3.1 Introduction

There are many software products, available today, provide varieties of services in different domains. Now a day, in some of the domains, there are increasing demands for computer programs those are interoperable, i.e. capable of exchanging information and services, with other programs. Here, in case of interoperation of programs, the main problem is the heterogeneity of the programs. The main reason behind this heterogeneity is different programmers write in different languages and with different interfaces. So a new technology becomes necessary to make the heterogeneous programs interoperable and hence the concept of Agent Technology evolves. This technology is called Agent-Based Software Engineering and here programs are written as software agent (software components) that communicates with each other by exchanging messages through a communication language.

3.2 Agent Technology

The growth in networked information resources requires information systems that can be distributed on a network and interoperate with other systems. Such systems cannot be easily realized with traditional software technologies because of the limits of these technologies in coping with distribution and interoperability. The agent-based technologies seem be a
promising answer to facilitate the realization of such systems because they were invented to cope with distribution and interoperability. [1]

Agent technology is a relatively new field of applying artificial intelligence (AI) to many practical areas. There is no universally accepted definition of the term agent. Agent is an artificial agent and its works in a software environment. According to Russel and Norvid, an agent as an entity that can be viewed as perceiving its environment through sensors and acting upon its environment through effectors [2]. According to Coen, software agents as programs that engage in dialogs and negotiate and coordinate the transfer of information [3]. An agent is defined as a self-contained software element that acts autonomously on behalf of a user (e.g., person or organization) [4,5]. An agent can also be defined as a self-contained, concurrently executing software process, which encapsulates the current state in terms of knowledge, and is able to communicate with other agents through message passing [6]. Each agent has its own thread of execution, which means that it can perform tasks on its own initiative.

Agents possess the characteristics of delegacy, competency, and amenability. Delegacy means discretionary authority to autonomously act on behalf of the client. Actions include making decisions, committing resources, and performing tasks. Competency means the capability to effectively manipulate the problem domain environment to accomplish the prerequisite tasks. Competency includes specialized communication proficiency. And Amenability is the ability to adapt behaviour to optimize performance in an often non-stationary environment in responsive pursuit
of the goals of the client. Amenability may be combined with accountability [7].

Today most of the agents are fairly fragile and being made for special purposes. Therefore the term "software agent" might best be viewed as an umbrella term that covers a range of other more specific and limited agent types [8]. In general, for the requirements of a particular problem, each agent might possess few attributes like [9,10]:

- Reactivity: the ability to selectively sense and act

- Autonomy: goal-directedness, proactive and self-starting behaviour

- Collaborative behaviour: can work in concert with other agents to achieve a common goal

- "Knowledge-level" [11] communication ability: the ability to communicate with persons and other agents with language more resembling humanlike "speech acts" than typical symbol-level program-to-program protocols

- Inferential capability: can act on abstract task specification using prior knowledge of general goals and preferred methods to achieve flexibility; goes beyond the information given, and may have explicit models of self, user, situation, and/or other agents.

- Temporal continuity: persistence of identity and state over long periods of time.

- Personality: the capability of manifesting the attributes of a "believable" character such as emotion.
• Adaptivity: being able to learn and improve with experience.

• Mobility: being able to migrate in a self-directed way from one host platform to another.

Several researchers have been working for standardizing agent technologies. The works done like, Knowledge Sharing Effort [12], OMG [13], FIPA [14], RETSINA [15], MOLE [16] and ZEUS [17] etc. Some predefined agent models and tools are provided by these development environments, which make the development of Agent-based systems easy.

3.3 Agent Classifications/Types

There are several dimensions to classify existing software agents. Firstly, agents may be classified by their mobility, i.e. by their ability to move around some network. This yields the classes of static or mobile agents. Secondly, they may be classed as either deliberative or reactive. Deliberative agents derive from the deliberative thinking paradigm: the agents possess an internal symbolic, reasoning model and they engage in planning and negotiation in order to achieve coordination with other agents. Thirdly, agents may be classified along several ideal and primary attributes which agents should exhibit. [18]

At various times, researchers of Agent, are characterizing the space of agent types by combining the possible attributes and thus several in the agent research community have proposed various classification schemes and taxonomies.
Figure 3.1 describes intelligent agents in terms of a space defined by the three dimensions of agency, intelligence, and mobility. [19]

"Agency is the degree of autonomy and authority vested in the agent, and can be measured at least qualitatively by the nature of the interaction between the agent and other entities in the system. At a minimum, an agent must run asynchronously. The degree of agency is enhanced if an agent represents a user in some way... A more advanced agent can interact with... data, applications,... services... [or] other agents.

Intelligence is the degree of reasoning and learned behaviour: the agent’s ability to accept the user’s statement of goals and carry out the task delegated to it. At a minimum, there can be some statement of preferences... Higher levels of intelligence include a user model... and reasoning.... Further out on the intelligence scale are systems that learn and..."
adapt to their environment, both in terms of the user's objectives, and in terms of the resources available to the agent...

Mobility is the degree to which agents themselves travel through the network... Mobile scripts may be composed on one machine and shipped to another for execution... [Mobile objects are] transported from machine to machine in the middle of execution, and carrying accumulated state data with them.” [19]

At BT Labs, Nwana identified a minimal set list of three attributes for agent classification and these are [18]:

- Autonomy refers to the principle that agents can operate on their own without the need for human guidance, even though this would sometimes be invaluable. Hence agents have individual internal states and goals, and they act in such a manner as to meet its goals on behalf of its user.

- Cooperation with other agents is paramount: it is the raison d'être for having multiple agents in the first place in contrast to having just one. In order to cooperate, agents need to possess a social ability, i.e. the ability to interact with other agents and possibly humans via some communication language.

- Lastly, for agent systems to be truly 'smart', they would have to learn as they react and/or interact with their external environment. In our view, agents are (or should be) is embodied bits of 'intelligence'. The key
attribute of any intelligent being is its ability to learn. The learning may also take the form of increased performance over time.

Using the above three minimal characteristics, Nwana derives four types of agents, namely - agents, collaborative learning agents, interface agents and truly smart agents.

Figure 3.2: Typology based on Nwana’s (Nwana 1996) primary attribute dimension.

According to Nwana, the classification criteria are not only limited to the above three minimal characteristics but sometimes the roles of the agents also is an important factor for classification. Again sometimes combination of agents’ philosophies also may become a classification factor. Thus agents can be classified in to seven types depending upon their behaviour and action and they are:

- Collaborative agents
- Interface agents
Mobile agents

Information/Internet agents

Reactive agents

Hybrid agents

Smart Agents

3.4 Agent Platform

Now, come to the point of agent’s execution. An agent cannot execute without an environment and thus requires an Agent Platform. So, basically Agent Platform is an execution environment for agents. It facilitates the agent with different services and functionalities, e.g. agent intercommunication, agent autonomy, agent yellow pages, agent mobility, agent cloning etc.

Agent platforms are deployed horizontally over multiple hardware devices (such as PCs, PDAs, cellular phones etc) through containers [20]. Each container is an instance of a virtual machine and it forms a virtual agent network node. On each device at least one container may be set up (but there may be much more, like hundreds of them) [20]. Containers make agent platform independent from the underlying operating systems. Mobile agents are able to migrate from one container to another. Consequently, when containers are deployed on different devices, mobile agents can migrate between different devices.
Agent platforms can be imagined as agent communities, where agents are managed and are given the means to interact (communicate and exchange services) [20]. Many agent communities may coexist at the same time. Depending on the implementation of the platform, agents may be able to leave one community (platform) and join another.

![Agent Platform Diagram](image)

Figure 3.3: Agent Platform (Adapted from Rafal et al. 2008)

Whenever a new technological concept evolves then there must be some standard for using the technology. Same case happens with concept of Agent Technology. Two main organizations namely: Foundation for Intelligent Physical Agents (FIPA) and Object Management Group (OMG) are the pioneer for setting standards for the agent technology concept. There are other standardization processes, like DARPA, 2006, University of Maryland Baltimore County, 2000, related to agent exist but their main focus is on separate methodologies and solutions (which include communication languages, development of semantic web etc).

### 3.5 Agent Life-Cycle

According to FIPA, agents exists on an Agent Platform and they utilize should utilize the different facilities and functionalities provided by the Agent Platform [21]. Also an agent has a physical life cycle and it is to be
managed by the Agent Platform. The life cycle of an agent according to FIPA is depicted in figure 3.4.

![Agent Life-cycle Diagram](image)

Figure 3.4: Agent life-cycle as defined by FIPA (adapted from Bellifemine et.al. 2007)

The state transitions of agents can be described as [21]:

- **Create**: The creation or installation of a new agent.

- **Invoke**: The invocation of a new agent.

- **Destroy**: The forceful termination of an agent. This can only be initiated by the management system component and cannot be ignored by the agent.

- **Quit**: The graceful termination of an agent. This can be ignored by the agent.

- **Suspend**: Puts an agent in a suspended state. This can be initiated by the agent or the management system component.
• **Resume:** Brings the agent from a suspended state. This can only be initiated by the management system component.

• **Wait:** Puts an agent in a waiting state. This can only be initiated by an agent.

• **Wake Up:** Brings the agent from a waiting state. This can only be initiated by the management system component.

• The following two transitions are only used by mobile agents (see [FIPA00005]):

  • **Move:** Puts the agent in a transitory state. This can only be initiated by the agent.

  • **Execute:** Brings the agent from a transitory state. This can only be initiated by the management system component.

### 3.6 Agent Environments

The critical decision an agent faces is determining which action to perform to best satisfy its design objectives. Agent environments are classified based on different properties those can affect the complexity of the agent’s decision-making process. They include [22]:

Accessible vs. inaccessible: An accessible environment is one in which the agent can obtain complete, timely and accurate information about the state of the environment. The more accessible an environment, the less
complicated it is to build agents to operate within it. Most moderately complex environments are inaccessible.

Deterministic vs. non-deterministic: Most reasonably, complex systems are non-deterministic – the state that will result from an action is not guaranteed even when the system is in a similar state before the action is applied. This uncertainty presents a greater challenge to the agent designer.

Episodic vs. non-episodic: In an episodic environment, the actions of an agent depend on a number of discrete episodes with no link between the performance of the agent in different scenarios. This environment is simpler to design since there is no need to reason about interactions between this and future episodes; only the current environment needs to be considered.

Static vs. dynamic: Static environments remain unchanged except for the results produced by the actions of the agent. A dynamic environment has other processes operating on it thereby changing the environment outside the control of the agent. A dynamic environment obviously requires a more complex agent design.

Discrete vs. continuous: If there are a fixed and finite number of actions and percept, then the environment is discrete. A chess game is a discrete environment while driving a taxi is an example of a continuous one.
3.7 Agent Communications

Agent communication means the way of exchanging information between software agents. According to Tim Finin, a software agent is an autonomous, goal directed process that is situated in, is aware of, reacts to its environment and co-operates with other agent to accomplish tasks [23]. Also according to Genesereth, any system that uses an Agent Communication Language (ACL) to exchange information can be termed as a software agent [1].

From the above definitions of software agents, it is possible for an agent to communicate with other agents and react to its environment only when there is an agent communication language. When communication becomes possible between agents then agents’ autonomy is encouraged neglecting their internal structure. Also it encourages grouping of agents to form societies of agents and thus solutions to some complex problems are coming out. So, communication can be further divided into:

- Interaction Protocol, a very essential strategy which governs the interaction with other agents.
- Through Communication Language, the attitude related to the content of the messages exchanged, are communicated.
- Transport Protocol mechanism is necessary for actual transportation for communication with the help of communication language, e.g., TCP, IP, HTTP etc.
3.7.1 Methods of Communication

There are various methods for communication between agents exists; Blackboard Communication Architecture, Message Architecture, Facilitation Architecture are some of them.

In case of Blackboard communication architecture, Figure 3.5, instead of direct communication between agents, a common information space in the system is used to make the information available to all the agents.

In case of Message Architecture, Figure 3.6, direct communication between agents exists. Here, the agent who wants to send message, should specify for whom the message is intended. When the message receives the receiving agent receives it.

Direct communication between agents has some disadvantages, and they are:

- Communication Cost: It basically depends on the community of agents. If it large then broadcasting overheads and processing the request will be on very higher side.

- Complexity: This is in terms of implementation of the agents' communication. Larger the community the implementation will be more complex.
There is another way, called Facilitation Architecture. In this architecture the interaction between agents can be possible only through some programs and these programs are known as Facilitators. An agent uses an Agent Communication Language (ACL) for documenting its needs and its capabilities for its local facilitator. The facilitator(s) use this documentation and transforms it to application level messages. After transformation, these are sent to appropriate places. Here, in this architecture, the facilitators take the full responsibility of the needs of the agents.
3.7.2 Agent Communication Language

As already mentioned, for communication between agents there must be a common communication language which can be termed as Agent Communication Language (ACL). This communication language should be one for removing of inconsistencies which may occur if more than one communication language exists. The researchers from the ARPA Knowledge Sharing Effort had defined the components of an ACL.

ACL has three main parts – a vocabulary, and "inner" language called KIF and an "outer" language called KQML. Typically an ACL is structured in three layers [24]:

1. a common agent communication protocol,
2. a common format for content of communication message and
3. a shared ontology.

There are various Agent Communication Languages. KQML, FIPA-ACL and KQML Lite are some of them those are widely used.

3.7.2.1 Knowledge Query and Manipulation Language (KQML)

Knowledge Query and Manipulation Language (KQML) is a language that is designed to support interactions among intelligent software agents. It was developed by the ARPA supported Knowledge Sharing Effort and separately implemented by several research groups. It has been successfully used to
implement as variety of Information Systems using different software architectures. [25,26]

It is a very high level message-oriented communication language and also protocol information exchange independent of context, syntax and ontology [24]. One major advantage of KQML is its transport method independence, i.e. it is independent of low-level transport methods such as TCP/IP, HTTP, CORBA etc. KQML is also independent of high-level protocols.

3.7.2.2 FIPA-ACL

The Foundation for Intelligent Physical Agents (FIPA) is an international non-profit association of companies and organizations sharing the effort to produce specifications of generic agent technologies. FIPA is envisaged not just as a technology for one application but as generic technologies for different application areas, and not just as independent technologies but as a set of basic technologies that can be integrated by developers to make complex systems with a high degree of interoperability. [27]

However, the use of a common communication language is not enough to easily support interoperability between different agent systems. The standardization work of FIPA is in the direction to allow an easy interoperability between agent systems, because FIPA, beyond the agent communication language, specifies also the key agents necessary for the management of an agent system, the ontology necessary for the interaction between systems, and it defines also the transport level of the protocols. [21]
It specifies a standard messaging language. In general, a messaging language has its encoding specifications, has its own semantics and message pragmatics. This standard does not set out a specific mechanism for the internal transportation of messages. FIPA specifies that the messages should be encoded in textual form as agents running on different platforms use different networking technologies.

Three important features of FIPA ACL are [28]:

1. It is a communication framework which only defines the intention of communication but not the actual content.

2. The formal semantic element of ACL is defined in terms of the feasibility preconditions and the rational effect that allows a communicative act to be scheduled and planned as a normal action. In addition to this, the semantics also allows an agent to consider a message in an explicit manner, if and when needed.

3. It provides the base for the specification of different protocols and common patterns for interaction between agents.

3.8 FIPA Agent Platform Specifications

The Foundation for Intelligent Physical Agents (FIPA) is an international non-profit association of companies and organizations sharing the effort to produce specifications of generic agent technologies [14]. FIPA proposes a set of basic technologies that can be integrated by developers to make complex systems with a high degree of interoperability.
However, the use of a common communication language is not enough to easily support interoperability between different agent systems. The standardization work of FIPA is in the direction to allow an easy interoperability between agent systems, because FIPA, beyond the agent communication language, specifies also the key agents necessary for the management of an agent system, the ontology necessary for the interaction between systems, and it defines also the transport level of the protocols. [28]  

The standard model of an agent platform, as defined by FIPA, is represented in the following figure 3.8:

![Figure 3.8: Reference architecture of a FIPA Agent Platform](image)

The FIPA agent platform reference model consists of the following logical components, each representing a capability set (these can be combined in physical implementations of APs) [28]:
An Agent is the fundamental actor on an Agent Platform which combines one or more service capabilities, as published in a service description, into a unified and integrated execution model. An agent must have at least one owner. Also an agent has a unique identity called Agent Identifier (AID) which can distinguish an agent from other agents within the Agent Universe.

The Agent Management System (AMS) is the agent who exerts supervisory control over access to and use of the Agent Platform. Only one AMS will exist in a single platform. The AMS provides white-page and life-cycle service, maintaining a directory of agent identifiers (AID) and agent state. Each agent must register with an AMS in order to get a valid AID.

A Directory Facilitator (DF) is an optional component of the AP. Actually DF provides the default yellow page service in the platform. So, this service is useful when an agent needs to find out what services are offered by other agents. Though it is optional but if it said to be present in an agent platform then it must be implemented.

There is no restriction for the technology used for the platform implementation but all should be FIPA compliant implementations. The FIPA standard also specifies the Agent Communication Language (ACL). Here, the communication between agents is based on message passing, i.e. agents communicate by formulating message and then sending formulated message to each other.
3.9 Expert System and Its Architecture

Expert system, shortly ES, is a branch of Artificial Intelligence. It is basically useful for solving problems at the level of human expert by extensively using the specialized knowledge. An expert system can be defined as computer program that emulates the decision making ability of a human expert.

"Expert System is an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significance human expertise for their solutions". [29]

The advantages of expert systems over conventional programs are:

(i) Expert Systems usually deal with knowledge,

(ii) Expert Systems have the ability to handle qualitative information,

(iii) using Expert System, the knowledge of multiple experts can be made available while working continuously on a problem at any time,

(iv) the Expert Systems those are web based, can give expertise to the end users at anytime and from anywhere,

(v) Expert Systems can be made to handle factual/heuristic knowledge or both,
(vi) the knowledge base of Expert System can be made more knowledgeable with accumulating experiences.

Most expert systems operate through the user supplying facts and different information to the expert system and in return the user obtains expert advice. The expert system consists of two main internal parts; the knowledge-base which contains the knowledge that helps the inference engine in drawing the related conclusions. These conclusions are considered to be the expert system's responses to the user's queries for expertise. [30]

![Figure 3.9: Basic concepts of an expert system function, Expert Systems “Principle and Programming”, Lotfi Zadeh, (1989)](image)

The basic concepts related to an Expert System and its components are represented in figure 3.9.

A general architecture of and Expert System is shown in figure 3.10 which illustrates all different components related to the expert system architecture.
There are three very important components and they are:

- the knowledge base,
- the context and
- the inference mechanism

Other components those can be part of the expert system architecture are:
- user interface,
- an explanation facility and
Further explanations on the expert system components are:

a) **The knowledge base** is the component of an expert system that contains the facts about the subject which is being dealt with. The facts could be presented in the form of rules and sub-rules. Since knowledge is continually changing and expanding it is considered to be important that the knowledge base is clearly structured and can easily be modified if required to do so.

b) **The context** is the component which is responsible for providing the information on the problem which is being solved.

c) **The inference mechanism** is the part of the expert system which contains the control information. It does that by using the knowledge base to modify and expand the context. The inference mechanism's main task is to relate rules or sub-rules to the facts and execute the most appropriate rules that can satisfy the facts.

d) **The explanation facility** in an expert system varies from a trace of execution to the ability to give the user the reasons behind reaching a particular solution.

e) **The knowledge acquisition facility** in an expert system is the component which is responsible for entering the knowledge to the knowledge base. This facility acts as an editor, and knowledge is
entered directly in a form acceptable to the way in which the expert system was structured.

f) **User interface** of an expert system is the component which is responsible for the communication mechanism between the user and the system.

### 3.10 Expert System Shell

There are two ways for developing an Expert System. One way is to design and develop expert system from scratch. And another way is to design and develop expert system using expert system shells or tools. The first method of designing and development of expert system is very much time consuming and it will cost a lot. In the second method, we get a framework for design and development of expert system.

As already mentioned, Expert System shells are the tools for construction of expert systems which provides knowledge representation facilities and inference mechanisms. Therefore an expert system shell is the software skeleton which provides an inference engine and reasoning techniques and can be customized through user interface to add knowledge of a domain.

![Figure 3.11 Components of Expert System Shell](image-url)
Figure 3.11 represents the generic components of expert system shell which are, knowledge acquisition, knowledge base, reasoning engine, explanation and user interface.

There are various expert system shells available today. Some of the most popular expert system shells are: Automated Reasoning Tool for Information Management (ART-IM), C Language Integrated Production System (CLIPS), Knowledge Engineering System (KES), Level 5, VAXOPS5 and Java Expert System Shell (JESS). In my research work, I have used the Java Expert System Shell (JESS) as the expert system shell.

**Java Expert System Shell (JESS)**

JESS took birth at Sandia National Laboratories in Livermore, CA, in the hands of Ernest Friedman-Hill. It is a rule engine and a scripting environment which is entirely written in Java Language. JESS’s powerful scripting language gives one access to all of Java's APIs. The following figure 3.12 shows the architecture of JESS and its different components.
JESS uses the enhanced version of Rete algorithm. It is basically used for the processing of rules. Using the JESS scripting environment (based on Java), one can create Java objects, call Java methods and also can implement Java interfaces.

REFERENCES:


[26] Intelligent Software Agents: Definitions and Applications by David Wallace Croft, Senior Intelligent Systems Engineer Special Projects Division, Information Technology.
[27] Intelligent Software Agents: Definitions and Applications by David Wallace Croft, Senior Intelligent Systems Engineer Special Projects Division, Information Technology.

