A Region-based Architecture for Service-Providing Distributed Systems

by

Neha Singh

B.S., Computer Science

The University of Texas at Austin, 2003

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science and Engineering

at the

Massachusetts Institute of Technology

June 2005

Signature of Author ……………………………………………………………………………...
Department of Electrical Engineering and Computer Science
May 17, 2005

Certified by ………………………………………………………………………………………...
Karen R. Sollins
Principal Research Scientist
Thesis Supervisor

Accepted by ……………………………………………………………………………………..
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Region-based Architecture for Service-Providing Distributed Systems

by

Neha Singh

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2005 in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

A service-providing system consists of hosts that provide services such as data, content, computational and memory resources and data-based services to other entities in the system. Consumers that wish to use services describe their needs with a set of high-level objectives. In this thesis, we address the problem of locating services in a large-scale distributed system using their descriptions, rather than their addresses. We propose a network architecture that is based on the concept of dividing the service-providing hosts into Regions. A Region is a grouping of elements of the network that share a set of common characteristics and policies. Members of a region manage their interactions with other regions and their elements according to some defined rules and policies. Hosts can be divided into regions based on various properties such as their content, their commercial model or their security characteristics to name a few. The service provided by a region is an aggregate of the services provided by all its member hosts. The region-based architecture routes a service request through the network efficiently based on its description and on the advertisements from regions providing services. Division of hosts into a set of independent regions partitions the search space and produces a scalable structure. The architecture also does not impose any rules on the internal organization of regions making the system flexible and dynamic.

Thesis Supervisor: Karen R. Sollins
Title: Principal Research Scientist

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Karen Sollins, for her incredible support and guidance during my research and throughout my stay at MIT. Working with her has been an amazing experience. I enjoyed all our discussions on research and non-research topics and was fascinated by her knowledge about so many different subjects ranging from networks to ancient Greece to art museums. I am forever indebted to her for supporting all my decisions – she is the most wonderful advisor and mentor that I could have wished for.

Secondly, I would like to thank my friends from UT – Heena, Poornima, Nidhee and Deepika – with whom I shared some very memorable years and who have been a constant source of love and support even after I left for grad school.

I am also grateful to all my friends in Boston for their help, friendship and company over the last two years – thanks for being there for me. I would like to thank the members of my research group, Advanced Network Architectures (ANA), for all that I learnt from them.

Finally, I dedicate this thesis to my parents and my sister. Thanks to my sister, Aishna, for being my constant phone companion for the last year and for providing the comic relief to my days. Thanks to my parents for their selfless love, support and encouragement throughout my life – I am where I am because of you and words cannot express my love and gratitude.

Contents

Chapter 1

Introduction

With the rapid increase of data and services offered through networks, there is a need for mechanisms that enable users to locate the network services and resources easily. Specifically, an architectural capability is needed for such large-scale distributed systems where services can be accessed by specifying their characteristics, rather than their addresses. We use the term '*Service-Providing Distributed System*' to refer to a distributed environment that contains hosts which provide services and resources for the use of other entities in the system. Consumers of services describe their needs by providing a description of their objectives. The question we are addressing in this thesis is the following: how can services be located in a large-scale service providing distributed system through their descriptions? The network architecture that we propose in the thesis is based on the concept of dividing the service-providing hosts into groups called *Regions*. The idea is that by dividing the hosts into smaller groups, we get a scalable structure where a *Region* becomes the element of scaling.

We start by introducing the problem and providing some background on it. We then introduce our proposed solution by discussing the concept of *Regions*, which is at the core of this work. In order to provide a context for the related work, we give an overview of the proposed architecture. After this, we present a summary of our contributions and an outline of the remainder of the thesis.

1.1 Motivation

The objective of our system is to search and locate services and resources using descriptions, rather than addresses. This is a valuable capability for a large-scale network where the elements that provide services are numerous and widely-distributed. The services we are considering could include data, content, resources such as computational and memory resources, and data-based services. For users, it is much easier to describe their needs in terms of properties or characteristics of the information or resource they are looking for, rather than to know the hostname or address of the host providing it. It is also not sufficient for the consumers to go through an entity that simply indexes all the resources in the system with their addresses because this method will not work well for a large system. Our aim is to provide a scalable architectural solution to this problem.

The problem of locating data using its description or content has been identified and addressed a number of times. Publish/Subscribe schemes deliver content to users by matching their criteria with the message ([2],[7]). The provider of the information is the sender of the message and the destination is decided based on the content of the message. A shortcoming of such schemes is that communication is initiated by the sender of information. In our system, we want the users to be able to initiate the search for services. We also want them to be able to avail themselves of a wider variety of services such as memory and computational resources, rather than being restricted to accessing content on a subject.

Resource Discovery Systems are another class of systems that are designed with the purpose of locating resources and services in a network. They specify a naming scheme for the resources and a discovery architecture that aids in the finding of these names. Most of them maintain an index of the resources and the hosts that provide them. In order to develop a more scalable system, we need a structure more complex than an index from the resources to their locations. Our research proposes a divide-and-conquer strategy for the situation in which hosts are divided into smaller, independently-managed groups called *regions*. Since the groups are self-configured and managed, the system only needs to concern itself with interfacing with them.

1.2 Regions

The current Internet structure consists of a loosely coupled federation of separately defined, operated, and managed entities, interconnected to varying degrees, and often differing drastically in internal requirements and implementation [40]. It is natural to think of each of these entities as existing in a region of the network, with each region having coherent internal technology and policies, and each region managing its interactions with other regions of the net according to some defined set of rules and policies. A **region** is a grouping and partitioning mechanism that is a key design element in architectures for extremely large scale, wide distribution and heterogeneous networks.

We apply the region abstraction to the problem of locating services in a large scale network. Individual hosts that provide services are members of regions. Regions are formed based on certain principles and thus, the members of a region share characteristics and policies. The service provided by a region is an aggregate of the services provided by all its member hosts. Essentially, the service search space is partitioned into regions. In order to avail themselves of services, consumers go to regions that have characteristics that they are interested in. Besides providing scalability, regions help in dealing with the dynamism of hosts and services by containing the changes within a boundary. This makes it easier to manage changes than attempting to do so altogether in a large-scale system.

1.3 System Overview

This section presents a brief overview of the proposed architecture. To enable the consumers in a service-providing distributed system to search for services, the providers have to advertise what they are offering. In our architecture, the hosts that provide services are members of *regions*. Each host has a description of the services that it offers. The service description of the region is the aggregate of the service descriptions of all its member hosts.

The intents of the consumers in the system are represented by *Mobile Agents*. Mobile agents are the mechanism that is used by the consumers of services to communicate with the providers. They are the means by which services are searched for and located. Mobile agents are entities that combine both program code and data i.e. they have state and algorithms. They are launched into the network by their creator and then travel from host to host performing computations or utilizing resources on the behalf of their owner. In our system, the agents are *autonomous* which means that they determine their own path through the network. They do not need to be given a list of addresses to visit when they are launched, but they compute their own path using information about the objectives of their creator. They essentially use a high level description of the consumer's objectives in order to find appropriate services that will fulfill the objectives.

Lastly, the consumers represented by the agents need to be connected with the regions advertising services. This role is performed by the *Directory Server Network*. Regions send their aggregate service advertisements to directory servers in this network. Agents go to the directory server network in order to obtain information about the locations of services. The directory servers are organized as an application-level overlay network. The information about the services of regions is distributed among the various servers. When an agent arrives at a directory server, its objectives are matched against the service advertisements that the directory server is carrying. The agent is referred to regions and other directory servers that match.

1.4 Contributions

In this thesis, we present a network architecture that enables location of services in a large-scale distributed system. Services in the form of data, content or resources can be located through their descriptions, without knowing their exact address in the system. This is a valuable and essential requirement for large-scale systems that provide any kind of services.

We apply the region abstraction [40] in the system and argue that it provides the basis for a scalable solution to the problem. The region is a construct proposed for designing large scale, widely distributed and heterogeneous networks. In this particular problem, regions provide a means to partition the search space of services by grouping together hosts that have common characteristics. We describe, in subsequent chapters, a detailed design and mechanism for how regions can be used to achieve the above objective and what other elements are needed in the system. We do not impose any properties on the regions or constrain their internal organization. They are treated as independent entities that organize themselves and have their own characteristics that determine their properties and membership.

In order to evaluate the proposed architecture, we implemented the entire system in Java. The implementation provides a general framework that can be used by applications with or without specialization. The base architecture uses a general-purpose language for representation of services and communication between the various entities. If this is sufficient for the application, it can use the framework directly. The architecture is also flexible such that the necessary components can be extended to make them more specific for the application. We also performed a set of simulations with the implemented system to measure its performance with respect to various factors. We present the results of the experiments and analysis of the system properties and features in the thesis.

This work is also a study in further exploring the regions abstraction. It investigates the utility of regions as an architectural mechanism by applying the concept to a specific network scenario. This work helps in providing insight about regions, their membership and their properties. As an exercise in building a system that involves various regions and entities that move in and out of regions, it has helped us explore what happens when processes cross the boundary of regions. Through the work, we have been able to generalize the actions that can take place upon a boundary crossing, which is an important property of regions.

The remainder of the thesis is organized as follows. In Chapter 2, we present related work in publish/subscribe systems and resource discovery systems that also aims at discovery of services or resources using their content or descriptions rather than addresses. In Chapter 3, we present the proposed architecture in detail by describing all its components and the communication between the components. Chapter 4 describes the implementation of the system and a few extensions to it. It then presents the results of the simulations that attempt to measure the performance of the base system. Finally, Chapter 5 concludes with a summary of contributions and suggestions for future work.

Chapter 2

Background and Related Work

In this chapter, we discuss the research that has been done on topics related to this thesis. First, we present summaries of other systems that aim at locating data or services based on their content or descriptions. Then we discuss the previous work done on the region abstraction and its applications. Lastly, we present briefly some previous work on the subject of mobile agents, which form one of the components of the architecture that we are proposing in this thesis.

2.1 Service Location Systems

The primary feature of the proposed architecture is that it enables us to find data and services residing at remote locations without knowing the exact destination address. This is achieved by matching service advertisements coming from regions with the objectives that describe the service needs of a user. There are a number of different models of addressing and routing in a network based on some form of 'content' instead of IP addresses. These are generally referred to as *data-based addressing* or *content-based addressing* techniques. In [11], Carzaniga *et al.* present foundational concepts of content-based networking, and relate them to the corresponding concepts in traditional networking.

The most common among these techniques are the content-based *Publish/Subscribe schemes* that treat the provider of information to be the sender of the message and the destination is decided based on the content of the message ([2],[7]). The information reaches consumers who have expressed interest in delivery of messages that satisfy some arbitrary predicates on their content (independent of who produced it). The responsibility of routing lies within the infrastructure which typically uses 'access points' where the producers of notifications advertise or publish and interested parties subscribe for individual notifications. The selection of notifications is performed using filters (which are attributes and constraints on values) and patterns (which are matched against two or more notifications). [30] defines a taxonomy for comparing and contrasting publish/subscribe systems and provides a survey of the publish/subscribe systems existing at the time. In [34] and [35], Mühl presents mechanisms to improve scalability of content-based publish/subscribe systems such as filter merging and also advanced routing algorithms that do not rely on global knowledge.

The publish/subscribe model is based on a 'push' methodology because the provider of information is the one that initiates the communication. Although this is suitable for some applications, it is a restrictive model in general because it does not allow consumers to search for content when they want it. Also, the current schemes are limiting because they can only publish and subscribe to data and not to services or resources. The architecture we propose is a scalable and dynamic content-based network where the consumers can access data and services they need without knowing the exact addresses of the hosts that provide them. Also, using mobile agents allows for a richer communication paradigm because agents can go to multiple hosts in order to find all the information they need in the same trip and collect and aggregate this information if desired by the consumer. Besides collecting data, they can perform computations and processing on the servers they visit, hence making use of services and resources in the network. In publish/subscribe systems, the access points receive messages and possibly store and forward them. In the proposed system, the directory servers and hosts in the region store aggregated and condensed versions of individual advertisements and not the entire data or message.

Another related area of research is that of *resource discovery systems*. These systems are designed with the purpose of locating resources and services in a network. They specify a naming scheme for the resources and a discovery architecture that aids in the finding of these names. Most of them are targeted for pervasive computing environments and use one or more directories that maintain an index of the resources and the hosts that provide them. [17] presents SDS (Secure Service Discovery Service), a secure and fault-tolerant service for locating services in the wide-area network. [52] introduces a resource discovery system for the World Wide Web that uses a vector space retrieval model, relevance feedback mechanisms and a hypertext mapping technique.

The *Intentional Naming System* (INS) is a resource discovery and service location system for dynamic and mobile networks that integrates resolution and routing and allows both anycast and multicast message formats [1]. Our architecture is based on INS in some of its features such as the spanning tree topology of the directory servers and the soft-state periodic advertisements from services to discover names. The *resolvers* in INS, which are like the directory servers in our system, form a spanning tree based on the round-trip latency between them. The paper states that time to process the name updates was the bottleneck in many cases and identifies the need for a technique to partition the namespace among resolvers to alleviate the problem and briefly discusses the idea of ‘virtual spaces’ as a solution. The most important difference between our architecture and INS is that our system uses *Regions* to group the hosts, hence using divide-and-conquer to make the system more scalable. Another difference is that in INS, each service attaches itself to a resolver and can advertise only a single application-based metric based on which incoming requests are routed. In our system, the aggregated advertisement of the region can contain any number of properties and the incoming agents will be routed based on any properties that the agent cares about. A matching algorithm is used that takes into account all the policy rules of the agent and the properties of the region. This makes the system more general purpose for any number of applications and also more scalable. Also, in INS, each service advertisement is propagated through the entire resolver network, which means that every resolver knows about every single service in the system. This solution does not scale very well for our purposes. We instead propose a method that considers the trade-off between the scalability and efficiency of the functioning of the directory server network[1].

There are other resource discovery and information retrieval systems that are based on the idea of dividing resources into groups. [12] proposes an application-level protocol that enables data sharing among different domains containing resources. This work mentions the idea of having 'data clusters' with a set of directories that share related information. A 'cluster' is distributed, self-organizing, responsive to data mobility, and robust to failures. Using application-defined data schemas, 'clusters' organize themselves into a hierarchy for efficient querying and network resource usage. 'Ingrid' [20] is a distributed, scalable and self-configuring information navigation infrastructure. In this system, links are automatically placed between individual resources based on their topic-similarity in such a way that clusters of term combinations are formed. This is again based on the idea of dividing the resources in some

[1] Details of the method are presented in the discussion of the directory server architecture in section 3.2.4.

manner. In our system, we formalize and generalize the idea of grouping services by introducing regions.

2.2 Regions

In this section, we will present some of the previous work on regions. This work is part of the Regions project ([40], [42]) and explores the use of regions as an architectural mechanism. The concept of regions is proposed as a key design element in an architecture for extremely large scale, widely distributed, and heterogeneous networks, and is a mechanism for grouping, partitioning, and formalizing boundaries around those groups and partitions. Regions are based on the notion that the Internet is an increasingly complex network of networks where elements are connected and interrelated by a set of common invariants. A region is thus a partition of the network where the member nodes share common state, policy or knowledge. An example of a region is the set of nodes within an autonomous system (AS) [36]. The regions project seeks to provide an architectural mechanism with which to define and use regions.

Previous work on regions includes the following. Law [28] looks at the issues that arise when entities of a region are no longer part of the region, due to a request to leave the region, an eviction by the region, or component or network failures. Consequently, the references to these entities must be garbage collected. In another work, Li [29] explores the utility of informing members in one region of the membership of those same entities in different regions and specifically improves peer-to-peer networks with information derived from Autonomous Systems. His work provides a general peer-to-peer simulation framework for different optimization schemes and also achieves performance improvements in the lookup, caching and replication of peer-to-peer systems. Beverly [8] proposes a reorganization algorithm, based on the region abstraction, to exploit the natural structure in overlays that stems from common interests. His architecture leverages the inherent heterogeneity of users and places within the system their incentives and ability to affect the network.

2.3 Mobile Agents

The use of mobile agents in our architecture was motivated by their rich functionality and the asynchronous communication model that they enable. A mobile agent is defined as an entity that combines both program code and data and has the ability to move around in a network from one host to another and execute on them [49]. There is a large body of research on the subject of mobile agents. The Open-Agent architecture [14] and Agent Cities Network architecture [50] projects each present frameworks for constructing agent-based systems, making it possible for software services to be provided in distributed systems. In [13], Chandra *et al.* address the feasibility of meeting resource management needs in an environment where multiple mobile agents are utilizing resources at a host. [31] discusses systems that use mobile software agents to manage complex real-world networks and describes a strategy for using a collection of cooperating mobile agents to solve routing problems for dynamic, peer-to-peer networks. The implementation of mobile agent systems can be classified into two categories based on the characteristics of the base language used to program the agent paradigm [47]. One category of mobile agent systems uses object-oriented languages. Agents are defined as first-class objects, and the system provides support for their migration in the network. Such systems offer the natural advantages of object-orientation, such as encapsulation and code reuse via inheritance. Most famous examples of such systems are Telescript [45] and Aglets [26]. The second class of agent-based systems uses script languages like Tcl, Python or Perl, for defining mobile agents. The advantage of such systems is their simplicity and the existence of mature interpreter environments which permit efficient access to local resources. Agent Tcl [22] is an example of a Tcl-based system. The agent system in our architecture is based on *Ajanta* [3] which is a mobile agent programming system aimed at building an infrastructure for mobile agent execution that incorporates security and robustness as integral features of its design [48]. Ajanta is an object-oriented mobile agent system implemented in Java.

In [24], Kotz *et al.* discuss several technical and non-technical hurdles along the path of adoption of mobile agents. Roth [37] presents the obstacles to the use of mobile agents, in particular, the lack of applications versus the lack of a sufficient installation base and security considerations. The main drawbacks of using mobile agents are the possible security concerns related to use of agents on remote hosts. It is hoped that these concerns can be circumvented in our system by security guarantees provided by regions and security measures taken by them such as screening and authenticating agents before they enter a region.

This concludes the chapter on background and related work. In this chapter, we discussed systems that aim to locate data and resources based on their content, including publish/subscribe schemes and resource discovery systems. In view of the existing work, we argued for the need of a scalable system where users can search for a wide variety of services based on their descriptions. We then presented previous work based on the regions abstraction. Finally, we briefly discussed some existing mobile agent systems. The next chapter presents details of the architecture that we are proposing in this thesis.

Chapter 3

System Architecture

This section describes the details of the proposed architecture for service-providing distributed systems that is based on Regions. A service-providing distributed system has been defined to be a distributed environment that contains hosts which provide services for the use of other external entities in return for monetary or some other form of compensation. Our architecture is a design for such a system where there are suppliers and consumers of services. Consumers have the ability to search for one or more services of which they would like to avail themselves. Services that are provided could include data, content, computational or memory resources and data-based services. The objective of the system is to provide a scalable solution to automate the process of searching and locating these widely distributed resources and then making use of the services and data they provide. The key idea in the architecture is to use the concept of Regions to group together the service-providing hosts that have similar characteristics and then use a Region as an atomic unit in the organization of the system.

There are four major components of the architecture: service-providing hosts, regions, directory servers and mobile agents. Hosts are the entities that make various services such as data, information and resources available to anyone who needs them. They send descriptions of these services to the directory servers through their regions in order to advertise them. Any entity that wants to use the services sends a mobile agent through the network to locate them. The intent of the agent's creator is expressed in the mission and policy of the agent and is used to derive a path for the agent through the network. The directory servers are the elements which aid in this process by providing agents with information about the available services from hosts. The directory server network performs the function of matching desired properties of agents with advertisements from regions, directing agents to regions or to other directory servers. Regions facilitate the interaction between the hosts and the agents by grouping hosts that have common properties together. They aggregate advertisements from their member hosts and send them to the directory servers. Therefore regions take the responsibility of advertising the services of hosts and could also provide the hosts with additional services such as security guarantees, commercial services and acting as the administrative entity. Once an agent finds out about the regions it is interested in from the directory servers and it visits a region, the infrastructure of the region directs the agent to the appropriate hosts. Hence regions provide services to both hosts and agents. Regions take care of hosts joining or leaving the network and changing properties as well as failure of hosts, hence making the system dynamic. The indirection introduced by the regions also makes the architecture scalable because regions can be treated as a single entity at the level of the directory servers.

The rest of this chapter discusses each of these components in further detail. Section 3.1 outlines the Regions model, the functions of regions in the architecture and other interesting problems related with regions. Section 3.2 describes the directory server network – its architecture and functions. Section 3.3 describes the mobile agents, the abilities they provide and the process with which they perform service location in the system. Section 3.4 outlines the communication model for the system including the language used for describing and searching for services and the communication between different entities of the system. Lastly, Section 3.5

talks broadly about the properties of the architecture – the ones it does have and the ones it does not.

3.1 Regions

A Region is defined as a grouping of elements of the network that have coherent internal technology or a set of common policies. It is a network entity that manages its interactions with other regions and their elements according to some defined set of rules and policies [40]. In the context of a service-providing distributed system, a region represents an entity in the system that groups together a set of service-providing hosts that share common characteristics. A region is independently managed and provides infrastructure to the member hosts to perform their activities. The membership of a region could range in size from a few hosts to thousands of hosts. It could be formed around various principals, or in other words a region could have several different characteristics that form a unique combination.

Regions are an integral part of the proposed architecture. They relate to the other elements of the architecture in the following way. The hosts that are members of the region provide services such as data, content and resources in return for monetary or some other form of compensation. Regions are the medium through which these hosts advertise their services to the outside world. A region collects the service advertisements from all its member hosts and advertises that to the directory server network as the services provided by that region. After obtaining information about these services from directory servers, mobile agents are directed to appropriate regions that have advertised services that the agent is interested in. Once an agent reaches a region, it is directed to the appropriate hosts using the infrastructure in the region.

3.1.1 Motivation for using Regions

The fundamental purpose of having regions in a service-providing environment is to reduce the size of the search space by partitioning it. Regions act as atomic entities at the level of the directory servers advertising some aggregate form of the services of all their member hosts. If there were no regions, each host would have to advertise its services directly to the directory servers. In the case of a large scale distributed system in this scenario, in order to search for one of these services, the agent would possibly have to go through the advertisements of a large number of services. There would also not be any relation between two hosts that are providing very similar services. This design does not entail a scalable search. By introducing regions that group together hosts with similar characteristics, the advertisements from a number of different hosts could be combined reducing the search space for the agent. At the level of the directory server network, a region is an atomic entity that provides a set of services and is independently managed. Since it actually represents a number of other hosts, it significantly cuts down on the number of entries in the directories. Also, two hosts providing similar services are ‘close’ to each other in the sense of belonging to the same region. This makes the location and use of similar services more efficient as an agent could visit all the hosts in the same region at one time.

Related to the concept of cutting down the search space for service location is the fact that regions provide a service to their member hosts in doing so. They advertise the hosts’ services to the world and provide infrastructure to direct incoming agents to the hosts where they can consume services they are interested in. They facilitate administration by providing natural boundaries for managing a set of hosts. The large scale system is hence made more manageable by division and an additional layer of indirection (between the service-providing hosts and the directory servers). Regions could also provide additional services to their members such as security, privacy guarantees, etc.

A region would be effective in partitioning the service space only if it groups together hosts with common properties and not just a random collection of hosts. This notion is perfectly consistent with the definition of a region as a region is inherently based on a set of invariants that are inherited by all its members [40]. This definition implies that a region could be created along certain specific principles which can be expressed in its invariants. The defining invariants of a region need not necessarily be unique i.e. there could be two or more regions with the same invariants. The invariants can be used to define a region in different ways to enforce the purpose it was created with. For example, a region could be defined such that all hosts that are members agree with its principles by providing services that are consistent with the principles. This means that the services will have certain properties desired by the region. Or a region could be defined such that hosts that would like to avail of the services that the region is providing could become members. This means that the world can be divided into regions in various ways. A few possible divisions of regions are discussed below:

1. Division into regions based on *content* – each region has an idea of the kind of content it will specialize in and all the members of that region provide services and serve content related to that category. In this scenario, all the hosts that are interested in a particular topic would be grouped together. For example, there could be a region that serves information about astronomy such as data, computational services, etc. There could be another one that serves information related to geography such as maps. The division based on content can be achieved by only admitting hosts that have a particular set of properties in their service advertisement.

2. Division into regions based on *commercial activity* – the region provides a differentiated service to the hosts that are its members and the agents that are visiting it and charges a fee in return for this service. Servers that agree with the service model of the region can become part of it. The region may or may not specialize in certain kind of content. For example, a region may provide certain guarantees on performance in return for a fee such as a bound on the time in which the agent will be able to process its query. Another region could guarantee the maximum amount an agent will spend at any one host. A region could charge a host for bringing a minimum level of traffic to that host.[1]

3. Division of regions based on *security and privacy characteristics* – the region will have some security credentials such as a guarantee that it conducts all of its transactions in a secure manner. Some agents might need that kind of security for their actions, and therefore they might choose to go to a region that offers them that service. Or a region can guarantee that it will not disclose the queries that an agent is making and so an agent seeking privacy might choose to go to such a region. Hence the defining characteristic of a region could be based on the kind of security and privacy it provides to its member hosts and to the agents that visit it. The kind of access control imposed by secure transactions has two sides: an agent might want to go to regions with specific security characteristics and also, a region might also want to admit only those agents that have certain security credentials.[2]

4. Division of regions based on *administration* – a region consists of all hosts that are administered by a single entity. The administrator can decide the policies of the region, negotiate billing contracts with other regions, etc. Boundary of the region is the administrative boundary of

[1] Commercial models for the existence of regions are discussed in detail in section 3.1.6.

[2] Access control by regions and agents is one of the actions that happens at the boundaries of regions and is discussed in greater detail in Section 3.1.5.

control of the owner of the region. In this case, all the hosts that are owned by the administrator would be part of the region.

The above divisions are examples of the kinds of characteristics that a region could be organized around. There could be any number of other possible divisions. An important observation is that one region could provide more than one or all of the above services, i.e. it could be created based on all the four kinds of divisions discussed above. For example, there could be a region that charges for providing some premium content on a particular subject matter. It could be administered by one entity and also provide some security and privacy guarantees. Hence, a region can be based on any arbitrary set of characteristics and all the hosts that are members of the region will inherit these characteristics.

3.1.2 Creation and Membership of Regions

Regions are created by some entity for the purpose of providing services in a distributed system. Each region has a unique name, which is used to identify it. There can be two forms of creation of regions: creation by an administrator and self-creation by a group of hosts. In the first case, an entity who is interested in providing some kind of service to hosts and/or agents in the system can create a region. This entity would be the administrator of the region and would decide on the policies and properties of the region. The administrator of the region is free to organize the region internally as it wishes, but has to make sure that the region provides the basic functionality that is required of all regions in the system for them to be functional and useful (required functionality is discussed in section 3.1.4). The administrator can also install hosts that are members of the region and provide services in the region. Also, there can be existing hosts that could want membership in the region because of its properties. New hosts can thus become members of the region at any point after it is created.

The other method of creation of a region is self-organization of hosts with similar interests into a group. In this case, a set of hosts that are providing services that are related to each other or that have common properties could decide to form a region in order to make the process of searching for their services more efficient. They could collectively decide on the policies and properties of the region. They would also have to share responsibility in providing the required functionality of the region. The region may or may not want to admit newer member hosts depending on its policies. The administrative tasks of the region could be handled by a single host or a group of hosts. If it is a group of hosts, they would have to act together and take atomic actions or make atomic decisions.

Following from the modes of creation of regions are the ways that hosts could become members of regions. They could either already be part of the region when it was created such as the case when a group of hosts self-organize into a region or when an administrator sets up some hosts while creating a region. The other case occurs when an existing host asks for membership in an existing region.[1] The region can grant or deny membership to any host depending on its policies. Also, another important point to note is that the same host could be a member of multiple regions. It could advertise part of its services through one region and the some other services through another region.

[1] For the purposes of this work, we do not discuss the process of how a host learns about new regions and goes about asking for their membership or how a newly formed region invites membership from hosts. We are assuming that hosts are already members of the regions since the focus of the work is on the architecture required for the hosts inside regions to advertise their services and for enabling search and usage of these services by agents.

When a host becomes a member of a region, it inherits all the invariants of the region. The host still has its own service advertisement and properties, but it now also has to follow the principles of the region it is in which translates to the host having certain properties that the region requires. This may or may not require a change in the host depending on whether the host already has the properties or not. Conversely, the content that the region will serve is aggregated over all its member hosts. Thus, even though the region has its own properties, the service description it advertises will be a combination of its own properties and the service advertisements of all its members. For an external observer, this aggregate description is the advertisement of the region since the region is viewed as an atomic entity externally.

3.1.3 Components and Internal Organization of Regions

A region is characterized by three components: its members, its invariants and its boundary. In our architecture, regions can have two kinds of member hosts. One is the service-providing hosts that offer some kind of content, resources or data services by advertising them through the region. The hosts providing services can move within the region and they could also move from one region to another as services could be mobile. The other kinds of hosts are those needed for the functioning of a region. These could include hosts that perform administrative functions such as deciding on policies of the region and controlling its boundary. There could be hosts that aid in the routing of incoming agents to the appropriate service-providing hosts and those that communicate with other entities in the system such as directory servers on behalf of the region. Also, there could be hosts for performing other functions that are specific to the particular region such as a host that collects billing information for a region that bills agents for its services and then processes it for each agent that is about to leave the region. All these hosts do not directly provide services to agents but are needed for running the region. They need to be maintained by the entity that owns and administers the region.

The second component of a region is the set of invariants it was created with. These are the defining properties that a region is based on and that are inherited by all its members. They can be used to represent the properties as well as the policies of the region. They will define the distinguishing characteristics of the region and the services it provides to its member hosts and agents. Some of these properties will determine what kinds of hosts are admitted into the region as members. Only hosts that agree with or satisfy the invariants will become members. They will also determine the kinds of agents that will be able to enter the region. Some instances of defining properties for a region are the subject of the content served by its hosts, the commercial model that the region uses and the security and privacy characteristics of the region. For example, a region could define a security standard in its invariants and all agents attempting to gain access to the region have to meet this standard.

The third component of a region is its boundary. The boundary represents the logical separation of the region from the rest of the world. The boundary has controlled crossing points which provide entry into and exit from the region. In our architecture, the boundary is physically manifested by hosts that sit at the edge of the region and are the first points of contact for an agent trying to enter the region from outside. There can also be exit border points that the agent has to pass though before it can leave the region. Both the entry and exit access points are called *Border Way Points*. The entry and exit boundary points can be used for various functions by a region. For example, a region can check the credentials of an agent for authenticity at the entry border hosts. The exit point could be used for charging an agent for all the services it has used while it was inside the region. There are also a number of other actions and exchanges that could take place at the boundary. Section 3.1.5 discusses the actions that take place when there is a boundary crossing.

The preceding paragraphs discussed the various components of a region. Different regions could implement these components in different ways or use them for different functions. Regions can be very diverse and could vary in size from a few members to thousands of members. They could also be providing a variety of different kinds of services to their members and to agents. Because of this variety in the form and functions of regions, we do not impose any strict rules on how the region should be organized internally in the architecture. Different organizations may be suitable for different kinds of regions and since a region is an independently managed and administered entity, the internal organization is left up to the region as long as it provides certain basic functionality. This functionality is needed for performing some basic functions for the architecture to work and also for providing a platform for communication between regions and the other entities in the system such as agents which are the consumers of the services and directory servers that aid in the process of locating services. For instance, once an agent enters a region, it needs to be directed to hosts that provide services that it desires. There has to be an infrastructure in place inside the region to provide this functionality to the region. How exactly it is implemented could depend on the region. In a region with 10 members for example, there could be a single server that stores all the host names and advertisements and directly directs the agent to the appropriate host by searching through all its entries. For a region with thousands of hosts, this approach would be inefficient and a possible approach is a hierarchical structure where the hosts are at the leaves and other hosts that serve as directories (referred to as *Way Points* [51]) are the internal nodes. The internal nodes direct the agent at every level. It would not be practical for deployment if each region in a large scale system had to have identical organization. Thus, as long the regions follow the required interface, the architecture is independent of the implementation of a region's internals. This is a key feature of the architecture and makes it viable for a large scale distributed system. We have implemented one possible internal organization that is flexible for regions of different sizes in our system and it is discussed in detail in the implementation section 4.2.

3.1.4 Functionality provided by Regions

In the previous section, it was discussed that a region can be internally organized as it chooses given that it provides some functionality that is needed for the system to work. This section discusses the functions that the region needs to provide so that it can advertise its services and agents can locate and avail of these services. It describes the protocol that regions need to follow to interface with other regions and with agents and directory servers. The functionality is independent of implementation or of the internal organization of a region. The functionality can be divided into three categories:

1. Communication with directory servers
2. Actions at boundaries of regions
3. Functionality for agents inside the region

Each of these areas of function sets is now discussed in detail.

1. Communication with directory servers

The purpose of directory servers is to advertise services of regions and direct agents searching for services to appropriate regions. The communication between a region and a directory server consists of advertisements of services sent from the region to the directory server. Each region has a designated directory server that it advertises with. Once a new region is formed, it obtains the name of its directory server from the Central Resolver and then registers with the directory server. It can then start sending its advertisements to the directory. The directory servers maintain a soft state [16] for the advertisements. This means that when a directory server receives a service advertisement from a region (called `REGION-TO-DS-AD-MESSAGE`), it stores the ad as an entry with a timeout value. After the region has sent the advertisement once, it can send it as often as it wishes depending on how frequently it is changing. If the services of a region are changing frequently, the region may wish to send messages to its designated directory within time periods much less than the timeout value. Or the region may choose to wait and only report the services once in a while if the changes are minor. This again depends on the nature of the region. However, if the advertisement of the region is not changing often, it has to keep sending a *Keep Alive* message (called `REGION-TO-DS-KEEPALIVE-MESSAGE`), to the directory at least once every timeout period so that the directory knows that the region still exists and its advertisement is valid even though it has not changed. The directory server maintains the same advertisement for the region until it receives a new one.

If the directory does not hear from the region again before the time period expires, it deletes the entry for the region. In this way, regions do not need to be explicitly deregistered and this also takes care of failure scenarios if a region has stopped functioning abruptly due to problems, the directory server will eventually delete the entry. When a directory server is deleting the entry of a region, it sends a message to the region (called `DS-TO-REGION-DELETING-ENTRY-MESSAGE`) saying that its ad is being deleted. This is useful because if the region's ads or keep alives did not reach the directory server for any reason, it is now aware that its entry has been deleted so that if it is still alive and functioning, it can send a new `REGION-TO-DS-AD-MESSAGE` to the directory. This is one of the many ways in which the system is made fault-tolerant and it can recover from failures of message delivery.

This function of communication with the designated directory server can be performed by any host or group of hosts in the region. The region can have a host that is permanently dedicated to sending out ads and keep alives. Or the region could change the servers it uses to do this over time. Or a group of servers could share the task. The directory does not care which particular host in the region is sending the message. It however does need to verify that the message is authentic i.e. it has legitimately come from the region it claims to be coming from and also possibly do an authorization check. There are a variety of techniques that can be used for this authentication such as certificates, keys, etc. Related work in the area of authentication shows how this can be done ([18], [19], [44]). The only case in which a directory server has to send a message to the region is `DS-TO-REGION-DELETING-ENTRY-MESSAGE`. The directory sends this to the server from the region that sent the last ad or keep alive message. Also, an integrity check needs to be done when the message is received at the directory server to make sure that it did not get corrupted or changed on the way.

We will now discuss the content of the ad messages sent by the regions to the directory servers. The message contains the name of the region which is used to identify its entry. The most important part of the ad message is the service description of the region. This is an overall description of all the services provided by the various hosts in the region. Each of these hosts can advertise their own services and the region has to aggregate all the ads to form the composite one that it will send to the directory server. The service advertisements of individual hosts are described as attributes and values. The ad of the region can have multiple values for the same attribute if more than one host has it. Therefore the final form of the service part of the ad will be a map from attributes to a set of values for each attribute. The directory server will do the matching of the agent's mission to the advertised attributes and all their associated values[1]. The region can choose to aggregate these properties of the hosts as it wishes. It may wish to advertise only the important attributes for example and therefore might create a summary or selection from all the advertised attributes. It chooses the properties that it wants to advertise and puts them in the `REGION-TO-DS-AD-MESSAGE`.

The other component of the region advertisement is the set of requirements that a region has for the agents that are going to visit it. It also contains the invariants, characteristics and policies of the region that would be relevant to the agent. This could include the language which the region communicates in, the credentials required of the agents and so on. Since there can be a large number of diverse regions, it is conceivable that they might communicate in a variety of different languages or use different protocols for communication. If an agent does find a match against a region at a directory, it can use these requirements and properties to check if it would actually be permitted to enter the region or if it does not meet the requirements. If it does not, it could decide to go back to its creator and get the additional features. This list of requirements that the region advertises need not be the entire list of all the possible requirements and policies

[1] It is assumed for simplicity throughout this work that the names used for attributes and values have identical meanings globally, which means that the same attribute means the same thing in all regions. If an agent is looking for A, an exact string matching of A is performed. For a detailed discussion of this point, see section 3.4.

since that might be very complicated and depend on the particular agent and the particular services that it is requesting. Furthermore, some of the policies can be negotiated at the entry point of the region.[1] The requirements advertised to the directory would preferably be the minimum ones that the agent absolutely needs to have before it attempts to gain entry into the region.

The last part of the advertisement from the region to the directory server is the name of the host that an agent interested in visiting the region should go to first. This is the host that acts as the entry point for the region and all the agents that want to visit the region have to go through this host. Once the agent is at the entry point and has been permitted to enter the region, it is directed by the entry point to other hosts in the region where it can get services that it is interested in. What the region advertises to the directory server in the `REGION-TO-DS-AD-MESSAGE` is an identifier or a name for the host that is the entry point. This name can be used to refer to different actual physical hosts at different times[2]. The region can again implement the process of choosing an actual host as an entry point (if it has multiple candidates) independently as long as it provides resolution of the advertised name to the correct host at any time. There could be many reasons that the region would want to have different hosts acting as the entry point at different times such as load balancing, choosing entry points closer to the directory server to improve efficiency, etc. The region implements an algorithm that picks a host to be the entry point when a query is made about the region's ad. This resolution is done at the directory server and then the agent is transferred to the chosen entry point to gain access to the region. As the gateway to the region, the entry point can perform many functions such as checking that the agent meets the region's requirements, negotiating the entry of the agent, charging the agent fees

[1] See section 3.1.5 for discussion of actions at boundaries.
[2] See section 3.4.2 for details on how the resolution to multiple addresses can be implemented.

for entering the region, referring the agent to the appropriate hosts within the region and so on. The exact functions that the border points perform can vary from one region to another.

The above section discussed the functionality set of regions with respect to communication with the directory servers. An issue with this system that might arise is that a region could advertise false properties about itself to the directory to attract more agents but when the agent reaches the region, it would find that the services did not exist. There are two forms in which this can happen. The first is the case of a malicious region when it is purposely trying to advertise false properties. In the second case, even if the region is not malicious, there is only an asynchronous relation between the services of the region and the advertisement at the directory. Services might have moved, died or new ones might have been added since the advertisement was sent to the directory or since the agent read the advertisement. The aggregated ad that the region itself has of its services might be outdated depending on the way it is compiled since there might be a delay between the time a service changed to the time the region discovers about it. Therefore, the agent might have the wrong impression about the services in the region and would discover after actually visiting the region that things have changed. This is a natural result of having a distributed system since the information cannot be up-to-date at all points at every instance. Therefore, such problems cannot be completely eliminated from the system. One of the ways in which the agent has been designed to handle the possible inaccuracy of the information it receives about a region from the directory server is that when it actually reaches the region, it can ask the entry point for the updated service map of the region. It then evaluates again whether it still needs the services of the region if they have changed since the time the agent read them. In similar spirit is the mechanism that has been put into place for the agent to get the updated information when it reaches a service-providing host. The agent always asks the host first which services it has available in case they are different from what it was told by other servers in the region.

For the first case of a malicious region problem, there are some possible solutions. Regions could be certified for their trustworthiness and agents could demand these certificates as a guarantee that the region will behave honestly. Agents could also blacklist regions or give them low ratings so that future business with the region is impacted. These ratings could be made available to the all the other agents from the region the agent originated from and to other regions if they are interested. Therefore, some incentives can be put into place for the regions to behave honestly.

2. Actions at the boundaries of regions

The boundary of a region is represented by the hosts called Border Way Points that sit at the edge of the region and are the point of entry and exit for the agents visiting the region. The actions that happen at the boundary are performed when the agent is attempting to gain access to the region and when it is leaving the region after using some services from its member hosts. The Border Way Points are not necessarily geographically or topologically at the boundary of the region, but functionally they are the crossing points for any entity or process wishing to enter or leave the region.

One of the most important actions that happens when an agent is at a border way point for entry is access control. Regions can control some agents from accessing them and agents can control themselves from accessing some regions. Details of why and on what basis the regions and agents can perform access control are discussed in section 3.1.5. In this section, we discuss the mechanism for how this is enabled in the system. Regions might want to screen agents at the entry point if they have some security or other kinds of constraints. For this the region would need to find out information about the agent. The agent has a file (called `AGENT-PROPERTIES`) that contains all its public properties in attribute-value format. The border way point can request the agent for this file at entry and then the region can process this information to determine whether it wants to admit the agent or not. The region could also decide to admit the agent with certain constraints on its behavior while it is within the region. The agent is programmed to handle denial of access by a region – it could go to the next item in its itinerary or to a directory server for more region recommendations or it could go back to its creator to get more information for being admitted into the region. There is no public information that is required for all agents to have. Each agent can put its own choice of properties into its `AGENT-PROPERTIES` file.

The other side of the access control is performed by the agent. An agent could have its own constraints on what kinds of regions it wants to visit due to security, privacy or performance standards. Also, the agent has to check again what the current services offered by the region are since they might have changed since it read the region's ad at a directory server. The agent can request this information at the border way point. The border way points should either have the aggregated ad or they should have a mechanism to request it from another place in the region or a mechanism to perform the aggregation of services at the point when it is queried. Along with the service advertisement, any other relevant public properties of the region are also returned to the agent. The agent would use its policy to decide whether to visit the region or not based on the properties and services. Also, if the region has imposed constraints on the functioning of the agent, then the agent could decide that it is not worthwhile for it to visit the region.

This exchange between the region and the agent at the entry point could be viewed as a negotiation in which both parties give each other information about themselves in order to reach an agreement (about entering the region). Therefore, this exchange could take place in stages like a handshake. Some of the information might be restricted and is not allowed to be revealed until the other party has proved itself trustworthy. Therefore at each stage of the handshake, the two parties might reveal more and more information. This will continue until both parties are satisfied. It is also possible that if the region required some information that the agent does not currently have, it could return to its owner to obtain this information and then come back and resume the negotiation. The exact mechanics of the negotiation would be very dependent on the application and could be programmed into the agents and the border way points.

Before the agent leaves the region, it has to pass through an exit point at the boundary of the region. This point can be added at the end of the agent's itinerary for the region at the entry point itself. This ensures that the agent will visit the exit point after it is done in the region whether it has visited any other hosts or not. When an agent exits a region, it is the responsibility of the exit border way point to transfer it to the next item in its itinerary. If the agent is going next to another region, the identifier for the entry point would need to be resolved to an actual hostname. Also, other functions could be performed at the exit point depending on the region. Some regions might want to bill the agent for all the services it has used while it was in the region at the end, for example. This task would be performed by the exit point. Thus, the entry and exit points have some basic functions to perform but they can also be made more sophisticated if required by the application.

3. Functionality for agents inside regions

Once the agent is inside the region, the region has to perform three basic functions for the agent: provide it the advertised services through its member hosts; direct the agent to appropriate servers that provide the desired services; and transfer the agent to hosts within the region that it is interested in. Each host should have the ability to transfer the agent to another host in the region. A region could have servers that do both the first two functions or it could have specialized servers to do each task separately. The agent has to identify what functions the host offers when it arrives at a host. For uniformity and ease of use, each host implements the interfaces for both the functions. This enables the agent to invoke either of the interfaces at every host and gather information. If the host does not provide the particular functionality, it could return no results when that interface is invoked.

Every host in the system will implement three basic methods for the first two functions from above. The first one (called `AVAILABLE-SERVICES`) would return the services that the host is currently offering. The services could include a variety of things like data, content, computational resources, memory resources, etc. The names of the services are represented as attributes and their characteristics by values[1]. Thus, the host returns a map from attributes to values. Each service will have one entry in the map with one value for its attribute. If the host does not offer any services but performs other functions such as helping to route the agent within the region or making recommendations for which hosts the agent should visit given its mission (such hosts are referred to as *Way Points* [51]), the host will just return `NULL` when this interface is invoked. As mentioned before, we are assuming that the names of attributes are

[1] For services which are identified only by their names and do not have any values as such, the value field in the map could just be a boolean representing TRUE. This would mean that the service is present at the host.

universal and mean the same thing semantically everywhere. This means that when the agent is looking for a particular attribute, it performs exact string matches for the services advertised by the hosts and if it finds a match, we assume that it is the same attribute it wanted. This is a simplistic view of the world because the same word or attribute could in reality have different meanings to different entities. In other words, two different entities could represent the same attribute with different names.

The second method (called `SERVICE-INFORMATION`) also pertains to hosts that offer services. It is used for obtaining detailed information about a service that the host offers. `AVAILABLE-SERVICES` only provides a listing of the services, but if the agent is interested in using a particular service, it might need other information about the host and the service such as security credentials, protocol used for communication, performance guarantee on using the service and so on. `SERVICE-INFORMATION` will take a particular service name as the argument and will return all the relevant information as an attribute-value map. This method is only meaningful for hosts providing services, and not way points.

The third method (called `FORWARD-AGENT`) is for hosts that direct the agent to service-providing hosts within the region. Directory servers are the hosts that do this at the level of directing agents to regions. But once the agent is inside the region, it has to know which hosts to go to find the services that it needs for completing its mission. Depending on the size of the region, this function could be performed just by one host within the region (this could be entry point or any other host) or it could be performed by a number of hosts each of which contains information about a few hosts within the region. Thus the agent might have to go to one or more way points in order to find out about the possible hosts that it could visit. The number and nature of way points in a region is determined by its internal organization. `FORWARD-AGENT` is implemented by the hosts that have information about the properties of other hosts. When an agent arrives at a host, it can invoke this method passing its mission and policy as arguments. The host will return a list of Uniform Resource Names (URNs) ([43], [39]) corresponding to hosts that match the agent's mission. The hosts that are returned could include both service hosts and other way points where the agent can get further information about specific service hosts. These are put into the itinerary of the agent and it will then visit them in order. If the host only provides services and does not have information about other hosts, it can return `NULL` for this method.

An issue that could arise with this kind of routing of the agent within the region is that the agent might keep getting randomly forwarded by a malicious region and not get to a useful host so that the agent can spend more time in the region and possibly be charged more. The argument against this kind of behavior by a region is that it is in the best interest of the region to provide good service to the region in order to ensure future business from the region the agent originated in. The agent may keep track of the number of hosts it has to visit in order to get to one that provides it the service it needed. If this number is very high, the agent can give the region a bad rating and it could also include it in a list of regions that have bad performance. The ratings and list could be circulated to all hosts in the originating region of the agent and it could also be made public to other regions. Also, there could be an incentive for good behavior built into the commercial model of the system. For example, if the agent only pays for the services it receives, it is wasteful for a region to forward it around aimlessly unless it receives a service. Thus, although this kind of behavior is possible by a region, it can be discouraged by building the appropriate mechanisms into the system.

Another possible issue with the design as presented so far is that of a malicious agent that does not pass through the exit border way point of the region after using its services. In some cases, this might not matter much but in others, the region might want to charge the agent for all the services only at the end and if the agent does not pass through the exit point, the region will not be able to do so. The exit point is put at the end of the agent's itinerary in the region at the entry border way point itself. But the agent could bypass it if it did not want to pay. The following method can be used to prevent this from happening. The Uniform Resource Names (URNs) ([43], [39]) of the hosts are put into agent's itinerary only by the way points in the region (including the entry point that puts in the exit point). The way points include a flag along with the URN that says whether the host belongs to the same region. They encrypt and sign this information before putting it into the agent's itinerary. All the hosts in the region have the public key so that they can decrypt the itinerary to transfer the agent to the next host. Before transferring, they check whether the next host is in the same region. If it is not, they send the agent to the exit point of the region for any exit processing. If it is in the same region, they just transfer it to the next host. Thus, the agent is forced to go to the exit point with this method.

The last point to discuss about this part of the architecture is the failure scenario when the agent is inside the region. The failures could consist of link failures and node failures. If there is a link failure, the agent will not be able to get transferred to the next item it its itinerary. The possibilities in this case are for the agent to attempt to transfer a few more times and if still unsuccessful, to proceed to the next item it its itinerary or look for more hosts. If the destination node fails, then the situation is also the same as link failure. The other case is failure of the node that the agent is currently at. Depending on the type of failure, it may be possible for the system to recover from the crash and restore the agent. Also, some resources might need to be released and some external reconciliation might need to be done (for example, if the agent had pre-paid for some services and the node crashed without the agent having used the service, the money needs to be credited back to the agent). It depends on the application how the transactions that the agent was performing are reconciled and how much of the information is still available and how the resources are released. In case the machine crashed completely and the agent died, the region might want to inform the creator of the agent of the event. To do this, it would have to find out if there were any agents present at the node when it crashed. Logs kept by the region of the basic information about agents visiting it such as their URNs, creators, etc and those kept by hosts about the times of agent visits can aid in the process of recovery. The region can then use these logs to backtrack and find out which agents died with the node crash and what their status was at that stage.

This concludes the discussion on the functionality required of regions for their communication with directory servers, actions that occur at their boundaries and functions that the agent needs while it is inside them. The next section discusses in further detail the processes that happen when the boundary of a region is crossed.

3.1.5 Actions at the Boundaries of Regions

The previous section outlined the mechanism in place in the regions for processes to take place when their boundary is crossed. This section discusses the different kinds of events that can happen when activities touch or cross the boundary and the reasons for their occurrence. The boundary is the logical separation of a region from the rest of the world. The key characteristic of the boundary is that we can know when it has been touched or crossed, and hence it is possible for an action to occur, such as modification of some piece of state or transmission of a notification [40]. In the context of a service-providing region, agents will be the entities that will cross region boundaries when they enter regions to avail of their services and then exit them.

There are a few different elements in the system that are affected by the crossing of a region boundary by an agent. These are the places where changes can take place because of a boundary crossing. The first is the region that the agent is entering and specifically, the entry border way point it is entering through and also possibly at other places in the region that keep track of the boundary. There might be a change at the host that owns the agent or is responsible for the agent because it might want to be notified of the agent's actions such as entering a new region. Also, if the agent is entering a new region from an old one or in other words, it is crossing over from one region to another, the region it is leaving will also be affected by the crossing of the boundary. There might be state changes at the exit border way point or other hosts in the previous region that have to track the agents in the region. Lastly, the agent itself will be affected since it will have to change its own state to reflect its new position and also might have to notify other entities like its owner.

There are two main categories of actions that can occur at the boundary. The first is the process of access control by the region and the agent. This involves the evaluation of the agent by the region to determine whether it should be allowed to enter the region and the evaluation of the region by the agent to determine whether the agent wants to enter the region. The second category of actions could be classified as side-effects of the boundary crossing. These are actions that occur as a result of the first process and can include notifications being sent between any two of the entities that were discussed above or state changes in any of them. State changes could occur at the agent (for example, in its itinerary), at the region which the host is entering and at the owning host or region of the agent or a third-party responsible for the agent. Notifications could also be sent between any of these entities or between hosts within the region.

A region will use some criteria to determine whether an agent that is requesting access should be approved or denied. There are different criteria that the region could use: characteristics of the agent; characteristics of the owning host or owning region of the agent; and characteristics of the region that is performing the access control itself. The characteristics of the agent that the region might be interested in evaluating before it grants access could include its mission and its policy, its security credentials, its spending policy, etc. The region can analyze the mission to find out what kind of data the agent is collecting. It can analyze the spending policy to find out the form of payment the agent uses or what is the maximum amount it is willing to spend at the region. There could also be other factors in the history of the agent's life that the region might want to know about such as the region the agent is transferring from, the regions or hosts that it has visited in the past and what its spending history and pattern have been so far. All these points aid the region in learning about the past behavior of the agent so that its future behavior can be predicted.

The second criteria that the region considers are the characteristics of the owning host and region of the agent. The region might have a policy regarding from which regions and hosts it accepts agents and from which ones it does not, because of its own past experience or information from other sources. Granting access could also depend on whether the region has a billing or other contractual relationship with the originating region, on the terms of the contract and the behavior of the other region regarding the contract. Security credentials of the originating host and region and behavior of past agents from those sources are other factors that could be considered.

The last criterion that the region uses is its own state at the time access is requested. If the traffic within the region is too high, it might temporarily suspend access for any new agents. It could also look at the state of the network and services within itself at the moment. For example, if too many services are down, it might not be able to support any more agents. Also, it could check whether it still has the services that the agent is looking for.

From the above three categories, the specific criteria that a region will use will depend on its policy and features. The parts of information that are needed from the agent will vary from one region to another and will depend on the application. Also, it will vary from one agent to another whether it keeps track of any of this information and whether it is willing to present it to the region. The region could also use other sources for obtaining information about the agent. There could be a public repository that maintains information about various agents that are currently active in various regions or maintains information about hosts and regions that have sent out agents in the past. Alternatively, the region could request information from other regions and hosts by sending them notifications. It could inquire about the history and behavior of the agent either from regions that the agent has been to by asking it for its past itinerary or the region could have contracts with regions to mutually share information.

The other important part of the access control action is the set of constraints that can be applied to the agent upon boundary crossing as a result of the region's analysis. This means that the agent can be allowed into the region but with restrictions imposed upon its abilities or the kinds of actions it can perform while it is inside the region or conditions it has fulfilled before it leaves the region. This can happen if the region decides for example that it is safe to let the agent use certain services but not others. Restrictions could be imposed on the resources that the agent can utilize such that it is only permitted to go to certain hosts or avail of certain services. There could be a requirement that the agent has to go to a minimum or maximum number of hosts or services. There could also be restrictions on the maximum and minimum amount of time and money that the agent spends in the region or the amount of time and money it spends at a host. In terms of capabilities of the agent, it could be restricted to performing only some transactions or there can be restrictions on what it can do with the data that it collects from the region.

After the region has decided on the set of constraints on the agent if any, it informs the agent of these. Then the agent can decide if it still wishes to visit the region with those constraints. As mentioned before, this exchange between the region and agent of information to decide on the access and constraints could be modeled as a negotiation. Both parties could give more and more information about themselves at each stage after they have decided on the trustworthiness of the other party. This continues until they reach a decision about the agent's visit to the region. Thus, the process of access control encompasses a rich class of actions not only because of the diverse characteristics of the regions and agents and the information they have and require but also because of the process of negotiation which can lead to different results in similar circumstances.

3.1.6 Commercial Models for the System

In section 3.1.1, it was discussed that commercial activity is one of the reasons for creation of regions. Moreover, commercial incentives are essential for the existence and functioning of the entire system and its deployment. [10] discusses an economic market in mobile agent systems where agents purchase resources from host sites and sell services to users and other agents. In this section, we present a few different examples of commercial models for service-providing systems. The models entail the commercial relationships between the different elements of the system and how each element receives compensation in return for the services it provides, thus motivating its presence. There are two cases that are considered: the first one is when the service-providing hosts are independent entities that make money for themselves in return for their services and the second case is when the hosts are owned by the regions such that the region as a whole is the entity that is making money for its services.

Independent hosts

In this scenario, the hosts that provide services to agents are members of regions, but operate as independent commercial entities such that they seek compensation for their services on their own. In this commercial model there are four actors: the service-providing hosts, agents, regions and third-parties. Hosts are the providers of content, data and resources and seek to make money by providing these services. They charge a fee for their services which can be either given to them by agents or by regions. They need to have a pricing scheme for their services. They also might need to pay regions for being their members. Since the same host can be a member of multiple regions, it might have to pay multiple regions. The second actors, the agents[1], are the consumers of the services and are willing to pay money in return for using them. They pay to the hosts and/or to the regions. The third parties are entities that facilitate the financial transactions of the system and provide other financial services. Examples are banks, credit card agencies, credit-checking agencies, etc. They make money both from merchants which are regions and hosts in this case and from customers which are the agents. They can provide a variety of

[1] Agents represent the interests of their owners i.e. the hosts that created them. When we say that an agent is consuming and paying for the service, it is really the owning host that is consuming the service through the agent and paying for it. Therefore, although we refer to the agents themselves as the consumers throughout the work, it is important to remember that they are only the means used by some host and could be replaced by another mechanism to avail of services.

services to the system such as facilitating financial transactions (banks), giving credibility to the customers and taking on some of the risk making money through interest payments (credit card agencies) and providing information that helps in financial transactions (credit agencies).

Regions, the fourth actor in the commercial model, act as the brokers between the providers and consumers of services. They provide infrastructure such as way points and border way points to route agents through the network to the hosts. Regions could make money from the agents because they ease the search of information and provide services of their own (such as security). They can charge a premium for the quality of infrastructure and other specialized services they provide. Regions could also make money from the hosts by bringing more traffic to them (since regions are areas of concentrated interest, it would be easier for an agent to locate them). They provide services to hosts such as screening the agents for security when they enter so that the hosts can be guaranteed a certain level of protection when they join the region. When a host joins the region, it agrees to the particular service model of the region and pays for the services it uses. Thus, the regions provide different sets of services to hosts and to agents and can charge both of them for the respective set of services. In essence, regions act as referral services, referring agents to hosts. Other examples of referral services are the telephone yellow pages and apartment locator services that act as sources of information and may do quality checking. Regions can charge the customers (agents) for using the referral service and the advertisers (hosts) for advertising and providing referrals which generate business for the advertiser (since the host is an independent entity that makes money on its own).

All models in this scenario where the hosts are independent will involve the agent paying the region and/or the hosts. There can be different relationships however between the hosts and the region and how they handle monetary exchange between themselves. There can be two cases: the hosts pay the region and the region pays the hosts.

In the first case, a host pays the region for one or more of several services. Being part of the region helps bring more traffic to the host because of the infrastructure that the region provides to advertise and search for services. The region also provides services to the hosts such as security, privacy and performance guarantees. Since the host is paying the region, it will charge the agents directly for its services with the aim of making more than it has to pay the region. The region will receive money from the hosts and possibly from the agent and it aims at making that higher than the cost of infrastructure setup and maintenance and the cost of providing the other services. The amount that the region can directly charge the agent however should be low or none since the agent will already be paying the individual hosts for their services.

When the region is charging its member hosts, it could either treat all its members equally, that is, have a uniform payment scheme for all members or it could charge different members differently based on the number or quality of services that the host provides. There are various schemes that the region could use to charge the hosts. The region could have a *joining fee* that each member host pays when it initially joins the region. There could be a *periodic flat fee* in which the host pays a fixed amount every time period. There could also be a *periodic variable fee* in which the host is charged every time period depending on how much traffic it brought into the region (i.e. how many hosts visited that host for its services). Depending on the expense mechanics of the particular region, this could either be an inverse relationship wherein the more traffic a host brings, the less it has to pay to the region or it could be the opposite, that is, the more traffic a host brings into the region, the higher it has to pay. This depends on whether the region incurs higher expenses as more and more agents come in or whether the cost per agent decreases as more agents come in. The last payment scheme is *share of earnings*, where each host pays the region a fixed percentage of the money it makes from the agents visiting it. These are a few schemes, but the region could also have its own variation of any of them. Also, it can use an arbitrary combination of the above methods.

The second case of the commercial relationship between regions and hosts is when a region pays hosts for being its members. This relationship is economically feasible when the region collects a lot of money from the agents and distributes it to the hosts. Then, the hosts cannot individually charge an agent because it would already be paying a huge sum to the region (so that the region can in turn make enough to pay the hosts). The region could charge either a constant amount to each agent or it could charge an amount proportional to the number of hosts that the agent has visited while it was in the region. The region could also give each host a fixed share every time period or it could give each host a share proportional to its popularity (i.e. the number of agents that visited the host in the time period). In order to make a profit, the region will have to charge the agents more than what it pays the hosts and its costs.

Thus, in this model, the region collects money from agents and in effect, distributes it among the hosts. Hosts do not charge agents directly. The region could use different schemes for paying the hosts. A region could give a host a *joining payment*, which is a fixed amount paid at the beginning when the host joins the region. There is no further exchange of money between the host and the region. Essentially, the host has been ‘bought’ by the region or ‘rented’ for a period of time for a fixed fee. The region could also give a *periodic fixed payment* to the host in every time period for being a member of the region or a *periodic variable payment*, where the amount depends upon the traffic the host has brought into the region. Lastly, the region could use a *share* *of earnings* payment scheme, where it distributes a fixed percentage of its earnings in every time period among all its members.

Hosts owned by regions

The second commercial model for the system is for the case when the hosts are owned by the regions and are not independent commercial entities. The region is the only entity that is making money as a whole and it operates the hosts which is part of its costs. There are three actors in this model: regions, agents and third parties. Regions are the providers of data, content, resources and infrastructure through their member hosts. They charge agents for the services they provide. Agents are the consumers of the services and pay regions for them. They could use several different methods to make the payment such as by cash, through contracts between regions or through third-parties like banks. Third parties, as in the previous case, facilitate financial transactions and make money from the merchants as well as the customers.

A region could use several schemes to charge an agent. It could have a *flat charge* where every agent is charged a constant amount regardless of how many hosts it visits or how many services it uses. The region could use to impose this charge either at entry or exit. The region could also use a *variable charge* where the agent has to pay an amount depending on the number and type of services it actually used. The region could use different metrics for measuring the 'quantity' of services the agent has used such as the time it was in the region, the number of hosts it visited, the average time spent at each host, etc. If the region only has a variable charge, the agent would be charged nothing if it just entered and exited the region without using any services. Different regions could use other schemes or combinations of the above schemes depending on the application.

This concludes the discussion on commercial models to motivate the existence of service providing regions and hosts. The section outlined a few models but there are many others possible. The agent will be informed of the particular scheme a region uses either through the requirements section of the advertisement of the region or at the entry point when the agent is attempting to gain access to the region. The agent can analyze whether it agrees with the commercial model of the region and that would be one of the factors affecting its decision to visit the region.

3.2 Directory Server Network

The next important component of the architecture is the directory server network, which connects the regions that provide services to entities interested in using these services. Regions send a composite announcement of all their services to the directory servers. Entities that are interested in using any service create mobile agents and encode their service needs and policies into the agents. Agents start out their journey through the system at the directory servers where they can get information about the services that regions are providing. Directory servers perform matching of the agents' needs with the service announcements that they have and refer the agents to appropriate regions if they find a match. Agents then visit the regions to use the services. If agents need to visit more regions, they can go back to the directory server network to find out about other regions that might be interesting to them.

Directory servers are an essential feature of the system because without the directory servers, the agents would have to visit each and every region to find out the services they offer which is not feasible in a large scale system. Also, directory servers give the agents other important information such as the requirements of the region for the agents that it permits and the border points where they can go to enter the region. The information about all the regions is distributed among a number of directory servers that coordinate as a network to provide results to agents when they query about a service. The exact number of directory servers depends on the size and nature of the system. In essence, the directory server network behaves as a search engine for services which an agent can query with services it is interested in finding and it will receive search results that match its query. The search results are a set of regions which the agent can then visit individually to use the services.

The directory servers form a spanning tree network which means that each server has one neighbor in the network. The regions are distributed among the directory servers. Each directory server is responsible for some regions – it carries the advertisements of those regions and if an agent comes to it, it checks if any of them contain services that the agent wants. The main functions of the directory server network are to collect service advertisements from regions and refer agents looking for services to appropriate regions by matching their objectives with the region advertisements. The first two sections will give an overview of how the two functions are performed by the directory server network. After that, the internal architecture and functioning of the network will be described.

3.2.1 Advertising for Regions

This section describes the process of advertising a region's services to the directory server network. Each region has one directory server that it advertises with. When the region comes into existence, it sends a message (called `REGION-TO-CR-GET-DS`) to a central server called the *Central Resolver* that is part of the directory server network and knows about all the directory servers[1]. The job of the *central resolver* is to find a directory server that will be responsible for the region. It finds a directory server for the region and sends a message (called `CR-TO-REGION-GET-DS-REPLY`) with its URN to the new region. This is now the designated directory for the region and it sends all its advertisements to this server. The directory server network itself might replicate the service ad data, but the regions do not advertise to multiple directories. Since the central resolver is responsible for assigning directory servers to

[1] We assume that the new regions are pre-configured with the URN of the *Central Resolver*.

regions, it can query the directories about their load and perform load-balancing for the network. Thus, the regions do not need to concern themselves with load-balancing or replication of their advertisements. They find out their directory server at the beginning and after that they can continue to advertise with it unless there is a failure scenario. If the directory server fails, the region sends another `REGION-TO-CR-GET-DS` message to the central resolver to find a new directory server.

The entry for each region in the directory server has four parts: name of the region, aggregated services of the region, agent requirements/region properties and the entry point of the region[1]. The region uses some mechanism to perform aggregation of service descriptions of all its member hosts. The aggregate has the format of attributes and values describing the services. Each attribute could have multiple values since different hosts within the region might have different values for the same attribute. Different hosts in the region can send the attribute-value map to the directory server in different messages – it does not have to be the same host every time. The directory server will do the matching of the agent's mission to the advertised attributes and all their associated values. The requirements for the agent consist of the information that the agent needs to know before attempting to gain access to the region and the relevant policies of the region. This could include the language which the region communicates in, the credentials required of the agents and so on. This part of the message could also contain the invariants of the region and any other characteristics that it wants to advertise. If an agent does find a match against a region at a directory, it can use these requirements and properties to check if it would actually be permitted to enter the region or if it does not meet the requirements. The last part of the advertisement from the region to the directory server is the identifier for the entry point for

[1] The advertisement message was discussed in detail in section 3.1.4.

the region. Since a region could use different entry points at different times or for different agents, the identifier is resolved to an actual location at the time an agent wishes to visit the region (which could be from a directory server or from another region). The region has an algorithm to pick an entry point when an agent wishes to visit it and it provides resolution of the advertised name to the correct host.

As mentioned in section 3.1.4, the directory servers maintain a soft state for the advertisements. When they receive a service advertisement from a region, they store it as an entry with a timeout value. After the region has sent the advertisement (`REGION-TO-DS-AD-MESSAGE`) once, it can send it as often as it wishes depending on how frequently its services are changing. If the advertisement is not changing often, the region sends *Keep Alive* messages to the directory so that its entry is not timed out. The *keep alives* should be sent by the region at least once every timeout period. The directory server maintains the same advertisement for the region until it receives a new one. If the directory does not get a message (either `REGION-TO-DS-AD-MESSAGE` or `REGION-TO-DS-KEEPALIVE-MESSAGE`) from the region in two consecutive time periods, it assumes that the region has died and deletes its entry. In this way, regions do not need to be explicitly deregistered and this also takes care of failure scenarios. When a directory server is deleting the entry of a region, it sends a `DS-TO-REGION-DELETING-ENTRY-MESSAGE` to the host in the region from which it received the last message saying that the region's ad is being deleted. This is useful because if the region's ads or keep alives did not reach the directory server for any reason, it is now aware that its entry has been deleted. If it is still functioning, it can send a new `REGION-TO-DS-AD-MESSAGE` to the directory. Thus the system can recover from failures of ad delivery. If the directory server is overloaded and needs to delete the entry of a region, it can send a similar message to the region.

In this case, the region would have to go back to the Central Resolver in order to find another directory server.

3.2.2 Locating Services for Agents

Agents are created with the purpose of using services which could include collecting data, getting content on a particular subject from a remote host, using resources such as memory to perform a computation on another machine and using data services on other hosts to process data and derive results. In order to use any of these services that regions (or specifically the member hosts within regions) provide, the agent first has to search and locate them. The search is based on the agent's mission that was given to it by its creator. The mission is represented in terms of a set of objectives each of which contains combinations of attributes and values that describe the desired services. The role of a directory server is to look at each objective in the mission of the agent and match it with the service advertisements of regions that it contains. Based on the results of the matching, it suggests regions that the agent should visit for separately accomplishing each of its objectives.

Since the information about the regions is distributed among the directory servers, the agent will possibly need to go to multiple locations to find out all the information it needs. Therefore, it is not enough for a directory server to refer an agent only to regions whose ads the directory contains because if no matches were found at this directory, the agent would have nowhere to go. In addition, there might be matches in regions that were advertised at other directories which the agent would want to know about. There might be matches for only some of the agent's objectives at one directory and it would need to go elsewhere to find matches for its other objectives. Thus, directory servers should also be able to refer agents to other directory servers. In order to accomplish this, other than region ads (called `REGION-ADS`), each directory server also carries ads of a few other directory servers[1] (called `OTHER-DS-ADS`). Therefore, it can refer the agent to regions and to other directory servers.

The initial itinerary of the agent consists of a single host, which is the directory server that it starts at. The agent is autonomous, so it figures out the rest of its path as it goes along. It is pre-configured with the first host by its creator. The creator finds the directory server from its region – the region could use the same directory server that it advertises with or it could use another one to start off its agents. The process of service location begins with the agent arriving at the directory server. The directory server looks for the attributes from each of the agent's objectives in the `REGION-ADS`. If it finds any of the attributes, it looks at the values of those attributes. If they match with the values that the agent requires, a match is found. At the end of this process, the directory would have found 0, 1 or more matching regions for each of the agent's objectives. The matching regions are sorted in the order of their usefulness to the agent which is computed from its preferences for the various objectives. The entry points of all matching regions are put into the agent's itinerary in this order.

After this, the directory server will look through the `OTHER-DS-ADS` and repeat the same process of matching, ordering and adding them to the itinerary. If no matches are found in the `OTHER-DS-ADS`, the directory puts the URN of its neighboring directory server after all the region entry points. This is necessary so that the agent has a directory server to go to after it has finished visiting the current set of regions and still not completed all its objectives. After visiting the first set of regions, the agent will go to the directory servers in its itinerary. The same process

[1] The details of the other ads a directory server carries and what they contain is described in section 3.2.4.

is repeated at every directory server. Thus, the agent is either forwarded to directory servers that have matches or it is forwarded down the spanning tree to find about other regions that the current directory server does not have information about. The agent updates its mission after visiting every region to reflect the services it has already used and the ones it still needs to locate. The process of searching for services continues until the agent has used all the services that it wanted that were present in the system.

3.2.3 Architecture of the Directory Server Network

The directory server network is a self-configuring overlay network. The directory servers share the tasks of advertising information of all the regions and locating particular services from these advertisements for agents. Each of them is responsible for advertising a few regions. They match the objectives of agents that come to them with the service announcements they are carrying. But they also need to have information about the other directory servers in order to forward the agent to other locations where it may find relevant information. In order to achieve this, they need to organize themselves into a network by establishing neighbor connections among themselves. The servers form a *Spanning Tree* network in which each new node connects itself to one other node in the network. In a spanning tree, every node has one out-neighbor – the one that it connects to. A node may have multiple in-neighbors if more than one node is connected to it. The neighbors can be chosen based on some metric that the network is trying to optimize. For example, in a network based on round-trip latency as the metric, each node would try to find the neighbor that is closest to it in terms of the time it takes to send a message from the node to its neighbor. Any messages that need to be exchanged in the system are passed along the spanning tree, which means each node passes it along to its one neighbor.

In the directory server network, when a new directory server joins the network, it discovers the other directory servers that are already part of the system and then chooses a neighbor from among them. Thus, the neighbor connections are formed dynamically and in a distributed way by each node that joins the system. In order to choose a neighbor, when a directory server joins the network, it needs to know the locations of the other servers so that it can send messages to them and measure the metric that the network is based on. There is a well-known entity in the system, called the *Central Resolver*, which maintains the locations of all the currently active directory servers. A new directory first obtains this list from the CR (*Central Resolver*). It then sends a message (called `DS-TO-DS-PING`) to each of the other directory servers that are currently in the system to obtain the metric from each of them. Once it receives their replies, it compares all of them to find the one with the maximum or minimum value depending on what property the network is based on. This node is made the neighbor and the resulting topology is a spanning tree[1]. Once the neighbor has been picked, the node forwards any messages that it needs to send through the system to its neighboring node. The spanning tree network of the directory servers is based on the 'resolver network' from the Intentional Naming System (INS), which is a resource discovery and service location system [1].

In order to maintain a list of active directory servers, the Central Resolver needs to communicate with them regularly. The directory servers send *Keep Alive* messages to the CR

---

[1] Each node except the first one in the system has one node ahead of it in linear order (assuming the nodes are joining one at a time). Thus, the resulting graph is connected. Also, since each node has exactly one neighbor, the number of links formed in an n-node network is exactly $n-1$. Any connected graph with n nodes and $n-1$ edges is a tree.

(called DS-TO-CR-KEEPALIVE) at least once every time period of length equal to the timeout value of directory server entries at the CR. In the keep alive message, the servers also report their current load to the CR. As discussed in section 3.2.1, when a new region is formed, it first goes to the CR to find a directory server. The CR uses the load of directory servers reported in the keep alive messages to assign new regions to them. It chooses a random directory from all the ones with the lowest load and gives its URN to the region. Thus, the keep alive messages serve the purpose of telling the resolver that the directory is alive and of reporting its status so that load-balancing can be performed.

The CR also maintains a soft state map of the neighbors of all the nodes in the system. Thus, after a directory joins the system and finds its neighbor, it sends a message to the CR (called DS-TO-CR-NEIGHBOR) to inform it of its neighbor. The neighbor map is needed so that in case of a node death, the CR can inform the node's neighbors that they need to find another neighbor. Such mechanisms make the system robust in failure scenarios. If the CR does not receive a DS-TO-CR-KEEPALIVE from a directory server for 2 consecutive time periods, it assumes that the directory is dead and deletes its entry. Thus, directory servers do not need to explicitly deregister and also in case of a failure, their entry is automatically deleted and no regions are assigned to them. When the CR deletes the entry of a directory server x from the system, it looks for all the nodes whose out-neighbor was x by doing a reverse lookup in the neighbor map. It then informs these nodes that x is dead by sending them a CR-TO-DS-NEIGHBOR-DEATH message which means that they need to find a new out-neighbor. They can do this by repeating the process they followed when they joined the network for the first time: query the CR for all active directories, sending them pings to make measurements, choosing a neighbor and reporting it back to the CR.

This section described the process of formation of the directory server network and the role of the Central Resolver in the process. It also discussed how the network is made self-organizing and dynamic through soft state message exchanges. A self-organizing network also has to protect itself against failures. There are mechanisms in place that help the directory server network recover from node failure such as the CR informing nodes of their neighbor's death. The CR could be a potential vulnerability since it is a single point of failure. However, this should not be a major concern in general for the following reason. The CR is mainly used when new directory servers and new regions wish to join the system. In general, the addition of new regions or directories would happen on a relatively large time scale and therefore, the potential for the CR becoming overloaded is low. Even if the CR fails, the only effect would be that new regions or directories will not be able to join until it is restored, but the functioning of the rest of the system is not affected. The CR can be replicated for fault-tolerance. There should be some redundancy built into the directory servers as well in order to protect the data that they hold (i.e. the service advertisements of regions). For this purpose, the directory servers could be replicated or some of their data could be stored in multiple locations. It is sufficient in most cases to maintain a weak consistency between the replicated data. A partial replication could be performed where only some of the data is replicated in a dynamic way depending on factors such as its popularity, the load on the server, etc. The replication would be hidden from the regions and encapsulated in the directory server network so that each region would only have to communicate with one directory server for its advertisement.

3.2.4 Exchange of Information among the Directory Servers

The itinerary decision process of the agent consists of charting out a path through the directory server network with the help of information from the directory servers. If scalability was not a consideration, each directory server could know about the ads of all the other directory servers, so it could direct the agent to the exact location that it needs to go to find a service. This is the approach followed in INS [1] where every service advertisement is propagated to the entire network of *resolvers* through the spanning tree. Thus, every *resolver* knows about every service in the system and when it receives a request for a service, it can directly point it to the original *resolver* that the service is attached to. This solution is not scalable for large-scale systems because each directory would need to contain advertisements of all the other directory servers and for our system, this means that each directory server would know about all the possible regions. The other extreme would be for the agent to take a random walk through the directory server network. This is the scenario when the directory servers do not have much information about each other's contents and therefore randomly forward the agent through the network. This solution, although scalable, has the problem that it might take the agent a long time to locate all the services that it needs since it is operating with very low information. The solution we propose is midway between the above two extremes and strikes a compromise between the scalability of the system and the efficiency of the agent's operation. Each directory server carries the ads of a few others, so it can possibly give the agent some locations where it can find matches. But there is also an element of randomness as the directory servers do not know about the advertisements of the entire network. The agent will have to be forwarded blindly at times when no matches are found in the information that a directory server has[1].

It was pointed out in section 3.2.2 that other than the advertisements of regions, directory servers also carry ads from other directory servers (called `OTHER-DS-ADS`). Each directory does not carry `OTHER-DS-ADS` of all the other directory servers in the network because that is not scalable. It is only required to carry the ad of its neighbor, but other than that, it only has to carry as many as `OTHER-DS-ADS` as it has excess capacity for. This section describes how the `OTHER-DS-ADS` are created and how they are exchanged among the directory servers.

Firstly, each directory server aggregates all its region advertisements. The directory only needs to aggregate the service announcement part of the region entries, and not the name of the region or its agent requirements or its entry point. This is because the itinerary decision of the agent only depends on the service announcement of the region. It can be forwarded to the appropriate directory server with all the detailed information once the services match and the exact region with which the agent matched can be found there. Thus, the region aggregates the services over all the regions and advertises that as its ad message. It is possible that the directory server does not include each and every service of the regions in its advertisement. It could filter out some of the less important properties if it has some way to learn which ones they are. Learning algorithms could be used to learn over time which services of the region can be filtered out. Also, encoding could be used to shorten the length of the advertisements [25]. Various techniques such as these could be used to reduce the load on the system by making the advertisements from directory servers more specific. Another point to note is that only the

[1] The agent is actually forwarded to the neighbor of the directory server in the spanning tree in such situations.

regions ads are included in the aggregate and not the ads from other directory servers that this directory might be carrying.

The next step is for each directory server to send the aggregated advertisement to its neighboring node in the spanning tree. This is sent as a `DS-TO-DS-NEIGHBOR-AD` message. It contains the URN of the directory server and its aggregated service announcement. The service announcement itself is a mapping from attributes to values aggregated over all the regions. The directory server can decide how often it wants to send this ad. If it is receiving frequent updates from its regions, such that its ad is changing often, it might perform the aggregation often and send it to the neighbor. Or it might choose to wait and not send very frequent updates if the changes are not that significant. The minimum requirement is that the advertisement has to be sent at least once every time period before it is timed out at the receiving server. This communication is similar to the one between a region and its directory server. The timing out of these ads also ensures that any stale entries for regions or directory servers that are no longer functioning are cleaned out periodically.

Each directory server will possibly have one or more in-neighbors, i.e. neighbors that send their messages to it. It will receive `DS-TO-DS-NEIGHBOR-AD` messages from all of them. Upon receiving an ad, the directory server puts in an entry for it in the `OTHER-DS-ADS`. The forwarding location for this entry would be the URN of the directory server that it came from instead of the entry point of a region. When an agent comes to the directory, it matches the objectives of the agent against the `OTHER-DS-ADS` as well as the `REGION-ADS`. If a match is found with another directory server, its URN is put into the agent's itinerary so that the agent can visit it for a detailed matching and for finding out the exact region that has services that it is interested in. The process of forwarding of neighbor ads and other ads is illustrated with an

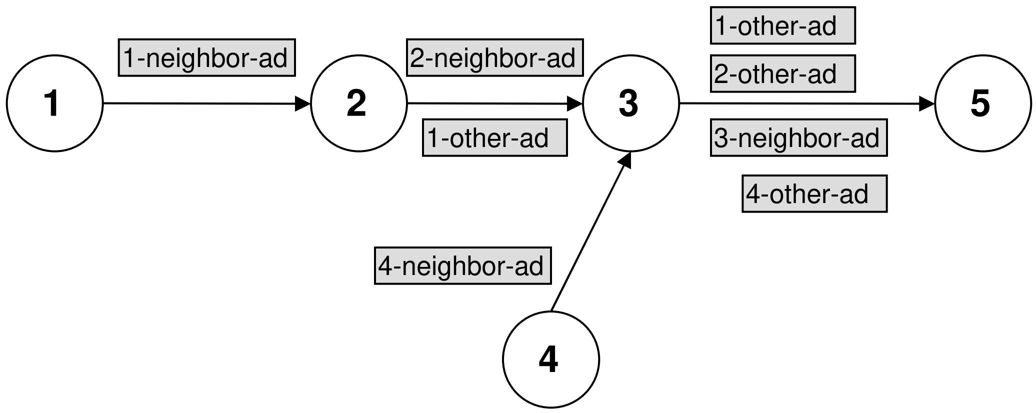


Fig 1. *Exchange of ads between the directory servers. Circles denote the directory servers. Arrows denote the directions in which ad messages are sent.*

example spanning tree in *Fig 1*.

Each directory server also forwards the ads from its in-neighbors on to its own out-neighbor as `DS-TO-DS-OTHER-AD` messages. The form of this message is similar to the `DS-TO-DS-NEIGHBOR-AD` message. The only distinction between these two kinds of messages is that each directory has to keep its neighbor's entry, where as it can choose if it wants to keep entries of other directories or not depending on its capacity. The `OTHER-DS` entries also have a timeout value and so the `OTHER-DS-ADS` should be forwarded at least once every timeout period to prevent them from getting deleted if they are still valid. When a directory server receives a `DS-TO-DS-OTHER-AD` message, it can decide whether it wants to add the entry to itself or not. Each directory server has a metric for the maximum number of other entries it can have that depends on its processing power and its current load. It can change this metric over time. This is the metric that the directory server uses to determine whether it should add another entry or not. Thus, each directory performs matching for an agent with all its region ads and neighbor ads. Depending on its remaining capacity, it can also match against ads of other directories. It will also forward `OTHER-DS-ADS` on to the network depending on its capacity to do so. Thus, the directory servers are not forced to forward or match against each and every ad they receive if they are busy. This maintains the scalability of the system and makes the directory servers more efficient.

If a directory server has reached its maximum capacity for `OTHER-DS-ADS` and it receives another `DS-TO-DS-OTHER-AD` message, it has a choice for what it can do. One option is to ignore the message completely and not process it since it has already reached capacity for what it can process. The other option is to replace an entry that currently exists with the new entry so that the total number is still the same. The replacement scheme is up to the directory server – for example, it could eliminate the least recently used entry, the oldest entry, etc. Also, if the directory server is operating at capacity and cannot process any other advertisements, it could send a message to the sender asking it to reduce the rate at which it forwards `OTHER-DS-ADS`. The sender can then perform a backoff and increase the rate it sends messages gradually.

Also, another issue that might arise with this mechanism is that a directory could receive two different `DS-TO-DS-OTHER-AD` messages for the same directory server from two different neighbors. It is possible that even though one of the ads is received later, it might actually be outdated if it has been forwarded through a longer path and the one received previously could reflect the latest information. To prevent this from happening, time stamps are included with all advertisements. When a directory server aggregates its information and first sends out an ad, it puts the current time stamp on it. This time stamp is carried through for all the neighbor-ad and other-ad messages. Whenever a directory server receives an ad for a URN that is already has an entry for, it checks the timestamps of the two entries and only replaces the original one if it is older. If the new entry has an older time stamp, it is discarded and not forwarded on in the network. In this way, the network will eventually converge to the latest copy of the ads and the older ads do not need to be explicitly removed.

Another optional functionality that could be added to the directory server network to improve its performance is caching. With the caching mechanism, the agent is not randomly forwarded to other directory servers, but its query is sent through the network and the answer is given to the agent at the same directory server. When a directory server receives a query from an agent that does not produce any matches from its own region and other ads, it sends the query to its neighbor while the agent waits at the first directory for the answer to come back. The query is forwarded as far as needed to find a match and the answer is returned back along the path that the query was sent. The contents of the answer are cached by every directory server on the way back. Once the directory that originally sent the query receives the answer, it caches it and also gives the response to the agent that was waiting for it. If it does not receive a response in a certain amount of time, it can assume that the answer was not found. In this case, it is up to the agent what it wants to do next – it could either try to send a query again or it could go to a different directory server or it could go back to its creator. The benefit of the cached entries is that in the future, if an agent queries for the same services, the directory server knows where they are located and can directly answer the agent without having to send out more query messages into the network.

This section described how the directory servers exchange information by forwarding their aggregated advertisements down the spanning tree. Each directory will only carry as many other entries as it has excess capacity for. This ensures that the system does not get overwhelmed with messages as it grows and maintains the balance between its scalability and the performance of service location. This also concludes the section on the functions and architecture of the directory server network. The next section discusses the purpose and functioning of mobile agents in the architecture.

3.3 Autonomous Mobile Agents

Mobile agents represent the consumers of services provided by regions and hosts. The entities in the system that wish to avail themselves of services launch a mobile agent into the system. A mobile agent is an entity that combines both program code and data and moves around in a network from one host to another and executes on them [49]. A mobile agent can be given a list of hosts to visit. This list is called the *itinerary* as the agent will travel to each of destinations in order and then possibly return to its creator with the results of its execution. The agent can also be programmed to perform common or specific tasks at each of these hosts. An autonomous mobile agent is one that can determine its path through the network on its own once it is given an objective at the time of its creation.

An autonomous agent does not need to be given an entire list of addresses to visit when it is launched. It is only given a high level description of its purpose. As it moves around in the network, it looks for information about the services that it needs for achieving its objective. Whenever an agent learns about a host that is providing such a service, it adds the location of the host to its itinerary so that it can visit it later. Thus, the agent constructs its itinerary on its own based on the information it discovers. There is a decision process that the agent goes through to decide which hosts it should visit. For our purposes, we have chosen a simple algorithm. However, the decision process can be made as sophisticated as the creator of the agent wishes in order to make the agent collect some data, perform a computation, use a service, etc. Additionally, the agent could also deliver data or provide an action to a host. It also has to decide where in the itinerary it should put a host. This depends on the rules that the agent has for deciding what properties it should prefer and how critical the service is to the agent's mission.

In order to support the operation of autonomous agents, there is an infrastructure needed through which an agent can gather information about services as it moves through the network. The agent needs to be able to enquire about services and also enquire about hosts that have information about services. This is where the directory servers and regions come in. Regions collect and combine information about a number of service-providing hosts and advertise them to the directory servers. An agent can go to a directory server to find out about regions and also about other directory servers. Once an agent is inside a region, there is infrastructure in the region to direct it to appropriate hosts.

There can be other models of communication that could be used for achieving the purpose of consuming services such as packets, remote invocation and continuations [42]. The advantage of using the mobile agent model is that it enables the consumers to avail themselves of a larger class of services rather than just data collection because the agents are programs and not static messages, and hence they can actually perform computation at remote hosts. They can do processing on the servers they visit, hence making use of services and resources in the network. They also facilitate asynchronous and loosely coupled communication between the senders of the agents and the providers of services, making the architecture more flexible [23]. Also, using mobile agents allows for a richer communication paradigm because agents can go to multiple hosts in order to find all the information they need in the same trip and collect and aggregate this information if desired by the creator. It is important to note though that mobile agents could be replaced by any other model and the architecture could still be used to perform the same functions with the appropriate modifications.

3.3.1 Components of Mobile Agents

The main function of autonomous mobile agents in our system is to locate and use services on their owners' behalf. A mobile agent is composed of the following components: an *itinerary*, a *mission*, a *policy*, a *properties file*, and procedures to compute the itinerary. *Fig 2* shows all the components with examples for each. The *itinerary* is the current list of hosts that the agent plans to visit. The initial itinerary of the agent consists of only one directory server that the agent will visit first. If the creator of the agent knows about more directory servers, it could put those into the initial itinerary as well so that the agent can visit them all. When the agent goes to a directory server, it gets referred to some regions and possibly some other directory servers. It puts the URNs of the region entry points and the directory servers at the end of the itinerary to visit them in the future and continues to the next host in the list. The agent may also remove items from the itinerary if it does not need to visit them any more. This can happen, for example, if it has already found a piece of data that it was looking for. Thus, the itinerary is constantly modified by the agent as it goes along.

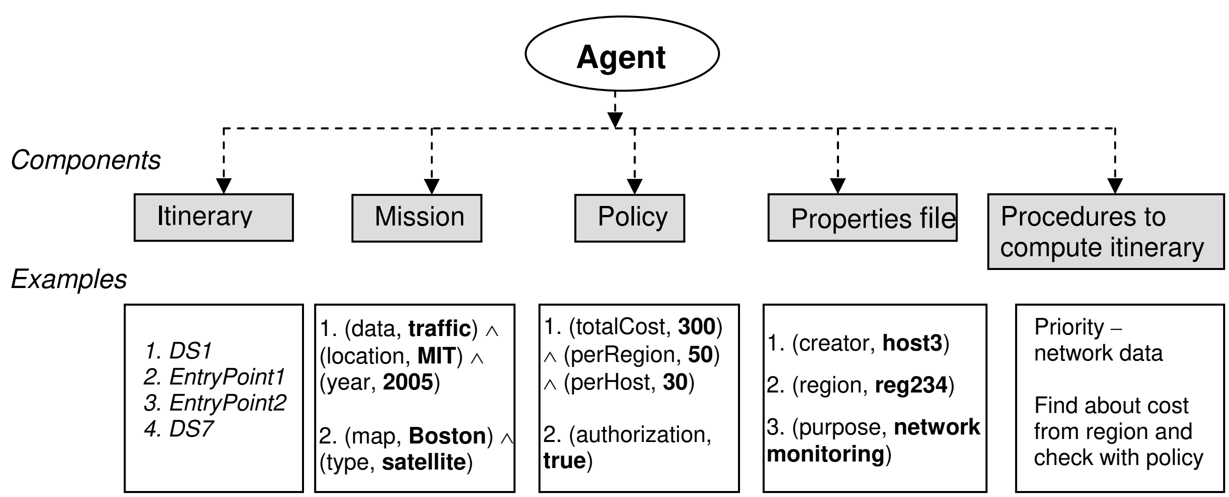


***Fig 2**. Components of a mobile agent. The five components are shown with examples for each below them for an agent.*

The *mission* of the agent is given to it by its creator. The mission expresses the intent of the creator in launching the agent by describing the services that it needs to use. On a high level, it can be considered as a set of conditions that should be met for the mission to be fulfilled. In the implementation of the mission, the conditions are represented as attributes and values[1]. When the agent visits a host whose service advertisement consists of the same attributes and values, then the agent would have fulfilled that part of the mission. Thus, the mission is subdivided into smaller independent objectives – each objective represents one service that the agent wants to use and it consists of one or more conditions that together represent that service. Each condition itself is an attribute-value pair and the objective is a boolean combination of the conditions[2]. For each objective to be completed, the entire boolean combination of conditions has to be true at the same host. The agent can independently complete the different objectives at different hosts. The mission as a whole is not modified by the agent, but it keeps track of the objectives that it has already completed at any point and the ones that it still needs to complete. Thus, after visiting each host where it uses some service, the agent updates the list of services that it is still looking for.

The *policy* is a set of guiding principles for the agent to follow when it is trying to achieve its mission. The policy consists of general constraints that do not relate to a specific objective, but the agent would prefer or require while going about its mission. They could be characteristics that the agent wants the regions and hosts it visits to have. For example, the security requirement of an agent is something that would be incorporated into the policy. If an agent expects a certain level of security in its operation, it would prefer to go to hosts that

[1] We have used a simple general-purpose attribute-value representation for the mission as well as the service advertisements. It can be imagined that this part would be replaced by a more sophisticated language for specific applications.
[2] See section 3.4.3 for an example of a *mission*.

provide it that security. It is something that pertains to all the functions the agent performs and is not directly part of the mission. Additional requirements and preferences for the agent's operation such as these are included in the policy. The agent can decide whether it should visit a region or not on the basis of its policy. The policy is also represented in terms of attributes and values. One of the components of a region's advertisement to the directory server is the set of requirements, characteristics and policies of the region that are relevant to an agent wishing to visit it. The agent would look for its policy attributes in this section of the advertisement after its objectives gave a match with the region's services. If its policy agrees with the region's properties, it would include it in the itinerary. Also, the agent would check its policy when it is at the entry point of the region and also before it decides to use a service at a host.

The *properties file* of an agent contains any public information that the agent wants to disclose about itself, its creator and its owning region. The purpose of the properties file is to make information available to a region when the agent is attempting to gain access to it. The region uses the properties of the agent to decide whether it will grant or deny access. For example, a region might not want the data collected from it to be used in a commercial product. Therefore, it would want to know from the agent what kind of use the data that it collects is put to. It could also be used by hosts to make sure that the agent has certain characteristics that they require the consumers of their services to have. The properties do not apply to a specific service of a region or to a specific objective of an agent, but in general to the characteristics of the exchange between the agent and the region or the host. The properties file is also expressed in the same attribute-value language so that a region or host could look for the value of a specific attribute that it cares about. The properties file is presented by the agent upon request to the region at the entry point and to a host before using its service. As mentioned in section 3.1.4, there could be a negotiation during the entry so that the agent could reveal parts of the properties file in stages. Thus, each agent would have its own mechanism for how it wants to use its properties file.

The last component of the agent consists of the procedures that are used to compute the itinerary of the agent. The itinerary decision process involves selecting the regions to visit (at the level of the directory servers) or selecting the hosts to visit (at the level of regions). It also involves ordering the matching regions or hosts before putting them into the itinerary so that the agent can visit them in an order that would be most efficient for its purposes. The decision process is thus a function of the agent's mission, its policy and any other considerations that it has for selecting and ordering items in the itinerary. The agent needs to be programmed with an algorithm to do these tasks. For example, in the example implementation discussed in Chapter 4, each objective in the agent's mission also has a numerical preference value associated with it. This represents which objectives are more important for the agent to complete and which are less important. When the agent finds 2 or more matching regions, it first looks through their services to find how many objectives each of them match against. It then computes a weighted sum for all the objectives that can be completed at each matching region based on the preference values. Thus, the agent creates an ordering such that it gives preference to regions that are most useful for completing its important objectives. Another example is that an agent might have a limit on the amount of money it can spend in a single region according to its policy. It will look at the region's properties and if the fee of the region is higher than its limit, it might decide not to visit it even if the region has services that the agent is looking for. Thus, agents can have a variety of factors that they can consider for selecting and ordering regions. The architecture does not impose any such factors, but leaves it up to the creator of the individual agent to decide on what is important for them. Therefore, each agent will have some form of an itinerary decision procedure which is derived from its mission, policy and other factors.

3.3.2 Life Cycle of an Agent

This section will discuss the path that the mobile agents take through the system, the role of the various components discussed above in their life cycle and their interactions with the other entities in the system including directory servers, regions and hosts. The life of an agent starts when it is launched by its creator. The agent is initialized with the mission, policy, properties and itinerary decision procedure. The initial itinerary of the agent consists of the URN of a directory server that the creator knows about. The creator can use the directory server that its region advertises with or could find out about directory servers from some other source (like the *Central Resolver*). If the creator host knows about more directory servers, it could put all of them into the agent's initial itinerary. The initial address is necessary because the agent needs a place to start finding out information to build its itinerary.

When the agent arrives at a directory server, it makes its mission available to it. The directory uses this mission to perform matching against the advertisements of regions that it currently has. The overall idea is to look for the attributes in the agent's mission and test if the desired values are present in any of the ads. The mission is subdivided into objectives and matching is done for each individual objective. An objective is a boolean combination of conditions and the entire expression has to evaluate to true for a match to be found. Each individual condition is an attribute and a corresponding value. The advertisements of regions also consist of attributes and one or more values corresponding to each attribute. The directory server matches the attributes and values and evaluates the expression for each objective. It gives the agent a list of all matching regions. The agent then uses its itinerary decision procedure to check which of these it would actually include in its itinerary and in what order. In order to do this, the agent might have to look at the other properties and requirements that the region has advertised and also the specific regions that matched with the specific objectives[1]. Once the agent decides the order of regions, it puts the entry points of all the regions in its itinerary.

Besides referring the agent to regions, the directory server also refers it to other directory servers where the agent can go for information about additional regions. Each directory server carries the ad of its in-neighbor and probably those of some other servers in the network. It performs matching similar to above for the ads of the directory servers and presents the results to the agent. In this case, the agent does not yet know about the specific regions advertised at the other directory servers that matched with its mission but only knows that the directory as a whole has some matching ads. The agent puts the URN of the directory servers at the end of its itinerary so that it can visit them after the matching regions. In case, the directory server does not find any other matching directory servers, it can put the URN of its out-neighbor at the end of the agent's itinerary so that the agent can at least have one other directory server to go to. The agent also has to make sure it has not already visited the directory server before. Similar to regions, the agent keeps track of all the directory servers it has visited as well and checks against them before adding a directory URN to its itinerary.

[1] The system may or may not want to allow repeat visits to regions depending on the needs of the particular application. In some cases, the agent might want to visit a region a second time if there are dependencies in functionality between regions. If the agent only has to visit each region once to get all the information it needs, it can keep track of the names of all the regions it has visited and check against them before adding any region to its itinerary.

The agent then proceeds to the entry point of the first region on its itinerary after its identifier is resolved to the actual hostname. First, the agent will ask the region for its current service advertisement so that it can confirm that the region actually does have services that the agent is interested in. Once the agent is positive that it wants to visit the region, it asks the region to grant it permission to enter. At this point, the region can ask for the agent's properties file. The agent can also ask the region for other detailed information that it might be interested in for conforming to the policy. The negotiation to grant entry to the agent continues until the region grants access to the agent. This could be either a simple one-step process where the region might grant permission to every agent that wants to visit or it could be a length exchange of information between the two entities to reach an agreement.

Once the agent has been authorized to visit the region, the entry point has to direct it to one or more hosts in the system that can provide it with services or information about where to find the services. The entry point itself might have information about where all the services in the region are located or there might be another server to do this task and the entry point can send the agent there. Also, the information could be distributed between various servers in the region and so the agent somehow has to be routed among them to discover the services. Each region can have its own method to route an agent that depends on its internal organization, its size and other factors. The agent has a standard interface to communicate with every host that it visits within a region[1]. It uses the function `AVAILABLE-SERVICES` to inquire about the services that the host offers. If it finds a service that it is interested in, it can use the method `SERVICE-INFORMATION` to get detailed information about a particular service. This information can include, for example, the name of the function that the agent has to invoke in

[1] The interface was discussed in detail in the section on the functionality of the regions with respect to agents (3.1.4).

order to actually use the service. The agent uses any services that help in fulfilling its mission. If the current host only provides routing information and not any services as such, it can return `NULL` for these functions. The third method `FORWARD-AGENT` is used to get information about the locations of other services from the current host. The hosts that implement this method are called *Way Points* and are used for routing the agent to service-providing hosts in the region. They contain information about services provided by other hosts in attribute-value form. They perform matching of the agent's mission with service advertisements similar to the matching at the directory servers. If they find any services that match the agent's requirements, they give the URN of the corresponding service host to the agent. The agent can put it in its itinerary for the region to visit later. The two functions of providing services and routing the agents can either be performed by the same hosts in the region or by different hosts. Thus, within a region, the agent will go through a path containing service-providing hosts and *way points* until it cannot find any more hosts that have services it is interested in.

At the end of the agent's journey in a region, it has to pass through an exit point which sits at the boundary of the region. The exit point can be put into the agent's itinerary at the entry point itself and it would remain at the end of the itinerary for that region. The region can perform any exit functions on the agent here. For example, the region might want to bill the agent for all the services it has used while it was within the region at the exit point. The agent would not be allowed to leave the region until it pays for the services. Also, for the agent, the exit point would be a place where its mission gets updated to reflect which objectives have already been fulfilled through services it used in this region and which services it still has to look for. The agent could actually do this every time it uses a service at a host or it could do it all together in the end. Once the exit functions have been performed, the agent can proceed to the next item in its itinerary, which could be a region or a directory server. If it is a region, its entry point identifier is resolved to the hostname at this point and the agent is transferred to it. Then the agent repeats the above process of visiting the directory servers and regions until all its objectives have been met.

At the end of its life cycle, the agent could return to a location that was specified by its creator, which could be the creator itself or a different host. It can report back the results of the functions that it performed such as the data it collected or the status of activities it carried out. In some cases, there might be no results to report back so that the agent will cease to exist after completing its mission without reporting back.

This concludes the discussion on autonomous mobile agents which are the means that consumers of services communicate with the providers. The life cycle of an agent was discussed as it travels through the network and determines its path by obtaining information from various sources. The next section focuses on the communication protocol between the different entities in the system and the language used for various advertisements and exchanges.

3.4 Communication Protocol

The various components of the architecture interact with each other at different points for functioning of the system. A protocol is needed for the communication that happens at all these points because different entities could have different languages for expressing their intents and properties. The languages and protocols we have used in the system have intentionally been made to be very simple and general purpose so that they can handle a diverse set of applications. The expectation is that each application will have its own specialized language which it would use for its internal communication. The architecture permits an application to define the grammar and semantics of its own language.

Most aspects of the communication protocol have been discussed at various points in the description of the architecture in the previous sections of Chapter 3. Section 3.1.4 discussed the communication of regions with directory servers and the contents of the messages exchanged between them. It also discussed the interface for agents when they are visiting hosts inside regions. Section 3.2.4 explained the messages that are exchanged within the directory server network. This section will first describe the mechanism for sending messages between any two entities in the system. It will then discuss the protocol for service advertisements of regions and the language used for expressing an agent's objectives.

3.4.1 Message Exchange in the System

The entities in the service-providing architecture have to communicate with each other in various situations when they need to inquire or relay information. The communication model in our system is for them to send messages to each other. Special-purpose mobile agents called *notification agents* are used to carry the messages. Notification agents are launched from the sender of the message to the receiver. They may or may not carry data depending on the purpose of the message. For example, a `REGION-TO-DS-AD-MESSAGE` is sent from a region to a directory server when it wants to advertise its services. It contains some data, which is the set of services of the region, its requirements and its entry point. There are other messages that do not have any data such as a `REGION-TO-DS-KEEPALIVE-MESSAGE`, whose purpose is only to inform the directory server that the region is still alive and its ad is unchanged. When a notification agent is received at a host, it has to take some actions in order to process the particular type of message it has received. For a message that is querying for information, the host has to send a notification agent in reply. For example, a directory server sends the `DS-TO-CR-JOIN-NETWORK` message to the Central Resolver when it first joins the network. When the Central Resolver receives this message, it has to reply with the URNs of all the currently active directory servers so that the sending directory can pick a neighbor from among them. Thus, the Central Resolver sends a `CR-TO-DS-JOIN-NETWORK-REPLY` message back to the directory.

There are a number of messages that have already been defined in the system. An application that has its own special communication requirements can define its own notification agents. A notification agent needs to have a sender URN, a receiver URN and could have some additional data. It might also need to carry some credentials with it for authorization. A notification agent also has a function that is executed at the receiving host and is used for carrying out the processing of the message. In this manner, all the functions associated with an agent are encapsulated within the agent itself and each host does not need to define special methods in order to handle different kinds of agents. This property of mobile agents being able to encapsulate both data and functions was the main reason they were chosen as the communication model for the system. They can be used ubiquitously for all communication needs and are easy to define and use. Thus, hosts do not need to need to have individual mechanisms to process all the different kinds of messages.

Here is an example of the life cycle of a notification agent. A `DS-TO-DS-NEIGHBOR-AD` message is sent from a directory server to its out-neighbor in order to advertise its regions. The directory server first has to aggregate the services of all the regions it is advertising in order to have the data to send in the agent. It initializes the notification agent with this data, the URN of its out-neighbor and a time stamp to record the time at which the ad was created. The agent travels to its destination. The receiving host invokes the processing function of the agent which involves adding a new entry in the receiving directory server for the sender. The aggregated ad from the agent is put into the entry along with the time stamp. If the directory server already has another entry for that sender, it replaces the old entry after comparing the time stamps. When the processing of the agent is over, it ceases to exist.

3.4.2 Service Advertisement Protocol

Since the regions are independently managed entities, they can follow any protocol for communication between the hosts within the region. The hosts can be informed of the protocol when they join the region. However, they need to have a standard interface in order to communicate with entities external to the region such as the directory servers. This section describes the protocol for the communication between a region and a directory server which consists of the region sending its service advertisement to the region.

The services of hosts are described in attribute-value form. An attribute is a category in which the service can be classified, for example 'location', 'name' or 'cost'. Value is the classification within that category such as '*MIT*', '*traffic data*' or '*20*'. Attributes and values can be free-form strings or they could be restricted to having a particular data type by the application. For example, an application could restrict certain attributes such as 'cost' to have only decimal values. The architecture also does not restrict the usage to a fixed set of attributes and values. However, an application may choose to impose such a restriction so that only a fixed set of attributes or values would be recognized and anything outside the set would be an error. Also, regions can have their own rules about what kinds of services they allow in the region, and hence what kinds of attributes their hosts can have. The attribute-value descriptions serve the same purpose as *name-specifiers* [1] or *resource descriptions* [6] in resource discovery systems. An application can also have the attribute-value descriptions be hierarchical such that a pair dependent on another is a descendant of it. For example, for a news service, <City, *Boston*> would be below <Country, *USA*> in the hierarchy. Thus, each service is expressed as a map from free-form attributes to their values.

The region has to aggregate the service advertisements of all its member hosts to present a consolidated ad to the directory server. Regions can have their own algorithms for performing the aggregation which depends on their organization and also on how the attributes and values are expressed. *Fig 3* shows an example of how a region could perform the aggregation: it collects the service ads from all its hosts and aggregates over all the attributes such that in the final ad, each attribute that has appeared in any of the hosts appears once. Each attribute could have multiple values if more than one host had the same attribute and different values for it.

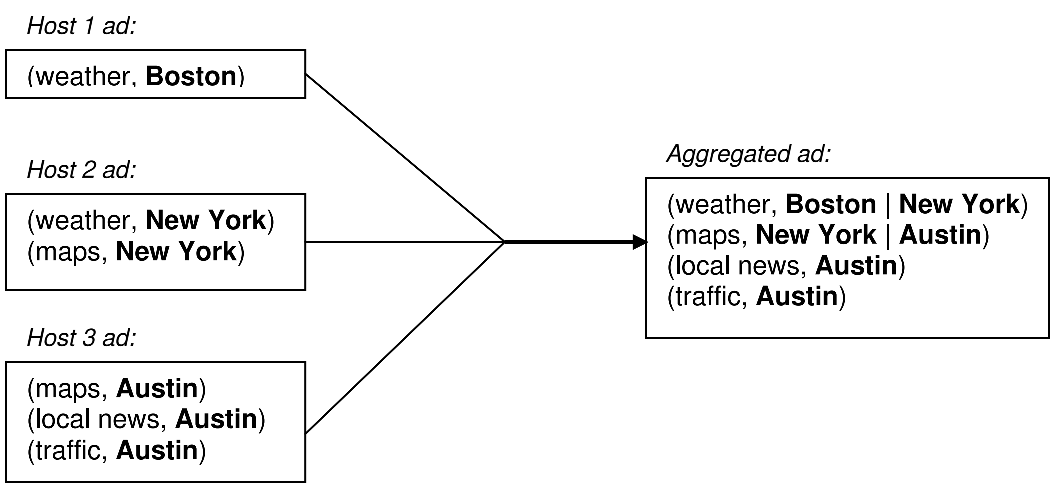


Fig 3. *Example of how aggregation of ads can be performed. This is a region that serves local city information. On the left are service descriptions from 3 different hosts. On the right is the aggregated advertisement that the region sends to the directory server.*

The region could choose to omit certain attributes and values in the final advertisement. It can also choose how often it updates the service advertisement. If the region's services are changing frequently in very short intervals of time, it might not want to update the ad at the directory server every time there is a small change, but wait for a certain period of time and then report the new status at that point. Along with the services, the region can also advertise some of its properties and requirements. The properties would be those that would be relevant to the agents that wish to visit the region such as the language it uses for communication, its billing policy, etc. These properties do not apply to a particular service within the region, but they consist of information that would be helpful to the agent in determining that it wants to visit the region and also in its operation within the region if it does decide to visit it. The requirements refer to properties that the agent should have for it to be allowed into the region. This could include the security credentials that the agent needs to present, the language it needs to understand and so on. Both the properties and requirements are also expressed as maps from attributes to values for simplicity. An agent can look for the particular attributes that it understands.

The final part of the ad message from a region to a directory server is the entry point identifier for the region. The identifier is resolved to an actual hostname and address when an agent wants to enter the region. It was explained in section 3.1.4 how a region can pick its entry point and vary it over time. The name to multiple address resolution is required for several reasons: load balancing, resolving requests based on their origin, and resolving requests to hosts that are 'closer' to the requesting host. The actual process of resolving the identifier to multiple hosts could be implemented in a few different ways. One of the methods is to use the round-robin resolution feature provided by DNS ([32], [33]). In the name record, there are multiple IP addresses corresponding to the same name. If a query is made for that name, a list of IP addresses is returned. If the round-robin option is turned on, the list is rotated for every new query. The clients usually use the first name in the list and hence this provides a simple solution to load balancing. However, since the rotation is completely deterministic, it does not rotate the list of servers based on load or the number of queries, but rather rotates the list the same way on every single query. Thus, this is a simple solution that might be sufficient for some cases, but other applications might require more sophisticated solutions.

Another option is to use load-balancing name servers such as *lbnamed* [38], which is an addition to the BIND implementation of DNS [4] that performs load balancing. The *lbnamed* server polls hosts that have the common address and collects information about their load. The DNS response goes through the *lbnamed* server which returns the host with the minimum load. It is transparent so that users do not need to know that this load balancing name server is actually selecting the least-loaded machine for them. Other than *lbnamed*, there are also many other commercially available hardware and software load balancers available that could be used.

3.4.3 Mobile Agent Language

The intent of a mobile agent's creator is expressed in its mission and policy. Agents have to follow a protocol for describing their missions and policies because these are the components that are visible to external entities such as directory servers and hosts inside regions. The directory servers have to match an agent's objectives with the service advertisements from regions that they carry. At the basic level, the service requirements of an agent are expressed as attribute-value pairs. But a more sophisticated representation is needed to express more complex combinations of these attribute-value pairs. This section describes how the mission and policy of an agent are structured.

A *Constraint* is used to express a restriction on the values an attribute can take. A few different kinds of constraints are defined in the architecture. More can be defined by an application. A *MaxConstraint* is used to express the condition that an attribute should have the maximum possible value that an agent can find. For a directory server that is trying to match this constraint for an agent, this means that it should look for all the services that have this attribute and the match would be the one that has the maximum value. For a *MinConstraint*, the directory has to look for the service with the lowest value for the given attribute. A *RangeConstraint* imposes the condition on the value of its attribute that it has to be between a given pair of values. This is only possible if the values can be ordered. The *RangeConstraint* consists on an attribute `a`, an upper bound `u` and a lower bound `l`. For satisfying this *constraint*, the service should have the attribute `a` with a value `x` such that `x >= l` and `x <= u`. The *NotInRangeConstraint* represents the opposite of a *RangeConstraint* such that the value has to less than the lower bound and greater than the upper bound.

A *Constraint Expression* is a used to express a boolean combination of individual *Constraints*. An *AndConstraintExpression* is used to represent a conjunction of individual constraints. In order to satisfy it, all individual constraints should be satisfied. The following is an example of an *AndConstraintExpression*:

```
RangeConstraint('Year', 1990, 2000) ∧ MaxConstraint('NumberOfDataPoints')
```

An *OrConstraintExpression* is similarly used to represent a disjunction of individual constraints and satisfying any one of the constraints is sufficient to satisfy the expression. A *NotExpression* expresses the negation of a constraint or a constraint expression.

These individual constraints can be combined to form constraint expressions. Each objective of the mission is a single expression that is a combination of constraint expressions. The mission consists of a number of objectives. Each objective of the mission has to be satisfied in order for the agent's entire mission to be achieved. The following example shows how the various building blocks are combined to form the mission for an agent.

An example mission:

Objective 1: (data, **network traffic**) ∧ (location, **MIT**) ∧ ((year, **2005**) ∨ (year, **2004**))
Objective 2: (map, **Boston**) ∧ (type, **satellite**) ∧ (color, **true**)

The policy of an agent is also constructed similarly out of constraints and constraint expressions. The constructs defined in the architecture are simple and general-purpose, but an application can define its own special constructs for expressing an agent's mission as long as it defines the algorithm to find a match for them. An important assumption has been made throughout the system about the attribute-value language used by agents and regions to describe services. The assumption is that the language is universal which means that a given word is semantically equivalent for all different entities in the system and has the same interpretation for all of them. This implies that if an agent is looking for a particular attribute x and a region has advertised an attribute y whose string representation is the same as x, then y produces a match for x. As long as their string representations are the same, two words are interpreted the same way. It is important to note this assumption because in a real world scenario with a diverse set of regions, it is possible that 2 entities can have different interpretations of the same attribute. For example, the word 'bank' can be intended to mean 'financial institution' by one region and could be interpreted as ‘the slope of land adjoining a body of water, like a river’ by an agent from another region. This and other considerations for designing a naming scheme for a system are discussed in [41].

The languages used to represent services and the mission and policy of agents are intentionally chosen to be simple in order for them to be general-purpose. The expectation is that an application might have its own specific language that it will use for communication and for expressing its services. The system is flexible such that it can be extended to use other languages. There are a number of languages that have been designed to express policies, constraints and capabilities of systems such as RuleML [9], Web Services Description Language (WSDL) [15], Resource Description Framework (RDF) [27] and so on. Each of these and other resource and policy description languages has its own features, which might be suitable for different applications.

This concludes the section that described the details of the communication protocol that is used in the architecture, including the exchanged messages, service advertisements and agents' language. This also concludes the chapter on the details of the architecture of the service-providing distributed system. In this chapter, we discussed the various components of the architecture that enable location of services in a large-scale system. Regions help to partition the search space by grouping together hosts that share common characteristics. They can be based on any combination of features. They manage themselves independently and provide some basic functions and an interface for communication with the other components. They perform aggregation of service descriptions of their member hosts and advertise it to the directory servers. The directory server network is a self-configuring overlay network that advertises services for regions and aids agents in locating services that they want to use. It can adapt itself to increasing load from regions by dynamically changing the number of advertisements that are exchanged between individual directory servers. The entities that wish to consume services are represented in the system by autonomous mobile agents. The intent of an agent's creator is expressed through a mission and policy for the agent's operation. Agents compute a path through the network based on their mission and policy and with the help of information from directory servers and regions. The next chapter discusses the implementation of the architecture described here and presents results from an experimental evaluation of the system.

Chapter 4

Implementation and Evaluation

This chapter describes the implementation and evaluation of the region-based service-providing system. The architecture that was discussed in the last chapter has been implemented in Java. It can be used for setting up regions to be integrated with the system. Service-providing hosts can be added to regions. The directory server component sets up the directory server network and provides complete functionality for it. Agents are implemented such that they can be initialized with a set of objectives and launched into the system. They will travel through the network to find services to complete their mission. The purpose of the implementation is to be a framework which can be extended by applications using it. There are two ways in which the extension can be done: one is to use the current general purpose framework and add regions and hosts in the application to it directly; the other is to inherit from parts of the current framework to develop a more specialized system for a specific application and implement rules and properties that are required for the application.

Regions are independently managed entities that can define their own internal organization and structure. A new region can be defined and added to the system. The only condition is that it has to implement a few methods of the standard interface for regions which is needed for their communication with the other entities of the system. For illustrating how a region can be defined, we have implemented an example internal organization for regions. This is not part of the basic framework, but an extension of it. Section 4.2 discusses this hierarchical internal organization which is a flexible structure and could be suitable for regions of various sizes. The section after that briefly discusses an example application for the framework. The application is for a billing system where the service-providing hosts charge for their services. It implements parts of the commercial model that was discussed in section 3.1.6. It describes the different schemes that can be used for charging for the services and for payment by the agents. The system also includes a basic banking service. Notifications are used to make payments and verify identities.

The last section in the chapter presents an evaluation of the architecture in terms of desirable properties for distributed systems. We performed simulations to measure the performance of the system as various factors affecting it were varied. The experimental evaluation and analysis of the design show that the system has some good properties in terms of scalability, robustness, and fault-tolerance and recovery. The results are presented in section 4.4.

4.1 Implementation Details

The major components of the system as discussed in Chapter 3 are: autonomous mobile agents, regions, service-providing hosts and the directory server network. The implementation of each of these components is discussed in the following sections.

4.1.1 Basic Agent Framework and its Extensions

The agent framework we have used in the system is called Ajanta ([3], [48]) and was developed at the University of Minnesota. The implementation for mobile agents and agent servers comes from this system and the rest of the components have been implemented over it. All servers and agents are identified by URNs in Ajanta. Agents can be launched to a particular host and can be given a method name that they will execute on that host (assuming that the server has defined that method). They can also be initialized with a static list of host URNs that they will visit once they are launched. The hosts should have servers running on them to receive agents and process their requests. Ajanta takes care of resolving the URNs to IP addresses and of routing the agents to them so that the rest of the system can just use URNs for identifying and addressing hosts.

The main focus of the Ajanta design is on mechanisms for secure and robust executions of mobile agents in open systems [46]. In Ajanta, the mobile agent paradigm is based on the generic concept of a network mobile object. Agents in this system are active mobile objects, which encapsulate code and execution context along with data. Ajanta is implemented using the Java language and its security mechanisms are designed based on Java's security model. It also makes use of several other facilities of Java, such as object serialization, reflection, and remote method invocation [3]. It uses java bytecode to transfer the agents.

For the implementation of our system, we first enhanced the mobile agents from Ajanta to be autonomous i.e. to figure out their own path through the network given a set of objectives in terms of attributes and values to describe services. A `Constraint` is used to express conditions on the values that attributes can take. `MaxConstraint`, `MinConstraint`, `RangeConstraint` and `NotInRangeConstraint` are inherited from `Constraint` and are used to define specific types of constraints[1]. A `ConstraintExpression` is used to express boolean combinations of individual constraints such as conjunction, disjunction and negation. The `Mission` of the agent is composed of a set of objectives. Each objective is a compound constraint expression that represents a particular service that the agent wishes to use along with its characteristics. The various objectives can be ordered by giving each of them a preference value that helps in deciding the order of regions and hosts to visit. The `Policy` is also constructed similarly but contains general requirements of the agent that are not related to a specific service. The agent has methods that it will execute when it arrives at a directory server and at a host inside a region. It also contains algorithms that can order a list of matching regions and/or hosts (returned to it by a directory server or a *Way Point* for example) and construct an itinerary out of them. The `AutoAgent` class combines all these elements to define a fully-functional autonomous agent for the system.

`AutoAgent` can be extended for specific agents if needed. However, in order to provide additional flexibility to the applications to implement different kinds of agents that might have different languages for expressing their requirements, an interface is also defined for agents in the system. It is sufficient for an application to implement this interface for its agents since the rest of the system uses this interface to communicate with them. The `AgentInterface` defines methods for the basic functions of an agent that can be implemented by the application. It includes methods that the agent will execute when it is at a directory server, at a host inside a region and at a border host. It has accessor methods that return the various components of an agent such as its itinerary, mission, policy, public properties and its URN. `AutoAgent` implements the `AgentInterface` for a general-purpose agent.

[1] See section 3.4.3 for a discussion on the structure of an agent's mission.

4.1.2 Interfaces for Regions and Hosts

The next part of the system is the interface for regions and hosts inside regions. Regions and hosts need to provide some basic functionality to the agents and to the directory servers. They can choose any implementation they like to provide the functionality for the methods in the interface. They could have additional functionality besides the basic methods in the interface. In the `Region` interface, there are methods to find a directory server from the *Central Resolver* when a region first joins the network and to send its ads and keep alives to the directory server after it joins. There are also methods which the agent interfaces with at the entry point of the region that ask for the agent's properties and determine whether to grant it access. Lastly, there are accessor methods that return the values of various fields of a region such as its name, latest ad, entry point, directory server, agent requirements and so on. A `RegionAdRecord` is sent in the ad message from the region to the directory server and consists of the service map, requirements and the entry point. `RegionID` is the unique identifier for a region, which currently just consists of a string, but can be extended to define any kind of identifiers for regions. There are a number of notification agents as well that are associated with regions and are used to exchange messages needed for the above functions. All notifications are spawned off as separate threads and hence the servers can continue execution while they are being transmitted.

The `RegionHost` interface is implemented by all the hosts inside a region that an agent can visit and consists of the basic operations that an agent needs to function at the host. It is required for the initial communication of an agent with a host. It has two methods dealing with services that the host offers: `AVAILABLE-SERVICES` returns the services that the host is currently offering, and `SERVICE-INFORMATION` returns detailed information about a particular service that the agent is interested in. Region hosts also need to direct agents to other hosts inside the region that have services that the agent wants. `FORWARD-AGENT` is the method that takes the mission and policy of the agent and matches them against service advertisements from other hosts. If the host does not have information about services of any other hosts, it can return `NULL` for this method. There are also accessor methods in the `RegionHost` interface to get the URN of the host and the `RegionID` of its owning region. Special hosts inside the region such as the *Border Way Points* could have special implementations of this interface because they may not directly provide services to agents. They do have to forward the agent to other hosts however and they could also perform special functions such as putting an exit border point at the end of the agent's itinerary for the region. A region can implement its own notifications for communication between its member entities.

4.1.3 Directory Server Network Implementation

The directory server network mainly consists of two components: `DirectoryServer` and `CentralResolver`. The `DirectoryServer` class has implementation for the various functions of directories. When they first join the network, they report to the *Central Resolver* (CR), obtain the list of currently active directories, send messages to each of them, measure the metric that the spanning tree is based on, choose one out-neighbor and report it back to the CR. This process results in a spanning tree organization of the directories. The next function is to receive ads and keep alives from regions that are advertising with each of them. Each ad entry that is placed in a directory (either from regions or from other directory servers) is associated with a separate timer thread that deletes it when the timer expires. If a keep alive or another ad message is received before that, the timer thread is reset to start from the beginning. When a timer thread deletes an entry from a region, it also sends a message to the region informing it that the entry is being deleted, so that in case the region is still functioning, it can resend the ad. Each directory receives `NEIGHBOR-ADS` and `OTHER-DS-ADS` from its in-neighbor. It has to store all the `NEIGHBOR-ADS`, but it only stores as many `OTHER-DS-ADS` as it has the capacity for. It also forwards all the ads that it stores on to its out-neighbor. The other set of functions of directory servers is related to navigating the mobile agents. There are functions that take the mission and policy of an agent and return the regions and directories that have advertised services which the agent wants.

The `CentralResolver` has entries for all the currently active directory servers in the network and it maintains a list of their neighbors as well. It also uses timer threads to time out entries of directory servers that are not functioning any more. The CR also keeps track of the status of each directory server in terms of its load which is reported by the directory along with its keep alive. The CR uses this load information to assign directory servers to new regions.

This concludes the section on the implementation of the basic service-providing architecture. The next section discusses a particular internal organization of regions that was implemented as an extension to the system.

4.2 An Example Internal Organization for Regions

This section presents a hierarchical scheme for organizing hosts within a region. This is an example of an internal architecture for regions that was implemented for illustration. It is a scalable and flexible tree structure that can be used for regions of various sizes. In this organization, there are a few different types of hosts inside a region: *Service Hosts*, *Way Points*, *Top-Level Way Points*, *Border Way Points* and the *Region Information Server*. They are described in the following sections. *Fig 4* illustrates the proposed internal organization with an example.

4.2.1 Components of the Internal Structure

The core of a region is formed by two kinds of hosts: *Service Hosts* are the ones that

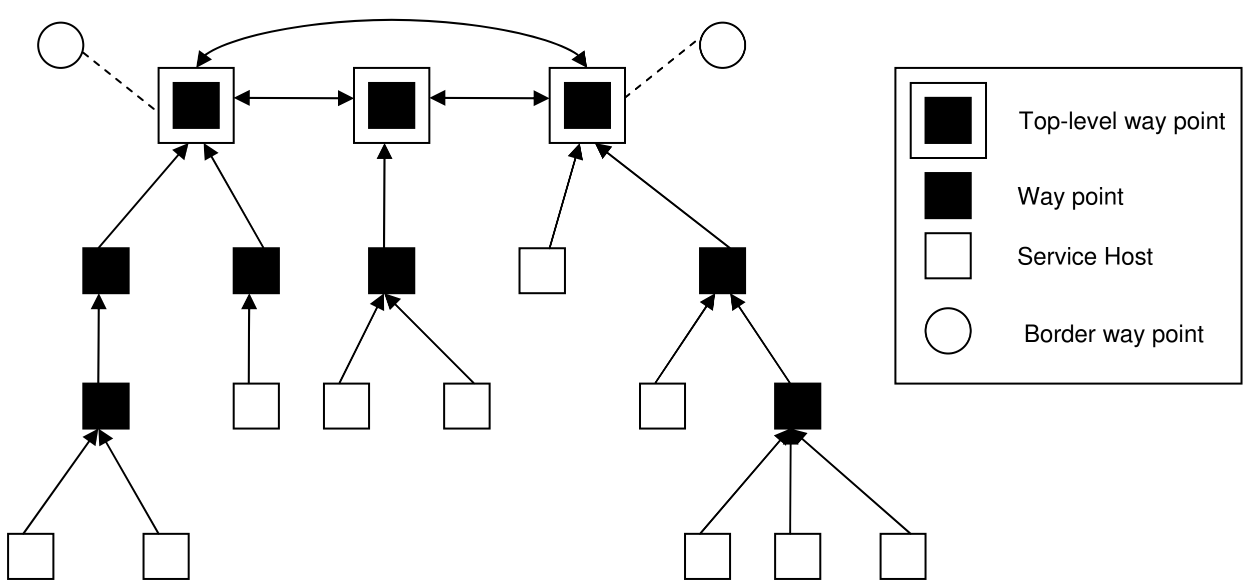


Fig 4. *Hierarchical internal organization of regions. Arrows point in the direction that ads are sent. Agents are forwarded in the reverse direction.*

provide services to agents visiting the region and *Way Points* are the ones that provide navigation facilities i.e. they direct agents to other hosts in the region. The hosts are organized as a tree structure and the number of levels of the tree depends on the complexity and size of the region. Way points are the internal nodes of the tree and service hosts are the leaves that attach themselves to the internal nodes. A way point can have both service hosts and other way points as its children and hence, the tree can be unbalanced. Service hosts have attribute-value descriptions of the services they provide that they advertise to attract consumers (agents). Each child advertises these services as its properties to its parent. The parent node aggregates the reported properties of all its children into a condensed representation. It then advertises this aggregated property as its own to its parent and this process continues recursively till the top of the tree. The properties are reported to parents through notifications. A child can add/delete/modify its properties at the parent at any time by sending a new notification. The parent then has to perform the aggregation again and report the modified properties to its parent and so on.

If the tree is rooted at a single way point, that node will become the central point of delay or failure and might have excessive load. To prevent this in our organization, the root of the tree is not necessarily one single way point, but the hierarchy ends at a few *Top-Level Way Points*. Thus, the load is distributed among these top-level way points and it can be considered that the organization actually consists of multiple trees rooted at these few points. The top-level way points are all connected to each other, which means that they forward their aggregate properties to each other. Thus, they can forward agents to each other because they know the aggregated properties of the other trees. Another way of looking at this structure is as a singly-rooted tree, with a cluster at its root that serves the purpose of providing resiliency.

The *Border Way Points* are the servers that sit on the boundary of the region and control the agent traffic entering and exiting from the region. They can perform several functions to implement the region's policies such as keep track of billing, authenticate the incoming agents according to the region's security policy, etc. They also have to forward incoming agents to appropriate hosts inside the region. For this purpose, each border way point is connected to at least one top level way point and it forwards all incoming agents to this top-level way point. From here, they can make their way into the internal network of the region and access the services they need.

The *Region Information Server* (RIS) is the administrative authority for the region. It creates the rules and policies of the region and also provides all the information that is global to a region. There is one RIS per region and its creation signifies the creation of the region. It is also responsible for reporting the aggregated services of the region to its designated directory server. The service descriptions are distributed among the different top-level way points for the different sub-trees. They need to be combined before being sent to the directory server. To do this, the top-level way points report the aggregated properties from their sub-trees to the RIS. The RIS aggregates them and sends the complete service advertisement of the region to the directory. The frequency with which updates to properties are reported to the RIS and to the directory from there could vary from region to region and is thus configurable. The RIS can be replicated for fault-tolerance. Also, it is not necessary that the RIS is a single server – a group of servers could cooperate to act as the central authority for the region. For performing a function, the group would have to take a single combined action or make a single combined decision.

The structure of the hierarchy of way points and hosts could depend on a number of factors such as size, complexity, functionality, resources and possibly the geographic location of the entities. For example, a region with 10 hosts could have a flat hierarchy - 1 top-level way point with 10 children. A region with 1000 hosts might have 8 levels in the hierarchy. A region with 2 locations one in North America and the other in Asia might have 2 large sub-trees. Thus, the formation process of the hierarchy is region dependent and will be determined by the characteristics of the particular region.

4.2.2 Routing of Agents

An agent enters a region through a border way point and if it is granted access, the border way point forwards it to one of the top-level way points. Way points match the agent's objectives with the service advertisements that they have similar to the directory servers. At the top-level, the results of the matching might include the children of the top-level point which the agent is at (which could include other way points or service hosts) and also other top-level way points. The agent makes its way through the tree by querying the way points for the properties that it is interested in at each level and including in its itinerary all the hosts that it finds matches with. Thus, the agent is routed by the way points deeper and deeper into the tree using the aggregated advertisements until it reaches a leaf of the tree which contains the service that had originally produced the matching.

Each way point returns a list of hosts to the agent that have services it is interested in. An agent then weighs each of these servers based on its preferences for its various objectives. The itinerary is formed by arranging the selected hosts in decreasing order of their weights so that it can visit the most closely matching, which are potentially the most useful, hosts first. If it comes across a way point, it performs the query again and thus keeps modifying its itinerary dynamically. The traversal will be similar to a depth-first traversal of the tree. Only matching hosts would be visited in the traversal. Another point to note is that the agent may sometimes not see the most recent version of an advertisement at a way point higher in the hierarchy if an update was sent from the service host but the modification process has not reached all the way up the tree yet.

4.3 A Billing System Application

We now discuss an application of the region-based architecture in a scenario where hosts charge money for the services they provide. In other words, agents are billed for the services they use in the regions that they visit. Regions can have different billing structures for how they charge agents. Agents can also use various methods for payment of the charges that they incur in these regions. We implemented an extension to the system described in section 4.1 to incorporate a general billing mechanism for regions. The extended system includes various policies for billing, different modes of payment and enhanced functionality of the components of regions to assist in the accounting of charges incurred by agents. The key elements of any billing system are the following: the payer, the payee, the product for which the payment is being made, location where the billing takes place (which may or may not be the same as the location where the product is exchanged) and possibly financial representatives of the payer and the payee that facilitate the transaction. In our system, the products are the services; the payers are the agents that use services on behalf of their creators; the payees are the regions and/or the hosts that provide services; the location could be a host or the exit point of the region; and the financial representatives could be banks of the payer and payee. Our system includes a *Bank Server* that facilitates financial transactions that occur as a result of the billing. The actual transfer of money can be performed by various electronic payment systems ([5],[21]).

In a billing environment, the hosts that provide services firstly need to be aware of the billing policy of the region they are in. They might have charges associated with each of the services they provide and a bank account where the payments they get will be deposited. Depending on the payment model of the agents that use their services and the billing model of the region they are in, the hosts will have to send notifications about charges to a designated authority in the region or communicate directly with the bank or deduct the amount from the agent's balance.

The purpose of the *bank server* is to serve as the authority that approves transactions involving transfer of money between accounts of regions or hosts within regions and accounts of agents. The bank is required when an agent is using the 'On-Use Authorized' model[1] of payment. In this case, payments are made by the agent from the account of the host/region that is responsible for handling its payments. The bank needs to authorize each of these payments after checking if there is enough money in the charged account. Authorization requests are sent to the bank in the form of notifications from the hosts whose services the agent uses. The bank sends authorization replies in turn informing the hosts of the status of authorization.

4.3.1 Billing Models for Regions

Different regions can have different policies for how they are going to bill the agents that visit

[1] See section 4.3.2 for payment models of agents.

the region and use the services of its member hosts. There are two kinds of models that have been implemented. The overall policy of a region can be any combination of these models. The two models are:

- *Flat Fee Billing model*: In this model, the region charges a fixed amount to agents for entering a region. This amount can be charged to the agents when they enter the region through an entry point or at the end at an exit point. It will be deducted from the agent's cash balance or authorization will be acquired from the bank depending on the payment model of the agent.

- *Per-Host Billing model*: In this billing model, agents are charged for every host that they visit and use services from. Individual hosts can have different charges. It can be specified in the model whether the agent is charged before it uses the services at a host or after it uses the services. Also, it can be specified if the agent will have to wait for its payment authorization to be approved before it will be allowed to use the services at a host (if the *charge-before* option is chosen). There are two ways in which accounting of charges incurred at the hosts can be done. In the first, each host can individually bill the agent and ask for authorization from the bank if the agent's payment model is to pay on use. In the second method, each host can send its bill to a designated server in the region (such as the Region Information Server described in section 4.2.1) which consolidates the bills for that agent from the entire region. Let this server be called the *Agent Billing Server* (ABS). The function of the ABS could be performed by an exit point in the region. Or it could be performed by another dedicated server which the agent will have to visit before it exits the region. The ABS will deduct the total amount from the agent's cash balance or ask the bank for authorization of the charge depending on the payment model of the agent. It can be specified, in the per-host billing model, which of these ways of accounting the region will be using.

4.3.2 Agent Payment Models

Agents can use different modes for making payments for the charges they incur when they use services of various regions. Three different modes of payment for agents are included in this system:

- *Contract Payment*: In this payment model, it is assumed that the region where the agent originated has a regular billing contract with the region whose services the agent is using. This means that the details of the billing and payment have been negotiated beforehand by some administrative entities. The agent operates under the assumption that the specifications of the contract are being fulfilled by the regions. Therefore, the agent does not need to pay every time it uses the other regions' services, but just has to show the contract to the billing host which can verify it if needed.

- *Cash Carry Payment*: This model represents the pay-as-you-go mode of payment. It can be likened to the agent carrying a bag of cash with it, which it pays from every time it is billed. It starts out with a fixed amount of money, makes an immediate payment every time it is charged for a service and continues until it has no money left. No other entity needs to be notified in this case because the agent makes payments on the spot.

- *On-Use Authorized Payment*: This is the payment model in which the agent makes the payment when it uses the services of a host through an authorized transfer from its bank account. The agent does not carry cash with it, but there is a designated entity that is responsible for handling payments for the charges incurred by the agent. This entity could be one of the following: the host that launched the agent, the final destination of the agent or a third-party. The agent carries around with it the URN of the host to be billed, the URN of the bank and the account number from which payments will be made. If an agent using the On-Use Authorized payment model incurs a charge, the billing entity obtains authorization from the bank for the charge. The authorization is granted if there is enough money in the account. The money is then transferred from the charged account to the account of the billing host.

This concludes the discussion of the billing system. In this chapter so far, we have presented the implementation details of the service-providing architecture that we proposed in Chapter 3. Over the basic system, we built an extension that implements a specific scheme for internal organization of regions and another extension that implements a billing system as an example application. This concludes the discussion of the entire implementation. Next, we present an analysis of the system through results of simulations performed with the implemented framework.

4.4 Simulation Results and Analysis

A set of simulations were carried out with the implemented system to measure the performance of the basic framework. The metrics we have used are application-independent and give an idea of the overall trends in the performance of the system as it increases in size and complexity. The exact numbers are not as important as the trends because the numbers will depend on the particular application and how it has implemented certain features of the system. The architecture is only an underlying framework that has general-purpose constructs and algorithms. For example, we cannot predict the performance of regions because they can be very diverse and their performance depends on how they are individually set up.

In the first set of experiments, we analyze the effect on the performance of the system that comes from introducing regions and directory servers, which are key components of the proposed architecture. We compare the results of the following 3 scenarios:

1. No regions and no directory server network: hosts report to a single central directory server
2. No directory server network: hosts are members of regions and regions report to a single central directory server
3. With regions and a directory server network: hosts are members of regions and regions report to a number of directory servers that form a network

We first analyze how the scalability of the system improves with the introduction of regions and a directory server network. The scalability is measured in terms of the number of advertisement messages that are sent by the hosts in order to advertise their services. We also discuss the overhead in terms of messages that occurs with the proposed architecture. We then analyze how regions help in handling the dynamism of hosts that enter and leave the system. We present experimental results for the behavior in the start-up phase when directory servers and regions join the system. Lastly, we analyze the service location performance by measuring the efficiency of an agent's operation as the size of the system increases.

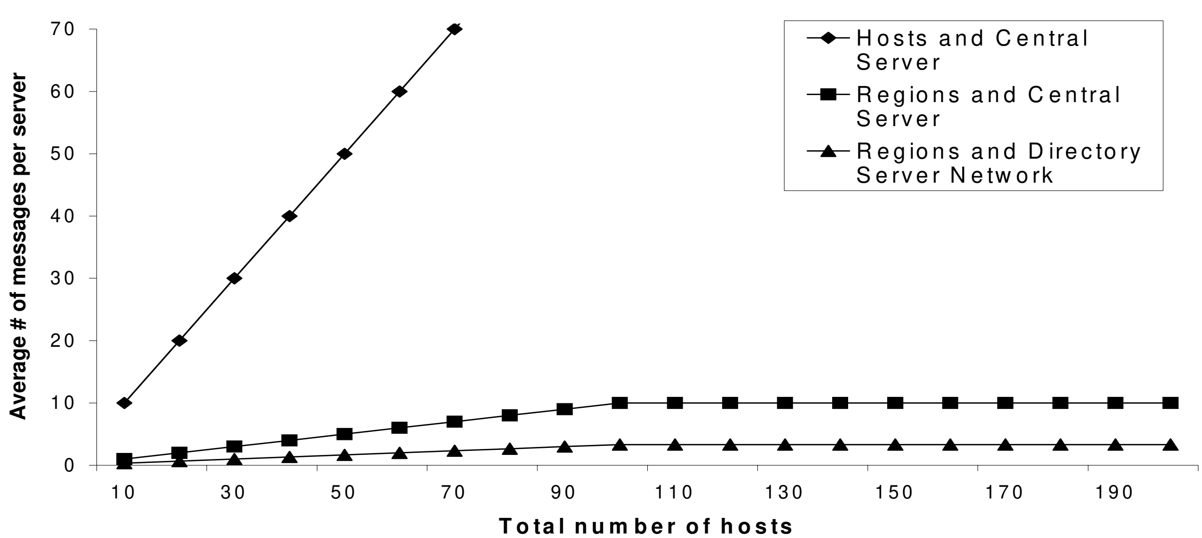
4.4.1 Analysis of Scalability

One of the main reasons for introducing regions in a service location architecture is the benefit they give in terms of the scalability. By grouping together hosts with similar characteristics and independently managing them, the load on the directory service for the system is significantly reduced since it now only has to deal with regions instead of individual hosts. The purpose of having a directory server network, instead of a single directory server, is to distribute the advertisement information. This eliminates a single point of failure and provides resiliency to the directory architecture. It also provides scalability as the load on each individual server reduces. In the experiments in this section, we attempt to measure this advantage in scalability that is provided by regions and directory servers. However, introduction of the directory server network also induces some overhead in the system that is required to keep the directory servers functioning as a network. This consists of the messages that are sent from the directory servers to the Central Resolver and also messages that are exchanged among the directory servers. We present results that measure this overhead, which is a cost to the system. We conclude that the improvement in scalability outweighs the cost and hence, the system gives better overall performance than the first 2 scenarios mentioned above.

For the following experiments, we are not considering the messages that are passed inside regions. This is because regions can have different internal organizations and methods for consolidating advertisements, and the number of messages that are passed depends on those two factors. We assume that each region will have sufficient infrastructure to perform the function of creating an aggregate service advertisement and hence, we do not consider the performance issue inside a region. Also, in the following experiments, for simplicity, we assume that the frequency of all message exchanges is fixed and that the directory servers are not performing backoffs in response to high load. In reality, each region could have a different frequency for sending its advertisements and it could change this over time. The same is true for ads that are exchanged between directory servers.

Graph 1 compares the scalability in the 3 cases as the total number of hosts is increased. It shows the average load per directory server which is measured as the total number of advertisement messages that are received by the directory in one time period. The topmost plot

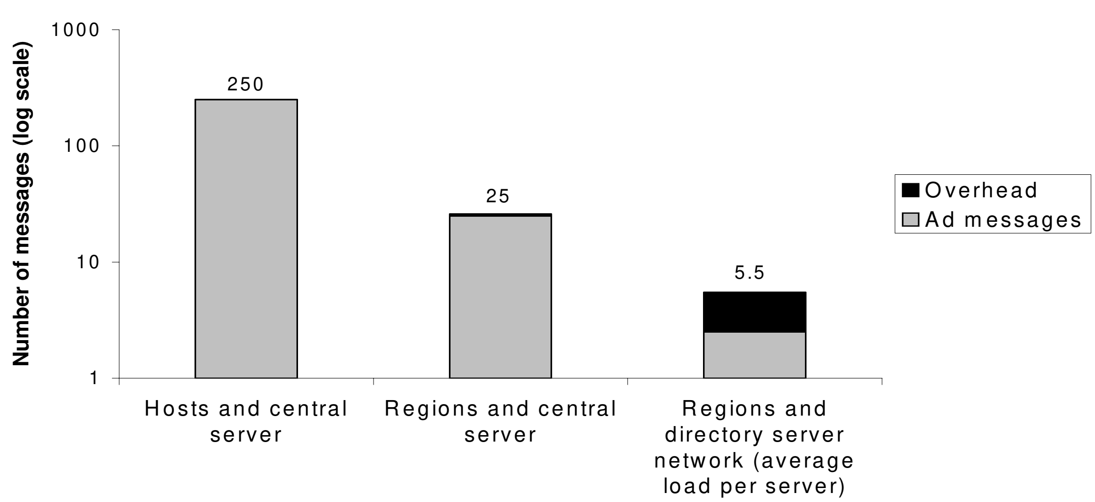
Graph 1 *: Load per server with increasing number of hosts*



line shows the case when there is only one central directory and all the hosts directly report to it. As expected, the load on the directory shows a linear increase. The middle plot line shows the case when regions are introduced. There are 10 regions in the system and the hosts are assumed to be uniformly distributed between them. A new region is formed every time an additional 10 hosts join until the number of hosts reaches 100. After that, the additional hosts join the existing regions. We can see that introducing regions significantly reduces the load on the central directory in terms of the number of entries, as it now has to store only 1 entry for each region. The bottom plot line shows the case when there is a directory server network composed of 3 directory servers along with regions. The plot shows the number of advertisements received from regions per directory server. The regions are randomly distributed among the 3 directory servers as they are formed. This shows that the load on each server is significantly reduced as one server has to only carry 3.3 entries on average.

The improvement in scalability can be seen more clearly in ***Graph 2*** which shows the average load per directory server for a fixed number of hosts (= 250) in 1 time period. The bars

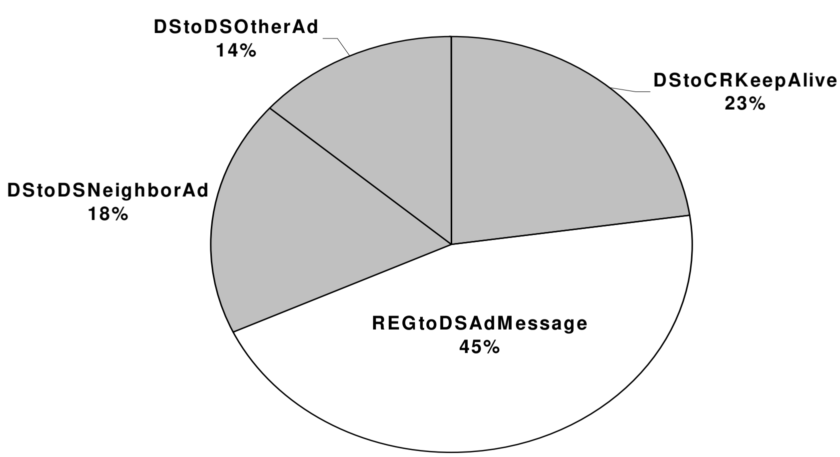
Graph 2 *: Scalability in terms of load per directory in the 3 cases for a fixed number of hosts (shown on a log scale)*



show the number of messages that are exchanged in each of the three cases on a log scale, in order to exhibit the third case more clearly. There are 10 directory servers and 25 regions in this scenario. Hosts are uniformly distributed between regions and regions are randomly distributed between directory servers. The black portion at the top of the third bar shows the overhead in our proposed architecture. The overhead consists of additional messages (other than the advertisements) that are needed for the directory server network. We can see that in spite of the overhead, the proposed architecture shows a 97.8% improvement over case 1 and a 78% improvement over case 2 in terms of the average load per directory server.

One of the major costs of having a directory server network is the overhead of the extra messages that are needed for its set-up and functioning. In ***Graph 3***, we show a breakdown of the different kinds of messages that are passed in the system after it has stabilized, that is, after all the directory servers and regions have joined and no more communication is needed for their setup. The advertisements from the regions to directory servers compose about 45% of the total number of messages in 1 time period. The other portion can be regarded as overhead and consists

Graph 3 *: Distribution of different kinds of messages after the system stabilizes (shaded area shows the overhead)*



of the following messages: keep alives from each directory server to the Central Resolver, `NEIGHBOR-AD` messages and `OTHER-AD` messages that are exchanged among the directory servers.

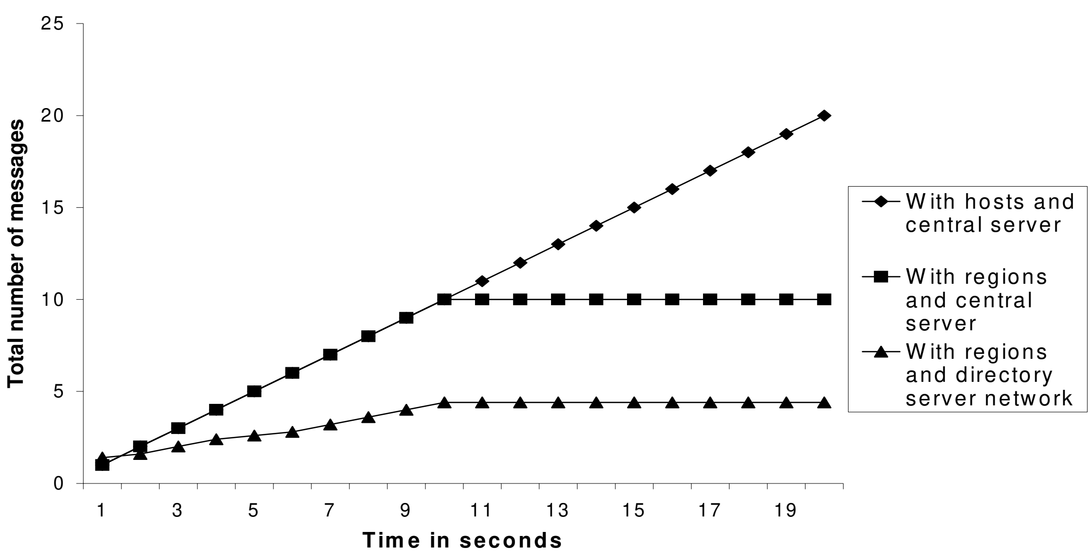
From the above results, we can conclude that the introduction of regions and directory servers significantly improves the scalability of the system. Also, having a directory server network provides better resiliency and reliability to the system than a single central server. There is a tradeoff between these benefits and the extra cost of maintaining the network. There are possible ways to reduce the message overhead such as reducing the frequency with which keep alives are sent to the CR since the directories will usually be relatively static. The number of `NEIGHBOR-AD` messages will be constant since each directory sends one message to its neighbor, but the number of `OTHER-AD` messages can be controlled depending on the load of the servers.

4.4.2 Analysis of Startup Phase

Besides the improvement in scalability, another benefit of introducing regions in the system is that they can handle dynamism of hosts at a lower level. This means that if hosts were entering and leaving the system and there was only one central server, it would have to change its state every time a change occurs in the hosts. A region is a grouping of a fewer number of hosts and hence is better suited to handle their dynamism. Also, since a region will be more stable relative to its member hosts, the entries at the directory will not have to be added or deleted as often. A region can make changes to its advertisement at an appropriate frequency and the directory does not even need to know of the entry or exit of individual hosts. The dynamism that our system needs to handle is limited to the addition and deletion of regions and directory servers, which would in general be more stable.

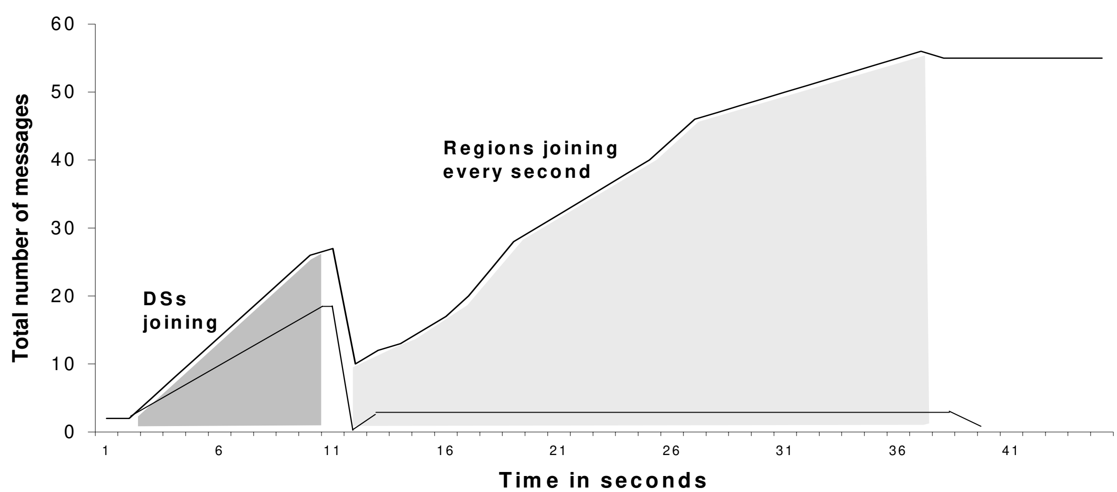
In the next experiment, we analyze the effect on the system as new elements are added to it. We refer to this as the startup phase of the system. In ***Graph 4***, the total number of messages that are passed in the system is measured in each of the 3 scenarios as time passes. 1 host is joining the system every second in each of the cases. The topmost plot line shows the results with a single directory. The number of messages passed increases linearly. The middle plot line shows the results when regions are introduced into the system. As the first 10 hosts join, 1 region is formed every second for each of them. The additional hosts join the 10 regions after that. The bottom plot line shows the total number of messages (including overhead) per server with our architecture. There are 5 directory servers in the system. As in the second case, there are 10 regions formed for the first 10 hosts and additional hosts join the existing regions after 10 seconds. It can be seen from the graph that the number of messages per server needed during the start-up phase is far less for our architecture than for the first 2 scenarios.

Graph 4 *: Performance with addition of hosts in the 3 cases*



We now focus on the start-up phase for our architecture and analyze it in ***Graph 5***. In this experiment, there are 10 directory servers and 25 regions. The directories join the system 1 per second for the first 10 seconds and the regions join 1 per second from seconds 13 to 37. The lower plot line shows the number of startup messages and the upper plot line shows the total number of messages. During the densely shaded phase, each new directory that joins first sends a message to the Central Resolver in order to obtain a list of the currently active directories. It then sends messages to each of the current directories, receives their replies and then picks one of them to be its neighbor[1]. This is the start-up cost of addition of directories. After this phase, the directories continue to send keep alive messages to the CR. In the lightly shaded phase, as each region joins, it sends a message to the CR to obtain the URN of its directory server. Thus, the startup cost of regions is much lower in terms of the number of messages. As the number of regions increases, the number of advertisements at the directories increases and hence the amount of information they exchange between themselves in the form of `NEIGHBOR-AD` and

Graph 5 *: Messages exchanged during the start-up phase (10 directory servers and 25 regions)*



[1] See section 3.2.3 for details on the architecture of the directory server network.

`OTHER-AD` messages increases as well. This accounts for the increase in the total number of messages during the addition of regions.

We can conclude from the above experiments that regions and directory servers help in containing the dynamism of hosts. The cost is dynamism of the regions and the directory servers themselves, which is significantly lower as seen in the graphs and will also occur less frequently. Hence, this is another area in which the proposed architecture presents an improvement over the first 2 scenarios.

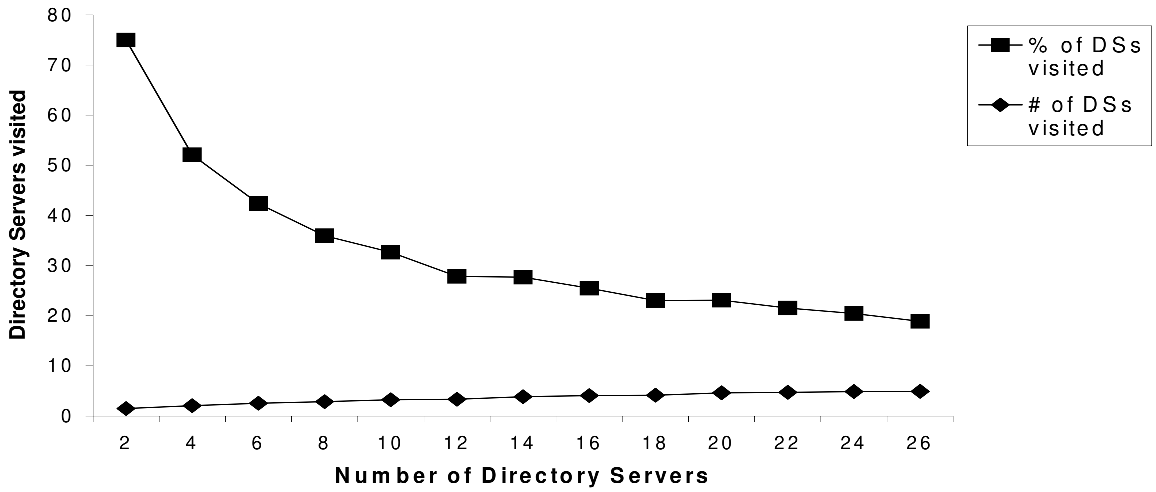
4.4.3 Service Location Performance

The last set of results demonstrates the performance of the system in service location for agents. One measure of the efficiency of an agent is the number of hosts it has to visit in order to achieve its objective. The architecture in general cannot control the number of hosts the agent will have to visit inside the region. Therefore, in the simulations, we restrict ourselves to measuring the number of directory servers that the agent has to visit to complete its mission. We present the effects of scaling different components of the system on this metric – first regions are increased while the directory servers remain constant and next, the directory servers are increased while regions remain constant. In the following experiments, the agent has only one objective in its mission. The service required to fulfill that objective is present in only one region out of all the regions. In this manner, we can isolate the effects of the agent searching for one location in the entire network and assess how long it takes to do so. The target region is picked randomly in every run. Also in every run, the directory server spanning tree is randomly generated and the agent starts out at a randomly picked node.

First, let us consider the number of directory servers that an agent will have to visit in order to find 1 region as the total number of regions is increased. As the number of regions increases, each directory server carries higher number of region advertisements. But the agent can still be directed through the network in the same number of hops, which means that the number of directories it visits remains constant even though the number of regions increases. This fact was verified experimentally. For instance, in a simulation with 25 directory servers and the number of regions increasing from 5 to 50, the agent visits 5 directory servers in every case. The matching time at an individual directory might increase because it would have to match against a higher number of ads. We are not considering the matching time as a measure of performance because it is highly dependent on the algorithm used which in turn depends on the specific application and its requirements. The number of directory servers visited is an absolute measure which is independent of the application. We can conclude that the system scales up well with respect to this property.

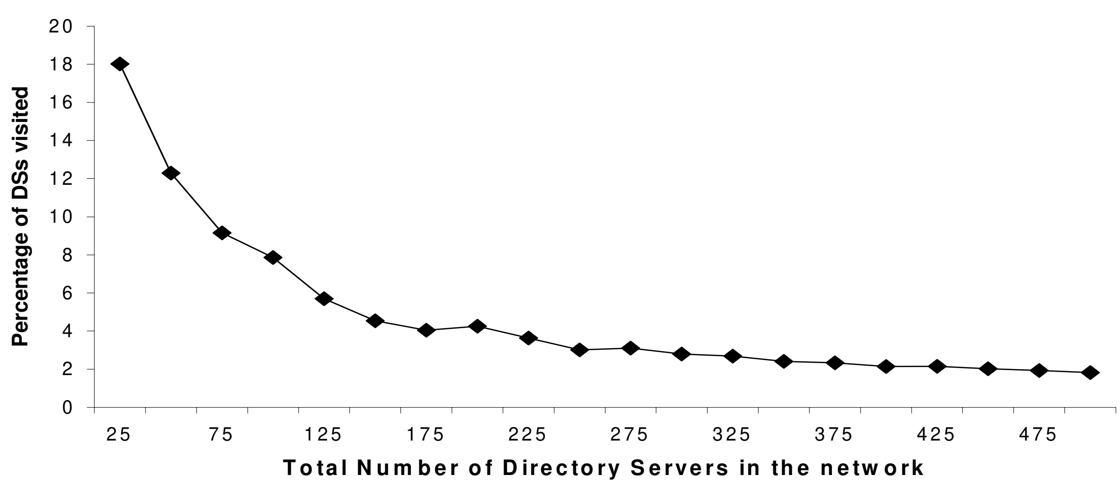
Next, let us consider the number of directory servers that the agent will need to visit in order to find 1 region as the total number of directories is increased, but the number of regions is held constant. The results of 2 simulations are presented. In ***Graph 6***, there are 25 regions and the number of directories increases from 2 to 26. The graph shows that the number of directories that the agent has to visit remains constant even as the total number of directories increases, and hence the percentage reduces. ***Graph 7*** shows the results of the same experiment repeated when the total number of directory servers is varied from 25 to 500. After the number of directories reaches around 400, the percentage visited stabilizes at around 2%. This result is independent of the number of regions in the system.

Graph 6 *: Scalability with increasing number of directory servers* **.**



The behavior exhibited in the above simulations can be attributed to the spanning tree structure of the directory servers. As more and more directory servers are added to the system, in the average case the network graph becomes denser and denser. Therefore, the number of directories that the agent has to visit to locate one region does not increase by a lot even though now there are many more servers in total. This is true under the assumption that directories can

Graph 7 : *Scalability with increasing number of directory servers.*



carry the load of the additional neighbors. The denser structure arises due to the fact that in a spanning tree, each new node that joins can pick its neighbor from among all the existing nodes in the network. Since in every case, the spanning tree graph is randomly generated, in the average case, the structure is dense enough that the average path length from one node to another remains relatively stable.

From the above results, we can conclude that the number of directory servers visited by an agent in order to locate a service scales well with increasing regions and directory servers in the system. The number of directory servers visited is an application-independent metric for service location performance.

This concludes the presentation of the simulation results for the system. In this section, we first analyzed the scalability in terms of the number of messages passed in the system. We concluded that the introduction of regions and a directory server network significantly improves the scalability by reducing the average load per server. We also presented an analysis of the overhead associated with the system. Next, we discussed the addition and deletion of elements of the system, specifically in the startup phase, when regions and directory servers are joining. We demonstrated that the startup costs in terms of additional messages are lower than if there are no regions and a single directory. We concluded that the proposed architecture is well suited to dealing with dynamism of hosts and other elements of the system. Finally, we analyzed the performance of service location for agents and concluded that it scales up well as the number of regions and directory servers is increased.

This also concludes the implementation and evaluation chapter of the thesis. In this chapter, we first presented details of the implementation of the architecture we are proposing in the thesis. We then presented two extensions to the system – an internal organization for regions and a billing application. Finally, we presented an analysis of the system that was done through simulations conducted with the implementation. In the next chapter, we present the conclusions of our work and discuss some areas for future research on the subject.

Chapter 5

Conclusions and Future Work

In this thesis, we presented a network architecture that can be used for location of services in a large-scale distributed system using their descriptions, rather than their addresses. The architecture that we propose is based on the concept of grouping together the service-providing hosts that share common characteristics. In this section, we present the conclusions of our study on this topic and then propose directions for future research on the subject.

5.1 Conclusions

We began this thesis with a study of the different kinds of existing service location systems. We argued the need for an architectural solution to the problem that would be suitable for large-scale systems and that would enable the use of services with a description of their characteristics. We proposed the idea of using regions as a fundamental architectural element that could be used for scaling such systems. The objective of automating the process of locating services is achieved through autonomous mobile agents that chart their own course through the network unsupervised by their creators and only with assistance from the infrastructure.

Our architecture addresses some of the key properties of good distributed system design. The introduction of regions presents a natural entity using which the system can scale. In the directory server network, the property that directories can change the number of advertisements they carry and process depending on their capacity, is a way to ensure that the network does not get overloaded as it increases in size. Analysis of the simulation results presented in Section 4.4 supports these observations as well. The architecture is flexible and suitable for wide-scale deployment because each region is independent and can manage and organize itself in its own preferred manner. It is also dynamic as hosts, regions and directory servers can enter and leave the system at any time. Services can be mobile within regions. The directory server network design is largely self-configuring. Regions may or may not be self-configuring depending on their individual design and properties. Reliability is built into various parts of the system. For instance, agents always verify that the information they received from a directory server is accurate when they actually reach the region and they do the same when they reach a service-providing host. Such actions help in improving reliability of advertisements. Lastly, there are many mechanisms built into the system that provide fault-tolerance and robustness. For example, regions and directory servers do not explicitly need to be deregistered from the system if they fail.

An important property that our architecture does not address directly is security. There are many places in the system where security could be a concern such as functioning of agents, advertisements of regions, etc. In order to restrict the scope of this work, we are assuming that the transfer and operation of agents is secure. We are also assuming integrity for transfer of all kinds of messages in the system. It is possible that individual regions will have their own policies for enforcing security within their boundaries. Another issue that could arise in the scenario of availing services, and that we have not addressed, is privacy. We are assuming that the objectives of agents as well as service advertisements by hosts are both going to be public to the infrastructure so that matching can be performed based on descriptions.

In this thesis, we also presented details of the implementation of the proposed architecture in Java. The implemented system can be used as the basis for an application that uses regions for service location. The system provides facilities for creating regions, adding directory servers and launching agents into the network. If the general-purpose language used for service descriptions is sufficient for the application, the above actions can be performed directly. On the other hand, if an application wishes to define its own language and rules, the system is flexible so that it can be extended and specialized for an application's purposes. Thus, the implementation provides a starting point for systems that wish to incorporate the ideas that we have discussed in this work.

Finally, an important contribution of this work was the exploration of the *Region* abstraction. We showed how the concept of regions can be applied to the scenario of locating services. We discussed how elements could be grouped together into regions based on various different characteristics such as the subject of their services, their commercial activities, their administrative owner, etc. We presented some commercial models that can be used to motivate the existence of regions in this scenario. We concluded from these models that it is possible to design the system such that regions as well as hosts can benefit commercially from the services that they provide to each other and to consumers. This work also presented a study on the kinds of actions that can take place when processes are crossing the boundaries of regions. The actions can be classified into those related to the access control performed at the boundary and the side-effects. Regions and agents exchange information at the boundary to negotiate access. This could result in permission to enter the region; denial; or permission with restrictions imposed on the capabilities or functioning of the agent. The side-effects can include notifications being exchanged between the concerned parties and also change of state at any of the concerned elements. The discussion about the access control by regions and agents and other actions performed at the boundaries of regions can be generalized to any kind of process that is entering or exitting the region.

5.2 Future Work

There are many aspects of this work that can be further explored. The first one of these has to do with the formation of regions and their membership. In order to restrict the scope of the work, we did not focus on the details of how regions are initially formed and how they get members. [8] proposes one model for formation of regions, in which users with shared interests cluster to form self-organizing regions. We presented some commercial models to motivate the existence of regions, but more work could be done on other kinds of incentives for regions to be formed. Further research needs to be done to outline the mechanism of region formation and decisions about their policies and membership such as addition and deletion of members. Also, there are other issues which could be explored to make the system complete such as overlap of regions and regions within regions.

The second area of future work could be in the topology of the directory server network. Currently, the directory servers are organized as a spanning tree where each node connects to another node in the network, based on minimizing (or maximizing) an application-specific metric. As mentioned in [1], despite each node making a local minimization (or maximization) decision, the resulting spanning tree will not in general be the minimum (or maximum) one because nodes join in order and can only decide from among the nodes that have joined before them. Also, another shortcoming of a spanning tree structure is that if one of the nodes in the tree dies, the entire tree (specifically, all the nodes that joined after the node that died) needs to be reorganized. Further work could be done in eliminating these limitations of the spanning tree structure and also in exploring alternative structures for the directory server network. There are a number of engineering tradeoffs to consider in the organizational structure of the directory server network. For example, exchanging more queries between individual servers will give the agent a higher probability of finding the information it is looking for. Here, we are trading the scalability of the system (which will decrease as more queries are exchanged) with efficiency of the agent's operation. Various parameters determine the performance of the system such as the number of neighbors of each directory, the amount of information that is exchanged between them, the directions of search queries, etc. Features of the spanning tree itself such as the branching factor and the depth will also affect the outcome. It is possible that structures resulting from different combinations of parameter values would be suitable for different kinds of applications.

The third possible avenue for further work is in the area of the communication protocol used in the system, and specifically, in the service advertisement protocol and the language used by mobile agents to describe their objectives and policies. As mentioned before, we chose a simple and general-purpose representation for all the languages in the system. It would be useful to explore more complex representations such as hierarchical attribute structures and corresponding algorithms that provide matching for them. It might also be useful to build more aggregation schemes for regions. We proposed one particular hierarchical aggregation scheme in section 4.2.

Also, before actual deployment of a system like the one we propose, all the related security and privacy issues would have to be explored. As mentioned, we did not focus on these aspects of the system. Regions could handle the security and privacy individually, but considerations need to be made for these issues in the functioning of the directory server network and the mobile agents as well. Lastly, the value of the proposed architecture will be realized through applications built over it. Future work could be done in building particular applications over our system to study it further and explore more issues with it. This would also be valuable in collecting more data about the performance of the system and in analyzing how it behaves in different scenarios.

We hope that the work we have done will be useful in providing the basis for future research in these and other directions.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The Design and Implementation of an Intentional Naming System. In *17th ACM Symposium on Operating Systems Principles*, December 1999.

[2] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *18th ACM Symposium on Principles of Distributed Computing (PODC'99)*, May 1999.

[3] Ajanta. Project homepage: `http://www.cs.umn.edu/Ajanta/home.html`. 2003.

[4] P. Albitz, and C. Liu. DNS and BIND. *O'Reilly and Associates*, 1998.

[5] N. Asokan, P. Janson, M. Steiner, and M. Waidner. The state of the art in electronic payment systems. *IEEE Computer*, *30(9)*, pages 28–35, September 1997.

[6] M. Balazinska, H. Balakrishnan, and D. Karger. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. In *Proc. of the 1st International Conference on Pervasive Computing,* pages 195-210, August 2002.

[7] G. Banavar, T. D. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, May 1999.

[8] R. Beverly. Reorganization in Network Regions for Optimality and Fairness. *Master's Thesis, MIT/EECS*, August 2004. URL: `http://www.mit.edu/~rbeverly/papers/beverly-thesis.pdf`

[9] H. Boley, S. Tabet, and G. Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Semantic Web Working Symposium (SWWS)*, August 2001.

[10] J. Bredin, D. Kotz, and D. Rus. Economic markets as a means of open mobile-agent systems. In *Proc. of the Workshop on Mobile Agents in the Context of Competition and Cooperation (MAC3) at the 3rd International Conference on Autonomous Agents (AA99)*, pages 43-49, May 1999.

[11] A. Carzaniga, and A. Wolf. Content-based Networking: A New Communication Infrastructure. In *NSF Workshop on Infrastructure for Mobile and Wireless Systems*, October 2001.

[12] P. Castro, B. Greenstein, R. Muntz, C. Bisdikian, P. Kermany, and M. Papadopouli. Locating application data across service discovery domains. In *Proc. of the 7th Annual International Conference on Mobile Computing and Networking*, July 2001.

[13] B. Chandra, M. Dahlin, L. Gao, A. Khoja, A. Nayate, A. Razzaq, and A. Sewani. Resource Management for Scalable Disconnected Access to Web Services. In *Proc. of the 10th International World Wide Web Conference (WWW10)*, May 2001.

[14] A. Cheyer, and D. Martin. The Open Agent Architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 4, no. 1, pages 143-148, March 2001.

[15] R. Chinnici, M. Gudgin, J. Moreau, and S. Weerawarana. Web Services Description Language (WSDL) 1.2, *World Wide Web Consortium (W3C)*, 2002.

[16] D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proc. ACM SIGCOMM,* pages 106-114, August 1988.

[17] S. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An Architecture for a Secure Service Discovery Service. In *Proc. of the 5th Annual International Conference on Mobile Computing and Networks (Mobicom'99)*, August 1999.

[18] W. Diffie, P. Oorschot, and M. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes, and Cryptography*, 2(2), pages 107-125, June 1992.

[19] J. Feghhi, J. Feghhi, and P. Williams. Digital Certificates - Applied Internet Security. *Addison-Wesley-Longman*, 1999.

[20] P. Francis, T. Kambayashi, S. Sato, and S. Shimizu. Ingrid: A Self-Configuring Information Navigation Infrastructure. In *Proc. of the 4th International World Wide Web Conference*, pages 519-537, December 1995.

[21] A. Furche, and G. Wrightson. Computer Money - A Systematic Overview of Electronic Payment Systems. *DPunkt Verlag*, 1996.

[22] R. Gray. Agent Tcl: A flexible and secure mobile-agent system. In *Proc. of the Fourth Annual Tcl/Tk Workshop (TCL '96)*, July 1996.

[23] C. Harrison, D. Chess, and A. Kershenbaum. Mobile Agents: Are they a good idea? *Technical report, IBM*, March 1995. URL: `http://www.research.ibm.com/massdist/mobag.ps`

[24] D. Kotz, and R. Gray. Mobile Code: The Future of the Internet. In *Proc. of the Workshop on Mobile Agents in the Context of Competition and Cooperation (MAC3) at Autonomous Agents '99*, pages 6–12, May 1999.

[25] J. Kulik. Fast and Flexible Forwarding for Internet Subscription Systems. In *Proc. of the 2nd International Workshop on Distributed Event-based Systems*, pages 1-8, June 2003.

[26] D. Lange, and M. Oshima. Programming and Deploying Java Mobile Agent with Aglets. *Addison-Wesley*, 1998.

[27] O. Lassila, and R. Swick. Resource Description Framework (RDF): Model and Syntax Specification. *World Wide Web Consortium (W3C),* 2000.

[28] C. Law. Garbage Collection in Regions. *Master's thesis, MIT/EECS*, May 2003. URL: `http://krs.lcs.mit.edu/regions/docs/law-thesis.pdf`

[29] J. Li. Improving Application-level Network Services with Regions. *Master's thesis, MIT/EECS*, May 2003. URL: http://krs.lcs.mit.edu/regions/docs/li-thesis.pdf

[30] Y. Liu, and B. Plale. Survey of Publish Subscribe Event Systems. *Technical Report TR574, Computer Science Department, Indiana University*, May 2003. URL: http://www.cs.indiana.edu/pub/techreports/TR574.pdf

[31] N. Minar, K. Kramer, and P. Maes. Cooperating Mobile Agents for Dynamic Network Routing. In *Software Agents for Future Communication Systems*, *Springer-Verlag*, 1999.

[32] P. Mockapetris. Domain names - concepts and facilities. *RFC 1034*, November 1987.

[33] P. Mockapetris. Domain names - implementation and specification. *RFC 1035*, November 1987.

[34] G. Mühl. Generic Constraints for Content-Based Publish/Subscribe Systems. In *Proc. of the 6th International Conference on Cooperative Information Systems (CoopIS '01)*, September 2001.

[35] G. Mühl, L. Fiege, F. Gärtner, and A. Buchmann. Evaluating Advanced Routing Algorithms for Content-Based Publish/Subscribe Systems. In *Proc. of the 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002),* October 2002.

[36] Y. Rekhter, and T. Li. A Border Gateway Protocol 4 (BGP-4). *RFC 1771*, March 1995.

[37] V. Roth. Obstacles to the Adoption of Mobile Agents. In *Proc. of the IEEE International Conference on Mobile Data Management (MDM '04)*, January 2004.

[38] R. Schemers. lbnamed: A Load Balancing Name Server in Perl. In *Proc. of LISA '95*, September 1995.

[39] K. Sollins. Architectural Principles of Uniform Resource Name Resolution. *RFC 2276*, January 1998.

[40] K. Sollins. Designing for Scale and Differentiation. In *Proc. of ACM SIGCOMM 2003 - Workshop on Future Directions in Network Architecture*, August 2003.

[41] K. Sollins. Effective Design of Naming Systems for Networks. URL: http://krs.lcs.mit.edu/regions/docs/naming.pdf, February 2003.

[42] K. Sollins. Regions: A new architectural capability for networking. *White paper*, August 2001. URL: http://krs.lcs.mit.edu/regions/docs/regions-wh.pdf

[43] K. Sollins, and L. Massinter. Functional Requirements for Uniform Resource Names. *RFC 1737*, December 1994.

[44] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open network systems. In *Proc. of USENIX Winter Conf.*, pages 191–202, February 1988.

[45] J. Tardo, and L. Valente. Mobile Agent Security and Telescript. In *Proc. of COMPCON Spring '96, pages 58-63*, February 1996.

[46] A. Tripathi, and N. Karnik. A Security Architecture for Mobile Agents in Ajanta. In *Proc. of the International Conference on Distributed Computing Systems*, April 2000.

[47] A. Tripathi, and N. Karnik. Protected Resource Access for Mobile Agent-based Distributed Computing. In *Proc. of the ICPP Workshop on Wireless Networking and Mobile Computing*, August 1998.

[48] A. Tripathi, N. Karnik, M. Vora, T. Ahmed, and R. Singh. Mobile Agent Programming in Ajanta. In *Proc. of the 19th International Conference on Distributed Computing Systems (ICDCS '99)*, pages 190-197, May 1999.

[49] J. White. Mobile Agents. In *Software Agents, J. Bradshaw (Ed.), AAAI Press and MIT Press*, Chapter 19, pages 437-472, 1996.

[50] S. Willmott, M. Somacher, I. Constantinescu, J. Dale, S. Poslad, and D. Bonnefoy. The Agentcities Network Architecture**.** In *Proc. of the First International Workshop on Challenges in Open Agent Systems*, July 2002.

[51] J. Wroclawski. The Metanet. *Workshop on Research Challenges for the Next Generation Internet, ed. Computing Research Association*, May 1997.

[52] B. Yuwono, S. Lam, J. Ying, and D. Lee. A World Wide Web Resource Discovery System. In *Proc. of the 4th International WWW Conference*, pages 145-158, December 1995.