CHAPTER 3

AGENT TECHNOLOGY

Agents are a new pattern for developing software applications. More than this, agent-based computing has been hailed as ‘the next significant burst through in software development’ [31], and ‘the new revolution in software’. Currently, agents are the focus of intense interest on the part of many sub-fields of computer science and artificial intelligence. Agents are being used in a more and more wide variety of applications, ranging from comparatively small systems such as email filters to large, open, complex, mission critical systems such as air traffic control.

Artificial Intelligence (AI) and agent systems have been closely related over the last thirty years. AI is concerned in studying the mechanism of intelligence (e.g., the ability to learn, plan) [32], while the study of agents deals with integrating these same mechanism. This distinction may seem to imply that all the problems within AI must be solved in sort to assemble an agent.

3.1 DEFINITION OF AGENT

“An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuator”

An agent can be a person, a machine, a piece of software, or a variety of other things. The basic dictionary definition of agent is one who acts [33]. However, for developing software systems, such a definition is too general: software-related agents need additional properties.
3.2 AGENTS & OBJECTS

There is thick line which creates the difference between objects and agents. Agents are autonomous, flexible and active according to the environment [34], but object cannot show this type of properties.

![Agent Environment Diagram](image)

Figure 3.1 Agent interacting with its environment

3.2.1 Agents and objects - main differences:

Agents are autonomous:

Agents embody stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent.

Agents are smart:

Capable of flexible (reactive, pro-active, social) behavior and the standard object model has nothing to say about such types of behavior.
Agents are active:

A multi-agent system is inherently multi-threaded, in that each agent is assumed to have at least one thread of active control.

### 3.3 AGENT CLASSIFICATION

The main classes of agents are defined as follows:

- Reactive agents
- Collaborative agents
- Interface agents
- Goal based agents
- Mobile agents
- Information-gathering agents
- Multiagent system
- Collaborative agent

Some agents are hybrids, which exhibit properties of more than one of the categories listed above. The eventual aim of most intelligent agent research is to develop smart agents, which would be fully autonomous and able to learn and cooperate with other agents [35]. Smart agents do not yet exist.

#### 3.3.1 Reactive agents

Reactive agent (Reflex agent) is a production system where inputs from the environment are compared with rules to determine which actions to carry out. They simply react to events in their environment according to predetermined rules[36]. A simple example of a reactive agent is the automatic mail filter that many e-mail systems now possess. This mail filter examines each e-mail as it arrives and compares it against a set of rules, or templates, and classifies it accordingly. A common use for such systems is to reject so-called “junk mail” or “spam.” A reactive agent does not tend to perform well when its environment changes or when something happens that it has not been told about.
For example, an e-mail–filtering system might have problems when it receives an e-mail that is entirely in Chinese. New rules can of course be written to deal with such situations, but it might be more desirable to have an agent that can learn to adapt to new situations.

### 3.3.2 Goal-based agents

Goal-based agents are more complex than reactive agents. Rather than following a predetermined set of rules, a goal-based agent acts to try to achieve a goal. This is often done by using search or planning. A goal-based agent might, for example, be given the goal of finding pages on the Internet that are of interest to an Artificial Intelligence researcher. The agent will be designed so that it is capable of carrying out actions (such as loading a web page, examining it, and following links from one web page to another). It is also able to identify when it has reached a goal (for example, by matching the pages it finds against a set of keywords whose presence indicates relevance to Artificial Intelligence).

### 3.3.3 Utility-based agents

A utility-based agent is similar to a goal-based agent, but in addition to attempting to achieve a set of goals, the utility-based agent is also trying to maximize some utility value. The utility value can be thought of as the happiness of the agent, or how successful it is being. It may also take into account how much work the agent needs to do to achieve its goals. Let us return to our example from the previous section of an agent that searches for pages on the Internet that are of interest to Artificial Intelligence researchers. The utility-based agent can use knowledge about the Internet to follow the most worthwhile paths from one page to another. It can use heuristic-based search techniques to minimize the amount of time it spends examining pages that are not of interest and to maximize the likelihood that if an interesting page exists, it will be found with information retrieval techniques.
3.3.4 Interface agents

An interface agent can be thought of as a personal assistant. Interface agents are typically autonomous agents, capable of learning in order to carry out tasks on behalf of a human user. Typically, interface agents collaborate with the user, but do not need to collaborate with other agents; although in some cases, interface agents can learn by seeking advice from other agent. A typical example of an interface agent is a tool that is used to help a user learn to use a new software package. Such an agent has the ability to observe what the user does and make suggestions for better ways to perform those tasks. It is also able to assist the user in carrying out complex tasks, possibly learning as it does so. Interface agents can thus take instructions from users and can also learn from feedback from users about whether they are doing a good job or not, in order to perform better in future.

3.3.5 Mobile agents

Mobile agents are those capable of “moving” from one place to another. In the case of mobile robots, this literally means moving in physical space. In the case of mobile software agents, this mobility usually refers to the Internet or other network. An agent that is not mobile is static. Mobile agents travel from one computer to another, gathering information and performing actions as needed on the basis of that information. A computer virus can be thought of as a form of mobile agent, although most viruses are not intelligent, merely autonomous. That is, they are able to act without being given direct instruction from a human, but they do not adapt intelligently to their surroundings—they simply follow a fixed set of rules that tells them how to infect a computer and how to reproduce.

The main advantages of mobile agents are in efficiency. An agent that has to communicate with a number of remote servers and request large quantities of information in order to make a decision uses a large amount of bandwidth, which can be avoided if the agent is able to physically move to the remote server and query it locally.
3.3.6 Information agents

The Information agents also known as the information-gathering agents, are usually used on the Internet and so are also sometimes called the Internet agents. An information agent is used to help a user find, filter, and classify information from the vast array of the sources available on the Internet. Information agents may be static or mobile. Some information agents are capable of learning, whereas the behavior of others is fixed. Additionally, information agents can be collaborative or can work independently of other agents. The distinctive feature of an information agent is the function that it provides, rather than the way it works.

3.3.7 Multiagent system

Multiagent systems are a common way of exploiting the potential power of agents by combining many agents in one system. Each agent in a multi-agent system has incomplete information and is incapable of solving the entire problem on its own, but combined together, the agents form a system that has sufficient information and ability to solve the problem. The system does not have a centralized control mechanism for solving the problem. An example of how many simple agents can combine together to produce complex behavior can be seen by examining the way that ant colonies function. Each ant has very little intelligence and very little ability to learn. Taken as a whole, however, the ant colony is able to deal with complex situations and in some ways behaves as a single living entity.

3.3.8 Collaborative agent systems

Collaborative agent systems are Multiagent systems in which the agents collaborate with each other to accomplish goals. This property, of cooperating to achieve a common goal, is known as benevolence. Collaborative agents typically do not have the ability to learn, although some have simple learning abilities. As with multiagent systems, the idea is that a combination of many simple agents can solve a problem that each agent individually would not be able to solve.
Collaborative agent systems are able to take advantage of their parallel nature in order to solve problems faster than would otherwise be possible. They are also more reliable than traditional systems because additional agents can be added to provide redundancy: if one agent fails, or provides incorrect information, this will not affect the overall performance of the system because other agents will provide corrective information.

3.4 STRUCTURE OF AGENTS

We have talked about agents by describing behavior - the action that is performed after any sequence of percepts [37]. Now we will have to see how the insides work. The job of AI is to design agent program that implements the agent function mapping percepts in to action. The program will run on same sort of computing device with physical sensors and actuators - we call this the architecture. The agent program is the function that implements the mapping of the percept to the action. The computing device that the program will run on is called the architecture. The program chosen has to be compatible with the architecture. For example, a computer could be the architecture the program could run on. In general, the architecture makes the percepts from the sensors available to the program, runs the program, and feeds the program’s action choices to the effectors as they are generated. As we discuss earlier the relationship among agents, architectures, and programs can be summed up as follows:

\[ \text{Agent} = \text{Architecture} + \text{Program} \]

3.4.1 Agent architectures

Agent architectures are the basic mechanisms underlying the independent components that support effective behaviour in real-world, dynamic and open environments. In fact, initial efforts in the field of agent-based computing focused on the development of intelligent agent architectures, and the early years established several lasting styles of architecture. These range from purely reactive (or behavioural) architectures that operate in a simple stimulus – response fashion, such as those based on the subsumption architecture, at one extreme, to more deliberative architectures that
reason about their actions, such as those based on the belief desire intention (BDI) model, at the other extreme [38]. In between the two lie hybrid combinations of both, or layered architectures, which attempt to involve both reaction and deliberation in an effort to adopt the best of each approach. Thus agent architectures can be divided into four main groups: logic based, reactive, BDI and layered architectures.

Logic-based (symbolic) architectures draw their foundation from traditional knowledge-based systems techniques in which an environment is symbolically represented and manipulated using reasoning mechanisms [39]. The advantage of this approach is that human knowledge is symbolic so encoding is easier, and they can be constructed to be computationally complete, which makes it easier for humans to understand the logic. The disadvantages are that it is difficult to translate the real world into an accurate, adequate symbolic description, and that symbolic representation and manipulation can take considerable time to execute with results are often available too late to be useful.

Reactive architectures implement decision-making as a direct mapping of situation to action and are based on a stimulus – response mechanism triggered by sensor data [40]. Unlike logic-based architectures, they do not have any central symbolic model and therefore do not utilize any complex symbolic reasoning. Probably the best-known reactive architecture is Brooks’s subsumption architecture. The key ideas on which Brooks realized this architecture are that an intelligent behaviour can be generated without explicit representations and abstract reasoning provided by symbolic artificial intelligence techniques and that intelligence is an emergent property of certain complex systems. Subsumption-designed agents perceive conditions and act, but do not plan. The advantage of this approach is that it will perform better in dynamic environments
as well as that they are often simpler in design than logic-based agents. However, the fact that reactive agents do not employ models of their environment results in some disadvantages.

BDI (Belief, desire, intention) architectures are probably the most popular agent architectures. They have their roots in philosophy and offer a logical theory which defines the mental attitudes of belief, desire and intention using a modal logic. One of the most well-known BDI architectures is the Procedural Reasoning System (PRS). This architecture is based on four key data structures: beliefs, desires, intentions and plans, and an interpreter (in Figure 3.3).
In the PRS system, beliefs represent the information an agent has about its environment, which may be incomplete or incorrect. Desires represent the tasks allocated to the agent and so correspond to the objectives, or goals, it should accomplish. Intentions represent desires that the agent has committed to achieving [41]. Finally, plans specify some courses of action that may be followed by an agent in order to achieve its intentions. These four data structures are managed by the agent interpreter which is responsible for updating beliefs from observations made of the environment, generating new desires (tasks) on the basis of new beliefs, and selecting from the set of currently active desires some subset to act as intentions. Finally, the interpreter must select an action to perform on the basis of the agent’s current intentions and procedural knowledge.

Layered (hybrid) architectures allow both reactive and deliberative agent behaviour. To enable this flexibility, subsystems arranged as the layers of a hierarchy are utilized to accommodate both types of agent behaviour. There are two types of control flows within a layered architecture: horizontal and vertical layering. In horizontal layering, the layers are directly connected to the sensory input and action output (in Figure 3.4), which essentially has each layer acting like an agent. The main advantage of
this is the simplicity of design since if the agent needs n different types of behaviours, then the architecture only requires n layers. However, since each layer is in effect an agent, their actions could be inconsistent prompting the need for a mediator function to control the actions.

The vertical layer architecture eliminates some of these issues as the sensory input and action output are each dealt with by at most one layer each (creating no inconsistent action suggestions). The vertical layered architecture can be subdivided into one-pass and two-pass control architectures. In one-pass architectures, control flow from the initial layer that gets data from sensors to the final layer that generates action output. In two-pass architectures, data flow up the sequence of layers and control then flow back down. The main advantage of vertical layered architecture is the interaction between layers is reduced significantly to m 2(n -1). The main disadvantage is that the architecture depends on all layers and is not fault tolerant, so if one layer fails, the entire system fails.

Figure 3.4 Data and control flow in the layered architectures
3.4.2 Abstract architecture for intelligent agents

For formulation the abstract view of agents we will assume that the state of the agent’s environment can be characterized as a set \( S = \{s_1, s_2, s_3, \ldots \} \) of environment states \(^{[42]}\). At any given instant, the environment is assumed to be in one of the states. The effective capability of an agent is assumed to be represented by a set \( A = \{a_1, a_2, a_3, \ldots \} \) of actions. Then abstractly, an agent can be viewed as a function:

\[
\text{Action: } S^* \rightarrow A
\]

Which maps sequences of environments state to actions? We will refer to an agent modeled by a function of this from as a standard agent. The intuition is that an agent decides what action to perform on the heuristics of its history - its experiences to date. These experiences are represented as a sequence of environments states those that the agent has thus far encountered.

The (non-deterministic) behavior of an environment can be modeled as a function:

\[
\text{Env.: } S \times A \rightarrow \omega(S)
\]

Which takes current state of environment \( s \in S \) and an action \( a \in A \) (performed by the agent) and maps them to of environment states envy – those that could result from performing action \( a \) in a state \( s \). If all the sets in the range of \( \omega \) are all singletons, then the environment is deterministic and its behavior can be predicted accurately.

So this can represent the interaction of agent and the environment as history. A history as sequence:

\[
h: S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \cdots \rightarrow S_n \rightarrow S_{n+1}
\]

Where \( s_0 \) is the initial state of the environment, \( a_n \) is the \( n_{th} \) action that agent chose to perform and \( s_n \) is the \( n_{th} \) environment state (which is one of the possible results of executing \( a_{n-1} \) in state \( S_{n-1} \)).
If action: $s^* \rightarrow A$, s an agent, env.: $S \xrightarrow{X} A \xrightarrow{\phi(S)}$ is an environment and so $I$ is the initial state of environment, then the sequence

Let $P$ be a (non-empty) set of percepts. Then see in a function

See: $S \xrightarrow{P}$

Which maps environment state to percepts, and action is now a function

Action: $P^* \xrightarrow{A}$

Suppose we have two environments $S_1 \in S$ and $S_2 \in S$, such that $S_1 \neq S_2$ but see $(S_1) = see (S_2)$, then two environment state $S_1$ and $S_2$ are mapped to the same percept and hence the agent would receive the same perceptual information from different environment states, as for as agent is concerned $S_1$ and $S_2$ are indistinguishable.

The agent program just takes the current percept as input because nothing more is available from the environments. If the agent action depends on the entire percept sequence, the agent will have to remember the percepts.
3.5 ENVIRONMENT

Environments come in numerous flavors [43]. The primary distinctions to be made are as follows:

3.5.1 Accessible (accessible vs. Inaccessible)

If an agent’s sensory apparatus gives it access to the complete state of the environment, then we say that the environment is accessible to that agent. An Environment is effectively accessible if the sensors perceive all aspects that are relevant to the choice of action. An accessible environment is convenient because the agent need not preserve any internal state to keep track of the world.

3.5.2 Deterministic (deterministic vs. Nondeterministic)

If the next state of the environment is completely determined by the current state and the actions selected by the agents, then we say the environment is deterministic. In principle, an agent need not worry about uncertainty in an accessible, deterministic environment. If the environment is inaccessible, however, then it may appear to be nondeterministic. This is particularly true if the environment is complex, making it hard to keep track of all the inaccessible aspects. Thus, it is often better to think of an environment as deterministic or nondeterministic from the point of view of the agent.

3.5.3 Episodic (episodic vs. nonepisodic)

In an episodic environment, the agent’s experience is divided into “episodes.” Each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself, because subsequent episodes do not depend on what actions occur in previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.
3.5.4 Static (static vs. dynamic)

If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static. Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. If the environment does not change with the passage of time but the agent’s performance score does, then we say the environment is semi-dynamic.

3.5.5 Discrete (discrete vs. continuous)

If there are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete. Chess is discrete—there are a fixed number of possible moves on each turn. Taxi driving is continuous—the speed and location of the taxi and the other vehicles sweep through a range of continuous values.