CHAPTER 1

INTRODUCTION

1.1 INFORMATION RETRIEVAL

The advent of large wide-area networks like the Internet has caused a vast increase both in the availability and number of information sources. The locations of these information sources are physically distributed, even though it is a requirement for a Distributed System (DS) to hide the fact that the resources are distributed across the network. A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware. It enables the computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility (Coulouris et al 2011). The ability to retrieve information from these distributed and heterogeneous resources offers great promise for obtaining and sharing diverse information conveniently.

Information Retrieval (IR) is the activity of obtaining information when needed from a collection of information resources. The goal of IR is to provide users with those documents that will satisfy their information need (Manning et al 2008). Users have to formulate their information need in a form that can be understood by the retrieval mechanism to identify the potentially relevant documents quickly. However, the multitude, diversity and dynamic nature of on-line information sources make accessing any specific piece of information an extremely difficult task. Information
retrieval tasks are concerned with a particular aspect of IR derived from a user’s point of view. At the same time it should not be confused with the tasks in an IR process, such as query formulation, query expansion, comparison, ranking, and document presentation.

An IR object can support one or more tasks, and a task can be a stand-alone or it can be integrated in a process to perform a larger task. In order to achieve this task, nowadays, there exist a number of approaches. Some of the approaches (Canfora and Cerulo 2004) are as follows:

- **Ad hoc retrieval**: A task that is characterized by an arbitrary subject of search and a short duration.

- **Known search item**: Similar to an ad hoc search, but the target of the search is a particular document or small set of documents, that a searcher knows exists in the collection and wants to find it.

- **Filtering**: Combines the aspects of text retrieval and text categorization.

- **Clustering**: Automatic recognition and generation of categories of entities that can be text documents. It is generally used to improve the retrieval process, because the search can be restricted to a set of interested categories.

- **Browsing**: Instead of searching, exploring the document space, looking for interesting references.

- **Mining**: Process of automatically extracting key information from text documents. The goal is to extract information from a data set and transform it into an understandable structure for further use.
- **Gathering**: An activity involving the pro-active acquisition of information from possibly heterogeneous sources.

- **Crawling**: The activity of selecting new, or updating the existing sources of information that will be processed by successive activities.

However, all the entire information retrieval techniques listed here are based on user queries that implement the client-server architecture (Kumaran and Allan 2006).

### 1.1.1 Client-Server Architecture

The Client-Server (CS) architecture, a best known model for network applications, assigns asymmetric roles to two collaborating processes: the server and the client. The server waits passively for the arrival of requests, which is the typical role of a service provider. On the other hand, the client issues specific requests to the server and awaits its response, as shown in Figure 1.1.

![Figure 1.1 Functional Diagram of Client-server Architecture](image)

**Figure 1.1 Functional Diagram of Client-server Architecture**

Many types of applications have been written using the client-server model. Standard networked functions such as e-mail exchange, web access and database access, are based on the client-server model. For example, a web browser is a client program at the user computer, that may access information at any web server in the world (Liu and Chen 2003).
this architecture, the components that are mobile are the requests from the client and the data from the server. The contents of the request, normally are, the name of the service and some parameters. A connection between the client and the server is essential in this architecture, and this connection has to be maintained throughout the communication process (Coulouris 2011). If the communication is connection-oriented, this architecture is costly in terms of bandwidth usage but less prone to transmission failure. At the same time, the communication is cut once the request is sent by the client in connection-less communication. A fresh communication is established by the server to send the result after processing the received request. As a result, this architecture is more prone to transmission failure (Umar 1993).

The major drawback of client-server systems is single point failure. The only component with the ability to dispense the service is the server. If the server breaks down, then the system stops working. Thus, the functional abstraction created by the client-server architecture also makes it vulnerable to failure. Another drawback is that resources become scarce if there are too many clients. The client-server concept is comparable to a procedure call in programming languages. To offer a convenient use of this client-server concept in programming languages, different programming concepts were developed, like the Remote Procedure Call (RPC) (Schildt 2011), and Remote Method Invocation (RMI) (Java 2002). In order to overcome the difficulties in bulk data transfer through a network, a new technique called code migration is introduced. This technique migrates the code to the data, instead of moving the data through the network.

1.1.2 Code Mobility

In distributed computing, code mobility is the ability of running programs, codes or objects to be migrated (or moved) from one machine or application to another. This is the process of moving a code across the nodes
of a network as opposed to distributed computation, where the data is moved. It is common practice in distributed systems to require the movement of a code or processes between parts of the system, instead of data (Fuggetta et al 1998). In the paradigm of client-server architecture, the remote evaluation method supports code mobility.

In the Remote Evaluation (REV) model, a pattern of CS architecture, the server receives the processing requests from a client like the typical CS architecture as shown in Figure 1.2. In addition, it also includes the whole code needed for the selected operations to be performed on the data stored. Here, only the information that can be actually used and required by the client is sent by the server as a response without any additional overhead. The REV is better than the CS architecture as it is based on the code to data strategy. The benefit of this model is that the data sent in the output is ready for use, and the user needs only negligible additional processing. The drawback is the initial cost, which is higher in comparison with the CS paradigm, as the clients are equipped with minimum processing potentialities. At the same time, the size of the code for the data processing might be higher than a simple retrieval request.

![Figure 1.2 Functional Diagram of the Remote Evaluation Architecture](image)

Figure 1.2 Functional Diagram of the Remote Evaluation Architecture
However, considering the reply stage the amount of data passing through the network is limited in the REV paradigm. Though this paradigm supports code migration, there is always a direct interaction between the client and the server. This means that the code sent by the client returns the data directly to the source. On process completion, the context of program execution is limited to the single host. One of the main drawbacks of this paradigm is the network load.

There was a move towards object oriented CS systems in the late 90s (Umar 1997). However, in the object oriented CS model for distributed computing, communicating systems have to interact through predefined interfaces supported by preprogrammed methods. Each interface has implications for how the resources at each system can be accessed, what information can be exchanged, how the control flow in an application can be distributed over the network, and what connectivity must be maintained between the interacting systems.

The purpose of code mobility is to support sophisticated operations. For example an application can send an object to another machine, and the object can resume executing inside the application on the remote machine, with the same state as it had in the originating application. However, the architectures discussed in this section with or without code mobility lack flexibility and adaptability, the most required characteristics for information retrieval in the present era. To match the rapidly changing and challenging environment, there is a need to develop a new distributed computing model that must possess these characteristics.

In developing new distributed system infrastructures, the next step towards increased flexibility is to communicate programs, not just data. Likewise, when a program is migrated over network, it must adapt to various changes in the network environment. By generalizing this principle,
transmissions can contain pieces of the program code, that can travel around a network and be executed at different nodes. In the field of computer science, these traveling programs are referred to as mobile agents (Genco 2008). By using communicating agents within a distributed system, applications can essentially program the network to suit their particular needs. Hence, mobile agents can serve a variety of purposes in distributed systems. One natural application of mobile agents, therefore, is collecting information from multiple and usually heterogeneous information sources that exist in a distributed environment. By considering the benefit of using a mobile agent in information retrieval, this work presents a mobile agent based system for information retrieval.

1.2 MOBILE AGENT BASED INFORMATION RETRIEVAL

The emergence of agent based systems signals the beginning of one of the most important paradigm shifts in computing, since object oriented methods and CS based distributed systems. The term agent is derived from the concept of agency, referring to employing someone to act on one’s behalf. A human agent represents a person and interacts with others to accomplish a predefined task. In the computer field, an agent represents a code and interacts with other codes to accomplish a task on behalf of the owner, who created that agent. As these codes are just program or instruction, they are also referred to as software agents (Essaaidi and Fortino 2012).

1.2.1 Software Agents

A software agent is a software entity which continuously performs tasks given by a user within a particular restricted environment. The involved software entity can be a computer program, or a software component, or, in object oriented programming language, just a simple object. Software agents are often more clearly understood through their attributes and behavior
(Franklin and Graesser 1996). It is commonly agreed among researchers that every agent exhibits some of the following characteristics (Wooldridge and Jennings 1995).

- **Autonomous**: Agents operate and behave according to a self-made plan, that is generated in accordance with the user-given task. Agents should operate without the intervention of external entities. They typically have control over their actions and internal state.

- **Adaptive**: Agents are characterized by their ability to set up their own goals and the strategy to achieve them. They typically acquire and process information in their environment both spatially and temporarily.

- **Social behavior**: Agents are able to communicate with other agents by means of an agent communication language. Communication can be restricted to pure exchange of information, or can include sophisticated protocols for negotiation.

- **Proactive / Reactive**: Agents do not only react to stimuli from their environment, but they are also able to take the initiative and do pro-active planning.

- **Temporally Continuous**: It is a continually running process.

- **Communicative**: Communicates with other agents.

- **Flexible**: Actions are not scripted.

Software agents that possess these characteristics may fall in one of the following categories (Turban et al 2004), even though there is no agreed taxonomy of categorizing the agents.
- **Collaborative Agents**: Agents in a multi agent system work together to perform a task. This uses the characteristics of autonomy and collaboration.

- **Interface Agents**: They have the learning capability, assist the user in the use of applications, learn user preferences and adjust accordingly.

- **Information Agents**: The main ability of these types of agents is to collect orders, sort and present information from different sources, that tend to be much used agents within the distributed system environment.

- **Intelligent agents**: Mainly studied in the context of robotics, planning and scheduling, and machine learning, to enable computer systems to act on partial and/or inconsistent information.

Apart from this, based on the mobility characteristics, two more of these types are used in the agent community, and are defined as stationary and mobile agents (Fischmeister 2002).

- **Stationary Agents**: The only system that a stationary agent can execute is, where it begins its execution. The purpose of the stationary agent is to provide the requested resources by other resident stationary agents or incoming mobile agents. If the required information is not available in that system or if it needs to interact with the agent of a different system, the stationary agent uses the communication transport mechanism like the RPC.

- **Mobile Agents**: A mobile agent is not bound to the system where it begins execution. A prominent feature of this type of
agent is the unique ability to transport itself from one machine, and execute itself on another machine on either the same or a different platform.

As the information retrieval system presented in this work is mobile agent based, the following section details the various aspects of the mobile agent.

1.2.2 Mobile Agents

In the domain of a distributed operating system, the issues of code mobility have been studied comprehensively in the context of process migration. Process migration deals with the problem of moving a running system level process from one machine to another. This was mostly done with the goal to allow a dynamic reconfiguration of the system. Through this, a process can be moved to another machine if the current server is not able to process it. The basic idea of code mobility is to transfer a package, that contains the code and parameters input by the user, from a sender’s site to a remote receiver’s site to be executed. One of the main differences between a mobile code, such as applets, and a mobile agent, is its itinerary. The general travel plan of a mobile code, usually, is just from point A to point B, whereas, mobile agents have an itinerary and can travel sequentially to many sites. During the execution at the remote server, the code may generate results that can be sent to another server or back to the original sender as shown in Figure 1.3. Mobile agents are the most powerful form of code mobility.

A mobile agent is regarded as an active entity that can migrate autonomously through a computer network and resume execution at a remote site, to access resources required in order to perform a task on behalf of its user (Cockayne et al 1998). It is able to observe its environment and to adapt dynamically to changes during its execution. A mobile agent can continue its
computations independently even if its user is not connected to the network, since both the state and the code of the mobile agent are transferred with it.

![Functional diagram of Mobile Agent Architecture](image)

**Figure 1.3 Functional diagram of Mobile Agent Architecture**

Individually, a mobile agent is a combination of three basic elements, the code, the state and the data. Code is the program that defines the agents’ behavior; state is an agent’s internal variables which enables it to resume its activities after moving to another host; and data is the information that is collected after processing at each remote server (Attiya and Welch 2004). Based on these elements, the function of a mobile code is to transfer the code and that of the mobile object is to transfer the code with the data. Similarly, the mobile process transfers the code, data and thread state, whereas a mobile agent transfers code, data, thread and the authority of its owner.
In a mobile agent system, the mobile agent is migrated with both the code and data. The mobile agent executes the code at the server and migrates to the next server and so on. After visiting all the servers, it returns to the source machine. Based on the state of the code execution at any server, the mobile agent migration is either strong or weak (Braun 2003).

- **In strong migration**, the agent system captures the entire state of the agent. The state and the code of the agent are then transferred to the destination. Once the agent is received at the destination, its state is automatically restored. In this scheme, the capture, transfer and restoration of the complete agent state is done transparently by the underlying agent system. Strong migration might also be a time-consuming operation, since the complete agent state can be large, especially for multithreaded agents.

- **In weak migration**, the agent system does not capture the entire state of the agent, but only its data state. The size of the data state can also be limited by allowing the programmer to select the variables that make up the agent state. Therefore, only the data state and the code of the agent are transferred to the destination. In this scheme, the programmer must encode the agent’s relevant execution states in the program variables. The programmer must also specify a start method, which will decide where the agent should continue its execution after migration. Therefore, although weak migration reduces the amount of information that must be communicated, it puts an additional burden on the programmer and makes agent programs more complex.
1.2.3 Technical Benefits of Mobile Agents

The mobile agent system offers a superior performance over the traditional CS system based on the various criteria, as listed below (Lange and Oshima 1999).

- The Network load is reduced. Multiple interactions are not required between the server and the client, as in traditional approaches. Instead, a mobile agent carries the complete set of interactions to the server. For a large amount of data, transmission over channels is rather expensive; hence the mobile agent that uses the data locally reduces the network load.

- No delays are witnessed as the mobile agent executes at the point of execution.

- Protocols do not need to be enhanced for a mobile agent that is migrating; a mobile agent can utilize the protocols at hand.

- Autonomous execution makes the mobile agent independent of its creator.

- Dynamism is enhanced as a mobile agent can be adapted according to the environment that is given to it.

- Heterogeneity is provided because the underlying specification is independent of the mobile agent, allowing better integration of heterogeneous systems and environments.

- Robustness is achieved, because dynamism allows agents to be configurable as the environment suggests; this also makes them fault tolerant if required.
These advantages lay a foundation for the mobile agent system to be applied specifically in the following domains (Cabri et al 2000).

- Network monitoring and management
- E-Commerce
- Parallel Processing
- Network security and Intrusion Detecting System
- Personal assistance
- E-Health
- Load Balancing
- Distributed Information Retrieval.

1.2.4 Mobile Agent System Architecture

The principal components that are included in the mobile agent system architecture are place, migration, meeting, connection, region, permission and obviously, the mobile agent (White 1997). A mobile agent system with these components is shown in Figure 1.4.

The agent technology models a network of computers as a collection of places. The service to a mobile agent is offered by a place, which the mobile agent enters. In this work, the component place is also referred to as, node, remote server, host, and station. A communicating application can be modeled by the agent technology as a collection of agents. A stationary agent occupies a particular place permanently. However, to occupy several places at various times, a mobile agent can migrate from one place to another. The function of a mobile agent after entering a node is shown in Figure 1.5. This migration concept allows an agent to obtain a service that is offered remotely, and return to the place from where it started.
During the travel, two agents are allowed to meet in a place. Meetings are what motivate the agents to travel. A mobile agent might travel to a place to meet the stationary agent, that provides the service the place offers. Agents belonging to different places shall be allowed to make a connection between them. Agents located in different places can communicate on connection. Connections are often made for the benefit of human users of interactive applications. Agent systems may be grouped in regions. A region represents a security domain where network-wide resources are accessed following a uniform policy.
Mobile agents are about moving codes around in distributed systems, allowing the code to be transported and executed in different locations and carrying out tasks on behalf of a user. Based on this definition, the success of an agent based information retrieval relies on how the following issues (Singh and Ahuja 2012) are handled by the agent system.

- Agent transmission
- Migration strategies
- Mobility issues
- Accommodation
- Security
- Communication
- Management
- Execution platform
- Fault tolerance
If not properly handled, any one or a combination of the issues listed here may lead to blockage, and thus, the failure of the mobile agent. In this context, the work described herein, identifies four specific issues that may result in mobile agent failure, viz, agent migration strategies, accommodation, security and fault tolerance. In order to provide solutions for the problems that may arise due to these issues, this work presents a mobile agent system for information retrieval, which ensures that the mobile agent is not blocked permanently due to the above mentioned issues. That is, the proposed system ensures that the travel of the mobile agent is not affected by these issues, and it successfully migrates back to the dispatcher with the retrieved information.

1.3 MOBILE AGENT MIGRATION STRATEGIES

The initial requirement of a mobile agent in the event of achieving its information retrieval task is the identification of the node(s) to be visited. Once the nodes are identified, determining a travel plan to visit them is an important issue. The sequence of the nodes the mobile agent is expected to visit from the source to the destination, is defined as the itinerary of a mobile agent, which has been widely accepted as a term to refer to a specification or plan of the agent’s movements (Cucurull et al 2010).

An itinerary contains one or many sequences. The basic form of a sequence $S_i$ can be expressed as a tuple,

$$S_i = \text{host}_i + \text{data}_i + \text{action}_i \quad (1.1)$$

More complex forms of sequences can contain several other types of sequence entries recursively, including (Baumann et al 2002):

- An order is a list $[S_1, \ldots, S_n]$ with $n \geq 1$ sequences defining that the nodes ‘$a_i$’ ($1 \leq i < n$) must have been visited prior to ‘$a_{i+1}$’ being visited.
• A set is a list \([S_1, \ldots, S_n]\) specifying that the nodes \(\{S_1, \ldots, S_n\}\) can be handled in any order as long as each element is handled exactly once.

• An alternative is a set \(\langle S_1, \ldots, S_n \rangle\) specifying that exactly one sequence is chosen from the set of sequences \(\{S_1, \ldots, S_n\}\) and this sequence is usually chosen by the agent.

Based on these definitions, the itinerary of a mobile agent is categorized into two types, fixed and flexible. A fixed, otherwise termed as static, itinerary can be expressed as a sequence composed of simple entries. A static itinerary is defined in the source and cannot be modified by the agent’s execution (Genco 2008).

The size and behavior of the network changes dynamically, much faster than the agent-owner’s perception. The mobile agent needs to develop its ability to construct and proactively adapt its own sequence, and this type of migration is termed as a flexible itinerary (Baek et al 2002). A flexible, also termed as dynamic, itinerary can result in any of the following cases (Xu and Qi 2006).

• The servers to be visited are predetermined by the agent’s owner but the agent is free to select the next node; hence, it is the combination of sets (containing several sequences).

• All the servers are known to the agent’s owner, but the agent is free to decide which \(m\) out of \(n\) nodes to visit and in what order. This could be a combination of different types of entries.

• All the servers may not be known to the agent’s owner at the agent’s departure. In this case, the agent might be able to migrate to a yellow pages server, where it will search for its
next destination; or the yellow pages server can provide a list of candidate servers, upon which the agent is able to select the optimal server for migration.

The agent based IR system presented in this work uses a dynamic itinerary. The main issue associated with the dynamic itinerary is the selection of an optimal node to be visited next. The selection criteria may vary, depending on the application and requirement of the user. However, it must be ensured that, for a given application the dynamic itinerary strategy used by the mobile agent should fetch a node that is better as per the user’s requirement. In the present work, this requirement includes the nature of information, its relevance, and the status of the node. The initial focus of this work is, in a direction that achieves this requirement. On arrival at a node after the selection of an optimal one, in order to achieve the task, the mobile agent is required to be provided with the necessary resources by the hosting node. Among the various resources, the initial resource required by the mobile agent for its accommodation is memory.

1.4 MOBILE AGENT ACCOMMODATION

A mobile agent is initially accommodated in memory before further processing. For this, it is essential for a node to be available with sufficient memory. In the agent based system, a node is expected to allocate a specified amount of memory to accommodate and process the incoming mobile agents. This allotted memory may get exhausted due to various reasons. Some reasons that are related to this work are:

- Excessive accommodation of mobile agents
- Excessive number of threads created by the mobile agents
- Excessive number of unattended or unintentionally duplicated agents
Each mobile agent occupies some amount of heap memory. The heap memory gets exhausted, if the node accommodates the mobile agent in excess. Similarly, for each process, the operating system allows a limited number of threads depending on the thread stack size and heap memory. For example, whenever a copy method is called in an agent system, a new instance of agent is created with the same state but with a new identity. Each of this type of agent occupies considerable amount of memory. This process may be repeated intentionally and due to this, the allotted memory is completely occupied by these duplicated agents. As a result, the node is not able to allow either the creation or reception of new agents. Likewise, an agent can create any number of threads. If the number exceeds the limit, then also the node may face an “out of memory” problem. Finally, the number of mobile agents increases if the unattended or unintentionally duplicated mobile agents are not disposed of. Allocated memory is released only after an agent is explicitly disposed. This is another situation in which the proposed system has a role to play. The mobile agent system presented in this work focuses on the problem of the non-availability of memory, that arises due to the reasons mentioned here. The system is equipped with a solution to this problem, and thus ensures that more number of mobile agents is accommodated in a node for execution.

The accommodated mobile agents are executed once they are given permission to execute in the agent execution environment. However, in order to get the permission, the mobile agents are subjected to security checks by the hosting platform. Similarly, the host is also subjected to a security check by the mobile agent to protect it from the host.

1.5 MOBILE AGENT SECURITY

As the Mobile Agent (MA) operates in the distributed environment, it is vulnerable to various security attacks. In particular, during their interaction, the MA and its platform pose some security problems. These attacks are primarily focused on the communications capability of the platform, to exploit the potential vulnerabilities.
1.5.1 Security Requirements

The users of mobile agent frameworks have four main security requirements, the same as the users of the networked computer systems. They are termed as confidentiality, integrity, availability, and accountability.

- **Confidentiality**: Any private data stored on a platform or carried by an agent must remain confidential. Agent frameworks must be able to ensure that their intra and inter-platform communications remain confidential. Monitoring the message flow between two entities may allow other entities to infer useful information without having access to the actual message content (Wayner 1995).

- **Integrity**: The agent platform must protect agents from unauthorized modifications of their code, state, and data, and ensure that only authorized agents or processes carry out any modification of shared data. The agent itself cannot prevent a malicious agent platform from tampering with its code, state, or data, but the agent can take measures to detect this tampering (Jain et al 2011).

- **Accountability**: Each process, human user, or agent on a given platform must be held accountable for their actions. In order to be held accountable each process, human user, or agent must be uniquely identified, authenticated, and audited.

- **Availability**: The agent platform must be able to ensure the availability of both data and services to local and remote agents. In addition, it must be able to detect and recover from system failures. While the platform can provide some level of fault-tolerance and fault-recovery, agents may be required to
assume responsibility for their own fault-recovery (Jain et al 2011). The agent platform must be able to handle the requests of hundreds or thousands of visiting and remote agents, or risk creating an unintentional denial of service.

The categories of attacks with regard to a mobile agent system are given as, a host attacking the incoming mobile agents, a mobile agent attacking other mobile agents of the same platform, a mobile agent attacking the hosting platform, and a host attacking other hosts through mobile agents.

1.5.2 Host Attacking Mobile Agent

The host provides the necessary execution environment for the visiting mobile agents, and for this reason, it has almost complete control over the mobile agent. This provides the host with the necessary opportunities to initiate several types of security attacks on the mobile agent and thus become malicious.

One of the ways in which an agent can be attacked by its host is by having its privacy invaded (Paracha 2009). This is generally performed with the aim of gathering or stealing information in order to benefit from the agents’ operations (unauthorized access). A host that could easily corrupt an agent by deleting or altering the agent’s code, data, flow control or status is a threat to data integrity (Montanari et al 2001). The host can also compromise the agent’s integrity (Jansen et al 1999), by interfering with its intended mission or responding falsely to requests for information or services, or diverting them to another platform not on the agent’s itinerary (denial of service). In the worst case, the host may even kidnap the mobile agent and prevent it from ever returning to its home machine (repudiation), or create its own similar agent for use in an unfair manner, by making use of the reverse engineering principles (Ssekibuule 2010). A host can act as another host
(masquerade) in an effort to lure an unsuspecting mobile agent to it, in order to extract sensitive information from these agents (Ma and Tsai 2008). The masquerading platform can harm both the visiting agent and the host whose identity it has assumed. The cloning of an agent together with masquerading creates an authentication problem (Li et al 2004).

1.5.3 Agent Attacking other Agent

In many mobile agent system architectures, system-level services such as directory services and inter-platform communication services are provided by components, that are themselves agents (stationary agents). A visiting mobile agent, in search of a particular service could expose itself to various forms of attack from these stationary agents or other mobile agents sharing the same execution environment. For example, a stationary agent could delay or prevent a visiting agent from accessing one or more resources or services available at the host (Borselius 2002). An agent can also intentionally distribute false or useless information to prevent other agents from completing their tasks correctly or in time (Greenberg et al 1998). In the process of direct communication between the mobile agents, an agent may attempt to disguise its identity in an effort to deceive the other agent into releasing sensitive information to which it is not entitled (Tomasek et al 2005). Another form of attack which one agent can perform on another agent is repudiation (Jansen 2000). Repudiation occurs when an agent participating in a transaction or communication later claims no knowledge of it ever having taken place. An example of this attack is the tailgating attack.

1.5.4 Agent Attacking Host

A mobile agent paradigm requires an agent host to accept and execute an agent. Without adequate defense mechanisms, an agent host is vulnerable to attacks from incoming agents. In order to provide adequate
security for the host, it is essential to know the types of attack that can be executed against the host.

A mobile agent can change or destroy resources or services by modifying, reconfiguring, or erasing them from memory or disk. When a mobile agent deliberately damages a mobile agent host, it could potentially damage all other mobile agents executing there at the time. An agent can block one of the processes by overloading its buffers to create a deadlock, and hence, preventing its use (Paracha 2009). Masquerading is yet another form of attack that an agent may execute on the host. The masquerading agent may assume the identity of another agent in an effort to gain access to services and resources to which it is not entitled.

1.5.5 Host Attacking other Host

Assuming that the agents on a host are well behaved, it is possible for a host to be attacked from other entities, such as another host in the system, in an attempt to disrupt, harm, or subvert the agent system. A host may attack the inter-agent or inter-host communication through intercept, forgery or replay.

Among the various attacks that are possible in a mobile agent system, this work focuses on an attack that is performed by a mobile agent on another mobile agent. In turn, these infected mobile agents may harm the host at a later stage. In this context, the two security attacks identified for this work are tailgating and eavesdropping. Tailgating is an attack that is performed by a Malicious Mobile Agent (MMA) on the Legitimate Mobile Agent (LMA) that is executing in the platform and thus creates a threat to the platform. An attacking agent appends itself with the LMA, and tries to gain the access to the agent execution platform, either by attacking or by injecting itself. After gaining access, the MMA detaches itself from the legitimate ones
and starts functioning on the platform. For instance, it can disrupt the normal execution of the remote platforms by launching the denial of service attacks, that consume more system resources to preclude other agents from accessing the host’s services.

Eavesdropping is an attack that can be performed by the MMA. In a flexible itinerary, the address of the next server to visit is obtained from the directory service. If this address is known to others, then it is easy for the attackers to know about the itinerary of the mobile agent and the servers that provide the information. Using this information, the action of the LMA can be repeated by the malicious ones.

In the view of the IR system presented in this work, another of its tasks is to ensure that the travelling mobile agent is not blocked due to the tailgating or eavesdropping at a node. If a mobile agent is not infected by these attacks and withstands the security check by the host, then it is permitted to enter the agent’s execution environment, in which the actual execution of the mobile agent takes place. As part of the execution, it is essential for an agent to communicate with the stationary agent or other mobile agents of that host. A mobile agent is able to interact, cooperate or coordinate with other agents of the system by using the communication principles of the agent based system.

1.6 MOBILE AGENT COMMUNICATION

It has been realized, that communication between mobile agents is not as simple as the common communication method in a computer network. To identify key components of a mobile agent system and to facilitate discussions on issues surrounding agent communication, the use of an agent communication model has been made (Hidayat 2011; Charu 2012). For the mobile agent system presented in this work, the model as presented by Charu
(2012) has been used. Based on this model, agent communication can be broadly divided into three categories.

### 1.6.1 Mobile Agent – Mobile Agent

Each mobile agent has its own plans, which it must initiate and control, in order to fulfil its needs and goals. Agents collocated at a place use this opportunity to provide references to each other, that could be used to invoke the desired operations. The communication patterns that may occur in this type of interaction are peer-to-peer and are not limited to only request/response (Cabri et al 2001). This type of communication is termed as intra place communication as the agents are located in the same place. The communication mechanisms used by most mobile agent systems are based on the RPC or message passing or streams as well (Moreau 2001). For mobile agents that are not located at the same place, a common approach to communicate (inter place communication) among agents takes place through the exchange of messages. The messages may be synchronous or asynchronous.

### 1.6.2 Mobile Agent – Service Agent

The style of the service agent’s interaction is typically client-server, in which the service agents provide operations or methods that can be requested by other agents, since the service agents are representatives of services in the agent world. For this purpose an RPC-like communication mechanism (intra place) is incorporated in the agent system. If the mobile agent system is developed and used exclusively in a Java environment, the RMI is used as the communication mechanism. To locate or enquire about a particular service, the mobile agent uses inter place pattern communication. Direct message passing is the predominant mechanism for this purpose (Peine 2002).
1.6.3 Mobile Agent – Group

The previous two types of communication assumes that the communicating partners know each other when communication takes place, i.e., the sender of a message or RPC is able to identify the recipient. However, there are situations, where a sender does not know the identities of the agents that are interested in the sent message. For example, a given task is to be performed by a group of agents, each agent taking over a subtask. In order to perform their subtasks, the agents themselves may dynamically create subgroups of agents. For an agent to communicate with all the subgroups of the same location, the agent-to-group intra-place communication is used. Although there has been a lot of research conducted between agents and agent groups with respect to inter-place communication, from the extensive literature survey conducted, it was not possible to conclude if there are any mechanisms implemented that provide support for this category of communication.

The various communication strategies presented in the previous sections play a major role in accomplishing the tasks of mobile agents. Besides, the success of mobile agents in task completion depends not only on the communication mechanism, but also on the network and node reliability. As the mobile agent system operates in an open environment, it is subject to failure, due to the possibilities of network component failure. It must be ensured that the failure of the network components does not affect the mobile agent migration. The mobile agent system presented in this work focuses on the failure of a mobile agent due to node failure. In this context, it must be ensured that the mobile agent reaches the destination successfully with the information retrieved during its travel. For this, the mobile agent system must be designed in such a way that the mobile agent is capable of tolerating faults.
1.7 MOBILE AGENT FAULT TOLERANCE

Mobile agent migration, communication and processing revolve around the distributed system environment. Any failure that is possible in a distributed system affects the mobile agent execution also. To avoid this, making a distributed system reliable is very important. To know about the possible failure situations of a mobile agent, it is essential to know about the failure situations of the distributed system.

1.7.1 Fault Models in Distributed System

The failure of a distributed system can result in anything from easily reparable errors to catastrophic meltdowns. A failure occurs when an actual running system deviates from its specified behavior. The cause of a failure is called an error. An error represents an invalid system state, one that is not allowed by the system behavior specification. The error itself is the result of a defect in the system or fault. In other words, a fault is the root cause of a failure. That means that an error is merely the symptom of a fault. A fault may not necessarily result in an error, but the same fault may result in multiple errors. Similarly, a single error may lead to multiple failures (Kumar et al 2011).

Based on duration, faults can be classified as transient or permanent. A transient fault will eventually disappear without any apparent intervention, whereas a permanent one will remain unless it is removed by some external agency. A particularly problematic type of transient fault is the intermittent fault that recurs, often unpredictably. This can be the most annoying of component faults (Laprie et al 2004). The fault tolerant mechanism presented in this thesis addresses the intermittent faults. Based on how a failed component behaves after failure, faults can be classified into the following categories (Jalote 1994).
- **Crash faults**: The component either completely stops operating or never returns to a valid state;

- **Omission faults**: The component completely fails to perform its service;

- **Timing faults**: The component does not complete its service on time;

### 1.7.2 Fault Handling Approaches

One way to avoid these faults in the distributed system is to design systems that minimize the presence of faults. The following methods (Attiya et al 2004) may be used to rectify these faults.

- **Fault Avoidance**

- **Fault Removal**

- **Fault Tolerance**

Fault avoidance is a process that ensures that the system avoids being faulty in the first place by analyzing the design and validation steps. This can include formal validation, code inspection, testing, and using robust hardware. Fault removal is an approach that detects and removes the faults from the system. This is done by testing, debugging, and verification, as well as replacing the failed components with better ones. Fault tolerance is the realization that the faults are always present (or its potential) in the system (Koren and Krishna 2007). Here, the system must be designed in such a way that it will be fault tolerant. That is, the system should compensate for the faults and continue to function. A fault-tolerant system is one that continues to provide the required functionality in the presence of faults. In other words, fault tolerance is the ability of the system to provide a service, even in the presence of errors. In the context of the work presented in this thesis, fault
tolerance means that the ability of the system to recover the mobile agents with data from failures.

1.7.3 Fault Tolerant System

The general approach to building a fault tolerant system is redundancy. Redundancy may be applied at several levels (Cristian 1991) like physical redundancy, time redundancy and information redundancy.

Physical redundancy deals with devices, not data. Extra equipment is added to enable the system to tolerate the loss of some failed components. Redundant Array of Independent Disks (RAID) and backup name servers are examples of physical redundancy. Time redundancy achieves fault tolerance by performing an operation several times. Timeouts and retransmissions in reliable point-to-point and group communication are examples of time redundancy. This form of redundancy is useful in the presence of transient or intermittent faults. It is of no use in the case of permanent faults. An example is protocol’s retransmission of packets. Information redundancy seeks to provide fault tolerance through replicating or coding the data. In the replication based technique, we have replicas of a task running on different machines, and as long as not all replicated tasks crash (i.e. host crash etc), chances are that the task execution would succeed.

For a mobile agent system to achieve fault tolerance for IR, this work combines the benefits of time and information redundancy, and is applied for failure recovery. That is, to recover a failed mobile agent, the mobile agent with its data is replicated, and is kept in different machines to achieve fault tolerance. The main benefits of the replication of data can be classified as,
- Performance enhancement
- Reliability enhancement
- Data closer to client
- Share workload
- Increased availability
- Increased fault tolerance

Though information redundancy provides fault tolerance through data replication, there exist various issues in achieving this task. For example, in order to keep data consistency, the system is required to ensure a satisfactorily consistent appearance for clients. The second one is where to place the replicas, and how are updates propagated? Another important issue is scalability. When applied to the mobile agent, these fault tolerance issues are addressed in this work based on the replication strategy.

1.7.4 Fault Tolerant Mobile Agents

Generally, the fault-tolerance of a computer system or component is designed in such a way that, in the event of a component failure, a backup component or procedure can immediately take its place with no loss of service. That is, the machine, equipment or system has the ability to recover from a catastrophic failure, without disrupting its operations (Jalote 1994). By the term fault tolerance, this work means guaranteeing the continuous execution of mobile agents even in the case of node or server failure. In a mobile agent system, normally, fault tolerance is achieved through replication or checkpointing mechanisms.
Replication resolves the problem of blocking (Pleisch and Schiper 2001). However, maintaining many physical servers and the consistency of data among them is difficult. Also, in an asynchronous system like the Internet, it is very difficult to detect whether the agent has actually failed or it is slow (Fischer et al 1985). When replication prevents blocking, it may lead to the multiple execution of agents, (i.e.) the violation of the exactly-once execution property.

Checkpointing is relatively a more popular fault tolerant approach used in distributed systems, where the state of the application is stored periodically in reliable and stable storage (Park 2010). In the case of a problem during execution, i.e. after a crash, the application is restarted from the last checkpoint rather than from the beginning (Chowdhury and Neogy 2009). Uncoordinated checkpointing (Elnozahy et al 2002) presents a good failure-free performance, but since processes checkpoint independently, there exists the possibility of creating useless checkpoints. Periodic checkpointing earns the advantage of resuming the execution of agents from the last saved state, when failure occurs. However, excessive checkpointing results in performance degradation, and too infrequent checkpointing results in too much recomputation overhead.

Some other approaches drawn in this category have been proposed by Yang et al (2008). The common idea of these approaches is to share some information among the agent replicas. Mobile agents, which are now part of some larger application, have to co-operate with each other, in order to achieve the desired result. It is assumed that the shared information is protected, as long as no critical number of agent executors collaborates with one another.
1.7.5 Failure Recovery

Once failure has occurred in many cases, it is important to recover critical processes to a known state in order to resume processing. There are two approaches for failure recovery in distributed environments (Gupta et al 2008).

- In **backward error recovery**, the system is restored to a previous known valid state. This often requires checkpointing the system state, and once an error is detected, rolling back the system state to the last checkpointed state. Clearly, this can be a very expensive capability. Not only is it necessary to keep copies of previous states, but also it is necessary to stop the operation of the system during checkpointing, to ensure that the state that is stored is consistent. In many cases, this is not viable, since it may not be possible to restore the environment in which the system is operating to the state that corresponds to the checkpointed state.

- In such cases, **forward error recovery** is more appropriate. This involves driving the system from the erroneous state to a new valid state. It may be difficult to do so unless the fault that caused the error is known precisely and is well isolated, so that it does not keep interfering. Forward error recovery is not used often in practice, as it is tricky in nature.

The failure recovery of the mobile agent in this work is achieved, by using a combination of both the replication and checkpointing strategies. At the same time, the forward error recovery mechanism is used in order to attain this failure recovery.
1.8 ORGANIZATION OF THE THESIS

This thesis is organized in seven chapters.

Chapter 1 gives a brief overview of the evolution of the mobile agent system and the issues related to its operation in a multi-region environment.

Chapter 2 presents a detailed literature review about the existing works on the various issues affecting the mobile agent based information retrieval.

Chapter 3 describes the various components and the functionalities of the Multifaceted Mobile Agent System for Information Retrieval (M-MASIR), for the efficient and reliable information retrieval.

In Chapter 4, the benefit of the location server based dynamic travel plan over the static travel plan is discussed with a comparative study.

Chapter 5 explains the functioning of the memory allocation algorithm that tends to provide more space for agent accommodation. It also analyses the prominence of the presented security mechanism in the event of tailgating attacks.

Chapter 6 discusses the functioning of the failure recovery model, in various failure situations and its mastery over the existing fault tolerant systems.

Chapter 7 gives the conclusion of this thesis and presents possible future enhancements.