiled into regular Java code before being executed. In JACK, a system is modelled in terms of agents defined by capabilities, which in turn are defined in terms of plans (set of actions), events, beliefs, and other capabilities.

3.5.2 JADE

Java Agent DEvelopment framework (JADE) is a free software framework fully implemented in the Java language. It allows for the implementation of the MAS through middleware and through a set of graphical tools that supports the debugging and deployment phases. The agent platform can be distributed across machines which do not even need to share the same operating system and the configuration can be controlled via a remote graphical user interface. The configuration can even be changed at run-time by moving agents from one machine to another one when required. JADE can be obtained from the website (http://jade.tilab.com)

3.5.3 ZEUS

ZEUS is an open source software for building a MAS. It has been developed at the British Telecommunication Laboratories. ZEUS is a synthesis of established agent technologies with novel solutions to provide an integrated collaborative agent building environment. ZEUS agents can pursue many goals and can be engaged in a variety of tasks. Details about ZEUS are available at the website (http://labs.bt.com/projects/agents/zeus/)
3.5.4 SWARM

SWARM is the first software tool that was developed at the Santa Fe Institute in 1994 for ABS modelling and simulation. SWARM was specifically designed for artificial life applications and studies of complexity. In the SWARM system, the fundamental component that organises the agents of a model is a “swarm”. A swarm is a collection of agents with a schedule of events over those agents. SWARM supports hierarchical modelling whereby an agent can be composed of swarms of other agents in nested structures. In this case, the higher level agent’s behaviour is defined by the emergent phenomena of the agents inside its swarm. This multi-level model approach offered by SWARM is very powerful. Multiple swarms can be used to model agents that themselves build models of their world. SWARM is available at the website (http://www.swarm.org/).

3.5.5 REPAST

The Recursive Porous Agent Simulation Toolkit (REPAST) was developed at the University of Chicago’s specifically for creating agent based simulations (North et al., 2006). It is very SWARM like, both in philosophy and appearance. REPAST is available at (http://repast.sourceforge.net/). REPAST provides a library of code for creating, running, displaying and collecting data from simulations. There are six modules in the Repast. These modules are the Engine, the Logging Module, the Interactive Run Module, the Batch Run Module, the Adaptive Behaviors Module, and the Domains Module. The Modules of the Repast are shown in Figure 3.3 and explained as follows:
Chapter 3. MAS

Figure 3.3 Repast modules

- **Engine Module**: The Engine Module is a fixed module that is responsible for controlling the activities in a simulation. It contains Engine Controller, Scheduler, Action, and Agent components. Controllers work with Interactive Run and Batch Run components to initiate, start, pause, step, stop, and restart simulation runs. Schedulers are responsible for managing the flow of time in a simulation using Actions. Schedulers are commonly implemented as discrete event clock managers. Agents cause actions to occur by registering them with a Scheduler. Agents are autonomous components that manage their own activities and coordinate with other Agents through one or more Schedulers.

- **Logging Module**: The Logging Module is a fixed module that is responsible for recording simulation results. Two types of logging components, namely Data Loggers and Object Loggers, are required. These types differ based on the complexity of the inputs. Data
Loggers are the simplest logging components. Data Loggers simply record primitive values such as integers, floating point numbers, or strings to a specified location. Object Loggers are more sophisticated than Data Loggers. Object Loggers record the state of full objects or sets of objects, rather than just primitives, to a specified location. Agents can be logged using object loggers since agents are normally implemented as objects with specific properties.

- **Interactive Run Module**: The Interactive Run Module is a fixed module that is responsible for managing simulation runs under the direct control of a user. The components in the Interactive Run Module usually act as intermediaries between users and Engine Module components.

- **Batch Run Module**: The Batch Run Module is a fixed module that is responsible for completing a set of simulation runs without requiring the direct intervention of a user. As with the Interactive Run Module, components in the Batch Run Module usually act as intermediaries between users and Engine Module components.

- **Adaptive Behaviours Module**: The Adaptive Behaviours Module is a flexible module that is responsible for providing adaptive components for implementing agent behaviours. The components can include genetic algorithms, neural networks, other artificial intelligence tools, and regression tools for building agents that can learn and adapt.

- **Domains Module**: The Domains Module is a flexible module that is responsible for providing area-specific functions. In some cases,
these functions may be sophisticated enough to become modules in their own right. Example components in the domains module include tools for general networks, social systems, geographical information systems (GISs), systems dynamics, and computational game theory.

3.6 MAS Applications

MAS technologies are most appropriate for application domains with the following Properties (oliveira et al., 1999):

- **Distribution**: The application domains that are distributed geographically or logically, e.g., distributed entities or distributed knowledge bases.

- **Complexity**: The overall problem, which has to be solved, is in its computational complexity only tractable with heuristic strategies which use local chunks of data or knowledge and which can be easily separated into autonomous problem solving entities.

- **Flexible interaction**: There is no a priori assignment of tasks to problem solvers and there are no fixed problem solving processes.

- **Dynamic environments**: Such environments require responsive and adaptive problem solving entities, e.g., autonomous robots acting in shop floor environment or softbots acting in virtual reality worlds on the Internet.

- **Openness**: In these settings it is not even possible to give a complete specification of the problem that has to be solved. An example of
such a setting is an electronic marketplace, in which a large number of users with different interests interact with each other. We can neither define a global utility function nor declare any user’s objective function as the one the overall system should use. In such settings users can interact in a collaborative or competitive manner.

By structuring such applications as MASs rather than as single agents, the system will have the following advantages:

- Speed-up due to concurrent processing;
- Less communication bandwidth requirements because processing is located nearer the source of information;
- More reliability because of the lack of a single point of failure;
- Improved responsiveness due to processing, sensing and effecting being collocated; and
- Easier development of the system, due to modularity coming from the decomposition into semi-autonomous agents.

The first MAS applications appeared in the mid-1980s and increasingly cover a variety of domains. For a more detailed description of agent-based applications, refer to (Jennings et al., 1998) and (Chaib-Draa, 1995). Some of MAS domain applications are:

- **Business and management**: Organisations have sought to develop a number of IT systems to assist with various aspects of the management of their business processes. Some MAS applications are:
Project adept (Jennings et al., 1996) tackles a business process as a community of negotiating and service providing agents. Each agent represents a distinct role or department in the enterprise and is capable of providing one or more services. Agents who require a service from another agent enter into a negotiation for that service to obtain a mutually acceptable price, time, and degree of quality. Successful negotiations result in binding agreements between agents.

WARREN financial portfolio management system (Sycara et al. 1998) is a MAS that integrates information finding and filtering from the internet in the context of supporting a user manage his/her financial portfolio. The system consists of agents that cooperatively self-organize to monitor and track stock quotes, financial news, financial analyst reports, and company earnings reports to appraise the portfolio owner of the evolving financial picture. The agents not only answer relevant queries but also continuously monitor internet information sources for the occurrence of interesting events (for example, a particular stock has gone up past a threshold) and alert the portfolio manager agent or the user. WARREN also includes agents that analyze user buy and sell decisions with respect to asset allocations and risk.

- **Electronic commerce:** Electronic commerce is rapidly becoming reality; the need for negotiation techniques that take into consideration the complexities of the real world, such as incomplete information, multiple negotiation issues, negotiation deadlines, and the ability to break contracts will be critically needed. Kasbah was developed at MIT's media lab (Chavez and Maes, 1996). Kasbah implements a marketplace allowing
users to create buying and selling agents for each good (books or music) to be purchased or sold respectively. Commercial transactions then take place by means of the interactions of these agents.

- **Telecommunication**: Telecommunication systems are large, distributed networks of interconnected components which need to be monitored and managed in real-time. There are a variety of MAS applications in telecommunications. In one such application (Weihmayer and Velthuijsen, 1994), negotiating agents have been used to tackle the feature interaction problem by utilizing negotiating agents to represent the different entities that are interested in the set up of a call. When conflicts are detected, the agents negotiate with one another to resolve them so that an acceptable call configuration can be established.

- **Scheduling**: Some MAS applications in this area are as follow:

  - Liu and Sycara (1994) have proposed a MAS with a coordination mechanism called Constraint Partition and Coordinated Reaction (CP&CR) for job shop. This system assigns each resource to a resource agent responsible for enforcing capacity constraints on the resource, and each job to a job agent responsible for enforcing temporal precedence and release-date constraints within each job. Moreover, a coordination mechanism called Anchor & Ascend is proposed for distributed constraint optimization. Anchor & Ascend employs an anchor agent to conduct local optimization of its subsolution and interacts with other agents that perform constraint satisfaction through CP&CR to achieve global optimization.
- Distributed airport scheduling (Neiman, 1994), where each arriving plane needs to be assigned not only a gate at which to land but also baggage handlers/trucks to unload and load baggage, and equipment and personnel for servicing the plane (fuelling, cleaning, etc.). If we assume that different concourses of an airport are scheduled by different scheduling agents with their own complement of resources, the need for cooperation (the lending of resources) occurs when there are insufficient resources assigned to a particular concourse given the set of flights that need to be handled during a specified period. This lack of sufficient resources will then delay the scheduled landing and departure times for planes.

- (Basra et al., 2005) applied MAS technology to the real-time scheduling problem of the London Underground, which represents a highly complex, dynamic and unpredictable environment.

**Transportation:** The domain of traffic and transportation management is well suited to an agent-based approach because of its geographically distributed nature. Some of MAS applications in this domain are as follows:

- Distributed vehicle monitoring (DVMT), by (Durfee, 1987), which is a set of geographically distributed agents monitor vehicles that pass through their respective areas, attempt to come up with interpretations of what vehicles are passing through the global area, and track vehicle movements.

- Ljunberg and Lucas (1992) described a sophisticated agent-based air-traffic control system known as OASIS. In this system, which is
undergoing field trials at the Sydney, Australia, airport, agents are used to represent both aircraft and the various air-traffic control systems in operation. As an aircraft enters Sydney airspace, an agent is allocated for it, and the agent is instantiated with the information and goals corresponding to the real-world aircraft. For example, an aircraft might have a goal to land on a certain runway at a certain time. Air-traffic control agents are responsible for managing the system.

- **Manufacturing**: Industrial applications of MAS technology were among the first developed, and have been applied in a wide range of industrial systems. One early application is YAMS system (Yet Another Manufacturing System) by Parunak (1987), which applies the Contract Net Protocol (see above) to manufacturing control. YAMS adopts a MAS approach, where each factory and factory component is represented as an agent. Each agent has a collection of plans, representing its capabilities. The contract net protocol allows tasks (i.e., production orders) to be delegated to individual factories, and from individual factories down to flexible manufacturing systems, and then to individual work cells.

- **Information handling**: The richness and diversity of information available to us via the Internet and World-Wide Web (WWW) represents a very real problem. Information agents have access to multiple, heterogeneous and geographically distributed information sources as in the Internet or corporate Intranets. This includes retrieving, analysing, manipulating, and integrating information available from multiple autonomous information sources. One MAS application in this domain is the minds project by (Corkill and Lesser, 1983), which was a distributed
information retrieval system, in which agents share both knowledge and tasks in order to cooperate in retrieving documents for users. Agents use a blackboard. Blackboard is a sharable data structure to facilitate the communication between the agents.

• **Entertainment:** Electronic entertainment, such as interactive and virtual reality-based computer games have been influenced by MAS technology. e.g., Grand and Cliff (1998) brought together agent technologies and concepts from Biology and designed the marvellous game Creatures. The agents are intended to be sophisticated pets whose development is shaped by their experiences during their lifetime.