CHAPTER-3
MIGRATING AGENTS AND THEIR SECURITY

3.1 INTRODUCTION

A software agent [17, 101, 102, 103] is a piece of software that acts for a user or other program in a relationship of agency. An agent under consideration is a software entity that shows several degrees of autonomy, since it has to take decisions and to carry out jobs without the direct participation of the user. Often an agent is an active object, i.e., an object with autonomous computational capability. In such situation, the programmers can exploit the characteristics of the object-oriented programming paradigm [104, 105].

In general, agent is an autonomous entity that [17, 103]:-

- acts on behalf of others in an autonomous fashion,
- performs its actions in some level of proactivity and reactivity, and
- exhibits some levels of the key attributes of learning, co-operation and mobility.

Various definitions in relation of mobile agents are [106]:

- The IBM agent, "Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires”.
- The Maes agent, “Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed”.
- General Definition of an agent, "an abstract or physical autonomous entity which performs a given task using information gleaned from its environment to act in a suitable manner so as to complete the task successfully. The agent should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result".
An agent can also be described as “an active object with the ability to perceive, to reason and to act. Moreover, it is assumed that agents possess their own knowledge and can communicate with each other.”

This concept of agents can be extended to the concept of intelligent agents, which have the following additional properties such as **Autonomy**: An agent does not depend on the actions of an external entity (human or other). It can act in an autonomous way and has the control of its behaviour, its state and its actions. **Mobility**: An agent can move and decide to join or not a system or an exchange platform. **Pro-activity**: An agent is able to make decisions and to take initiatives. It works towards personal goals, has its own behaviour depending on its individual objectives. It can achieve complex actions, which can be divided into sub-tasks organized according to its behaviour. **Sociability**: Agents can interact and communicate with others. **Reactivity**: An agent has its own view of the world and it can react to the changes in the environment. **Temporal continuity**: Agents are continuously running processes.

### 3.2 TYPES OF SOFTWARE AGENTS

Agents can be classified according to the actions they perform, their control architecture, the range and effectiveness of their actions, the range of sensitivity of their senses, or how much internal state they posses. There are several dimensions to classify existing software agents [17]. 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 agents. 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. These agents on the contrary do not have any internal, symbolic models of their environment, and they act using a stimulus/response type of behaviour by responding to the present state of the environment in which they are embedded. Thirdly, agents may be classified along several ideal and primary attributes which agents should exhibit. A minimal of three parameters are autonomy, learning and cooperation.
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. A key element of their autonomy is their pro-activeness, i.e. their ability to ‘take the initiative’ rather than acting simply in response to their environment. Cooperation with other agents is paramount; it is the reason 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. Having said this, it is possible for agents to coordinate their actions without cooperation.

Several types of agents identified are [17, 103, 107] collaborative agents, interface agents, information agents, reactive agents, hybrid agents, smart agents and mobile agents, as shown in Figure 3.1.

![Figure 3.1: Various Types of Agents](image)

### 3.2.1 COLLABORATIVE AGENTS

Collaborative agents (see Figure 3.1) emphasise autonomy and cooperation; with other agents in order to perform tasks for their owners. They may learn, but this aspect is not typically a major emphasis of their operation. In order to have a coordinated set up of collaborative agents, they may have to negotiate in order to reach mutually acceptable
agreements on some matters. In brief the key general characteristics of these agents include autonomy, social ability, responsiveness and proactiveness. Hence, they should be able to act rationally and autonomously in open and time-constrained multi-agent environments. They tend to be static, large coarse-grained agents. They may be benevolent, rational, truthful, some combination of these or neither. Typically, most currently implemented collaborative agents do not perform any complex learning, though they may or may not perform limited parametric or rote learning.

The goal of collaborative agents is to interconnect separately developed collaborative agents, thus enabling the ensemble to function beyond the capabilities of any of its members. Implementing efficient ways of cooperation among agents is actually one of the central issues for Multi-Agent Systems development [103].

3.2.2 INTERFACE AGENTS

Interface agents (see Figure 3.1) emphasise autonomy and learning in order to perform tasks for their owners. The key metaphor underlying interface agents is that of a personal assistant who is collaborating with the user in the same work environment. For learning, interface agents learn typically to better assist its user in four ways i.e. by observing and imitating the user (i.e. learning from the user); through receiving positive and negative feedback from the user (learning from the user); by receiving explicit instructions from the user (learning from the user); by asking other agents for advice (i.e. learning from peers).

3.2.3 INFORMATION/INTERNET AGENTS

Information agents (see Figure 3.1) have come about because of the sheer demand for tools to help user manage the explosive growth of information they are experiencing currently, and which they will continue to experience henceforth. Information agents perform the role of managing, manipulating or collating information from many distributed sources.

Interface or collaborative agents started out quite distinct, but with the explosion of the web, there is now a significant degree of overlap. This is inevitable especially since information or internet agents are defined using different criteria. They are defined by
what they do, in contrast to collaborative or interface agents which are defined by what they are. Many of the interface agents are autonomous and learn, but they have been employed in WWW-based roles; hence, they are, in a sense, information agents. One distinction between interface and information agents, however, is that information agents are defined by *what they do*, in contrast to interface agents which are defined by *what they are*.

### 3.2.4 REACTIVE SOFTWARE AGENTS

Reactive agents (see Figure 3.1) represent a special category of agents which do not possess internal, symbolic models of their environments; instead they act/respond in a stimulus-response manner to the present state of the environment in which they are embedded. The agents are relatively simple and they interact with other agents in basic ways. There is no a priori specification (or plan) of the behaviour of the set-up of reactive agents. Secondly, is that of ‘task decomposition’, a reactive agent is viewed as a collection of modules which operate autonomously and are responsible for specific tasks (e.g. sensing, motor control, computations, etc.). Reactive agents tend to operate on representations which are close to raw sensor data, in contrast to the high-level symbolic representations that abound in the other types of agents.

### 3.2.5 HYBRID AGENTS

Hybrid approach (see Figure 3.1) brought together some of the strengths of both the deliberative and reactive paradigms. Hence, hybrid agents refer to those whose constitution is a combination of two or more agent philosophies within a singular agent. These philosophies include a mobile philosophy, an interface agent philosophy, collaborative agent philosophy, etc. The key hypothesis for having hybrid agents or architectures is the belief that, for some application, the benefits accrued from having the combination of philosophies within a singular agent is greater than the gains obtained from the same agent based entirely on a singular philosophy. Otherwise having a hybrid agent or architecture is meaningless. Clearly, the motivation is the expectation that this hypothesis would be proved right; the ideal benefits would be the set union of the benefits of the individual philosophies in the hybrid.
3.2.6 HETEROGENEOUS AGENTS

Heterogeneous agent systems unlike hybrid systems, refers to an integrated set-up of at least two or more agents which belong to two or more different agent classes. A heterogeneous agent system may also contain one or more hybrid agents.

3.2.7 MIGRATING (MOBILE) AGENTS

Migrating agents [108] are computational software processes capable of roaming wide area networks (such as WWW), interacting with foreign hosts, gathering information on behalf of its owner and coming back having performed the duties set by its user [17]. Mobility allows an agent to move, or hop, among agent platforms (as shown in Figure 3.2). The agent platform provides the computational environment in which an agent operates. The platform from which an agent originates is referred to as the home platform, and normally is the most trusted environment for an agent. One or more hosts may comprise an agent platform, and an agent platform may support multiple computational environments, or meeting places, where agents can interact. They may cooperate or communicate with other agents making the location of some of its internal objects and methods known to other agents without necessarily giving all its information away.

![Figure 3.2: Mobile Agent System Model](image)

3.3 MIGRATING AGENTS

Now, after having discussion on various types of agents, let us have a look on related to mobility in migrating agents, their characteristics and challenges.
3.3.1 MOBILITY IN MIGRATING AGENTS

Dany Lange [109] states that mobility is an orthogonal property of agents, that is, not all agents are mobile. An agent can just sit there and communicate with its environment through conventional means, such as remote procedure calling and messaging. A stationary agent executes only on the system on which it begins execution. If it needs information not on that system or needs to interact with an agent on another system, it typically uses a communication mechanism, such as remote procedure calling. In contrast, a mobile agent is not bound to the system on which it begins execution. It is free to travel among the hosts in the network. Created in one execution environment, it can transport its state and code with it to another execution environment in the network, where it resumes execution. The term state typically means the attribute values of the agent that help it determine what to do when it resumes execution at its destination. Code in an object-oriented context means the class code necessary for an agent to execute.

A migrating agent has the unique ability to transport itself from one system in a network to another in the same network. This ability allows it to move to a system containing an object with which it wants to interact and then to take advantage of being in the same host or network as the object. Since computers were connected by links, there was the idea to exploit these connections not only to exchange messages, but also to move entities. Starting from simple data, the mobility has had an evolution that has led to move the execution control, the code and the execution environment (see Figure 3.3).

![Figure 3.3: Various type of Mobility](image-url)
While considering the mobility of more complex entities, such as active objects, one must take into account that they have also a state, composed of the values of the variables. Two kinds of mobility have to be distinguished weak and strong. The former case of mobility permits the migration of the code and of the values of the agent variables. After the migration, the agent is re-started and the values of its variables are restored, but its execution starts from the beginning or from a specific procedure (a method in case of objects). In case of strong mobility, not only code is moved, but also the whole execution state, in order to restart the execution exactly from the point where it was stopped before migration.

Further, the mobility can be explicit or implicit. In the former case, an agent asks explicitly to change its execution environment. In case of implicit mobility, the execution environment hosting the agent decides when to move the agent; such decision can be taken on the base of different needs, such as load balancing, resources retrieving or meetings request.

3.3.2 CHARACTERISTICS OF MIGRATING AGENTS

The key characteristics of mobile agents are as follows [17]:-

- **Migration**: Migration is the function which controls the transfer process. In process migration, migration is normally forced upon a process by the system, due to resource location, load balancing and similar other factors.
- **Data Acquisition**: Mobile agents interrogate their local environment to acquire the information necessary to achieve their goals. This information needs to be filtered locally by the agent before it is either stored with the agent or forwarded to some receiving destination
- **Route Determination**: Once an agent has finished with a network node, it must make a decision of where to move to next. This decision can be derived with various methods like Predetermination, Dynamic Determination or Hybrid Determination.
- **Communication**: The ability for agents to communicate is the fundamental requirement for mobile systems. Two methods for agent communication to take place are Network-oriented agents and Node-oriented agents.
3.3.3 ADVANTAGES OF MIGRATING AGENTS

Various advantages of migrating agents are [17, 109, 110, 111, 112] as follows:-

- **Bandwidth**: Distributed systems often rely on communication protocols that involve multiple interactions to accomplish a given task. This results a lot of network traffic. Mobile agents allow packaging a conversation and dispatching it to a destination host where the interactions can take place locally. Mobile agents are also useful when it comes to reducing the flow of raw data in the network. When very large volumes of data are stored at remote hosts, these data should be processed in the locality of the data rather than transferred over the network. The motto is, move the computations to the data rather than the data to the computations.

- **Latency**: By migrating to the location of the resource, a mobile agent can interact with the resource much faster than from across the network. In critical real-time systems, such as robots in manufacturing processes, need to respond in real time to changes in their environments. Controlling such systems through a factory network of substantial size involves significant latencies. For critical real-time systems, such latencies are not acceptable. Mobile agents offer a solution, because they can be dispatched from a central controller to act locally and execute the directions of the controllers.

- **Asynchronous task execution**: While the agent acts on behalf of the client on a remote site, the client may perform other tasks. Tasks can be embedded into mobile agents, which can then be dispatched into the network. After being dispatched, the mobile agents become independent of the creating process and can operate asynchronously and autonomously. A mobile agent can be delegated to perform a certain task and the agent is intelligent enough to decide, at execution time, with whom it needs to communicate.

- **Fault Tolerance**: The mobile agents are ideally suited for mobile computing in which computers can be disconnected from the network for long periods of time as they do not maintain permanent connections and their state is centralized within themselves. Instead of being online for a longer period, a mobile user may develop an agent request while being disconnected, launch the agent during a brief connection session, and receive back the agent with the result at some later time.
Mobile agents can be used to increase availability of certain services by assigning individual agents for each service. If the client/server, the transaction state is generally divided between the client and server. In case one server is down, the client can resume the situation and resynchronize with the server because the network connection is lost. However, since the mobile agent need not keep the connection permanently, in case of network failure it will continue to run in the node.

- **Peer-to-peer communication:** Mobile agents are considered to be peer entities and as such, can adopt whichever stance is most appropriate to their current needs. A failure in the paradigm case of client/server is the inability of servers to communicate. Mobile agents are considered peer entities and, as such, can act as either client or server is like.

- **Space savings:** Mobile agent code and state do not need permanent storage on the hosts they run on.

- **Reduction in network load/traffic:** Mobile agents are based on the concept of bringing computation to data rather than data to computation. Distributed systems often rely on communication protocols involving multiple interactions to accomplish a given task. The result is a lot of network traffic. Mobile agents allow users to package a conversation and dispatch it to a destination host where interactions take place locally. Mobile agents are also useful when reducing the flow of raw data in the network. When very large volumes of data are stored at remote hosts, that data should be processed in its locality rather than transferred over the network. The motto for agent-based data processing is simple: Move the computation to the data rather than the data to the computation.

- **Interaction with real-time systems:** Mobile agents can be installed close to real-time systems to prevent delays caused by the network congestion.

- **Support for heterogeneous environments:** Mobility frameworks provide the required support for mobile agents, and thus agents are separated from the host and its operating system. Network computing is fundamentally heterogeneous, often from both hardware and software perspectives. Because mobile agents are generally computer- and transport layer-independent (dependent on only their execution environments), they provide optimal conditions for seamless system integration.
Online extensibility of services: Mobile agents can be designed to extend capabilities of applications.

Convenient development paradigm: Mobile agents can be easily used to implement distributed systems.

Client customization: Mobile agents are customized for end-users to carry out specific operations.

Efficiency: If an agent moves through the network to the node where resources reside, then the resulting traffic is reduced since it can pre-process data locally and decide what are the most important information to transfer. It is a crucial aspect for users who are connected by a link of lower bandwidth.

Persistence: When a mobile agent is launched, it is no longer connected to its creator node, and will not be affected if this node fails.

They encapsulate protocols: When data is exchanged in a distributed system, each host owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data. However, as protocols evolve to accommodate new requirements for efficiency or security, it is cumbersome if not impossible to upgrade protocol code properly. As a result, protocols often become a legacy problem. Mobile agents, on the other hand, can move to remote hosts to establish channels based on proprietary protocols.

They adapt dynamically: Mobile agents can sense their execution environment and react autonomously to changes. Multiple mobile agents have the unique ability of distributing themselves among the hosts in the network to maintain the optimal configuration for solving a particular problem. Mobile agents have ability to react dynamically to unfavourable situations and events makes it easier to build robust and fault tolerant distributed systems. If a host is being shutdown, all agents executing on that machine are warned and given time to dispatch and continue their operation on another host in the network.

3.3.4 CHALLENGES IN MIGRATING AGENTS

The major challenges or technical hurdles are the following [113]:-

- Portability and standardization: Nearly all mobile-agent systems allow a program to move freely among heterogeneous machines, e.g., the code is compiled into some
platform-independent representation (like Java bytecodes), and then either compiled into native code upon its arrival at the target machine or executed inside an interpreter. For mobile agents to be widely used, however, the code must be portable across mobile-code systems, since it is unreasonable to expect that the computing community will settle on a single mobile-code system. Making code portable across systems will require a significant standardization effort. The mobile-agent community must take the next step of standardizing on some specific execution environment(s) as well as on the format in which the code and state of a migrating agent are encoded.

- **Transportation:** How does an agent move from place to place? How does it pack up and move?
- **Authentication:** How do users ensure the agent is who it says it is, and that it is representing who it claims to be representing? How do users know it has navigated various networks without being infected by a virus?
- **Secrecy:** How do users ensure that our agents maintain our privacy? How do ensure someone else does not read our personal agent and execute it for his own gains? How do ensure our agent is not killed and its contents ‘core-dumped’?
- **Security:** How do users protect against viruses? How do users prevent an incoming agent from entering an endless loop and consuming all the CPU cycles? It is possible to deploy a mobile-agent system that adequately protects a machine against malicious agents. Numerous challenges remain, like protecting the machines without artificially limiting agent access rights, protecting an agent from malicious machines; and protecting groups of machines that are not under single administrative control. An inadequate solution to any of these three problems will severely limit the use of mobile agents in a truly open environment such as the Internet.
- **Cash:** How will the agent pay for services? How do users ensure that it does not run amok and run up an outrageous bill on our behalf?
- **Performance and scalability issues:** What would be the effect of having hundreds, thousands or millions of such agents on a WAN? Mobile agents save network latency and bandwidth at the expense of higher loads on the service machines, since agents are often written in a (relatively) slow interpreted language for portability and security reasons, and since the agents must be injected into an appropriate
execution environment upon arrival. So, in the absence of network disconnections, mobile agents often take longer to accomplish a task than more traditional implementations, since the time savings from avoiding intermediate network traffic is currently less than the time penalties from slower execution and the migration overhead. Fortunately, significant progress has been made on just-in-time compilation software fault isolation, and other techniques which allow mobile code to execute nearly as fast as natively compiled code.

- **Interoperability/communication/brokering services:** How do users provide brokering/directory type services for locating engines and/or specific services? How do users execute an agent written in one agent language on an agent engine written in another language? How do users publish or subscribe to services, or support broadcasting necessary for some other coordination approaches?

There remain several non-technical issues that may deter the widespread adoption of mobile-agent technology. Internet sites must have a strong motivation to overcome inertia, justify the cost of upgrading their systems, and adopt the technology. The technology can be installed only if it provides substantial improvements to the end-user’s experience for more useful applications, each with fast access to information, support for disconnected operation, and other important features.

- **Lack of a killer application:** The mobile agent paradigm is in many respects a new and powerful programming paradigm, and its use leads to faster performance in many cases. Nonetheless, most particular applications can be implemented just as cleanly and efficiently with a traditional technique, although different techniques would be used for different applications. The advantages of mobile agents are modest when any particular application is considered in isolation.

- **Getting ahead of the evolutionary path:** It is unlikely that any Internet service will be willing to jump directly from existing client-server systems to full mobile-agent systems. Researchers must provide a clear evolutionary path from current systems to mobile-agent systems. In particular, although full mobile-agent systems involve all the same research issues as more restricted mobile-code systems, researchers must be careful to demonstrate that the switch to mobile agents can be made incrementally. As the technologies mature in intranets, site administrators will
become comfortable with them, and their practicality, safety and potential uses will become clear. Then they will find their way into the Internet.

- **Revenue and image:** A final important hurdle is the problem of revenue flow and commercial image. Although it is not yet clear whether advertising is a viable economic foundation for websites, many websites earn money solely from advertisements. If these sites allow mobile agents to easily access the content of the site, the number of human visits to the web pages will presumably decrease, and the advertisements will not be seen. Then, how will the site earn revenue. Similarly, when users are accessing a service with a front-end backed by mobile agents, the distinction between the service and the front-end agents starts to blur. Since the agents will likely be provided by middleware developers, the Internet service will no longer have complete control over its image. A poorly implemented agent may lead to a negative view of the service, even though the service is blameless. The mobile agents can be deployed in the near-term in many applications where the existing services do not rely on advertising; in the long-term, both the Internet and mobile-agent communities will need to explore different revenue models.

### 3.4 TRENDS IN USE OF MIGRATING AGENTS

The trends lead to the conclusion that mobile code, and mobile agents, will be a critical near-term part of the Internet. Not because mobile code makes new applications possible, nor because it leads to dramatically better performance than (combinations of) traditional techniques, but rather because it provides a single, general framework in which distributed, information-oriented applications can be implemented efficiently and easily, with the programming burden spread evenly across information, middleware, and client providers. Several trends leading to mobile agents are [14, 97] information overload, diversified population, bandwidth gap, disconnected operation, customization (re-formatting, filtering, meta search etc) and avoid large transfers. In other words, mobile code gives providers the time and flexibility to provide their users with more useful applications, each with more useful features.

Mohamad Eid et al [110] present a comprehensive study of mobile agent applications. They classify the application fields as Network monitoring and management, information searching and filtering, multimedia, Internet, intrusion detection, telecommunications,
military, and others. They discuss the potential uses of mobile agents in the various fields and present the many systems and architectures that have been proposed and implemented.

Despite the great deal of interest of the research community to effectively merge distributed systems design and code mobility paradigms, a number of problems still exist [110]. Mobile code-based systems seem to be quite complex both to design and to maintain. Security issues, such as protecting the agent from being abused by malicious hosts and protecting the host from being attacked by malicious agents, have to be resolved. Some of the other open issues include the development and standardization of new programming models for Agent Communication Languages (ACL), the integration and interoperability with legacy systems, and the design and standardization of feasible infrastructures and their basic services to carry out complex management tasks.

Jason C. Hung [114] states that Agent technology is one of the most vibrant and active areas of research and development in information technology. Multi-agents and mobile agent technologies are making significant impacts upon almost all aspects of computing discipline. Boldly speaking, they are underpinning the profound changes taking place in our information society. One of the admittedly most successful applications is in global business partnerships, directly due to the nature of the problem. In fact, virtual organizations and e-commerce have become fashionable practices of today's global businesses. At the right time, agents act as key components in the Internet-wide information and e-commerce systems that are currently being developed across the globe, and multi-agents and mobile agents provide most feasible and effective computational frameworks for contemporary business practices. Wireless communication and Mobile networking technology has been received much attention in the last decade. In particular, wireless sensor networks, Bluetooth networks, and wireless ad-hoc networks are deemed as the next generation mobile systems. Research in these systems is emergent and crucial, not only in academic and research community, but also in industrial field. In this decade, the web service researches are very popular. This special issue focuses on the state-of-the-art software technology related to these issues, including multimedia networking, agent technology, web services, Computer-Supported Cooperative Work, Information Systems Development, Data Mining, Mobile and Ubiquitous Computing.
The agent approach use a bandwidth of the network where the agent is mobile, and allow agents to continue to run after leaving a node, even if they lose connection with the node where they were created. It reduces traffic in the network thereby increasing the communication speed, and allows parallel distributed applications. An agent can move on to other machines when necessary and can delegate tasks to other mobile agents in order to achieve real parallel applications. It allows reliability as mobility and autonomy allow the agent to move from one point to another in the network and provide services and meet predefined goals without intervention.

Mobile agent applications in different domains such as network management, electronic commerce, energy efficiency and metering; Wireless Multimedia Sensors, grid computing and grid services, distributed data mining, multimedia, human tracking, security, affective computing, climate environment and weather, e-learning, location, recommendation and semantic web services.

Future work plans mainly concern social energy management. The mobile agents share and compare the data collected in home energy metering sensors with others in an online community. The mobile agent can then alert the user of over consumption of each household appliances and solutions for reducing energy consumption in connection with the use of the green community.

Gujar et al work [115] states that mobile agents are emerging as a promising paradigm for the design and implementation of distributed applications. While mobile agents have generated considerable excitement in the research community, they have not translated into a significant number of real-world applications. One of the main reasons for this is lack of work that quantitatively evaluates the effectiveness of mobile agents versus traditional approaches. Their paper contributes towards such an evaluation. Their paper identifies the underlying mobility patterns of e-commerce applications and discusses possible client-server (CS) and mobile agent (MA) based implementation strategies for each of these patterns.

Vuda Sreenivasa Rao [107] in his study states that data mining technology has emerged as a means for identifying patterns and trends from large quantities of data. The Data Mining technology normally adopts data integration method to generate Data warehouse, on which to gather all data into a central site, and then run an algorithm against that data
to extract the useful Module Prediction and knowledge evaluation. However, a single data-mining technique has not been proven appropriate for every domain and data set. Data mining techniques involving in such complex environment must encounter great dynamics due to changes in the system can affect the overall performance of the system. Agent computing whose aim is to deal with complex systems has revealed opportunities to improve distributed data mining systems in a number of ways. Multi-agent systems (MAS) often deal with complex applications that require distributed problem solving. In many applications the individual and collective behaviour of the agents depends on the observed data from distributed sources.

Distributed data mining is originated from the need of mining over decentralized data sources. The field of Distributed Data Mining (DDM) deals with these challenges in analyzing distributed data and offers many algorithmic solutions to perform different data analysis and mining operations in a fundamentally distributed manner that pays careful attention to the resource constraints. Since multi-agent systems are often distributed and agents have proactive and reactive features which are very useful for Knowledge. Their paper presents the integration of multi-agent system and distributed data mining, also known as multi agent based distributed data mining, in terms of significance, system overview, existing systems, and research trends.

Lisa Zyg [116] in the study states that the mobile agents must decide for themselves the best course of action to take in order to handle users' requests most effectively, with the results of the decision impacting their survival. Currently, these methods are far from being practically implemented since they lack a basic theory of the migration behaviors of mobile agents on a large scale.

Another study [116] has performed a macro dynamics analysis of mobile agents’ migration behaviours, which could provide a fundamental basis for the development of a more ubiquitous future Internet. The study not only favours the design of composite services in a type of self-organizing network architecture, but also benefits the future deployment of an Internet-scale mobile agent system that holds myriads, hosts, and migratory movements of mobile agents. Previous studies have predicted that the future Internet will have vast numbers of these mobile agents that can effectively handle users’ requests. The future Internet will be treated as a global integrated platform for communication, education, entertainment, business, and other human activities. Network
services are required to be highly available, ubiquitous, highly secure, self-managing, and adaptable to dynamic network environments and user requirements. The characteristics of the future Internet user envision resemble the self-organizing and the self-healing properties of natural ecosystems that have evolved over billions of years. The harmonious properties of natural ecosystems have shown us a promising way to build the future integrated platform.

Web information retrieval needs of maintaining the up to date pages in the collection cause a crawler to revisit the websites again and again. Due to this, the resources like CPU cycles, disk space, and network bandwidth, etc., become overloaded and sometime a website may crash due to such overloads. The problem of bandwidth consumption can be reduced by introducing an efficient indexing system based on mobile crawlers [37, 117, 118, 119].

3.5 SECURITY ISSUES IN MIGRATING AGENTS ENVIRONMENT

Security is a fundamental precondition for the acceptance of mobile agent technology. Security issues in migrating agents involve two main components, the agent and its platform. An agent is comprised of the code and state information required to carry out some computation. Mobility allows an agent to move, or hop, among agent platforms. The platform from which an agent originates is referred to as the home platform, and normally is the most trusted environment for an agent. One or more hosts may comprise an agent platform, and an agent platform may support multiple computational environments, or meeting places, where agents can interact.

Since the migrant’s code generated by search engine side transfers and executes on web servers, an environment controlled by another party, it gives rise to several security issues in mobile agent computing. Such issues include authentication, authorization (or access control), intrusion detection etc. Because of mobility of crawling agent, the security problems have become a bottleneck for development and maintenance of mobile agent technology. So, there is a need to develop secured mobile agents and to firm upon certain issues like, maintaining integrity of the network, to be protected from the attacks of other agents and remote platforms.
3.5.1 THREAT CATEGORIES

Wayne [18] identifies four threat categories: threats stemming from an agent attacking an agent platform, an agent platform attacking an agent, an agent attacking another agent on the agent platform, and other entities attacking the agent system. The last category covers the cases of an agent attacking an agent on another agent platform, and of an agent platform attacking another platform, and some more conventional attacks against the underlying operating system of the agent platform.

3.5.1.1 Agent-to-Platform attacks

This category represents the set of threats in which agents exploit security weaknesses of an agent platform or launch attacks against an agent platform. This set of threats includes masquerading, denial of service and unauthorized access.

- **Masquerading:** Masquerading is when an unauthorized agent claims the identity of another agent. The masquerading agent may pose as an authorized agent in an effort to gain access to services and resources to which it is not entitled. The masquerading agent may also pose as another unauthorized agent in an effort to shift the blame for any actions for which it does not want to be held accountable. It may damage the trust the legitimate agent has established in an agent community and its associated reputation.

- **Denial of Service:** The denial of service attacks can be launched intentionally by running attack scripts to exploit system vulnerabilities, or unintentionally through programming errors. Mobile agents can consume an excessive amount of the agent platform's computing resources. The mobile computing paradigm requires an agent platform to accept and execute an agent whose code may have been developed outside its organization and has not been subject to any a priori review. A rogue agent may carry malicious code that is designed to disrupt the services offered by the agent platform, degrade the performance of the platform, extract information for which it has no authorization to access, completely shut down or terminate the agent platform.

- **Unauthorized Access:** Access control mechanisms are used to prevent unauthorized users or processes from accessing services and resources for which they have not been granted permission and privileges as specified by a security
policy. Each agent visiting a platform must be subject to the platform's security policy. An agent that has access to a platform and its services without having the proper authorization can harm other agents and the platform itself. A platform that hosts agents representing various users and organizations must ensure that agents do not have read or write access to data for which they have no authorization, including access to residual data that may be stored in a cache or other temporary storage.

3.5.1.2 Agent-to-Agent

This category represents the set of threats in which agents exploit security weaknesses of other agents or launch attacks against other agents. This set of threats includes masquerading, unauthorized access, denial of service and repudiation. Many agent platform components are also agents themselves. These platform agents provide system-level services such as directory services and inter-platform communication services. Some agent platforms allow direct inter-platform agent-to-agent communication, while others require all incoming and outgoing messages to go through a platform communication agent.

- **Masquerade**: An agent may attempt to disguise its identity in an effort to deceive the agent with which it is communicating. An agent may try to convince another unsuspecting agent to provide it with confidential and private information. Masquerading as another agent harms both the agent that is being deceived and the agent who's identity has been assumed, especially in agent societies where reputation is valued and used as a means to establish trust.

- **Denial of Service**: The agents can also launch denial of service attacks against other agents. Repeatedly sending messages to another agent, or spamming agents with messages, may place undue burden on the message handling routines of the recipient. Agents that are being spammed may choose to block messages from unauthorized agents, but even this task requires some processing by the agent or its communication proxy. If an agent is charged by the number of CPU cycles it consumes on a platform, spamming an agent may cause the spammed agent to have to pay a monetary cost in addition to a performance cost. Malicious agents can also
intentionally distribute false or useless information to prevent other agents from completing their tasks correctly or in a timely manner.

- **Repudiation**: Repudiation occurs when an agent, participating in a transaction or communication, later claims that the transaction or communication never took place. Whether the cause for repudiation is deliberate or accidental, repudiation can lead to serious disputes that may not be easily resolved unless the proper countermeasures are in place. Repudiation often occurs within non-agent systems and real-life business transactions within an organization.

- **Unauthorized Access**: If the agent platform has weak or no control mechanisms in place, an agent can directly interfere with another agent by accessing and modifying the agent's data or code. Modification of an agent’s code is a particularly insidious form of attack, since it can radically change the agent's behaviour. An agent may also gain information about other agents’ activities by using platform services to eavesdrop on their communications.

3.5.1.3 **Platform-to-Agent**

This category represents the set of threats in which platforms compromise the security of agents. This set of threats includes masquerading, denial of service, eavesdropping, and alteration.

- **Masquerade**: One agent platform can masquerade as another platform in an effort to deceive a mobile agent to its true destination and corresponding security domain. An agent platform masquerading as a trusted third party may be able to lure unsuspecting agents to the platform and extract sensitive information from these agents. The masquerading platform can harm both the visiting agent and the platform whose identity it has assumed. An agent that masquerades as another agent can harm other agents only through the messages they exchange and the actions they take as a result of these messages, but a malicious platform that masquerades as an authorized platform can do more harm to the duped agent than a single agent can do on its own.

- **Denial of Service**: When an agent arrives at an agent platform, it expects the platform to execute the agent's requests faithfully, provide fair allocation of resources, and abide by quality of service agreements. A malicious agent platform,
however, may ignore agent service requests, introduce unacceptable delays for critical tasks, simply not execute the agent’s code, or even terminate the agent without notification. An agent can also become live locked if a malicious platform, or programming error, creates a situation

- **Eavesdropping:** It involves the interception and monitoring of secret communications. The agent platform can not only monitor communications, but also can monitor every instruction executed by the agent, all the unencrypted or public data it brings to the platform, and all the subsequent data generated on the platform. Since the platform has access to the agent’s code, state, and data, the visiting agent must be wary of the fact that it may be exposing sensitive information.

- **Alteration:** When an agent arrives at an agent platform it is exposing its code, state, and data to the platform. Since an agent may visit several platforms under various security domains throughout its lifetime, mechanisms must be in place to ensure the integrity of the agent's code, state, and data. A compromised or malicious platform must be prevented from modifying an agent's code, state, or data without being detected. Modification of an agent's code, and thus the subsequent behaviour of the agent on other platforms, can be detected by having the original author digitally sign the agent's code.

The agent platform may be running without the agent's knowledge, and the modified virtual machine may produce erroneous results. A mobile agent that visits several platforms on its itinerary is exposed to a new risk each time it is in transit and each time it is instantiated on a new platform. The party responsible for the malicious alteration of an agent's code, state, or data if not immediately detected may be impossible to track down after the agent has visited other platforms and undergone countless changes of state and data.

### 3.5.1.4 Other-to-Agent Platform

This category represents the set of threats in which external entities, including agents and agent platforms, threaten the security of an agent platform. This set of threats includes masquerading, denial of service, unauthorized access, and copy and replay.
• **Masquerade**: Agents can request platform services both remotely and locally. An agent on a remote platform can masquerade as another agent and request services and resources for which it is not authorized. Agents masquerading as other agents may act in conjunction with a malicious platform to help deceive another remote platform or they may act alone. A remote platform can also masquerade as another platform and mislead unsuspecting platforms or agents about its true identity.

• **Unauthorized Access**: Remote users, processes, and agents may request resources for which they are not authorized. Remote access to the platform and the host machine itself must be carefully protected, since conventional attack scripts freely available on the Internet can be used to subvert the operating system and directly gain control of all resources. Remote administration of the platform's attributes or security policy may be desirable for an administrator that is responsible for several distributed platforms, but allowing remote administration may make the system administrator’s account or session the target of an attack.

• **Denial of Service**: Agent platform services can be accessed both remotely and locally. The agent services offered by the platform and inter-platform communications can be disrupted by common denial of service attacks. Agent platforms are also susceptible to all the conventional denial of service attacks aimed at the underlying operating system or communication protocols.

• **Copy and Replay**: Every time a mobile agent moves from one platform to another it increases its exposure to security threats. A party that intercepts an agent, or agent message, in transit can attempt to copy the agent, or agent message, and clone or retransmit it. The interceptor may copy and replay an agent message or a complete agent.

3.5.2 **SECURITY OBJECTIVES**

Generally a secure software system should meet the following security objectives [120]:-

• **Accountability (Responsibility)**: It requires that users and administrators will be held accountable for behavior that impacts the security of information. Accountability is often an organizational policy requirements that directly supports non-repudiation, deterrence, fault-isolation, intrusion detection and prevention and after- action recovery and legal action [18]. For example in an electronic business
both the user and the online store from where the user (customer) buys products are accountable for their communications and behaviors.

- **Assurance:** Assurance grounds for confidence that other security goals including integrity, availability, confidentiality, and accountability are adequately met by specific implementations [121]. These include functionality that performs correctly, sufficient protection against unintentional errors (by users or software) and sufficient resistance to intentional penetration or by-pass.

- **Authentication:** It requires verifying the identity of a user, process or device before allowing access to resources in a system. Authentication requires that the identity of an entity or the originator of the data can be verified and assured to prevent it from faking or masquerading [122].

- **Authorization (or access control):** This is to grant or to deny access rights to a user, program or process. This objective requires that only legitimate users have rights to use certain services or to access certain resources keeping unauthorized users out. Here, apart from password access, digital signatures are also required to decide whether or not to grant a request to an entity [123].

- **Availability:** It requires that data and system can be accessed by legitimate users within an appropriate period of time [124]. Some attacks like Denial of Service or instability of the system can cause loss of availability.

- **Confidentiality:** It requires that data should be protected from any unauthorized disclosure i.e. data can only be read by persons or machines for which it is intended [25]. A loss of confidentiality hurts data privacy.

- **Integrity:** It is divided into two aspects i.e. data integrity and system integrity. *Data integrity* is the objective that data should not be altered or destroyed in an unauthorized manner to maintain consistency. *System integrity* is the objective that a system should be free from unauthorized manipulation.

- **Non-Repudiation:** It requires that either side of a communication cannot deny the communication later. To achieve this, important communication exchanges should be logged so as to prevent denials by any party of a transaction. It relies on authentication to record the identities of entities [125]. To increase the availability level, a system compromises its confidentiality or integrity levels. Both confidentiality and integrity can effect and also be affected by each other. Based on
them [120], availability and accountability is achieved, and an overall security policy is often preferred.

![Figure 3.4: Relationship among Five Main Security Objectives](image)

3.5.3 SAFETY POLICIES

Ramdoss et al [126] presented various issues regarding safety policies for the agents:

- **Safety Policies**: When downloading software from an unknown or untrusted source, it would be a good idea to ensure that the software is 'safe' to run. This can be done by applying certain restrictions to the untrusted software. For example, a user may want to ensure that the untrusted program will not overwrite critical system data, thereby causing a system crash. One may want to ensure that the program only accesses memory in its own address space and not memory belonging to other applications and processes. One may also want to ensure that the program does not perform illegal disk I/O.

In this context, the definition of 'safe' is known as the application's safety policy. In other words, a safety policy is a set of restrictions placed upon locally run untrusted code to ensure that the program does not behave in a manner that is detrimental to the system or to the system security. At the very least, a safety policy should guarantee the following fundamental safety properties:
**Control Flow Safety:** The program should never jump to and start executing code that lies outside of the program's own code segment. All function calls should be to valid function entry points and function returns should return to the location from where the function was called.

**Memory Safety:** The program should never be allowed to access random locations in memory. The program should only access memory in its own static data segment, live system heap memory that has been explicitly allocated to it and valid stack frames.

**Stack Safety:** The program should only be allowed to access the top of the stack. Access to other areas of the stack should be completely restricted. These three properties, combined, offer the minimum nontrivial level of security for mobile code. More complicated security policies are possible depending on the application.

**Trust:** Security is based on the notion of trust. Basically, software can be divided into two categories, namely the software that is trusted and the software that is not, separated by an imaginary trust boundary. All software on our side of the trust boundary is trusted and is known as the trusted code base.

All security implementations rely on some trusted code. As a result, a trust model of a particular implementation can be made. The trust model basically specifies which code is to be included in the trusted code base and which code lies outside of the trust boundary. At the very least, the trusted code base should include the local operating system kernel, but can also include other items of trusted software like trusted compilers or trusted program runtime environments (e.g. the Java interpreter). It is desirable, however, to keep the trusted code base as small as possible to reduce the security vulnerabilities.

**Performance and Security:** Unfortunately, as it is in most applications, performance is sacrificed for increased security. It would, however, be profitable to have applications that are both secure and perform well at the same time. For this reason, there is huge research going on concerned with resolving the conflict between these concepts in some way.
3.6 SECURITY IN AVAILABLE PLATFORMS

Now, let us have a look on security features of various available mobile agent systems [127], i.e. Agent TCL system, Aglets, Anchor Toolkit system, ARA, Concordia, Grasshopper, HTTP-Based Infrastructure for mobile agents, MAF, Mole, TACOMA, TAgent, Tracy, Trinity, Tryllian and Voyager.

The Agent TCL system [128] is most flexible architecture. It supports state oriented migration, multiple languages and networking protocol. It has a separate security module that prevents an agent from performing malicious actions. Security in Agent TCL is provided in various capacities. To protect migrating agents and to provide authentication, Agent TCL uses Pretty Good Privacy (PGP) for its digital signatures and encryption. To protect resources, a resource manager assigns each agent a set of access permissions. So, when an agent tries to access a resource, the request is sent to the resource manager that checks the agent’s access permissions with the resource. If the agent does not have the proper permission, it is denied access to the resource. To prevent agents from performing malicious acts, each interpreter is extended to include a security module that prevents such acts. Agent TCL has been used in both information-management and information retrieval applications.

Aglets is able to autonomously and spontaneously move from one host to another. It is flexible since it allows user to extend the platform in order to implement new functionalities. It is developed in Java and is compliant to the Java2Security Manager, and is poorly secured and provides basic security mechanisms.

The Anchor toolkit [129] provides strongly secure management of mobile agents. It handles the transmission and secure management of mobile agents in a heterogeneous distributed computing environment. The toolkit protects the agents being dispatched between hosts through encrypted channels. A mobile agent’s host platform is required to sign the agent's persistent state before dispatching the agent to the next platform. The signed persistent state can be used later to detect potential problems with the agent's state. The architecture of the Anchor toolkit consists of an Agent Viewer, Agent APIs, Anchor Server, Anchor Security Manager, Anchor Class Loader, Secure Agent Transfer Protocol handler, Anchor Java Naming and Directory Interface and Anchor Java Native Interface.
Ara [130] is a platform for the portable and secured execution of mobile agents in heterogeneous networks. Ara is concerned with designing of secure and portable execution of mobile agents. Its security model is flexible as domains of protected resources can be dynamically created in the form of places, and the admission of agents to such a domain, as well as their actual rights at that place, can be controlled in a fine grained manner down to individual agents and resources.

The Concordia [131] security model provides support for two types of protection i.e. protection of agents from being tampered with, and protection of server resources from unauthorized access. Concordia uses encryption to protect agents during transfer and, to protect resources on each server, Concordia relies on its security manager component to manage resource protection, which authenticates each agent by verifying its identity. If the identity matches, then the agent is able to access the resource.

Grasshopper [132] provides services like communication service, management service, persistence service, registration service, security service and transport service. Grasshopper supports two security mechanisms i.e. external and internal security, where external security protects remote interactions between the distributed Grasshopper components and internal security protects agency resources from unauthorised access by agents.

The HTTP based infrastructure for mobile agents [133,134] is designed to provide a low-level infrastructure to support agent mobility and communication through the use of HTTP. Here, the agent server must also organise the transfer of mobile agents to other hosts, manage communication between agents and their users, and perform authentication and access validation. Mobile agents that move between agent servers are launched from a home server, which keeps track of the agent’s progress through the network. Upon receiving this agent, the agent server parses the code of the mobile agent and determines whether it is acceptable. If the agent can execute on the site, then the agent server launches it within an appropriate runtime environment and forwards a URL to the home server of the agent to inform the user of its new location.

Mobile Agent Framework (MAF) [106] aims to provide a set of primitives to facilitate the development of distributed mobile agent; second, it strives to meet the application requirement from distributed sensor field which is to provide a light-weight, self-
organized and secure agent platform. Mole [106, 135] provides a stable environment for the development and usage of mobile agents in the area of distributed applications. Mole uses a Sendbox security model in which service agents have access to system resource, providing controlled, secure abstractions of these resources inside the agent system.

In TACOMA (Tromoso And Cornell Moving Agents) [136] an agent is a piece of code that can be installed and executed on a remote computer. Such an agent may explicitly migrate to other hosts in the network during execution. It has mechanism for firewall agents (tag_firewall) and RPC-style communicating agents (tag_rpc). The tag_firewall agent provides an entry point to a site and convenient point to perform such functions as authentication, access control and accounting. Tag_rpc allows client agents to block while they are awaiting results from a server agent.

Travel Agent (TAgent) [112] acts on behalf of their owner without the requirement for the user to interact. It provides an easy service extending an easy agent development platform and a secure design of the agents. It's main focus is on extendibility and flexibility. The TAgent’s platform was designed to be easy to extend. Tracy [106, 137] has a plugin-oriented architecture. Plugins are the software components that can be added dynamically to a running agency, if required, in order to provide high-level services such as inter-agent communication, migration, security etc. Tracy agencies are lightweight and extensible. The platform also offers several migration strategies for agents.

Trinity [112] allows designing wide range of mobile agent. It allows rapid development of non-complex agents. The goal is to provide a tool for rapid development of non-complex agents. Trinity mobile agent framework is an open source software that perform Intelligent Agents tasks. Tryllian [106, 137] is based on a sensing-reasoning action mechanism and allows programmers to define a reactive and proactive behavior of agents. It also provides a persistency service. The main disadvantage of Tryllian is that it does not offer location transparency.

Voyager [112] provides programmer to create state of the art distributed programs quickly and easily while providing a lot of flexibility and extensibility for the products that are being created with the voyager system. Agents are special type of objects in voyager applications and are simply remote objects. Voyager provides location transparency through forwarding chains of proxies.
Some solutions partially achieve integrity by replicating agents and by exploiting replication to compare the results obtained by following different paths. These approaches ensure that destroying or tampering with one or more agent do not compromise the correctness of agent computation because even in case of attacks a still adequate number of agent is expected to return with meaningful results.

Having gone through the available literature, various problems identified related to the agents are as follows:-

- Due to the dynamic nature of the web, it becomes very difficult for a search engine to provide fresh information to the user.
- Since all the web pages do not change with same frequency and user too don not show same level of interest in all web sites, there is a need to improve the revisit frequency of migrating agent as per the level of interest shown by users in various web sites.
- The present search engines are unable to search for the keywords based on sense behind the words. There is a need to prepare an index that searches the related terms based on user perception i.e. the word for which he is intended to search. This will help the search engines to provide better results to the user based on his mental vision.
- The migrating agent goes and executes at remote platform, so there is a need to develop secured migrants and to fix issues like maintaining security and integrity of the agent, data it carries and the remote platform on which it executes.

The next chapter discusses the users interest based revisit frequency model.