An Adaptive Communication Model for Mobile Agents in Highly Dynamic Networks based on Forming Flexible Regions via Swarming Behaviour

Dissertation

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

seit 1558

vorgelegt dem Rat der Fakultät für Mathematik und Informatik

der Friedrich-Schiller-Universität Jena

von

MSc. Phuong Thanh Nguyen

geboren am 10. September 1979 in Vietnam

1. Gutachter: Prof. Dr. Wilhelm R. Rossak, Friedrich-Schiller-Universität Jena.

2. Gutachter: Prof. Dr. Christian Erfurth, Ernst-Abbe-Fachhochschule Jena.

Tag der Abgabe: 20. Juni 2012

Tag der öffentlichen Verteidigung: 28. August 2012

Zusammenfassung

Im letzten Jahrzehnt gilt die mobile Agententechnologie als eines der wichtigsten Forschungsgebiete der Informatik. Mobile Agenten sind Software, die Aufträge im Namen ihrer Besitzer erfüllen können (ZK02). Mobile Agenten können selbstbestimmend von Server zu Server migrieren, sie können ihren Arbeitsstand speichern und dann ihre Arbeit am neuen Aufenthaltsort fortsetzen. Ihre wichtigsten Merkmale sind: autonom, reaktiv, opportunistisch und zielgerichtet. Diese genannten Merkmale sind für verteilte Anwendungen geeignet, z. B: Ressourcenverteilung (TYI99), Netzwerkmanagement (MT99), E-Commerce (BGP05), Fernüberwachung (CMCV02), Gesundheitssysteme (Mor06), um nur einige zu nennen. Es ist die Mobilität der Agenten, die mobile Agenten zu einer guten Computing Technologie macht (Pau02).

Kommunikation ist wesentlich in verteilten Systemen, und dies gilt auch für mobile Agentensysteme (LHL02). Neben den eher technischen Aspekten mobiler Agententechnologien, wie Migration (Bra03) und Kontrollmechanismen (Bau00), wurde die Kommunikation zwischen den Agenten als eine der wichtigsten Komponenten in der mobilen Agententechnologie identifiziert (FLP98). Es ist diskutiert worden, ob Agentenkommunikation ausschließlich lokal sein sollte, angesichts der Tatsache, dass mobile Agenten erfunden wurden, weil man die Verarbeitung zu den Daten tragen möchte, anstatt umgekehrt (SS97). Allerdings hat es sich gezeigt, dass es in vielen Fällen lohnt, wenn die mobilen Agenten kommunizieren anstatt migrieren (BHR+97),(FLP98),(ea02). Kommunikation hilft mobilen Agenten, eine bessere Leistung zu erreichen (Erf04). Kommunikation ist daher aus unserer Sicht die Basis mobiler Agentensysteme.

An der Friedrich-Schiller-Universität Jena ist das interdisziplinäre Projekt Speed-Up seit April 2009 durchgefürht worden (FSU11). Das Projekt entwickelt ein Unterstützungssystem für Rettungs- und Einsatzkräfte bei einem Massenanfall von

Verletzten (MANV). Im Projekt ist das Konzept mobiler Agenten als eine der Basistechnologien ausgesucht worden. Die hohe Netzwerkdynamik stellt neue Herausforderungen für mobile Agentensysteme dar, die in MANV Rettungsszenarien arbeiten. Es wird erwartet, dass die Kommunikation sich an die dynamische Umgebung zur Ausführungszeit anpassen kann. Dazu fehlen heute tragfähige Konzepte.

In dieser Arbeit wird daher ein Ansatz zur adaptiven Kommunikation mobiler Agenten in hochdynamischen Netzwerken des SpeedUp-Typs vorgestellt. Nach unserer Beurteilung sollte die Kommunikation zwischen den mobilen Agenten nicht nur Interoperabilität und Standortunabhängigkeit, sondern auch Anpassungsfähigkeit aufweisen. Wir schlagen ein Kommunikationsmodell vor, das sich auf den koordinierenden Aspekt und das Zusammenspiel der Agenten konzentriert, sowie die Zuverlässigkeit und die Fehlertoleranz unterstützt. Um die Netzwerkdynamik zu managen, planen wir einen selbstorganisierten Mechanismus zu verwenden, der sich ”honey bee” inspiriertes Verfahren nennt. Wir werden dazu eine Software für ein adaptives Kommunikationsmodell mobiler Agenten, basierend auf das mobile Agentensystem Ellipsis gestalten, implementieren, und evaluieren.

Abstract

In the last decade, mobile agent technology has been considered as one of the most active research fields in computer science. Mobile agents are software agents which run on behalf of their owner to fulfil jobs that have been ordered (ZK02). They have the ability to migrate from location to location in the network, they can temporarily save their work state at the time of migrating and then restore their tasks when arriving at the new location. Their outstanding characteristics are to be autonomous, reactive, opportunistic, and goal-oriented. Those characteristics are suitable for distributed applications, such as resource allocation (TYI99), network management (MT99), remote supervision (CMCV02), e-commerce (BGP05), health care systems (Mor06), to name but a few. It is the mobility of mobile agents that makes them to be a powerful computing technique, especially for pervasive computing (Pau02).

Communication is an essential component of distributed systems and this is no exception for multi-agent systems (LHL02). Besides technical aspects of mobile agent technology, such as migrations (Bra03) and control mechanisms (Bau00), communication between mobile agents has been identified as an important issue in mobile agent technology (FLP98). It has been argued whether agent communication should be remote or restricted to local, considering that the main reason for the birth of mobile agents was to move computation to the data instead of moving the data to the computation. Therefore, remote communication could be avoided completely (SS97). However, it has been shown that in many cases mobile agent systems can benefit from performing communication instead of sending agents to remote platforms (BHR+97),(FLP98),(ea02). The communication between agents helps to increase the chance that an agent attains its objectives (Erf04). Communication is one of the bases of multi-agent systems; it is difficult, if not impossible for a group of agents to solve tasks without communication (Hel03).

At Friedrich Schiller University Jena, an interdisciplinary project, named SpeedUp, for the support of handling mass casualty incidents (MCI) has been in development since April 2009 (FSU11). In the project the mobile agent concept has been selected as one of the main technologies on the communication infrastructure level. The dynamic nature of MCI networks poses new challenges to mobile systems working in a rescue scenario. For mobile agent systems working in highly dynamic networks, communication between mobile agents is expected to adapt easily to environmental stimuli which occur at execution time. Much research has been done into the design of an appropriate, highly flexible model for mobile agent communication in dynamic networks. However, to the best of our knowledge none of the suggested solutions has been able to achieve the necessary performance and quality attributes to count as a practical solution. In most cases, these existing approaches seem to neglect the inherent dynamics of modern networks.

In this dissertation, we present our approach for an adaptive communication model for mobile agent systems in highly dynamic networks of the SpeedUp type. In our opinion, communication in mobile agent systems should deal not only with interoperability and location-transparency, but also with adaptability. To achieve industrial strength, we propose a model for agent communication that focuses on the cooperation aspect of agent interaction and supports reliability and fault tolerance as the key qualities, while keeping up an acceptable overall performance at the same time. For the management of highly dynamic communication domains we use a self-organizing mechanism, a so-called honey bee inspired algorithm. In order to ensure message delivery, we propose a resilient mechanism for the management of a mobile agent's location. Based on this thesis, we will design, implement and evaluate a software prototype for an adaptive model for mobile agent communication based on the Ellipsis mobile agent system.

Acknowledgments

First and foremost, I would like to express the sincerest thanks to my Doktorvater, Prof. Dr. Wilhelm R. Rossak. Without his help I would not have been able to finish this dissertation. His crisp advice is always value instructions that help me promote ideas and prevent me from veering off course. I owe a debt of gratitude to him because he accepted me as his PhD student. I have learnt a lot from him, not only in computer science but also in acquiring necessary skills of a researcher. His preciseness and compassion should always remind me of him.

I am thankful to Prof. Dr. Christian Erfurth who has been willing to write recommendation letters for me so that I was able to extend the DAAD scholarships during my study. I appreciate to Prof. Dr. Birgitta König-Ries for her willingness to provide me with a WiHi job. I also thank Dr.-Ing. Volkmar Schau for helping me with the first steps in my work.

I thank Ms. Cornelia Müsse for proofreading the German abstract of my dissertation as well as for helping me by the secretarial procedures. My thanks also go to my colleagues: Uwe Krüger, Frederik Schulz, Aygul Gabdulkhakova for a homely atmostphere. Without friends the past years would have been harder, so I thank them for being with me: An Van, Son Dinh, Thang Truong, Ha Le, Tri Pham, Tung Vo and many others.

I feel a tremendous sense of obligation towards my wife Thu Hai. She sacrifices a lot for me. She has been bringing up our beloved daughter Phuong Anh while I was away. Thinking about them is one of the incentives that encourage me to go on. I also thank my big family, my parents and my parents-in-law for supporting me unconditionally during my stay in Germany.

I gratefully acknowledge the financial support for my study from the Vietnamese people, represented by the Government of Vietnam. I also thank the German Academic Exchange Service (DAAD) for backing me on my study.

Contents

List of Figures

List of Tables

Listings

Part I

Problem Statement and State of the Art

Chapter 1

Introduction

1.1 Preamble

Besides technical aspects of mobile agents, such as agent migrations (Bra03) and control mechanisms (Bau00), communication between mobile agents has been identified as an important issue in mobile agent technology. It has been argued whether agent communication should be remote or restricted to local, considering that the main reason for the birth of mobile agents is to move computation to the data instead of moving the data to the computation, and therefore remote communication could be avoided (SS97). However, it has been shown that in many cases, mobile agent systems can benefit from performing communication instead of sending agents to remote platforms (BHR+97). The communication between agents helps to increase the chance that an agent can achieve its objectives (Erf04). Communication is one of the bases of multi-agent systems; it is difficult, if not impossible for a group of agents to solve tasks without communication (Hel03). Information sharing gives agents the opportunity to learn more about what other agents have already undergone so that they can fulfil their tasks faster (CGL07). It can be summarized that, in mobile agent systems, remote communication cannot be completely replaced by agent mobility and therefore mobile agents should be equipped with remote communication capabilities (AP95).

It is the mobility of agents that makes the communication between them complicated. Communication in mobile agents consists normally of two phases: agent

tracking and message delivery. Since agents migrate from agency to agency to fulfil their tasks, there must be a mechanism that monitors the current location of an agent in the system, this is called agent tracking. The second phase, message delivery, takes place after the location of the targeted agent has been found. The two phases seem to be contradictory in performance since the more effort we pay for monitoring the agents, the easier the message delivery phase is, and vice versa. The mobility of agents may lead to severe message losses when communication excutes, especially in large scale networks. To achieve system's reliability, these losses need to be mitigated.

At the Chair of Software Engineering of Friedrich Schiller University Jena, mobile agent technology is considered as one of the leading research fields. The chair has been concentrating not only on improving the performance of mobile agent system (MAS), but also on spreading the applications of the technology. Many intensive studies have been conducted to pursue this objective. Peter Braun had dedicated his whole PhD Dissertation to the migration process of mobile agents in 2003. Soon after, Christian Erfurth successfully developed ProNav - a proactive autonomous navigation framework for mobile agents (Erf04). With the framework, mobile agents are equipped with the abiltity to percept the network conditions and therefore they can route and plan themselves in a changing environment (DER04),(ED04). In MAS level 2, agents are able to plan their route through the network themselves, based on relevant service descriptions and by using dedicated routing services, they have the ability to navigate the network in a highly autonomous fashion. However, to achieve useful autonomous navigation, a large number of up-to-date service descriptions must be handled effectively. The PhD Dissertation of Arndt Döhler deals with this issue (DÖ8). From the fact that current infrastructure frameworks are not scaling well with the potential number of service descriptions, cannot map the dynamics in the network, the thesis focuses on integrating scalability and flexibility into a unified solution framework - QuickLinkNet - that supports mobile agents in their autonomous behaviour (DER05).

Since April 2009, a project for the support of mass casualty incidents (MCI) rescue has been executed by Friedrich Schiller University Jena and its partners. The project, named SpeedUp, consists of two main areas: SpeedUp Practice and SpeedUp Technology. SpeedUp Technology is responsible for the IT support of

rescue forces in MCI situations so that in disaster events, rescue actions can be performed in a more effective way. The IT solution supports the usual rescue measures by providing a framework for improved information handling, additional preprocessed sensor data, and a basis for a highly flexible communication infrastructure (FSU11). In SpeedUp the mobile agent concept has been selected as one of the main technologies on the communication infrastructure level. It is the inspiration for our work; we expect useful practical inputs from it and in return, our work is expected to contribute to the implementation of the communication architecture for the project. We are, therefore, going to design, implement, and evaluate in this project environment a prototype for adaptive mobile agent communication in highly dynamic networks. To achieve industrial strength, we propose a model for agent communication that focuses on the cooperation aspect of agent interaction and supports reliability and fault tolerance as the key qualities, while keeping up an acceptable overall performance at the same time.

For mobile agent systems which work in highly dynamic networks, communication between mobile agents is expected to adapt easily to environmental stimuli which appear only at execution time. Communication should deal not only with interoperability and location-transparency, but also with adaptability. Several studies have been published so far which address the design of an appropriate, highly flexible model for mobile agent communication. However, to the best of our knowledge none of the suggested solutions has been able to achieve the necessary performance and quality attributes to count as a practical solution. In most cases, these existing approaches seem to neglect the inherent dynamics of modern networks. Thus, in our view adaptive mobile agent communication remains a challenging topic. We are interested in developing an adaptive communication model for mobile agents which is suitable for tasks in highly dynamic networks.

1.2 Prerequisites and Goals

Our goals and prerequisites can be summarized as follows:

- We will base our mobile agent implementation on the mobile agent system Ellipsis (Sch12) as the basic agent instantiation and execution environment.

All solutions will be implemented as an extension of this MAS.

- To ensure maximum interoperability, we opt for the FIPA Agent Communication Language Specification as the messages' semantics representation. Messages for the communication between mobile agents will, thus, be formatted according to this FIPA Standard.
- We will implement as a basic feature a stable location-transparency technique that supports instantiated mobile agents by keeping their communication independent from the currently allocated execution platform and the platform's type.
- We will improve the collaboration between instantiated mobile agents, based on location transparency, by making their communication and its structure adaptive to the topological changes within the network. The targeted qualities of collaboration will focus on fault tolerance and reliability.

1.3 Dissertation Structure

The dissertation is divided into three parts. Part I, which is made of chapters 1, 2, 3, and 4, provides the reader with a view of mobile agent communication. Chapter 1 is the introduction to the work. Chapter 2 provides the fundamentals of agent communication. Chapter 3 presents state of the art in mobile agent communication. Chapter 4 introduces the context in which mobile agent technology is going to be employed and the need for agent communication. Part II begins with Chapter 5, which presents the thesis of the dissertation. Chapter 6 gives a biological background as well as describes self-organizing mechanisms and their application for highly dynamic networks. This chapter also presents our approach for an adaptive communication model for mobile agents in highly dynamic networks of the SpeedUp-Type. Chapter 7 clarifies the design of the software prototype and its constituent components. Chapter 8 provides the implementation details. Part III deals with the evaluation of the software prototype. Chapter 9 focuses on the system's functionalities. Chapter 10 evaluates the targeted qualities. Some conclusions and a look at future work will be given in the last chapter of the dissertation.

Chapter 2

Fundamentals of Agent Communication

In recent years agent technology has attracted attention of the research community. Agent is a software entity that is created and owned by a user. On behalf of its owner, an agent acts in a responsive manner to external stimuli to fulfil bespoken tasks. The prominent advantages of agent should be autonomous processing, flexibility and intelligence. These characteristics are particularly suitable for tasks in distributed environment (Bra97).

Multi-agent systems facilitate distributed problem solving since agents are able to work to solve common tasks in a pervasive scale. The ability to communicate features in multi-agent systems, communication has been identified as one of the major issues in these types of systems (VMWB07). Agent communication may present at various forms; it can be the exchange of comprehensible messages, a request to perform an action, or cooperation negotiation (Cal02). Through communication agents share information and eliminate conflicts in goals.

Communication in multi-agent systems holds main features belonging to communication in conventional distributed systems. What makes it differ from that is the exchanging of semantic contents to achieve a higher level interactivity (FLP98). Agent communication, when being confronted with, means not simply sending a stream of bits from a sender to a receiver, but the ability of both sides to process messages meaningfully. Thus, there are two independent levels in agent communication: low level and high level communication. For low level communication,

it is about to transfer packets of bytes over networks, like in any other distributed system. By the high level communication, which is specific to intelligent and agent systems, the meaning of exchanged messges is parsed and processed.

Agent communication has been built based on some main theories. High level communication in agent systems is inspired by speech act theory, the discipline that deals with the study of human conversations in real situations. The semantic representation of agent messages is achieved by means of a shared ontology, which is defined as a conceptualization of objects and relations. The type of communicative acts is represented by an agent communication language (FIP02d). In this chapter, these foundations namely the speech act theory, agent communication language, and ontology will be highlighted. In addition, the topics related to low level communication will also be discussed. Afterwards the major issues regarding mobile agent communication will be examined.

2.1 Introduction

Given that agents are created by different programming language and run on different operating systems as well as hardware platforms, if two software agents want to communicate, there must exist an infrastructure for communication.

In order to communicate, agents must comply with the following requirements (Obi07):

- At the physical level, agents are able to send and receive packets that contain agent messages over existing network infrastructure.
- At the syntactic level, agents are able to parse messages correctly into different fields, such as: message header, message content, language, sender.
- To guarantee that the meaning of is preserved, agents must commit to a shared ontology. An ontology facilitates the process of understanding message fields meaningfully.

In Sections 2.2, 2.3 and 2.4, we will present speech act theory, agent communication and ontology in details. These are the most important bases of agent communication.

2.2 Speech Acts

Speech act theory aims to study how humans use language to attain objectives in everyday life. The philosopher Austin and his student Searle are the two pioneers, whose contributions are significant in this field. Speech act theory has been described thoroughly in Austin's book named *"How to do things with words"* (Aus75).

Speech act theory is based on the premise that "language is action." A speaker utters to put influence on the intention of the hearer. Austin classifies two categories: utterance-as-description and utterance-as-doing, he also points out different facets of speech acts:

- Locutinary act: The act of forming an utterance following grammar rules.
- Illocutionary act: The act of performing an action by uttering meaningfully. For example, a father tells his son: "I will buy you a bicycle tomorrow" he conducts the illocutionary act of making a promise.
- Perlocutionary act: The act of having influence on the hearer. For example, a teacher reminds his pupils: "Please remember to do the homework!", he performs the perlocutionary act of forcing the pupils to do the homework.

Speech acts are sorted into five groups:

- Representatives: A speaker expresses his belief, e.g., "The train is coming."
- Directives: A speaker shows his desire, or tells a hearer to perform some action, e.g., "Please get in the train."
- Commisives: A speaker indicates his intention to do something, e.g., "I'll awake you when the train stops."
- Expressives: A speaker expresses a mental state, e.g., "Thank you for awaking me."
- Declaration: A speaker declares something, e.g., "On behalf of the jury I declare the train's team has won the prize "Best Service" of the year."

A speech act is considered to have two components:

- a performative: accept, inform, request, etc.
- a propositional content: ”It is raining outside!”

Agent communication emulates speech acts from human conversations. Utterances may be considered as a foundation for communication among agents, with the intention of changing the mental state of the hearer of the utterance. Likewise, agents send message to other agents to put some influence of the receivers, with the intention that the receiver will perform some action suggested in the message's content and message's type (VMWB07).

2.3 Agent Communication Language

An agent communication language facilitates communication between agents, it allows an agent to exchange knowledge with other agents regardless of their platform, architecture, programming language, and reasoning system (LF98). Essentially, an agent communication language can be considered as a set of message types, called performatives. Each performative has a pre-defined meaning and represents an intention of a sender towards a receiver. An agent communication language can be used for communication between following entities (FLP98):

- Static agents.
- Static and mobile agents.
- Mobile agents.
- Mobile agents and information sources.

FIPA-ACL and KQML are two most widely used communication languages for inter-agent communication. They both emulate speech acts from human and share a very similar syntax and semantics.

2.3.1 KQML and KIF

KQML (The Knowledge Query and Manipulation Language) represents performatives. KIF (Knowledge Interchange Format) defines message content, properties of things and their relationships in a domain.

A KQML message is made of three layers: the content layer, the message layer, and the communication layer. The message layer specifies the performative, the content language, and the ontology. Performatives are multiple types of messages, each performative specifies the intention of a message. Currently, there are over 40 performatives in KQML. A message content does not tell much which actions need to be executed, in certain circumstances, the message's content might be confused because it can be understood in different ways. A performative allows an agent to narrow the scope of a message it has received. As a result, it is able to parse the message in the right way as expressed at the sender side.

2.3.2 FIPA ACL

FIPA ACL has been developed mainly based on KQML with some improvements. FIPA ACL is specified with precisely syntax definitions, semantics and pragmatics. FIPA ACL does not deal with the physical exchange over the network, but with the specification of the exchange content. Like in KQML, FIPA ACL distinguishes between different types of messages also called performative. Currently there are total of 22 performatives in the FIPA Specification (FIP02d).

An agent of FIPA type must be able to (PC00):

- interpret and send a not-understand message,
- correctly implement ACL messages according to their syntactic definition,
- correctly make use of ACL performatives according to their semantic definition, and finally
- generate messages in the transport form that corresponds to the messages they want to send.

A FIPA-ACL message is made of a set of one or more message elements. The mandatory field in this type of messages is *performative*. A normal FIPA ACL message also contains other fields, such as *sender*, *receiver*, and *content* as shown in Figure 2.1. The *content* field represents the domain dependent component of the communication; The *language* field identifies the content language (CL), *ontology* specifies the ontology used for knowledge sharing. In conjunction with ACL; The ontology field supports the interpretation of the content expression. Agents may use same or different ontology, but they must have the ability to interpret between them.

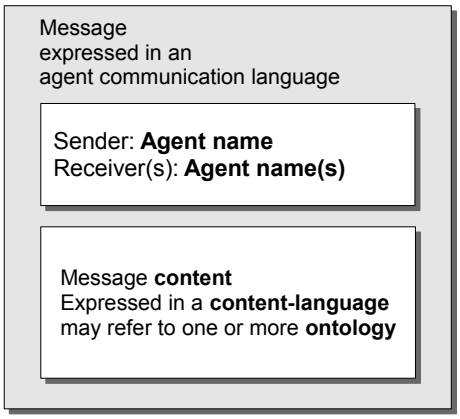

Figure 2.1: A FIPA Message (FIP00a)

2.4 Ontology

Sharing and reusing knowledge across heterogeneous systems were problematic since each system has its own way of defining and representing concepts. It is a common that systems refer to the same concepts but use different representation. In a conversation between agents from different background, agents may have different conceptualizations of objects and relations. In order to facilitate knowledge sharing between agents, it is necessary for them to have an agreement of a shared model of the world. That means, the involving agents must agree on a common specification of objects, entities and relationships.

Ontology has been invented in need for interoperability of heterogeneous systems. An ontology for a domain is a conceptualization of the world, i.e., objects, qualities, distinctions and relationships in that domain (FLP98). An ontology assists computers to process and exchange information automatically and rapidly, without intervention from humans. Ontology defined by Gruber *"is a particular conceptualization of a set of objects, concepts and other entities about which knowledge is expressed and of the relationships that hold among them. It provides the basic structure for building knowledge base"* (PB02),(Gru93),(Gru95).

Ontology provides a vocabulary of terms to describe entities in some domain of interest, as well as a set of assumptions about the meaning of the terms in the vocabulary. New knowledge can be deduced from the existing terms represented by ontology.

In agent implementation, ontology can be stored in an ontology service or coded within the software so that no ontology service is needed (FIP01).

2.5 Communication Architecture

2.5.1 Message Transport Model

FIPA defines a Reference Model for Message Transport. The model comprises of three levels (FIP00b):

- The Message Transport Protocol (MTP) is used to transfer physically messages between two Agent Communication Channels (ACC).
- The Message Transport Service (MTS) supports the transportation of FIPA ACL messages between agents on any given Agent Platform (AP).
- The ACL represents the payload of the messages carried by both the MTS and MTP.

There are two typical transport models: Message passing, and RPC/RMI (Remote Procedure Calls/Remote Method Invocation):

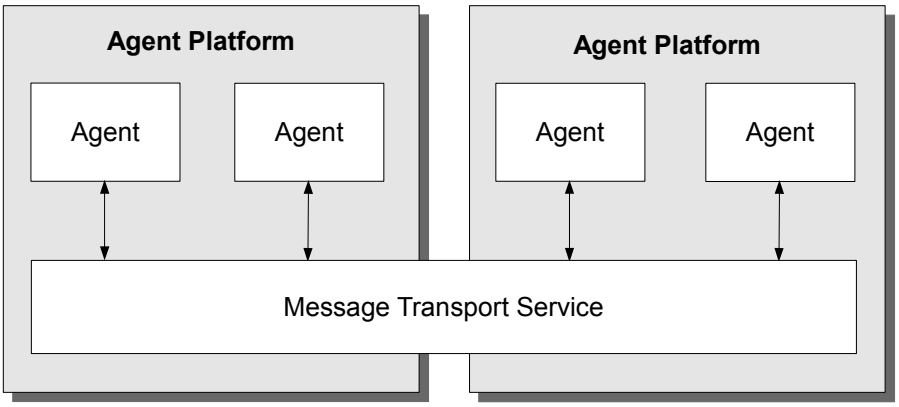

Figure 2.2: Message Transport Reference Model (FIP00b)

- Message passing is a form of communication used in parallel computing, and inter-process communication. In this model, processes or objects can send and receive messages.
- Remote Procedure Call (RPC): RPC is an inter-process communication technology that provides a computer program the ability to call a procedure in another address space, normally a remote server.
- Remote Method Invocation (RMI): RMI provides for remote communication between programs written in Java. This is the object-oriented version of RPC. It allows an object running Java Virtual Machine to invoke methods on an object running in another Java Virtual Machine.

2.5.2 The Layered Architecture

Abstraction methods play an important role in system design as they can be used to represent the structure of a system in a new space so that reasoning in the new model is simpler than that in the original (Cal02). For agent communication, an abstract for agent communication had been introduced in (Hel03) as illustrated in the upper part of Figure 2.3.

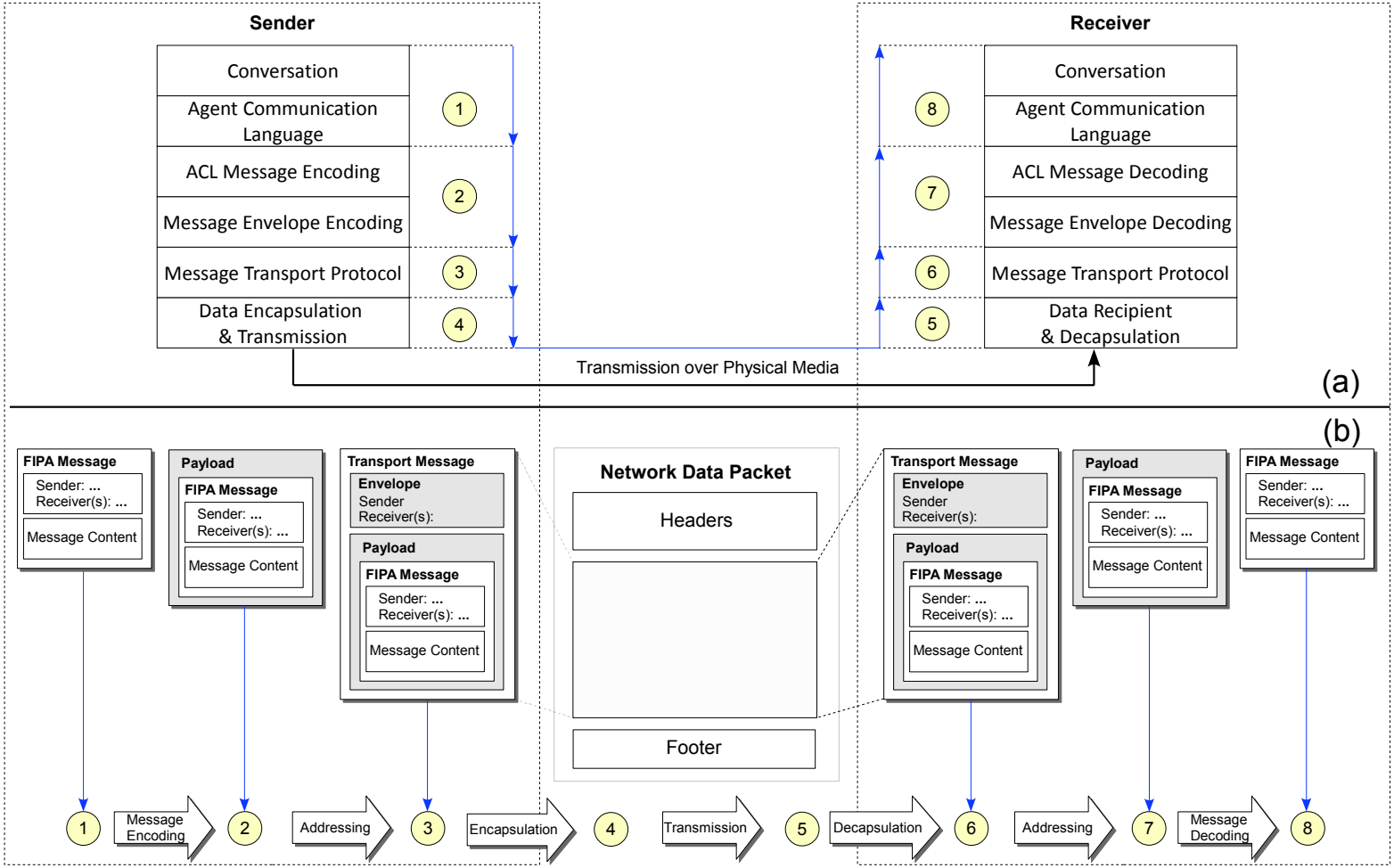

Figure 2.3: Layered Architecture of Agent Communication (a) (Hel03). Message Encapsulation and Decapsulation (b)

Figure 2.3 depicts the agent communication architecture. The upper part of the figure illustrates the layered architecture of agent communication (a). The lower part (b) of the figure presents the message encapsulation process.

The layered model resembles the OSI model (Sim07). The low-level data transport layers are common in conventional distributed systems. In this level, the two layers Transport Protocol and Data Transmission route and deliver data packets consisting of agent messages through networks. The high-level semantic layers are exclusive to systems with semantics communication. They are comprised of the four upper layers and parse, interpret semantics messages. The aims of the layered architecture are to categorize the processing into different levels, thereby facilitating communication optimization in different levels independently (Hel03).

The encapsulation process depicted on the lower part of the figure gives an overview of how an agent message is processed at different layers. At the sender side, down the communication stack each layer adds one ore more fields to the original message. And at the receiver side, the message is then parsed correspondingly to get to the original format. The circle numbers of the upper part are used to explain the message format at the corresponding stages. First, an ACL message consisting of the message content is created (1). To the next two layers, the message is encoded using ACL Message Encoding and Message Envelope Encoding (2). The data is then encapsulated using Message Transport Protocol (3). It is transferred over the network (4) and received at the destination (5). The message is then parsed to the transport format (6). The parsed data is converted to the form managed by Message Transport Protocol (7). Finally, the original ACL message is obtained (8).

2.5.3 Interaction Protocol

The process of exchanging messages between coordinated parties like agents typically falls into common patterns. It means, during the communication process, a sequence of messages is expected. At any point of conversation, a certain type of messages is awaited. The patterns of message exchange are called interaction protocol and they represent the highest abstraction component in the agent communication stack (Cal02),(SSB02).

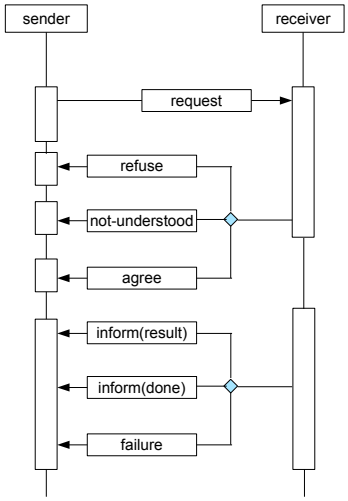

Figure 2.4: FIPA Request Interaction Protocol (FIP00d)

FIPA has defined a set of protocols which called Interaction Protocols. These specifications deal with pre-agreed message exchange protocols for ACL messages (FIP00c). Figure 2.4 gives an example for one of standards of FIPA Interaction Protocols, the FIPA Request Interaction Protocol. This protocol is invoked when an agent wants to request another to perform some actions.

2.5.4 Message Transport Protocol

Message transport protocol defines the structure of messages using a transport protocol. It delivers the messages to the destination agent in the correct order and informs the sender when communication error occurs.

2.5.5 ACL Message Encoding

The Agent Communication Language Layer in the layered model of agent community communication specifies the syntax and semantics of agent messages. For the purpose of optimization based on network environments, ACL messages can be represented in different formats. FIPA has defined three standard encoding schemes for ACL messages as listed as follows:

- Bit-efficient encoding technique encodes message by replacing fields of ACL messages with a combination of pre-defined hexadecimal byte. This technique can reduce the size of ACL messages considerably. Consequently, bit-efficient encoding scheme is suitable for transferring messages in wireless environment (FIP02a).

- String ACL encoding uses string to represent message fields. This encoding technique is not optimal as performing operations on strings consumes more time than other types of data, for example byte (FIP02b).

- XML ACL encoding is based on XML [1] which is used to represent and exchange data over the Internet. The ACL message representation in XML has been specified in (FIP02c).

XML ACL encoding is preferred because of the following reasons (CLC08), (GL99):

- XML is easy to manipulate; it enables users to format their own document structure.

- XML can be used to represent different types of information in human-readable format.

- XML is an open standard and is used by various software systems. Encoding ACL messages using XML allows MASs to be more open to other systems. As a result, the interoperability between heterogeneous systems can be improved.

- As XML has been used widely in web services, the deployment of XML in ACL messages facilitates collaboration between MASs and web services.

[1] eXtension Markup Language

2.6 Mobile Agent Communication

Mobile agents are software agents equipped with the ability to migrate. They can jump from a host to an other host in the network to fulfil tasks ordered by their owner. Mobile agents can temporarily save their work status before they migrate and restore tasks once they arrive at a new location. The basic principle of a mobile agent is to send the code to the data instead of doing inversely. The code performs processing on data and returns results. Consequently, the benefit of using mobile agents is to reduce the amount of data transfered through networks. The most important characteristics of mobile agent technology are to be autonomous, reactive, opportunistic, and goal-oriented. The mobile agent concept is highly beneficial to pervasive and ambient computing systems (Pau02).

A mobile agent inherits all characteristics of a conventional software agent. Communication among mobile agents encompasses all features of inter-agent communication. In addition to these characteristics, communication in mobile agents must deal with mobility. The Euler diagram (HBF+07) in Figure 2.5 depicts the relationships between communication in ordinary distributed systems, agent systems and mobile agent systems.

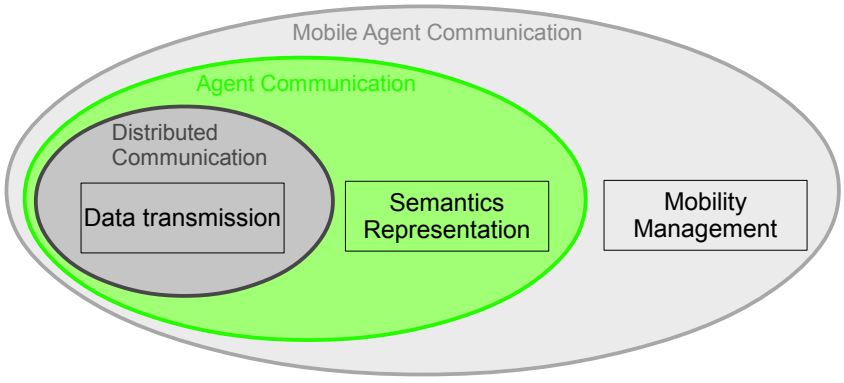

Figure 2.5: Relationship between mobile agent communication and other types of communication

On the application level, communication in conventional distributed systems relates to transferring bytes of data over networks. Agent communication consists of issues related to communication in distributed systems and issues regarding

semantics representation. Mobile agent communication, when being confronted with, means the combination of agent communication and mobility management.

Communication in mobile agent systems consists normally of two phases: agent tracking and message delivery. In the first phase an agent's whereabouts are tracked when it migrates over the network. The second phase, message delivery, takes place after the location of the targeted agent has been found. The two phases seem to be contradictory in performance, since the more effort we spend to monitor a mobile agent, the easier the message delivery phase is, and vice versa. A so-called location-independent communication protocol facilitates message deliverying between mobile agents regardless of their current residence.

A communication protocol for mobile agents is expected to satisfy the following requirements:

- Location independence: An agent can send messages to other mobile agents irrespective of their location.
- Reliability: Messages are expected to be delivered reliably to targeted agents.
- Efficiency: The protocol should produce low location updating and message delivery overheads.

So far, there have been many location tracking techniques for mobile agents proposed. In the succeeding sections we are going to give an overview of some popular location management schemes.

2.7 Location Management for Mobile Agents

2.7.1 Central Server

In the whole system, there is a central server that stores location information of all mobile agents. When an agent wants to communicate with an other agent, it sends message to the server, which in turn forwards to the destination. The server may also work as an address lookup, where agent asks server for the address of the receiver and then sends messages to the destination (Figure 2.6).

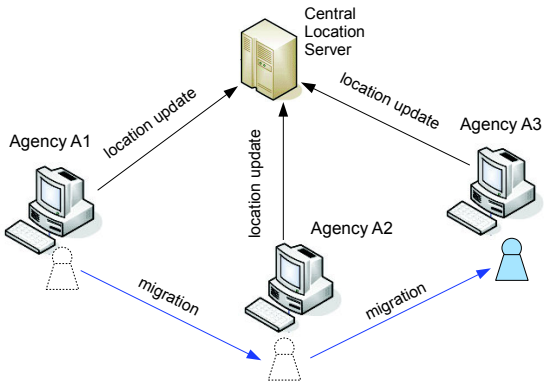

Figure 2.6: Central Server (BR04)

This approach is not suitable for systems with a huge number of mobile agents working in large scale network. A bottleneck may occur when many mobile agents send request to the central server at the same time. A single point of failure cannot be avoided if the server fails.

Bottleneck can be evaded with another approach called Home Server. Each home server manages location information for all mobile agents that have been created in it. When an agent migrates, it sends a location update to its Home Server. The name of an agent contains the Home Server address of the agent, so when an agent wants to communicate, it sends location lookup request to the receiver's home server to get the current location of the receiver, the message then will be sent directly to the destination address (JYBz07),(SWU10).

2.7.2 Forwarding Pointer

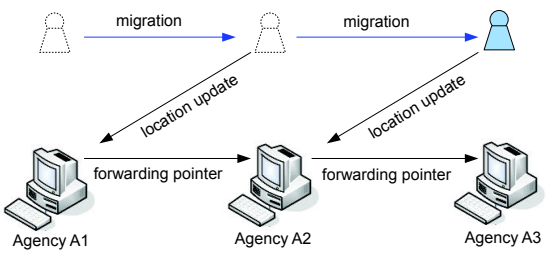

Figure 2.7: Forwarding Pointer (BR04)

Every agent knows the initial home site of all other agents. When an agent migrates, it leaves a pointer at the current agency points to the destination. When an agent wants to communicate with an other agent, it sends the message to the receiver's home agency. This agency forwards the message following the anchors that have been cached (Figure 2.7) (PS),(Woj01).

2.7.3 Broadcast

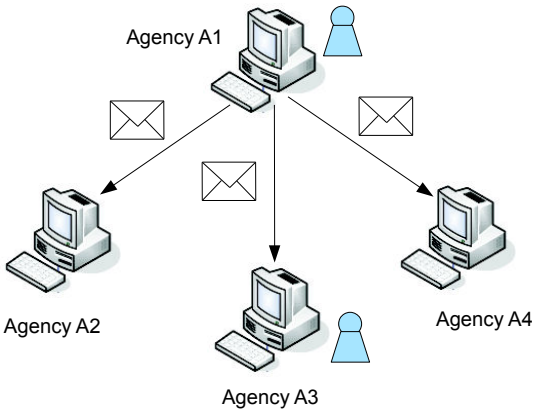

Figure 2.8: Broadcast

Figure 2.8 illustrates the Broadcast approach. There is no need to store an agent's location, thus the cost for agent tracking is null. However the cost for delivering messages or looking for addresses is considerably high, depending on the scale of network and the number of agent platforms. There are two types of broadcast: message broadcast and address lookup broadcast. When an agent wants to send message to an other agent, it may broadcast the message to all platforms or ask for the address of the destination by broadcasting a lookup query. Like the Central Server approach, the Broadcast approach is impractical in large scale networks (BR04).

2.7.4 Group Communication

A set of agents organizes as a group based on their common activities. The group maintains a record of location of every agent in the group. Before migrating, an

agent informs other agents of the group about its future location. It then waits for ACKs and all the messages due to arrive. The agent then migrates. Once it arrives at the destination, the agent informs other group members that it has finished migrating. Because an agent knows addresses of all other members, when it wants to send message to an other agent, it sends the message to the agency where the receiver is currently working (BR04),(SWU10),(Woj01).

2.7.5 Hierarchical Location Directory

The location directory of the servers in the system is structured as a tree. Each node corresponds to a leaf in the tree and each region is like a subtree. Each server keeps a record of location of some agents. For each agent, there is a unique path from the root to the actual server where the agent resides. Before an agent migrates, it informs its upper node. When the agent arrives, it registers itself with the new node, and informs the previous node and then waits for ACK. The algorithm is able to update the path when agent finishes migrating. Messages targeted to an agent are sent along the path from the root to the agency where the agent stays (Figure 2.9) (BR04),(SWU10).

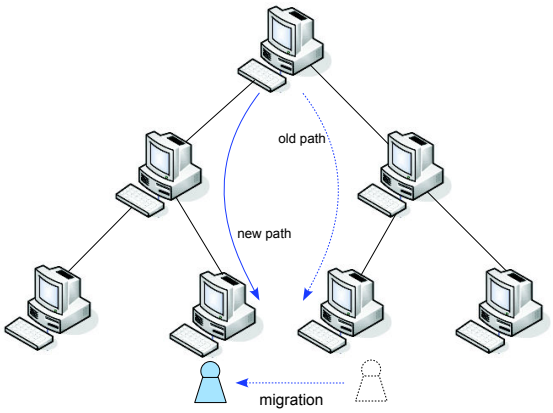

Figure 2.9: Hierarchical Directory

2.7.6 Arrow Directory

Each agent connects to a mailbox. If there is a message targeted to an agent, the message will be sent to the mailbox instead of to the agent itself. Agent checks the mailbox periodically for incoming messages (Woj01).

Agent platforms are organized as a connected graph, each vertex corresponds to a node and each edge models a reliable link. Nodes can send message to its neighbours along established paths. When an agent platform sends message to another platform, the message will follow the path until it reaches the destination.

Chapter 3

Some Existing Communication Models for Mobile Agents

So far quite a number of communication models for mobile agents have been proposed in many mobile agent systems (CFLD02),(BSP+05),(CKB+06). Each of these models has its own characteristics. In this chapter, we summarize some existing communication models for mobile agents. Based on the basic knowledge presented in Chapter 2, we are going to concentrate on analysing the main features regarding communication among mobile agents. For each system, the first section presents the system model. The succeeding section deals with the management of a mobile agent's location. Afterwards, the third section clarifies communication between agents. The last section will look at the pros and cons of each model.

3.1 An Improvement for the Forwarding Pointer Mechanism

Compared to the Central Server approach, Forwarding Pointer is considered to be able to reduce load on the server by distributing the load to all nodes along the route of an agent, and therefore evading bottleneck. However, this approach has its own drawback, if agent migrates with a high frequency, it might introduce a race between the sender and the receiver. When the chain of forwarding pointer is long, there will be unnecessary routes which need to be removed.

In (Ahn04),(Ahn10) the author presents a new approach to improve original forwarding pointer mechanism. Each mobile agent may leave trails of forwarding pointers only on some of the visiting nodes depending on some parameters such as location updating and message delivery costs, security, network latency and topology, communication patterns, etc. This aims to reduce the length of anchor for forwarding pointer. This approach is thought to be more efficient in terms of message forwarding and location management than the original Forwarding Pointer. In addition, it reduces considerably dependency on the home node in agent location updating and message delivery compared to the Home Server approach.

3.1.1 System Model

The model distinguishes between locator and forwarder. A forwarder of an agent is a node which has a forwarding pointer of the agent on its storage; the locator of an agent is the forwarder managing the identifier of the node where the agent currently resides. It is assumed that there is only one locator for an agent at a time, but there may be more than one forwarder for an agent (Ahn04). Each platform in the system holds the following data:

- A table that stores location of every agent currently running on the platform. Each record of the table consists of the identifier of the agent, the identifer of the agent's locator, and the timestamp when the agent located at the node.
- A table that has the location of every agent which has visited the node, but is not currently operating on it.

3.1.2 Location Management

After an agent has finished migrating, the current agency sends a location update message to the previous agency. All messages buffered for the agent will be sent directly to the current agency. In the case that the agent migrates to a new agency but this agency could not be its locator, no location update is sent to the home agency.

For message delivery, each node keeps a location cache, which stores all location information of agents which have communicated with agents on the node.

3.1.3 Communication

When an agent in a platform wants to communicate with an other agent, the platform will check its cache for the address of the receiver. If there is a record for the receiver, the message will be sent to the requesting platform regardless of the actual address. If there is no record found then the sender sends the message to the home agency of the receiving agent. Because the home agency is frequently updated about the current address of its agents, the length of forwarding pointer is partially cut.

3.1.4 Discussion

This technique has been designed to reduce the amount of agent location information maintained by each node. It also reduces the length of forwarding pointer chain. However, the cost for location update is considerably high. It introduces high location update per agent migration than that of the Forwarding Pointer approach.

3.2 IMAGO Prolog Agent Development Kit

IMAGO (Intelligent Mobile Agents Gliding On-line) is a mobile agent project developed by University of Guelph, Canada. It consists of two major parts: MLVMA (multithreading logic virtual machine) which presents a logic-based framework in the design of mobile agent server, and the IMAGO Prolog - an agent development kit based on Prolog which uses messengers as a means of agent communication facility (Li01),(LA05).

3.2.1 System Model

The model distinguishes between workers and messengers. Workers are normal mobile agents while messengers are a special type of agents which transfer messages between workers. When a worker is born either by creating or cloning, its identifier will be sent to and saved at the Home Server. It is called distributed registration. In each home server, there is a worker record for agents which have

been created on this server. Each worker is associated with a messenger queue which holds messages destined to it.

3.2.2 Location Management

For agent tracking, IMAGO uses a combination of forwarding pointer and home server. When a worker leaves agency A_1 for agency A_2, its worker record at A_1 will be updated with a pointer to A_2. If the worker continues to migrate and then arrives at A_3, A_3 will send update message to A_1, thus the path is compacted. Each remote server also stores worker records in its cache. Whenever an agent is created or cloned at the server, its information will be saved at the local server. In addition, the location of the sender is also cached.

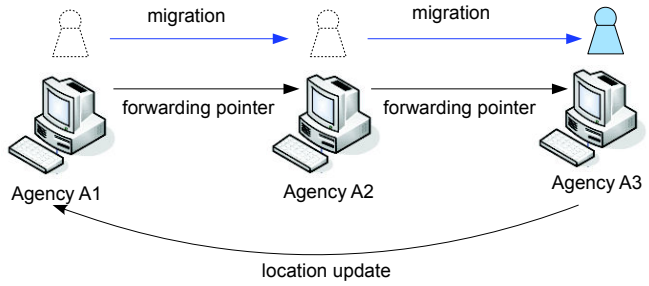

Figure 3.1: IMAGO location management

3.2.3 Communication

If an agent wants to send message to an other agent, it contacts the local agency to ask for the address of the destination. The local agency then looks up its cache to see whether the address is already cached or not. When the receiver has been created or already visited this agency then its address would be in the cache, in this case the local agency returns the address to the agent; otherwise it returns the address of the home server of the receiver. After the agent has the receiver's address, its message will be attached to a messenger. The messenger then tries to catch the receiver by calling move method until its message is accepted.

3.2.4 Discussion

With the caching strategy, this technique can help to reduce location management cost in the manner that one agent first contacts its local agency to ask if the agency has the desired address or not. When there is a cache hit, the sender has the address with a low cost, when there is a cache miss, the technique works exactly as the Forwarding Pointer approach does.

3.3 ARP: Mailbox Model for Agent Communication

A communication protocol called ARP (Adaptive and Reliable Protocol) is described in (CFLD02). This protocol associates each mobile agent with a mailbox. Mobile agent can migrate to a new agent platform but its mailbox may remain stationary. The decoupling allows to seperate the concerns of locating the mailbox and delivering a message to an agent. The mechanism is said to be able to guarantee reliable delivery of messages to mobile agents.

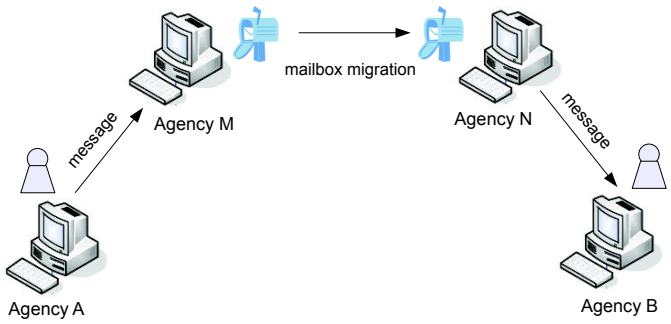

Figure 3.2: Adaptive and Reliable Protocol

3.3.1 System Model

Each agent has a mailbox associated with it; a mailbox is a message buffer that stores all messages destined for its owner agent. The mailbox and its owner are not necessary to be on the same agency. An agent can migrate while its mailbox remains stationary; this aims to reduce the cost of location tracking. For frequently migrating agent, mailbox can migrate but with a lower frequency.

3.3.2 Location Management

An agent decides itself whether it takes the mailbox with it or not. When an agent decides to take its mailbox with, it detaches the mailbox from the current agency. Once the agent and its mailbox have arrived at the new agency, the agent sends location update to all agencies in the migration path.

3.3.3 Communication

The communication process consists of two phases: the transmission of messages from the sender to the receiver's mailbox and the delivery of messages from the mailbox to its owner. For message-forwarding phase, the sender will forward the message along the addresses that have been cached. Messages can be fetched by the destined agents periodically or they can be pushed independently to agents.

3.3.4 Discussion

Although this technique may reduce the cost of agent tracking, it increases the cost of delivering message when the distance between mailbox and agent is far. When the chain of pointers grows, the cost of the search process is also increased.

3.4 ODDUGI: Ubiquitous Mobile Agent System

ODDUGI project was developed by a group of Korean universities. It was designated for multi-region mobile agent computing environment.

The ODDUGI architecture deals with multiple regions of agent platforms. In each region, there is a Mobile Agent Region Server (MARS) which provides the naming service for all agents of the region. The MARS maintains a region naming table for mobile agents which have been created or migrated in its region. A server called Multi-Region Mobile Agent Registration/Management Server (MRMARMS) provides the lookup service for mobile agents. The communication protocol in ODDUGI is RCP (Reliable Communication Protocol) (CBK+10),(CCB+09).

3.4.1 System Model

Figure 3.3 depicts an ODDUGI mobile agent system (CBK+10),(CKB+06):

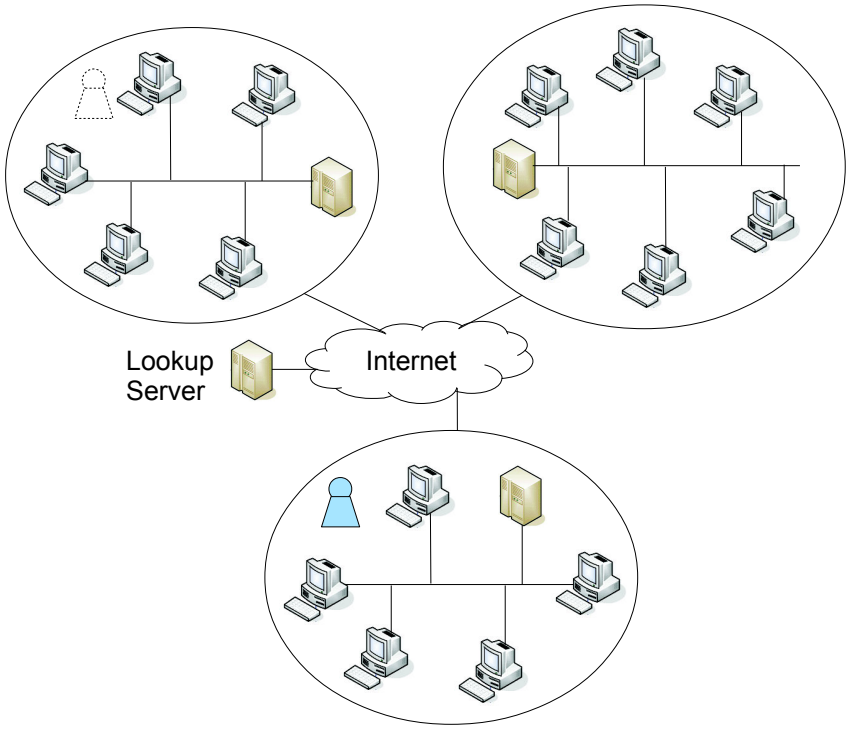

Figure 3.3: ODDUGI: Multi-region mobile agent system

- Mobile agent: A software entity that consists of code, data, state, and mobility metadata. A mobile agent can migrate when it is told to do.
- Node: A node provides execution environment for mobile agents. The Home Node of an agent is the node where the agent has been created.
- Region: A region consists of a set of nodes that have the same authority. Each region has one Region Server. The Region Server is responsible for naming mobile agents created within the region; it also manages location information and delivers message for all agents residing in the region.
- Lookup Server: The server that handles location information for every agent in the system. A node contacts the Lookup Server if it wants to send a message.

3.4.2 Location Management

After an agent has been created at a home node, it registers itself with the home Region Server. The Region Server sends a creation message to the Lookup Server. There are two types of migration: Intra-region and inter-region migration. With intra-region migration, the agent only informs its Region Server about its location. In inter-region migration, an agent sends location update information to its previous Region Server, the current Region Server, and the Home Node.

3.4.3 Communication

If an agent wants to send a message to an other agent, it contacts the Lookup Server to get the address of destination's Home Node and then the address of the current Region Server. The message will be sent to the Region Server. Each Region Server has a blackboard associated with it, whenever the Region Server receives a message from agent, it saves the message to the blackboard. The message will be delivered to the targeted agent when the agent updates its location by sending message to the Region Server. RCP also supports proactive message sending, in which message will be sent to the destination as soon as it arrives at the Region Server.

In ODDUGI some performance metrics have been introduced for evaluating system efficiency:

- Location management overhead: The overhead that incurred when tracking agents' location.
- Message delivery overhead: The overhead for sending message from sender to destination's Region Server and from Region Server to destination.
- Availability: Probability that a system will be correctly operational and able to deliver the requested services at any time.
- Storage Usage: This parameter indicates how much storage is needed to store the location information.

RPC has lower communication overhead than SPC (Search-by-Path-Chase) and ARP but higher than that of Home Server, Forwarding Pointer, and Broadcast approaches. RCP also has lower message delivery overhead than the Forwarding Pointer, Shadow, and ARP protocols. It has higher availability than Forwarding Pointer, Shadow, SPC, and ARP protocols.

3.4.4 Discussion

ODDUGI uses Central Server for location management, in which a server manages location information for all agents of the system. As shown in last sections, this approach may lead to bottleneck when there are too many connections to the Central Server. In addition, single point of failure cannot be evaded if the Lookup Server breaks down.

In inter-region migration at least three location updates must be sent, which is considered to be an expensive cost in communication. The technique introduces more overhead, and it is slower and less scalable than the other approach. With RPC, some metrics have been used to measure the efficiency of the system. Such metrics could also be useful for evaluating other communication models for mobile agent systems.

3.5 UbiMAS: Mobile Agent System for Wireless Sensor Networks

In (BSP+05),(Bag05), the authors at the University of Augsburg, Germany, introduce a mobile agent system called UbiMAS (later version called UbiMASS).

Thanks to its lightweight, UbiMAS can work on wireless sensor networks. UbiMAS has embedded in an IT support system for offices. In this system, so-called smart office building, walls, doors, and office equipments are integrated with microcontrollers and sensors, and users are also equipped with portable devices as well as microchips integrated in badges. Each node in UbiMAS has its own processor, memory and application specific sensors (BWSU10). Mobile agents in a node can receive information from the surrounding environment via appropriate

interfaces, e.g. RFID. The wireless sensor IT system has been used to assist users in finding ways within the building or to help them perform activities, such as email checking or file uploading/downloading (BSP+05),(BWUB09).

3.5.1 System Model

UbiMAS is a service that runs on top of a JxTA based peer-to-peer middleware. It provides basic functions for agent communication and migration. Each agent node may reside in one ore more peers. A peer can manage one or more agent nodes. Every agent and node has a unique ID. UbiMAS distinguishes between user-agents and service-agents. User-agents accompany users on their way. Service-agents offer services which user-agents can call on behalf of the user. Each user-agent has a home node, normally it is the office node of the user (BSP+05),(BPTU03).

Figure 3.4 illustrates UbiMAS's architecture. A UbiMAS node is comprised of five components: Agent, Message Delivery Engine, Migration Engine, PoBox, and PoBoxAdder.

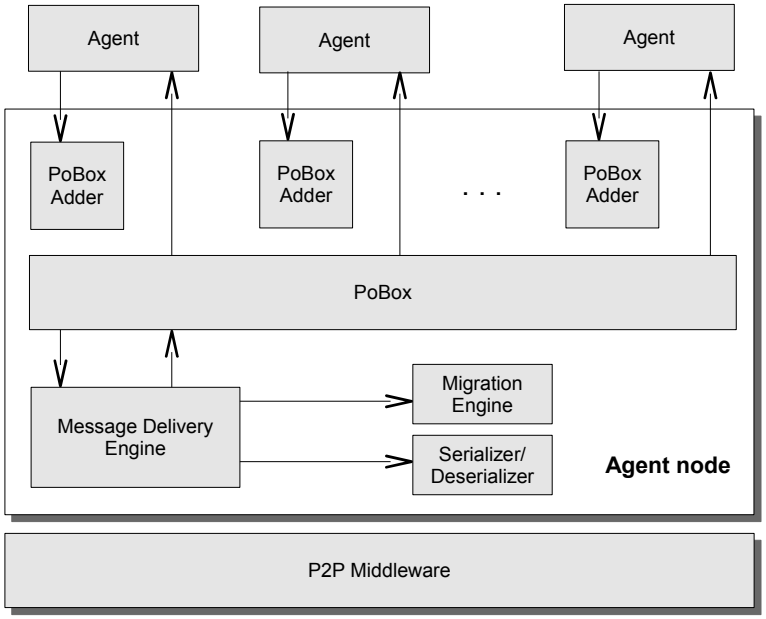

Figure 3.4: UbiMAS Architecture (BSP+05)

- Agent: A software unit with certain autonomy. It executes tasks that have been ordered by users or other agents. It can autonomously migrate, transfer program code, data and continuation pointer to a remote computer and resume with the program execution. Agent can also communicate with each other in order to exchange information.

- Message Delivery Engine (MDE): MDE sends request to the home node of an agent or to other peers to get location information of the receiving agent. It performs security functions on raw message data to secure messages, builds message header, encapsulates messages into right format, and sends messages to receivers. At receiver side, it performs reverse functions to get the orignial data.

- Migration Engine (ME): ME serializes agent code into byte stream and sends to the other nodes. It also receives byte stream and deserializes byte stream at receiver.

- PoBox: PoBox transfers messages between agents and nodes or between agents and agents. PoBox keeps a message queue which stores incoming messages for an agent.

- PoBoxAdder: PoBoxAdder is used to write messages into PoBox's queue.

3.5.2 Location Management

User-agents frequently inform home node about their current location. Other nodes can query home node for location information. If an agent wants to migrate to a new node, it sends a request to the node. MDE relays this request to the ME, which in turn serializes agent code into byte stream and sends to the destination node. The new node will store agent's ID and agent home node's ID.

3.5.3 Communication

Each agent is associated with a PoBox. The PoBox stores incoming messages for the agent. If the agent wants to send a mesage to another, it writes the message

into PoBox's queue using PoBoxAdder. The message is then processed by MDE. MDE sends a request to the home node of the agent or to other peers to get the address of the receiving agent. Once the MDE has the address, it secures message by performing security functions on raw message data, building message header. Afterwards, the message will be encapsulated into the right format. Eventually, the MDE sends the message to the receiver.

Once the receiver gets the message, it sends an acknowledgment to the sender. The receiver's MDE then checks to see whether the message is targeted at the node or at its agents. If the message is for an agent which still resides on the current node, the message will be fed to the PoBox. Otherwise, if the agent has migrated to another node, MDE will forward the message to this node.

Nodes can organize themselves as a group. Each group has a unique ID and is known by all members of the group. Inside a group, a node communicates with the others using a secure communication channel. Only members in the same group have the right to receive messages sent by other members.

3.5.4 Discussion

UbiMAS seems to be suitable for working in wireless sensor network environment. The size of a mobile agent is shrinked thus mobile agents can work on sensor nodes. UbiMAS provides software developers with a way to reprogram tiny devices with wireless connections. The communication model uses the Home Server technique for managing location information which is suitable for small and medium scale systems. It is unlikely to be an appropriate solution for large scale network.

3.6 JADE: Java Agent DEvelopment Framework

3.6.1 System Model

JADE (Java Agent Development Framework) is a framework for developing multi-agent systems (BCPR03). It has been designed in compliance with FIPA97 specification, which consists of three components: Agent Communication Channel

(ACC), Agent Management System (AMS), and Directory Facilitator (DF). Agent communication is performed through message passing, and messages are formatted following FIPA Agent Communication Language (BPR01).

A JADE system is made of *Agent Containers*, which reside in separate Java Virtual Machine and communicate using Java RMI. A special *Front-End* container is also an IIOP server, listening at the official platform ACC address for incoming messages from other platforms. JADE's architecture is shown in Figure 3.5.

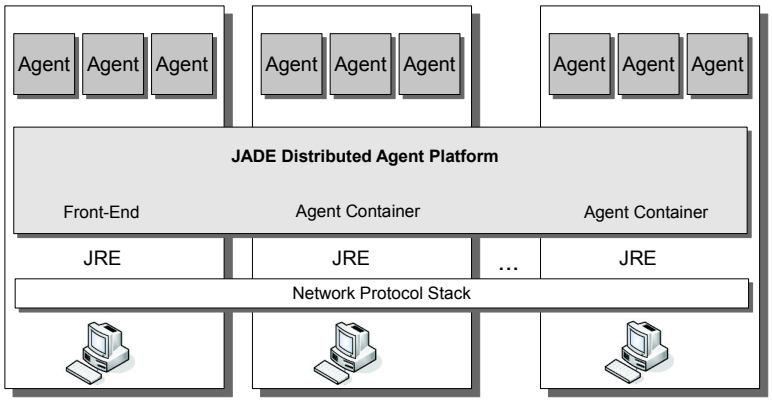

Figure 3.5: JADE Software Architecture (BPR01)

3.6.2 Location Management

Each agent has a globally unique identifier (GUID), this identifier works as the global address for the agent. Each container has a table called *Local Agent Descriptor Table (LADT)* for storing list of agents which currently reside on it. For the special container *Front-End*, besides its own LADT, it also maintains a *Global Agent Descriptor Table (GADT)* mapping every agent into the RMI object reference of its container. *GADT* works as the Global Location Database for the whole system. In JADE caching technique is used to avoid invoking *Front-End* frequently.

3.6.3 Communication

Each agent has a mailbox, a receiver agent will be notified whenever a message is posted in the message queue. However, software developers have to manually program if and when the agent fetches the message from its mailbox.

When a message needs to be sent, the GUIDs of the sender and the receiver will be inserted into the sender field and receiver field, respectively. JADE supports three types of communication (BPR01):

- Communication between agents in the same agent container: the ACL message is simply cloned using Java events.
- Communication between agents in different containers in the same platform: Java RMI is used, message is serialized at sender side, a remote method is called and the message in unserialized at the receiver side.
- Communication between agents in different platforms: IIOP is used, at the sender side, ACLMessage is converted into a string and marshalled. At the receiver, a remote CORBA call is executed, the message is unmarshalled and finally parsed into the original form.

3.6.4 Discussion

JADE complies with FIPA specification. In addition, JADE also supports large MAS. Both the number of threads per host and the network load among hosts can be considerably reduced by deploying clusters of related agents on separate agent containers. JADE uses *Front-End* for location management. It is vulnerable to a single point of failure since it relies totally in this component. Furthermore, the selection of transport protocols in JADE is not optimum as RMI is not suitable for transferring messages in wireless environments (Hel03). This framework is not able to deal with dynamic changes of environment and activities of mobile agents.

The authors in (BP01) propose a solution for a communication protocol for agents on handheld devices. A so-called JICP protocol has been developed to replace RMI in JADE and designed to support an efficient, peer-to-peer command/response based communication in both wired and wireless environment. In this approach, agent messages and other platform management communications between agent platforms are converted into proper commands and passed to the local JICP peer. Each JICP consists of both client and server parts, thus facilitates the open connection and accept connection processes.

Chapter 4

Communication in MCI Rescue Scenarios

4.1 Overview

Since April 2009 the Chair of Software Engineering of Friedrich Schiller University Jena has been taking part in a collaborative research project between academe and industry. The project, named SpeedUp, aims to provide support for rescue forces in mass casualty incidents. The project consists of two main areas: SpeedUp Practice and SpeedUp Technology. SpeedUp Technology is responsible for the IT support of rescue forces in mass casualty situations. The Chair of Software Engineering and its partners have been participating actively in designing and implementing a communication infrastructure for SpeedUp Technology. The IT solution supports the usual rescue measures by providing a framework for improved information handling, additional preprocessed sensor data, and a basis for a highly flexible communication infrastructure (FSU11). Essentially, the final aim of SpeedUp Technology is to convey necessary information to the right personnel so that in disaster events, rescue actions can be performed in a more effective way.

The communication network of the SpeedUp-Type is characterized by multiple working domains, distributed over a wide area. It exhibits the characteristic of dynamics. Because of its appropriateness, the mobile agent concept has been selected as one of the main technologies on the communication infrastructure

level for SpeedUp Technology. This fact is the inspiration for our work. We expect useful practical inputs from the project and in return, our work is expected to contribute to the implementation of the communication architecture in SpeedUp.

In Chapter 3, some existing communication models for mobile agents have been investigated. Although each model has some advantages in certain circumstances, in our view they are unlikely to be suitable to work in networks of the SpeedUp-Type. Most of them do not pay adequate attention to the dynamic properties of network environment. In this chapter, we will look into the context of rescue scenarios and identify challenges in rescue operations. We regard identifying the information flow among rescue teams as the sine-qua-non for pursuing the aim of the framework. Based on the behaviour of rescue workgroups, we figure out the information needs as well as information exchange flow among them.

4.2 Mass Casualty Incidents

On Wednesday at 16:15 in Weimar (Thuringia, Germany) an accident occurred when a school bus full of pupils after class lost its way and slipped on the highway. The bus then crashed into other cars and the central barrier, overturned, and finally stopped on the highway's right side. The cause of the accident remained unknown. It was a wonder that the highway's central barrier prevented the bus from hitting into the opposing traffic.

The succeeding vehicles tried to avoid the bus or to brake, but 11 of them did not manage it. More collisions occurred at all four traffic lanes. A VW bus with a family of seven people crashed into the school bus. Fortunately, the family only got lightly injured. The emergency center was alarmed via the number 112 about an accident with multiple cars and an overturned bus. Rescue forces estimated the accident with over 50 casualties; some of them were seriously injured. Because of the mass of casualties, the emergency rescue center in Weimar had to ask for reinforcements from other centers in neighbouring cities like Jena, Erfurt, and Apolda.

For such an incident scenario, forces like fire brigade, rescue service, and medical service are summoned and have to cooperate closely. The mission of the

teams, with the help of technical devices under instructions from rescue service, is to rescue casualties from collapsed cars and bring them to the nearest treatment station. Communication among involving parties takes place through different types of networks (SKEE10).

Figure 4.1 illustrates an example of a rescue scenario.

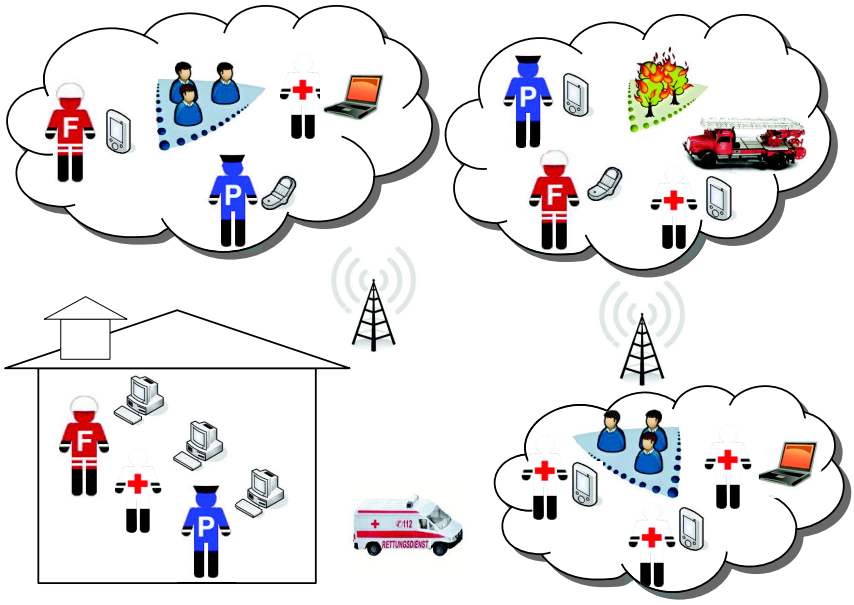

Figure 4.1: An abstract view of rescue scenarios

In typical casualty incident scenarios, rescue forces handle rescue missions based on exercise and experience. Police appeal for witnesses and evidence to the incident; fire brigades extinguish the fire; and medics give urgent medical treatment to injured civilians. Police and medical staff work together to evacuate casualties from the incident. Rescue forces decide quickly and correctly those casualties who are seriously injured will get medical treatment first. The quicker the seriously casualties get medical treatment, the bigger the probability that they survive.

In general, rescue tasks are (BJL06):

- Deployment of personnel and relief.
- Allocation of resources: ambulances, equipments, medical supplies.

- On-site classification and transferring of casualties to treatment stations.
- Medical treatment of casualties.
- Communication and coordination among rescue staff to improve rescue efficiency and to eliminate potential conflicts in goals.

In large scale casualty scenarios, a so-called mass casualty incident (MCI, hereafter), handling tasks based on experience without any additional technical supports seems to be an impractical mission. Due to the dynamic context and the growing chaos in the working environment, rescue staff might be overwhelmed by a torrent of events. As new tasks appear, different groups join and leave the area, work conditions change dramatically, rescue staff are unlikely to process incoming information in an effective way. In addition, as each team performs tasks with its own criteria and under control of its own authority, conflicts in their goals might arise at anytime.

Rescue tasks in MCI are problematic as many difficulties exist:

- Rescue scenarios spread over a wide area.
- Rescue tasks are executed by a large number of involved parties.
- Unknown number of casualties.
- Unknown location of casualties.
- Limited capacity of rescue forces.
- A huge number of critical requirements needs to be satisfied.

4.3 Information Needs in MCI Rescue Scenarios

As rescue tasks are challenging in an MCI, an inaccurate decision may pose many new risks. In constrast, with a reasonable strategy, supplies can be delivered to the right place at the right time, the number of casualties who have adequate medical treatment can be maximized, devastation is mitigated. From the existing

challenges, one question arises: Which technical infrastructure can be deployed to support rescue forces?

A technological system is expected to help rescue forces to cooperate, to share information, to identify the number and the severity of casualties, and to allocate resources efficiently. The system facilitates the process of collecting and processing information related to the incident. Information collected and processed is used to build a global view of the working scene (ea08). Rescue efficiency can be improved with cooperation of different rescue teams in nearby locations. As a result, emergency staff are able to handle casualty rescue scenarios in a more effective way by having questions answered, such as: *"What measures should be enforced where?" "Which kind of ambulances need to be deployed to zone Y?" "How big are the reinforcements needed for the site Z?"*.

In our type of MCI rescue scenarios, each rescue force has its own organizational structure and own way of processing information. Therefore, the information flow varies from team to team. In the command centre, desktop computers with stable network connection are used to collect rescue data from rescue teams and to distribute commands back to them. In the field, rescue personnel are equipped with portable devices such as laptops, PDAs to collect and exchange on-site data. Since tasks are partitioned between nodes, all nodes are equally privilledged, they form a peer-to-peer network. Communication takes place among members of the same rescue team or among multiple teams through heterogeneous networks like WLAN, UMTS/GPRS, SAT (SKEE10).

To face the difficulties in MCI rescue tasks, the key issue is to provide the right personnel with the right information. This means to identify information needs and flow of information exchange. In the first place, it is of crucial importance to know which information is transferred to whom (CLM$^+$08).

The crisis management group (CMG) works as the control unit for police units; it is responsible for getting information from on-site working police. Policemen collect information regarding the incident, like the cause of incident, people involved and send back to the CMG. The subordinate groups of the CMG receive and distribute information among them. The CMG has information from other sources, so it can provide information back to the police who are working on-site. The information flows in both directions (GKRM$^+$11).

For the fire brigades, the local operation manager plays the central role of the group. He has a tremendous amount of information regarding the incident to deal with. He receives, analyses information and demands to know the availability of the subgroups. The information exchanged within the fire brigades is considered to be heterogeneous, context specific, redundant or even contradictory (GKRM$^+$11).

The organizational structure of the medics has another form, with the presence of two leaders. The leading medical doctor is responsible for triaging - which means deciding who will get medical treament first based on the severity of the casulaties - and transferring casualties. The organizational medical leader is in charge of allocating resources. On-site doctors and other medical staff work as information sources for the two leaders. The leaders also ask other force leaders for additional information (GKRM$^+$11).

In the SpeedUp approach, a service oriented architecture (SOA) has been chosen. All available personnel, resources, functionalities are modelled as services and information needs, resource queries are defined as service requests (GKRM$^+$11). The aim of the MCI technological framework, which is to provide the right personnel with the right information, turns out to be matching requests with the available resources. Consequently, the communication infrastructure for MCI rescue scenarios is expected to fulfil the following requirements: Profile-based personalization, discovery of relevant resources, and automatic matching of individual profiles.

4.4 Network Dynamics

The network in SpeedUp's MCI rescue scenario, by its very nature, exhibits high dynamics. In order to find a suitable communication architeture for the MCI rescue scenario, it is necessary to identify environmental stimuli present in the context of the SpeedUp network. In our view, these factors play a dominant role in determining a feasible solution to deal with network dynamics.

In an MCI rescue scenario, different emergency staff performing rescue operations in a geographical area form a working region. There are multiple regions

scattered over a wide area and they are either isolated or connected to the global network. In a region, the movement and relocation of emergency staff make the availability of devices unstable. Network nodes join and quit rescue operations in an unforeseeable manner. Under particular circumstances, when communication of a region is obscured by the surrounding terrain, local rescue personnel can only communicate within the region. Upheaval in network topology is an imminent occurrence. The topology change is the most important stimulus that affects communication behaviour among nodes. It will be followed by other stimuli, such as changes in connection quality, in routing strategy, and in communication pattern.

In summary, environmental stimuli in the SpeedUp context are identified as follows:

- The change in network topology.
- The appearance and disappearance of network nodes.
- The connectivity between network nodes.

The characteristics will be the base for our approach in the next chapters.

4.5 A Coherent Application Architecture for MCI Communication Infrastructure

The network dynamics described in Section 4.4 have a significant impact on efficiency and reliablity of the communication system for MCI rescue scenario. The proposed system is able to adapt to network dynamics only, if there exists a decent application architecture. In this section, we are going to perform some analysis in order to find an application architecture that is suitable for MCI rescue scenario of the SpeedUp-Type.

We consider the client-server communication model as shown in Figure 4.2. In this type of computing architecture, a server possesses a powerful computing capability and it takes the central role of a system. Clients send requests to the server, the server receives requests as well as data from clients, processes and returns results to them. If the model could be applied in the SpeedUp context, then

all working network devices of the rescue staff would be clients, they send request to a remote central server. The central server will accept processing demands from on-site working forces and send the results back to them once it has finished.

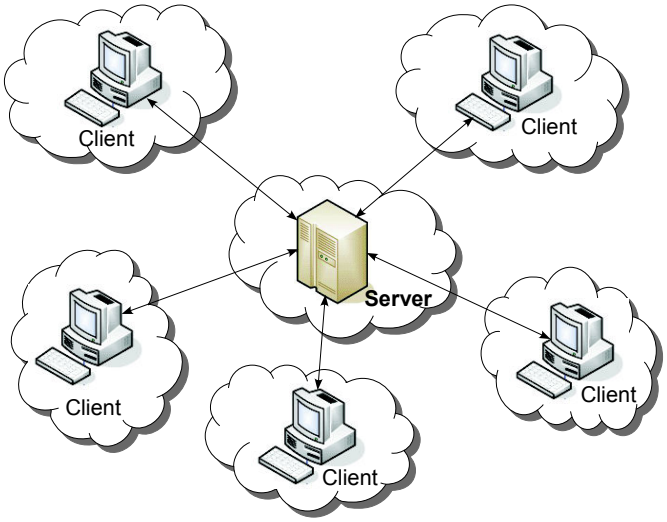

Figure 4.2: The client-server model

Despite some advantages, such that the model does not require maintenance at the client side and it is suitable for some specific applications, e.g., banking, e-mail services; the client-server architecture is not suitable to work in pervasive applications like the one in the SpeedUp context. The model does not bring benefit since the server and the nodes are widely distributed. A single point of failure occurs when the connection between the server and the remaining nodes is cut off, which is a regular occurrence in networks of the SpeedUp-Type. The communication cost would be expensive and it adds up to the overall complexity. In addition, it is the nature of the region architecture, network nodes do not always need to communicate to the central server; in some cases, it pays off if relating nodes group and perform tasks to achieve common objectives. Another disadvantage of the client-server model is that it produces growing network traffic when the number of clients increases (Hur95). A centralized IT structure like the client-server, therefore, is unlikely to be a good solution for the communication infrastructure in MCI support systems. As a consequence, a decentralized system is necessary.

In recent years, the peer-to-peer computing architecture, depicted in Figure 4.3, has appeared to be a prominent technology. The technology has been used widely in applications of sharing resources on a large scale. Resources in peer-to-peer (P2P) networks encompass a wide range of applications, such as: processing service, disk storage and file sharing. In this type of network, a network node plays the role of a client as well as of a server. Nodes are resource feeder and resource consumer at the same time. A pair of nodes can share and exchange information via a communication channel. The constituent nodes of a P2P network work without a central administration.

Compared to the client-server model, the P2P model has some more advantages; it guarantees fault tolerance and reduces cost of maintenance. A P2P network infrastructure provides scalablity, autonomy and flexibility (FSS05). As network tasks are distributed over all nodes, each node contributes an equal role to the processing of the whole system. No individual is to have superiority and, therefore, a single point of failure can be avoided (GWZ02). The features make P2P robust in distributed environment with highly dynamics.

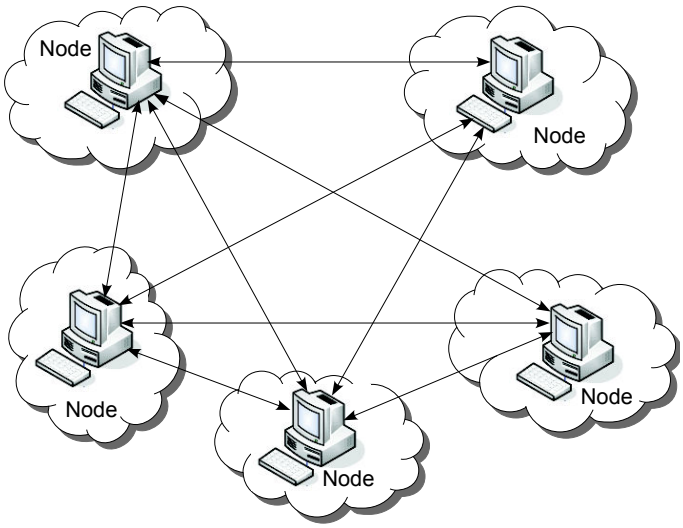

Figure 4.3: The peer-to-peer model

The P2P computing model meets the SpeedUp context's requirements, it appears to be a sensible solution to deal with the network dynamics and to provide a flexible communication infrastructure for the MCI network of the SpeedUp-

Type. With the employment of P2P architecture, each node works as information provider and information requester; the role of a node is dependent upon the type of tasks it is performing. The P2P architecture provides the system with a convenient way to manage information flow; it facilitates communication and coordination among rescue forces in nearby workplaces without needing a central admission.

4.6 SpeedUp Scenario Solution: Flexible Regions of Communication and Employment of Mobile Agent Technology

The SpeedUp solution is a communication and data platform for coordination and integration of all rescue teams in catastrophic situations. The aim is to provide rescue forces through the deployment of a technological framework. The entire solution is based on a hardware and software system which works as a self-organizing communication architecture. With self-organization, the system is able to operate autonomously in a changing environment, handling information and processing sensor data. It provides methods for describing incidents, suitable interaction models and supports information exchange between multiple rescue forces. SpeedUp solution is expected to help rescue forces to have a swift view of the incident and to localize and determine the severity of casualties.

With the adoption of P2P model, it is important to deploy an adequate software architecture for the network infrastructure. Because of its noteworthy characteristics, such as autonomous, reactive, opportunistic, and goal-oriented, the mobile agent technology is considered the right approach for the MCI communication infrastructure (BJL06). Mobile agents can percept their surrounding environment and find an appropriate route on the network (Erf04). They can migrate to different devices, collect data, access a database to keep them updated with their private store of the environment data. Mobile agents provide users necessary data and send results back in the form of reply messages. They can also undertake their tasks while being disconnected from the network, and exchange or submit

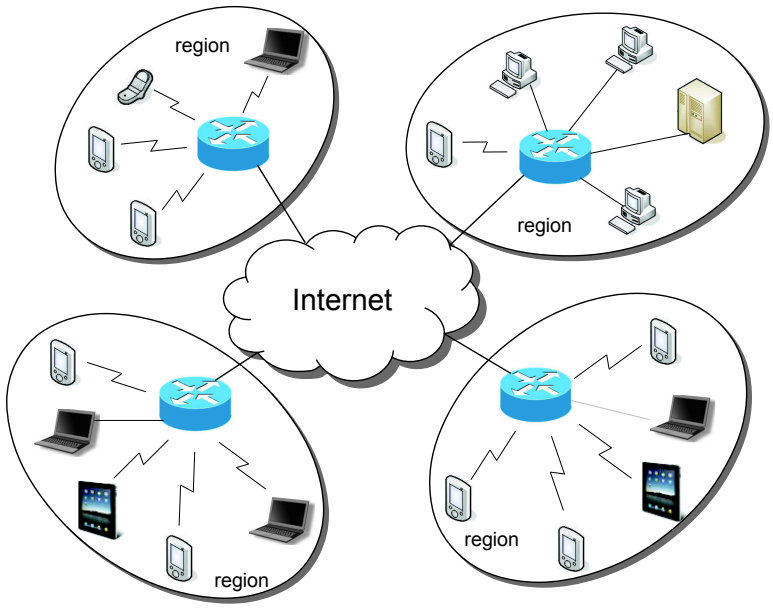

Figure 4.4: SpeedUp communication infrastructure

processed results to pre-defined destinations as soon as the user gets back to the coverage zone.

In most cases, mobile agents can be used to coordinate activities between multiple teams and to transfer required data from one system to others (BJL06). As migration is not always cheaper than communication, mobile agents can use communication technology to exchange or transfer data between involving parties. Data collected by mobile agents can be delivered to various parties with different roles based on their profiles (SKEE10).

Some applications of mobile agent communication in MCI rescue scenarios are illustrated in Figure 4.5 and explained as follows:

- Case 1: mobile agents work as a bridge between unconnected agent platforms.
- Case 2: mobile agents are used to coordinate rescue operations among personnel of the same force but located in different working domains.
- Case 3: mobile agents are used to impose collaboration on members of various teams working in the same domain: police, fire brigades, and medics.

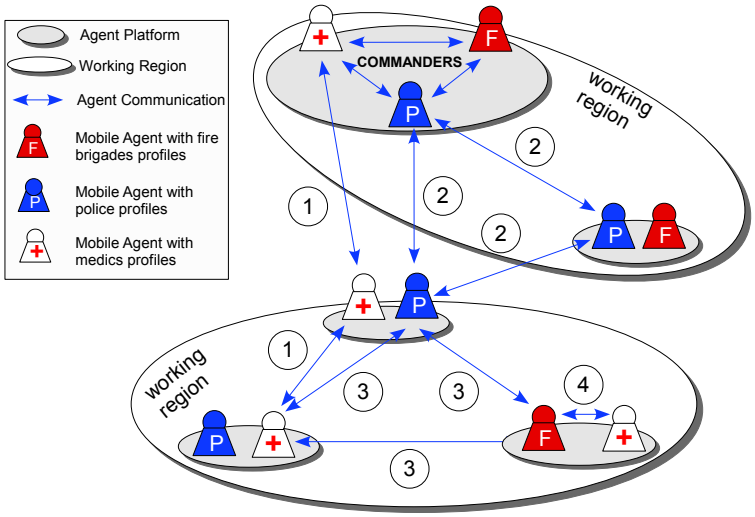

Figure 4.5: Applications of mobile agents in MCI rescue scenarios

- Case 4: mobile agents in a platform exchange data they have collected.

4.7 MCI-Scenario Summary

In this chapter, we have introduced the SpeedUp rescue scenario as well as the need for a technical support framework for rescue personnel. We address the issue first by describing information needs in SpeedUp rescue scenarios. In large scale rescue operations of the SpeedUp-Type, each rescue force has its own information needs and flow of information exchange. The identification of information needs is a contributing factor to find a suitable communication architecture. Environmental stimuli, e.g. the change in network topology and connection quality among network nodes, have been identified as the main factors that constitute network dynamics.

Network nodes in MCI rescue scenarios are scattered over a wide area, each of them plays the role of either information requester or information provider at the same time. The peer-to-peer application architecture fits well into such applications since each node of a P2P network has an equal role in processing information. The architecture provides applications with scalability, autonomy, and flexibility. Along with the adoption of the P2P architecture, the mobile agent technology has

been chosen as the middleware working on the surface of the architecture. Mobile agents are used as medium to convey information among rescue forces. By the SpeedUp context, we found out that mobile agent communication contributes to the performance of the whole system.

In networks with presence of dynamics, a change in network topology may adversely affect communication performance. There is a question left: How to organize the logical connection among agent platforms in a feasible way in order to deal with environmental stimuli? Given that communication between mobile agents is important, it is expected that agent platforms are able to self-organize in order to be adaptive to perturbations. As a consequence, fault tolerance and reliability in transferring of agent messages can be attained. The issues are considered as our research problems.

It is our firm belief that a communication framework that is autonomous, considerating the dynamics of working regions would be the most constructive approach. We are going to introduce our main thesis in the second part of the dissertation, a solution for the communication infrastructure in the SpeedUp's context.

Part II

Suggested Solution: Concept, Design and Implementation

Chapter 5

Thesis

Based on the preliminary introduction and the application scenario introduced in Part I, we are going to tackle challenges regarding the communication between mobile agents working in highly dynamic networks. Our suggested solution is going to bear on the following issues:

- The distributed nature of mobile agents.
- The parameters related to fault tolerance and reliability of communication between mobile agents.
- The scalability of the system with regards to the availability of heterogeneous end devices.
- The dynamics of the network.

In the SpeedUp architecture, regions of working devices are the prime elements which make up the global system. In our view, these facets should be handled in an effective way. Environmental stimuli affect the overall performance, therefore they need to be monitored so that the system can self-adjust accordingly. Since mobile agent communication is communication between individual agents, issues related to direct agent communication must be managed adequately from the start. Communication for mobile agents should reuse existing results from agent communication research. The main objective is, therefore, a model for agent communication that focuses on the cooperation aspect of agent interaction and supports fault tolerance and reliability as the key qualities.

5.1 Contributions

- Introduction of a self-organizing mechanism for the management of working regions in highly dynamic networks based on a honey bee algorithm.
- Development of a communication model that is able to adapt in a mostly autonomous fashion to topological changes of the underlying network.
- The proposed model is, in general, suitable for mobile agents working in highly dynamic networks of the SpeedUp-Type.

5.2 Outcomes

- Implementation of a software prototype for the proposed communication model that is adaptive to environmental stimuli.
- Improvement of overall qualities of mobile agent collaboration concerning fault tolerance and reliability.

5.3 Targeted Qualities

The aims and objectives of the dissertation are to gain overall better qualities regarding fault tolerance and reliability in mobile agent communication.

- Fault Tolerance: The system is able to operate even when some of its agent platforms disconnect. It avoids a single point of failure and prevents defects from spreading out. In the worst case, when the disconnection escalates, the system is able to degrade gracefully.
- Reliability: The proposed mechanisms assist system in delivery agent messages reliably in accordance with the changing environment.

The targeted qualities will be used as the evaluation criteria in Part III.

Chapter 6

An Adaptive Communication Model for Mobile Agents in Highly Dynamic Networks

In a highly dynamic network environment, communication between mobile agents is expected to adapt accordingly to environmental stimuli that happen at execution time. So far, several studies have been conducted to design an appropriate, highly flexible model for mobile agent communication. However, comparatively little of them has addressed the issue of the inherent dynamics of modern networks. To the best of our knowledge none of the suggested solutions has been able to achieve the necessary performance and quality attributes to count as a practical solution. Thus, in our view adaptive mobile agent communication remains a challenging issue.

In this key chapter, we present our approach for an adaptive communication model of mobile agents in highly dynamic networks. From our perspective, environmental stimuli and mobile agents' tasks shall have considerable influence on the performance of agent communication. As a result, they need to come under intense scrutiny in agent communication. To achieve industrial strength, we propose a model for agent communication that focuses on the cooperation aspect of agent interaction and supports reliability and fault tolerance as the key qualities, while keeping up an acceptable overall performance at the same time.

Based on our analyses, we concentrate on dealing with the following issues: The network dynamics and the requirements for a reliable communication between mobile agents. We attach significance to the self-organizing ability of a communication model for mobile agents. To manage the dynamics of the network, we employ self-organizing mechanisms, with which the appearance and disappearance of agent platforms as well as their connectivity will be monitored adequately. As being inspired by the exotic behaviour of honey bees, we propose an approach for the management of working regions based on honey bees activities. The inspired model is expected to be able to be aware of the changes in network topology as well as the connectivity between network nodes. For reliability, we propose a resilient mechanism for tracking of a mobile agent's location and deliverying messages to targeted agent platform (NSR11b).

6.1 Proposed Basic Architecture

The main components of the MAS, as we use in a SpeedUp-Type of scenario, are described as follows (see Chapter 4 and Figure 6.1):

- Agent: A software program that contains code, data, state; it acts on behalf of its owner to fulfil a designated task. An agent has a unique ID to distinguish it from all other mobile agents. An agent can migrate to another agent platform and communicate with other agents locally or remotely.

- Agent Platform: The software platform that provides the executing environment for agents. An agent platform has a unique ID.

- Lightweight Agent Platform: A special kind of agent platform which is designated to energy limited devices such as PDAs and smart phones. Only basic functionalities are available on this platform. A lightweight agent platform differentiates itself from other platforms also with a unique ID.

- Region: A region is made up of multiple agent platforms, which have the same authority. Agent platforms in a region are preferably in the same geographical location.

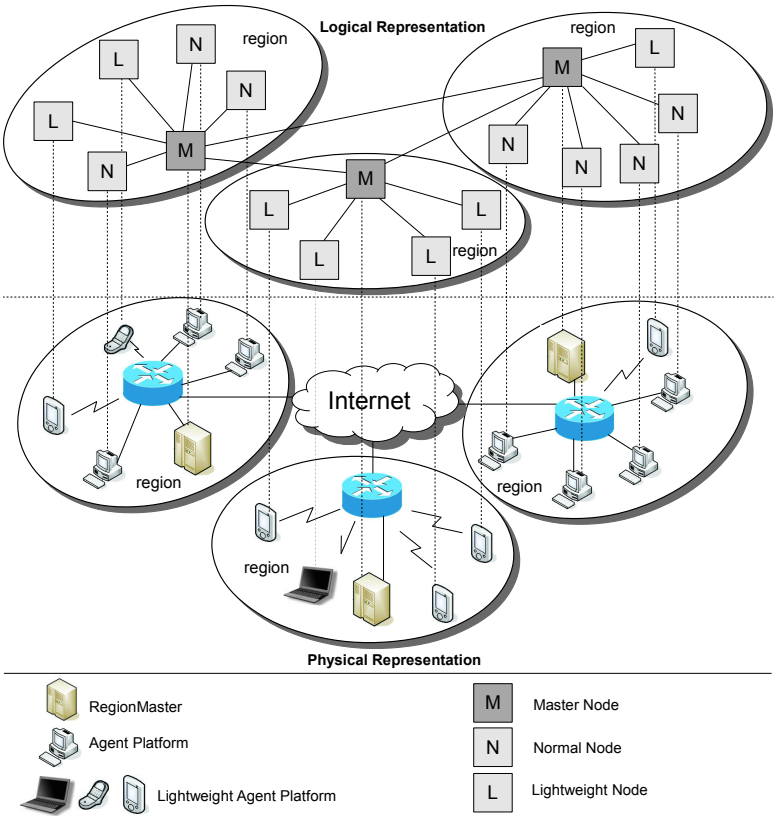

Figure 6.1: Network Representation

- RegionMaster: An agent platform in a region which has enough computing power to be the node that performs self-organizing tasks, manages location information, accepts and replies location inquiries.
- Global System: The highest logical view of the system; it consists of all regions of the system.

6.2 Self-Organization for Highly Dynamic Networks

6.2.1 Self-Organization

In MCI rescue scenarios, rescue forces may be distributed widely. Each geographical location forms most likely a technological region in which different forces work together to do rescue tasks. The key point is how to manage the dynamics

of mobile agent platforms in these regions efficiently, as they join and quit spontaneously, or they swarm onto a new location. In this section, we concentrate on analysing and finding an appropriate technique.

The technique is expected to help the system to fulfil the following requirements:

- The control of the whole system does not rely on any central component, it is distributed to all constituent members.
- The system is able to adapt automatically to the surrounding environment.
- The system is resilient to errors, it can get to a stable status after changes or damages through autonomous interaction of its members.

By bearing on these points, our analysis converges on the concept of self-organization (HG03). Self-organization is the way a system adjusts its behaviour to adapt to the surrounding environment. Through interaction among the constituent members on a local scale, the system reaches a globally sensible pattern. The control of a self-organizing system does not rely on any specific member, it is distributed to all of them (Hey99).

6.2.2 Swarm Intelligence

In recent years, the concept of swarm intelligence has become popular in computer science. Swarm intelligence can be considered as a branch of biomimicry which covers a wide range of applications, including architecture, mechanical engineering, nano technology, and computer science. The main idea of swarm intelligence in computer systems is to emulate activities in nature so that they can have alike features. As a consequence, they can react to environmental changes and therefore be resilient to perturbations.

Self-organization is a common phenomenon in nature. Throughout thousands of years of evolution, biological systems have adjusted themselves to adapt to a changing environment. Plants' leaves always lean to the direction where they can get sunlight, roots grow towards water sources and nutrients. In the animal world, flocks of bird and fireflies, schools of fishes, swarms of honey bees are

typical examples of self-organization (Hey99). The common property for these examples is that there is no central control or external intervention to the control of the community, the final decision is made based on the communication among individuals. The whole system fits itself ”bottom-up” to the environmental stimuli.

Robustness, flexibility and adaptability are the outstanding characteristics of natural systems (Yeo10). As a system can gain stability through interaction among its members, it is able to recover when error occurs. The system becomes more robust since the control of the whole system does not rely on any individual component. Self-organization has been proven to help a system to solve ”The complexity problem” (HG03).

A system that applies self-organizing mechanisms can exploit the following advantages:

- The system is able to adapt easily to environmental catalysts.
- New members can be integrated seamlessly into the system.
- The disappearance of members does not cause the whole system to stop working.
- The system can scale flexibly to the number of members.
- The system has the ability to recover itself from errors or perturbations.

In the following sections, we introduce the inspiration from the honey bee community to a self-organizing mechanism for the management of highly dynamic working regions of the SpeedUp-Type.

6.3 Self-Organization in the Honey Bee Colony

6.3.1 Overview

Honey bees can be considered as one of the most sophisticated colonies in the insect world. The bee colony is a typical example of self-organization in nature. A bee population consists of a queen bee who is reproductive, some male drones, and thousands of female workers. These are the worker bees who undertake the most

important tasks of the whole colony. They build the hive, keep it clean, regulate the temperature, search for food sources, and find a new resident location when the bee colony needs to split up (Kim10).

Teamwork is the distinctive character of the colony, honey bees are a well-organized team and they work in strict disciplines. Surprisingly, the queen bee has only an indirect role in organizing the colony, she gives birth to bees. When and how many bees the queen bee must create is actually controlled by the workers. The control of the whole colony is not dependent on any individual, it is done through the interaction among thousands of worker bees. The self-organizing model of honey bees has inspired much work in computer science and shown to improve considerably computational performance in particular circumstances (NT04),(KA09),(PAK07).

Regarding the network model in MCI rescue scenarios, we found that the honey bee community and the community of agent platforms have a substantial coincidence of behaviour. If we call honey bees and agent platforms simply *members*, then the following clauses hold for both communities:

- Members work in a distributed environment.
- Members cooperate to perform common tasks.
- Members work to maximize the global work performance.
- Congestion occurs when the population grows.

In addition, a honey bee colony has already mechanisms to deal efficiently with their daily routine:

- Control of the colony takes place without any external intervention.
- The colony can respond dynamically to the surrounding environment.
- The colony is still able to work, if some of the workers are missing.
- The colony can adjust its population to avoid conflict and maintain reproduction.

That the two phenomena have a lot in common and honey bees possess a good mechanism to deal with their daily routines encourages us to apply honey bee behaviour to manage the dynamics of a networked environment that reflects a SpeedUp-Type of system. We decided to employ swarm intelligence in the context of distributed systems. In the next sections, we are going to provide the reader with an overview of the honey bee colony, and then introduce our approach.

6.3.2 Bee Communication

Foraging is the most essential activity of the bee colony, without nectar and pollen bees cannot produce honey. Searching for food is a vital skill of honey bees since food sources are normally distributed far away from their hive. Foraging takes place in the summer when flora is blooming, worker bees collect and accumulate food for the whole year. Forager bees spread out to search for food sources. Each individual bee may find different food sources, but as time goes by, the colony tends to head for rich food source in which they can get more nectar and pollen. It is exciting to know that a honey bee has a mechanism to notify other bees. A worker bee can communicate with her colleagues to inform about the food sources she has found. A scout bee, after finding a new food source, returns to the hive and performs a *waggle dance* to notify other honey bees about the food source (CTE07). It has been shown that, with the waggle dance, a forager provides the following information to her colleagues (Kim10),(Hay07):

- Direction to the food source compared to the sun.
- Distance to the food source.
- The amount of food: the more exciting and the longer she dances, the richer the food source is (SB01),(SV04).

With this information, other honey bees are able to know where and how far they must fly in order to get to the food source. Rich food sources attract more honey bees to come and get pollen than other sources that contain lower amounts of food. Thanks to this information sharing, the honey bee colony can maximize the amount and quality of food. The discovery of bee communication brought

the biologist Karl von Frisch the Nobel Prize for Physiology or Medicine in 1973 (Gad96).

6.3.3 Swarming

Swarming in bee colony occurs when the bee population increases and causes congestion and difficulties in maintaining good hygiene in the beehive. Swarming is also one of the honey bees' instincts, they maintain reproduction by splitting.

Before swarming, worker bees build a cup in which the old queen lays eggs, in order to raise a new queen bee. When the new queen bee arrives, the old queen bee takes a number of workers with her and they fly to a new temporary location, e.g. a tree branch, that is near to their hive. The swarm remains on the temporary residence for a short time while waiting for a new residence. Scout bees, which are the most experienced foragers in the swarm, are then deployed to find new suitable locations. When the exploration is finished, every scout bee returns to the bee cluster to inform the whole colony about the place she has found. This is done by the same waggle dance the scout bee does when she informs the community about food sources.

The scouts fly back and forth until the number of honey bees that gather at a site constitutes a quorum, then the scout bees return to the swarm and the swarm decides to leave for the new location (SV03). The relocation may prolong, if the colony finds the new location does not satisfy expected requirements regarding available space, light intensity and hygiene conditions. The swarm will continue to depart for the next location. Eventually, it is summoned to the new residence.

6.3.4 Consensus

Although many sites may have been nominated during swarming, eventually all scouts make a unanimous decision, they reach a final agreement for a site, normally the best one (See03). The honey bee colony has a mechanism to build consensus among all scout bees. The exact mechanism of searching for consensus in honey bee colony remains a mystery. However, there have been many coherent explanations for the mechanism, one of them is the work from Thomas D. Seeley

and his colleagues (See03),(SKV04). This section provides a summary of these explanations.

Scout bees dance for good sites vigorously than for sites with inferior quality. In addition the dance for these sites also lasts longer (SV03). It has been shown that the strength of a dance is decreased linearly over the time, each time a scout flies to a site and then back to the swarm, it dances less stronger (SV04). When the dance expires, which means the strength is equal or lower than 0, the scout bee stops dancing for the site and chooses ramdomly a new site to follow and dance for. The rate of recruitment for a site is proportionate to the number of waggle dances by bees. Since dances for good sites prolong, a good site will attract more bees. As a result, a good site gains a quorum much easier than sites with inferior quality (SB01),(PSV07).

6.4 A Honey Bee inspired Algorithm for the Management of Highly Dynamic Working Regions

6.4.1 Preliminary Considerations

For small rescue scenarios with a handful of devices and consistent availability, network connection is stable. In this case, self-organization would not be necessary, nodes can communicate seamlessly, no collision or congestion occurs. However, as we have shown in the preceding sections, this is not the case in the SpeedUp scenario where environmental instability is a given.

For the SpeedUp-Type of system, in each region, the RegionMaster is responsible for handling location information of all mobile agents working in the region. When the number of nodes increases, a bottleneck may occur at the RegionMaster, if too many requests are sent simultaneously. The load of a RegionMaster changes dramatically. It becomes apparent that mechanisms which monitor the changes of a region and help the whole system to adapt to these changes are essential.

The phenomenon resembles the necessity for swarming in the honey bee community. The node population exceeds the region's capacity and needs to be adjusted, it requires some kind of ”swarming.” Like in the bee colony, when the

number of nodes in a region exceeds a predefined value, or when the platforms solving a common task move to a new location, the region will split into two regions. One of the nodes in the region will be promoted to be a new RegionMaster. The new RegionMaster manages to form a new region from the nodes it inherits. The two regions are then independent from each other, but logically connected.

In the first place, we need to take the following issues into account:

- When does the node colony need to split into two regions?
- Which node will be selected as the new RegionMaster?
- Which nodes belong to the new region, which nodes belong to the old region?

We are going to tackle the questions in the next sections. In the proposed algorithm, two mappings from the honey bee colony to the SpeedUp's network model will be employed. In the first mapping, the RegionMaster is considered as the beehive and mobile agents are scout bees. All other platforms of the region are viewed as potential beehives. In the second mapping, we consider all agent platforms as honey bees, and the RegionMaster as the queen bee. Each mapping has its own function. The first mapping is used to measure the connectivity between platforms; the second mapping is used to impose self-organization on the node community.

6.4.2 Mapping 1: Getting the Blueprint of a Region

The first mapping is used for measuring the connectivity between an agent platform and all the remaining platforms of a region. Once a calculating process has been performed for all nodes of a region, then the connectivity map of the region can be constructed.

In this view, the RegionMaster plays the role of the beehive. At regular intervals, the RegionMaster deploys scout agents to all platforms of the region and gets information about them (Figure 6.2). The information related to connectivity to platform's neighbours, and platform's performance will be collected by the scout agents.

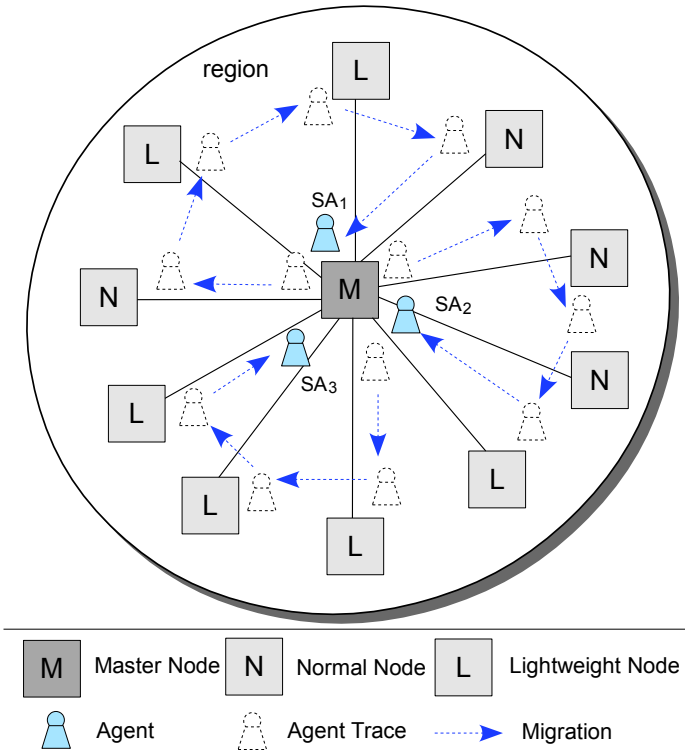

Figure 6.2: Honey Bee inspired algorithm; View of all scouts

When a scout agent arrives at a platform, it sends ping messages to all nodes of the region, including the RegionMaster to measure the transfer time between the current platform and every other platform (Figure 6.3). After collecting the node's information, the agent migrates to the next platform. At the new platform, the scout agent performs the same routine. The process repeats until all nodes of the itinerary have been visited.

A platform holds a boolean value $split$ to indicate whether it expects its region to split or not. At each platform once a scout agent has obtained transfer time to all neighbours by sending ping messages, it calculates the average transfer time τ_{avg}. Since every node in a region has the same number of neighbours, this value is the measure of the closeness between a node and other remaining nodes in the region, it will be specified later on in this chapter.

When the value is calculated, a platform compares the τ_{avg} with the transfer time to the RegionMaster τ_{RM}. If $\tau_{RM} >> \tau_{avg}$ then the node expects the region

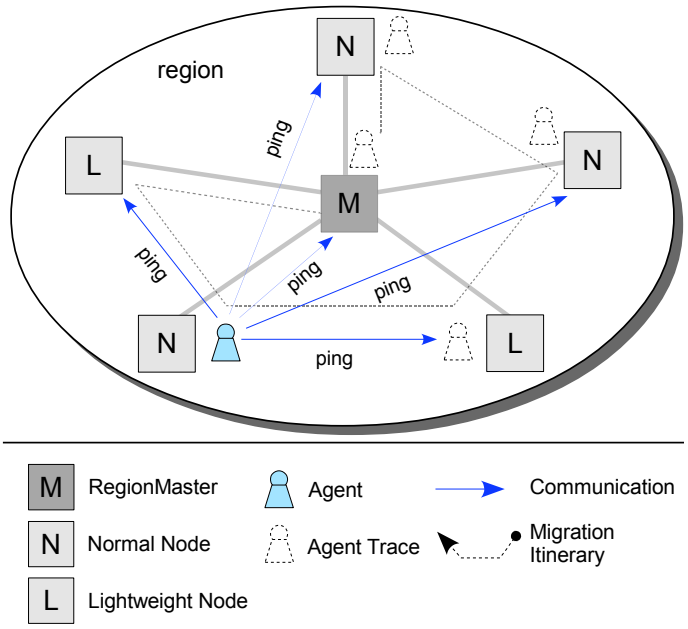

Figure 6.3: Scout agent measures connectivity; View of a single scout

will be splitted; it sets the value *split* to *true*. At a platform, a scout agent collects the two values τ_{avg} and *split*.

A platform has the potential to become a RegionMaster, if it satisfies at least two out of the following requirements:

- It has a good computing performance.
- It is a fixed network computer.
- It has a low average transfer time τ_{avg}.

On its itinerary a scout agent nominates the first node to be a RegionMaster. At the next node, the scout compares the information it has about the last node, if the node is more powerful than the last node, it replaces the current candidate to become a candidate for a RegionMaster. The process continues until all nodes have been visited and the scout returns to the RegionMaster. Finally, each scout agent can nominate one node as a candidate for a new RegionMaster, like a worker bee promotes a potential beehive.

When all scout agents finish their tasks and return to the RegionMaster, they unload information they have collected. The RegionMaster has node's information and a list of candidates for the new RegionMaster. From the list, it votes a reserve RegionMaster.

At this stage we should anticipate the effects of connection lost, that means while the process is still going on, one or more nodes lose their connection and as a consequence the process cannot execute smoothly as described above. In the worst case, if the RegionMaster fails during execution then a substitution for it needs to be made. Given the circumstances, fault tolerance mechanisms are about to be invoked. These mechanisms will be discussed in detail in the next sections of this chapter.

6.4.3 Mapping 2: Re-Organizing a Region

In Section 6.3.1, we already showed that there are considerable analogies between the honey bee colony and the agent platform colony. In the second mapping, the RegionMaster plays the role of the queen bee, and all other platforms of the region are worker bees. We use this mapping to impose self-orgarnizing mechanism on the node community.

The decision of splitting the region is made based on the thought of the majority of platforms in the region. In mapping 1, at every platform, the scout agents collect the boolean value *split*. The RegionMaster counts the number of platforms that want the region to be segregated. When the number constitutes a quorum, then the region is about to be splitted. The RegionMaster promotes a new RegionMaster from the two candidates it has already nominated, like the queen bee chooses her successor. Afterwards, the RegionMaster broadcasts the name of the new RegionMaster to all platforms. Each platform decides itself, which region it belongs to by sending a joining message to the new RegionMaster. The two RegionMasters organize their own region (Figure 6.4).

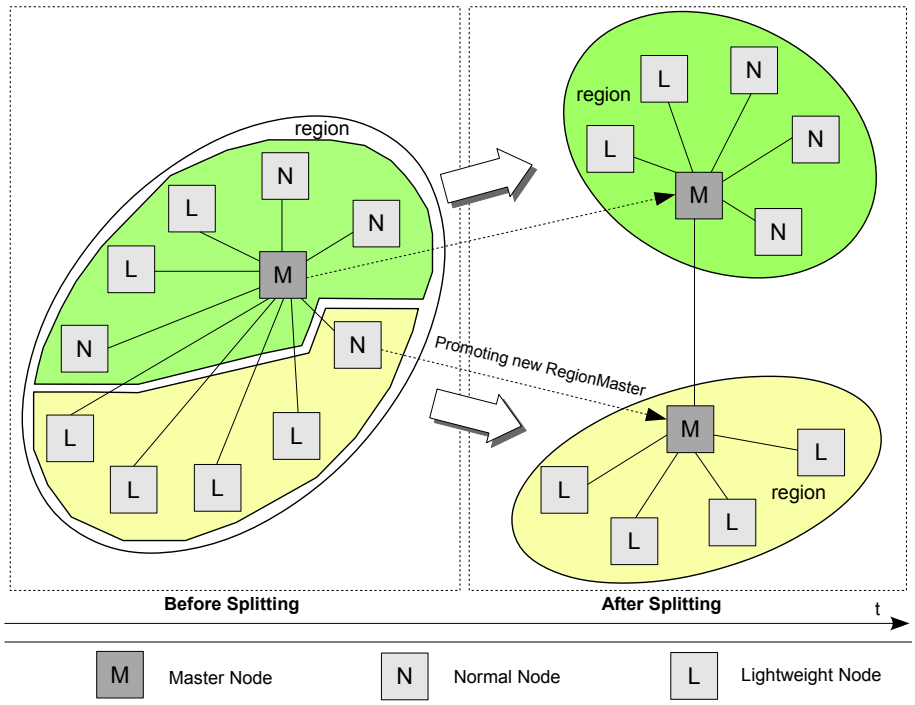

Figure 6.4: Splitting of a Region

6.4.4 The Honey Bee inspired Algorithm

The algorithm is summarized in four steps as follows:

Step 1: In the beginning, we assume that there is already a region of the SpeedUp-Type, in which devices belonging to different forces work in an area to perform rescue tasks. An agent platform is already specified as the RegionMaster of the region.

Step 2: At regular intervals, scout agents are deployed to visit all nodes of the region. The number of scout agents is dependent on the scale of the local region. At a platform, the values τ_{avg} and *split* are calculated as follows:

$$\tau_{avg} = \frac{1}{n} \sum_{i=1}^{n} t(i) \tag{6.1}$$

and

$$split = \begin{cases} 0 \text{ if } \tau_{RM} \leq \tau_{avg} \\ 1, \text{ otherwise} \end{cases} \tag{6.2}$$

In which n is the number of nodes in the region, $t(i)$ is the transfer time between the current platform and the i^{th} neighbour platform.

The values τ_{avg} and *split* will be collected by the scout agent.

Step 3: En route each agent nominates two candidate nodes as possible new RegionMaster, based on their performance and their connection quality. After visiting all dedicated nodes, the scout agents go back to the RegionMaster and submit the information they have collected.

Step 4: Based on the boolean values *split* of all nodes, if a quorum is reached, the RegionMaster splits its population into two separate colonies. The RegionMaster promotes a new RegionMaster from the candidates. The new RegionMaster manages to form a new region from the nodes it inherits. The two regions are then independent from each other but logically connected via the two RegionMasters. Once the logical network has been re-organized, the RegionMaster notifies other RegionMasters in the global network about the restructuring.

If failure occurs while any of the steps is executing then fault tolerance mechanisms must be activated. The mechanisms will be described later in the next sections.

The honey bee inspired algorithm is used to employ self-organization to the community of agent platforms. As mentioned in Chapter 1, we have been aiming for fault tolerance and reliability in mobile agent communication. However, these issues have not been yet addressed. With respect to the honey bee inspired algorithm, in the next sections, we present our approach for these targeted qualities. For dealing with fault tolerance, we introduce redundancy and self-recovery. In order to guarantee reliability, we propose location management mechanism and message exchanging technique between mobile agents.

6.5 Fault Tolerance: Self-Recovery by Employing Redundancy

The honey bee algorithm described in Section 6.4 seems to rely much on the RegionMaster, which may generate a single point of failure. According to the first mapping, in a region, the RegionMaster acts as the beehive, where the scout agents unload information they have collected. By the second mapping, the RegionMaster plays the role of the queen bee, it nominates a new RegionMaster and informs other platforms in the region about the change. At this point an issue arises: What would happen if the RegionMaster breaks down or disconnects suddenly? The node community has no information about the network and, therefore, cannot re-organize wherever necessary. The occurrence is obviously to be in conflict with one of the goals we have been pursuing, fault tolerance. In addition, in the given case the proposed self-organizing mechanism does not work.

In both views, the RegionMaster plays a decisive role in the whole region, the issue that we have been attempting to evade. As a result, we need a constructive approach to fulfil the fault tolerance requirement and to assure self-organization.

We propose the solution of using redundancy. A reserve for the RegionMaster is voted based on information fetched by the scout agents. On the way of the exploration, each scout agent compares information it has collected from the visited platforms, and the connection quality as well as the density of each platform, it nominates two substitutions. When arriving at the RegionMaster, scout agents submit the list of candidates. Apart from the RegionMaster, every region has a reserve RegionMaster. The reserve RegionMaster maintains a close contact with the RegionMaster, it keeps a list of all nodes that are present in the region. The reserve RegionMaster communicates periodically with the acting RegionMaster and updates the status of this node.

We examine two scenarios, on which fault tolerance will be imposed, as follows:

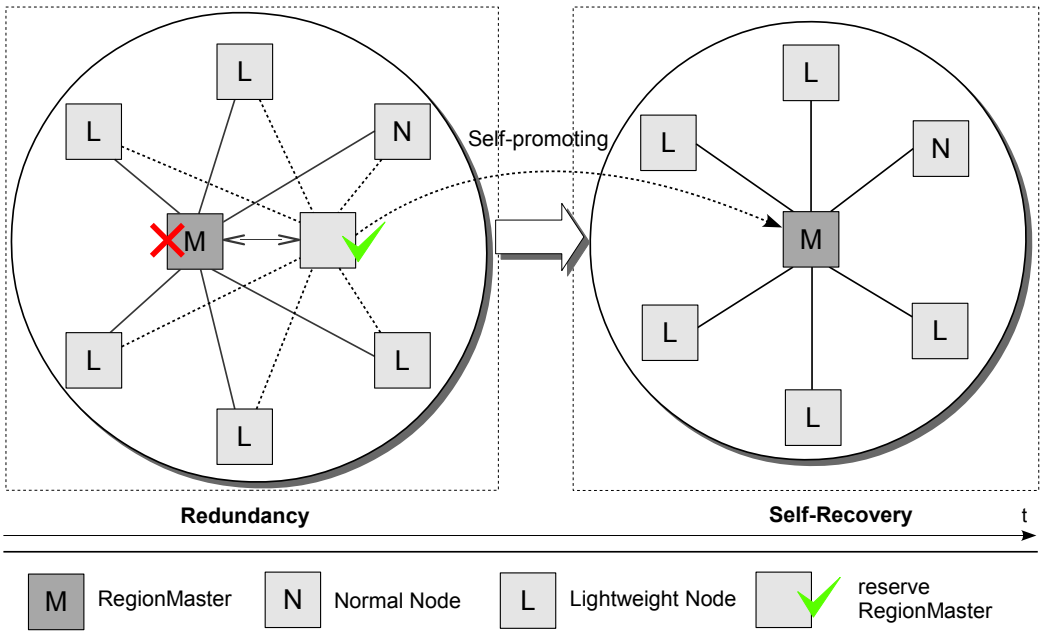

Figure 6.5: Self-Recovery from Redundancy: The acting RegionMaster goes haywire (left). The reserve RegionMaster takes over as the acting RegionMaster (right).

6.5.1 Scenario 1: The acting RegionMaster breaks down

The acting RegionMaster may go haywire during execution. We utilize the heartbeat failure detection strategy (ACK$^+$97) to detect crashes when they occur. The incumbent RegionMaster sends a heartbeat message at every $\tau_{heartbeat}$ to the reserve RegionMaster. If after $\tau_{heartbeat}$ the reserve RegionMaster gets no message then it assumes that the RegionMaster has crashed, it promotes itself to be the new RegionMaster. The new RegionMaster builds its list of mobile agents from the beginning by broadcasting a message to all nodes it inherits. If the nodes receive the message, they change the information about the RegionMaster and send a list of agents that are being active. Once the new region has been established, the new RegionMaster appoints a new reserve RegionMaster.

6.5.2 Scenario 2: The acting RegionMaster and the reserve RegionMaster break down concurrently

In scenario 1, the system can recover if the RegionMaster is out of service because a reserve RegionMaster already exists. But we may wonder what would occur if the substitution also does not work?

The first platform is aware of the disappearance of the RegionMaster once it sends location update or location query to the RegionMaster, but receives no answer. This platform must activate the scouting procedure. That means, it temporarily takes the role of the RegionMaster. Since every node has a list of platforms in the region, the node can create and deploy scout agents to other nodes. The scout agent performs its surveillance and comes back to the node as described in Section 6.3.4. Eventually, a new region is established under the chairmanship of the new RegionMaster.

6.6 Reliability: Management of Mobile Agent's Location

The mobility of agents may cause severe message losses when communication executes, especially in large scale and dynamic networks. Such, these losses must be minimized in order to achieve systems reliability. The management of mobile agents location should also be adaptive to network changes. With respect to the dynamic region architecture established by the self-organizing algorithm in Sections 6.4 and 6.5, we introduce a technique for the management of mobile agent's location.

6.6.1 Location Update

Each agent platform maintains a cache for storing a list of IDs of mobile agents that are currently working on the platform. To avoid flooding, the cache works as a FIFO list with limited size. In the RegionMaster of a region, there is also a cache that holds location information for all mobile agents which currently reside in the region. RegionMasters are organized as a peer-to-peer network, each of them

has pointers to all other RegionMasters of the global system. A RegionMaster can query its counterparts for information of mobile agents' IDs.

If an agent migrates to another platform, the current platform deletes the corresponding ID in its cache. The platform then tells the RegionMaster to remove the ID from RegionMaster's cache. When the agent arrives at the new agent platform, its ID will be added to this platform's cache. The platform notifies the RegionMaster to insert a new entry for the mobile agent's ID. Eventually, the address of the mobile agent becomes known with the new location.

6.6.2 Location Query

The process of searching for the location of a mobile agent is illustrated in Figure 6.6. The circled numbers depict the steps that are to be executed. These numbers will be used to explain the diagram.

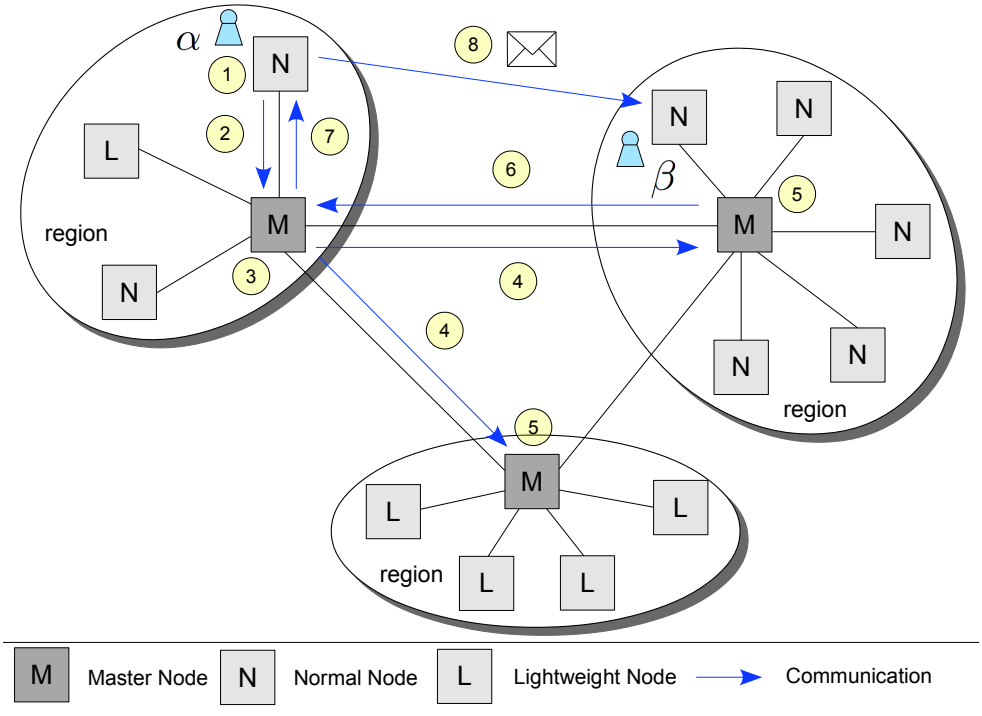

Figure 6.6: Location Query

In the example, we assume that agent α wants to communicate with agent β, whose location is unknown. Agent α asks the current agent platform for the ID of

agent β. This platform checks its cache to see if there is any information regarding the location of the requested agent (1). When the address is already in the cache, the platform returns the address to the agent. Otherwise, the platform queries its RegionMaster (2). The RegionMaster then looks into its cache for the address (3). When the address is found, the RegionMaster returns it to the requesting party (7); if not, the RegionMaster broadcasts a query to all RegionMasters of the global system (4). Each RegionMaster of the global network checks its cache for the address (5). Whenever the address is found by one of the RegionMasters, it will be sent back to the requesting RegionMaster (6). This RegionMaster in turn forwards the address to the requesting agent platform (7).

After the location of the agent β has been determined, the agent sends message to the targeted platform (8).

In case the destination agent resides in a lightweight platform that has disconnected from the network, its address is still cached in the last RegionMaster. Because the agent cannot be reached at that moment, the RegionMaster sends the address back but with the status *offline* so that the requesting party knows.

If a region is splitted the RegionMaster advertises the change to all other RegionMasters to make other RegionMasters aware of the new region's appearance.

6.7 Reliability: Message Exchange among Mobile Agents

To ensure reliability, along with tracking of mobile agent's location, it is necessary to guarantee message storing and delivery. Messages can be delivered to destination reliably only when they are stored and delivered credibly.

6.7.1 Message Format

We choose the existing FIPA Agent Communication Language Specification as the messages' semantics representation. Messages for the communication between mobile agents are formatted according to this FIPA Standard.

6.7.2 Message Cache

Each node has a message queue for agents which are currently working on it. Each agent is associated with a message cache, all incoming messages for an agent will be saved to its cache. When an agent leaves a platform, its cache will be removed. The message cache for an agent stores every message targeted to the appropriate agent. A cache for an agent is identified by an ID associated with the agent's ID.

6.7.3 Message Delivery

After the location of the receiver agent has been determined, the sender agent sends messages directly to the platform where the receiver resides. We adopt the layered architecture for agent communication introduced in Section 2.5 and in Figure 2.3.

6.8 Summary

In this chapter, we have presented our approach for an adaptive communication model for mobile agents in highly dynamic networks of the SpeedUp-Type. By identifying the need for a self-organizing mechanism, we suggested applying a honey bee inpsired algorithm for the management of dynamic working domains. Along with the algorithm, we also proposed supplementary mechanisms to pursue our goals, that means achieving fault tolerance and reliability in message transferring. It is expected that the proposed mechanisms will bring benefit in the SpeedUp context.

In a honey bee community, worker bees search for food sources and promote new beehives. Likewise mobile agents survey network nodes and collect information, as well as evaluate and nominate candidates, whom they think can be potential RegionMaster. A logical network is organized by consensus based on interaction of network nodes. Although the RegionMaster seems to play a crucial role in a region, it is replaceable. As a result, no individual of the colony is allowed to have superiority. With the introduction of self-recovery from redundancy, we aim to provide fault tolerance and preserve self-organization.

An interesting aspect of the approach is that mobile agents are used as tools for monitoring network dynamics and, at the same time, they are also a beneficiary of the algorithm. A reasonable restructuring of logical regions is of benefit to mobile agent communication. Mobile agents can cooperate and share information flawlessly thanks to a flexible communication infrastructure.

We believe that the honey bee inspired algorithm can be applied not only for managing regions of agent platforms, but also for imposing self-organizing mechanism on other types of systems with multiple network nodes working in a dynamic environment. The mechanism is beneficial to those systems, in which there is a need to segregate nodes according to their relations and to organize them in a flexible manner. Under the circumstances, the mobile agent concept is considered to be highly suitable for this purpose. The characteristics, e.g. reactive, opportunistic, and goal-oriented are intrinsic to successful management of a distributed region architecture of the SpeedUp-Type. One of mobile agents' advantages has been shown off.

In the proposed algorithm, the scout agents are sent to all nodes of a region to get information. What not yet specified there is how often they need to be deployed as well as how many of them need to be created. In our view, these issues can only be tackled in practical experiments. In the succeeding chapters of the dissertation, we are going to present our design and implementation for the software prototype based on the proposed algorithm.

Chapter 7

Design

This chapter deals with the design of the software prototype for an adaptive communication framework for mobile agents in highly dynamic networks of the SpeedUp-Type. The design aims to develop a plan for the implementation of the software prototype. It will bear on the honey bee inspired alogrithm with the aims and objectives of the targeted qualites: fault tolerance and reliability in transferring agent messages.

The chapter starts with some consideration. Afterwards, it gives an introduction to the mobile agent system Ellipsis. The next part of the chapter clarifies the functionalities of the software prototype. This part serves as a base for the succeeding part, which describes in detail the constituent software components. We address each software component by specifying their functions, sub-components and the relationships among the sub-components; afterwards, the format for the data exchanged, stored by each component will also be characterized. The final section summarizes the main points presented in the chapter.

7.1 Preliminary Considerations

- We base our implementation on the mobile agent system Ellipsis (Sch12) as the basic agent instantiation and execution environment. All solutions will be implemented as an extension of the system.

- To ensure maximum interoperability, we utilize the FIPA Agent Communication Language Specification as the message's semantics representation. Messages exchanged among mobile agents will be formatted following this FIPA Standard.

7.2 Introduction to Ellipsis

Ellipsis is an open source mobile agent system developed by the Chair of Software Engineering of Friedrich Schiller University Jena (Jen11). Ellipsis system provides a framework for hosting and executing mobile agents. Ellipsis has been written in the programming language Java, it runs on any operating system that supports Java, thereby adhering to the slogan "write once, run anywhere." As a consequence, it can be deployed on any device as long as the device is equipped with Java Virtual Machine (JVM).

Since we are going to base our design and implementation on the mobile agent system Ellipsis, it is necessary to have a look at its features as well as its architecture.

7.2.1 Architecture

Figure 7.1 illustrates Ellipsis and its relationship with other external components.

- Java Virtual Machine (JVM) provides an executing environment for Java bytecode. It can be combined with different application programming interfaces to form Java Runtime Environment (JRE). Every operating system that supports JVM can be a platform for Ellipsis.

- JBoss Application Server works on JRE and provides a server framework in which applications can execute.

- Java Management Extension (JMX) is a tool for managing server components and applications written in Java. It works as a component bus, an interface to JBoss so that programmers can seamlessly integrate their software components into JBoss as well as manage both JBoss server components

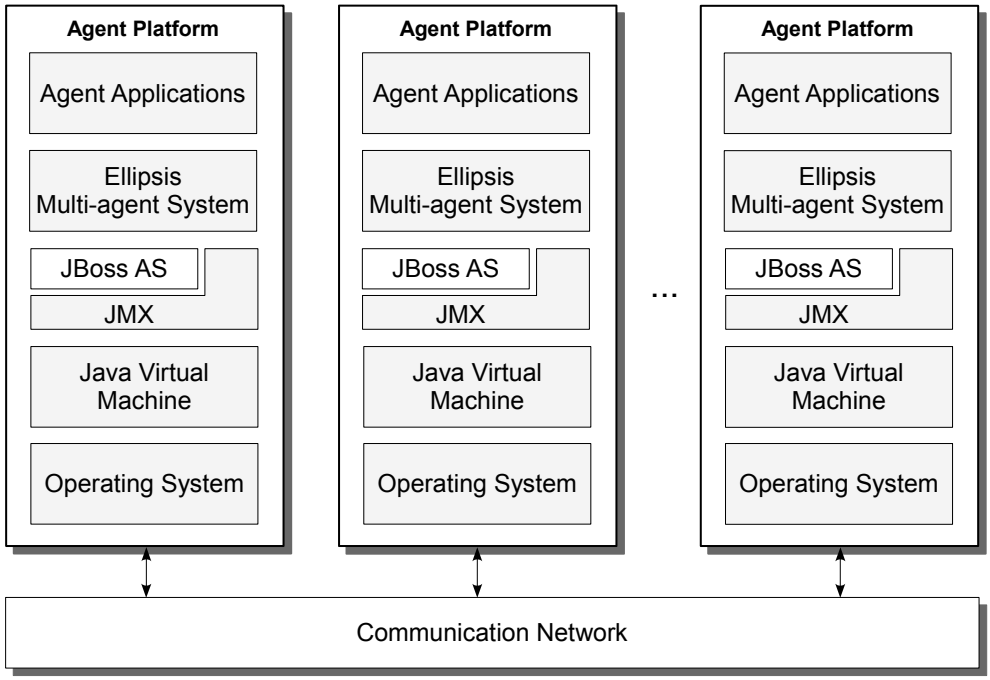

Figure 7.1: Ellipsis Infrastructure

and applications executed on it. JMX also supports remote administration through Remote Method Invocation (RMI).

- An object called MBean or Managed Bean accompanies the existence of JMX. An MBean is a Java object that works as a representation for the resources that need to be managed. To be called by other components an MBean must be registered to an MBean server. An MBean server hosts MBean objects and provides them with an interface to external objects.

- Ellipsis has been implemented to work in conjunction with Java Management Extension (JMX). Ellipsis's components are defined as MBean services, therefore they can easily be integrated into JBoss through the use of JMX.

- Applications run above Ellipsis can utilize the functionalities supported by Ellipsis.

7.2.2 System Model

The internal architecture of Ellipsis is shown in Figure 7.2 (Sch12).

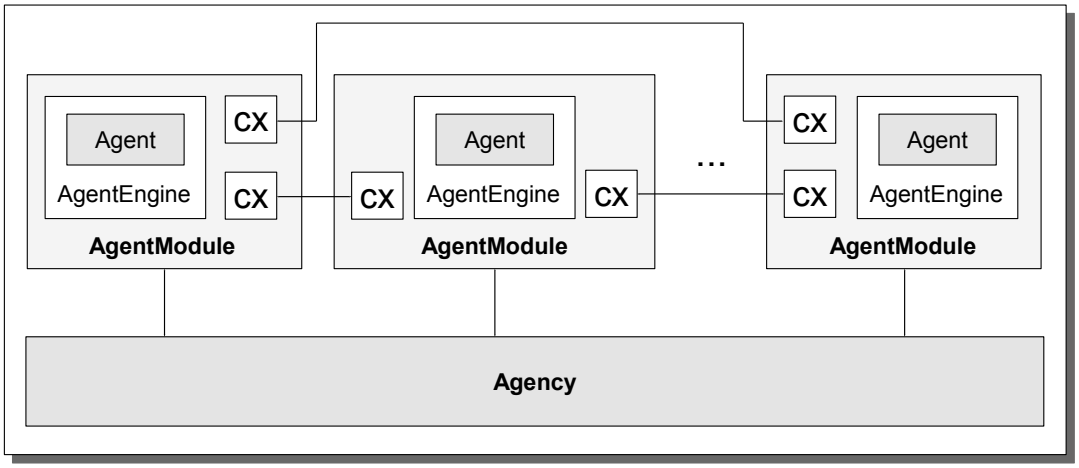

Figure 7.2: Ellipsis's internal architecture

- Agency is an MBean and is the core component of the entire system. It hosts and deploys mobile agents and performs tasks related to agent services. Agency holds different service directories for the management of agents: agent directory, service directory, node directory and protocol directory.

- AgentModule works as the container for an agent. It manages agent contexts, messages and events. AgentModule is also present in the form of an MBean.

- AgentEngine is integrated into AgentModule and provides an agent with an executing environment. This component manages the life cycle of an agent. With the use of this component, a foreign agent from other multi-agent systems can be imported to and executed in Ellipsis.

- Context (CX) creates a communication channel between two agents. Context provides other components with a repository of information related to current working context of the system.

7.2.3 Agents

Agent is a software entity, it can migrate and work proactively to fulfill tasks ordered by its owner. Ellipsis distinguishes between standard agents, service agents, and system agents.

- To be identified in different executing environments, an agent has an identity associated with it, the AgentCard. AgentCard helps to characterize an agent within a global context. Given an agent, AgentCard provides the following information about the agent:
  - The identifier of an agent within the system.
  - The owner of the agent.
  - The type of the MAS where the agent can work.
  - The name of the class that constitutes the agent.
  - A group of URIs for identifying agent resources.
- If an agent finishes its work but has no open contexts and no services then it switches to the status FINISHED. The agent and its corresponding AgentCard are then transferred into the holding area, which is managed by HoldingAreaAgent. The agent works as a root agent for the system. Other services that want to use an agent need to inform the HoldingAreaAgent.
- AgencyAgent is a root agent and it is started directly after the Agency. This agent provides search and register services to agents.
- LoaderAgent provides functionalities for starting agents. It works in conjuntion with Agency to create new mobile agents.
- MigrationServiceAgent provides an engine for migrating agents from a platform to another platform.
- EllipsisAgentEngine re-builds and assembles mobile agents from the transferred code when they migrate to a new platform.

7.2.4 Migration

In Ellipsis, scout agent's class data is supposed to be available at every platform thus an agent does not have to bring its class data with it while it is migrating. The only data that needs to be transferred over the network during migration is the data used for representing agent platforms.

In the honey bee inspired approach, agent migration will be employed in monitoring network. The issues concerning the migration process of mobile agents are beyond the scope of this dissertation. For interested readers we recommend the work of Peter Braun (Bra03).

7.2.5 Naming Convention

In the context of an MAS, the name of an agent or an agent platform must be unique within the entire context. In Ellipsis, naming convention is described in detail below:

- Agent platform is named as follows: **Platform's Private Name + IP Address**. For example, a platform with private name **BeeHive** and IP Address **10.35.12.85** has a unique name: **BeeHive.10.35.12.85**.
- An agent has a name in the form: **Agent's Private Name + "@" + Platform Name**. For example, an agent with the private name **Scout** and has been created in the platform with name **BeeHive.10.35.12.85** has a unique name **Scout@BeeHive.10.35.12.85**.
- The name of an agent remains intact during its lifespan, no matter what its whereabouts is.

7.3 Functionalities

Before going into further details regarding the design, it is of necessity analysing the functionalites of the software prototype. These functionalities are the realization of the proposals in Chapter 6. To achieve fault tolerance and reliability in transferring messages between mobile agents in a highly dynamic network, the

software prototype is expected to be able to perform the tasks as specified as follows. For our convenience, from now on the notations *agent platform*, *agency*, and *node* will be used interchangeably.

7.3.1 Network Monitoring

A network can be organized in harmony with the current environment only if there exists necessary information related to the network's condition. In the first place, the software prototype needs to chart environmental stimuli, which are the external changes occuring during execution.

The following parameters will be monitored (refer to Section 4.4 for more details):

- Changes in network topology.
- The accession and disjunction of network nodes.
- The connection quality between network nodes.

The monitoring process takes place at every platform of a region at regular intervals. It is comprised of the following sub-routines:

- Collecting network node's information and connectivity information.
- Nominating potential candidates for a new RegionMaster.

7.3.2 Self-Organizing

After all network nodes of a region have been surveyed, the collected information needs to be processed and served as a base for the decision making process. Self-organizing activities are conducted to maintain an equilibrium between the internal organization of the platforms and the external perturbations. These activites allow the colony of agent platforms to recover to a stable state if change or failure occurred.

The self-organizing activities are summarized as follows:

- Re-organizing a region when connectivity between members degrades.
- Voting new RegionMaster.
- Bringing up a reserve RegionMaster.
- Self-recovering whenever failure occurs.

The state diagram in Figure 7.3 shows the cooperative behaviour of the network monitoring and the self-organizing functionalities.

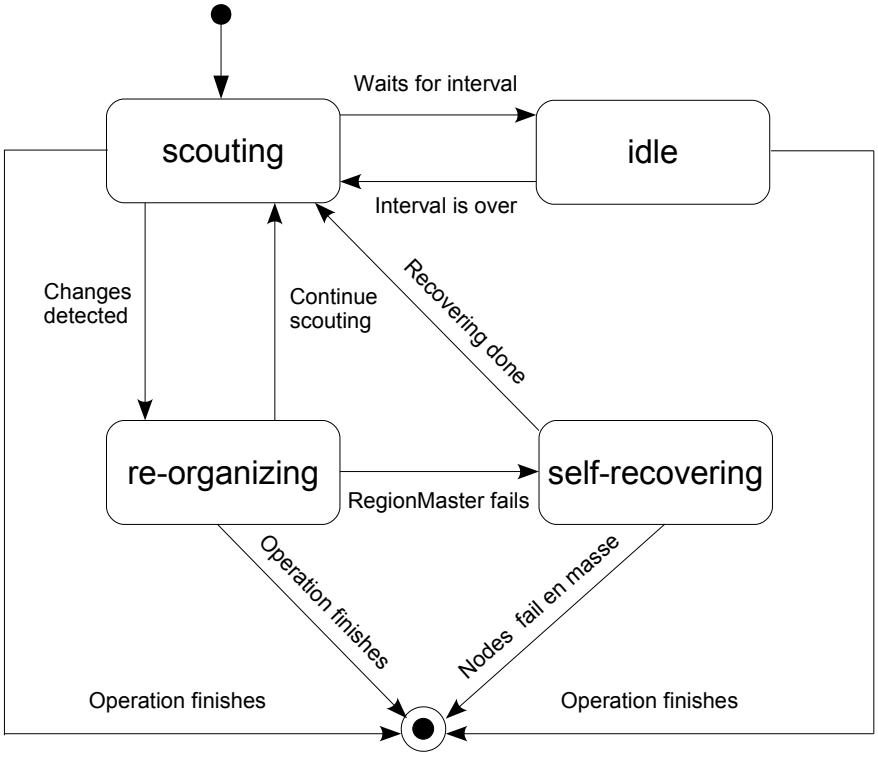

Figure 7.3: State diagram for network monitoring and self-organizing

7.3.3 Mobile Agent Positioning

Network monitoring and self-organizing tasks assist the system to maintain a resonable connection between agent platforms and to react to changes. Location

tracking mechanisms are to be adaptive to this organizational architecture after the adaption process took place.

The positioning tasks are divided into the following sub-tasks:

- Maintaining a location information cache.
- Receiving look up request from agents and agent platforms.
- Updating location cache once the logical network has been re-organized.
- Looking for location of mobile agents.

7.3.4 Agent Message Encoding, Delivering, and Storing

The layered model for agent communication provides us with a way to perform optimization in different levels (Hel03). A FIPA ACL message is comprised of multiple fields and each of them can be encoded, presented using suitable encoding techniques (NSR11a). Afterwards, messages can be transferred using appropriate message transport protocols (refer to Section 2.5.2 for more details).

For transferring agent messages across the network, agent platforms establish a network connection between communicating parties like in other conventional distributed systems. After the location of a targeted agent has been determined, messages will be delivered to the recipients. An incoming message will be decoded to the original form and saved into cache. The corresponding agent receives its messages by checking the cache.

The task is divided into different routines:

- Establishing a transmission channel between two agent platforms.
- Encoding and decoding messages.
- Tranferring messages.
- Saving messages into cache.
- Unloading messages from cache.

7.4 Software Components

To carry into effect the functionalities, we build a prototype software framework consisting of four functional components listed as follows:

- NetworkMonitor.
- NetworkOrganizer.
- LocationManager.
- Communicator.

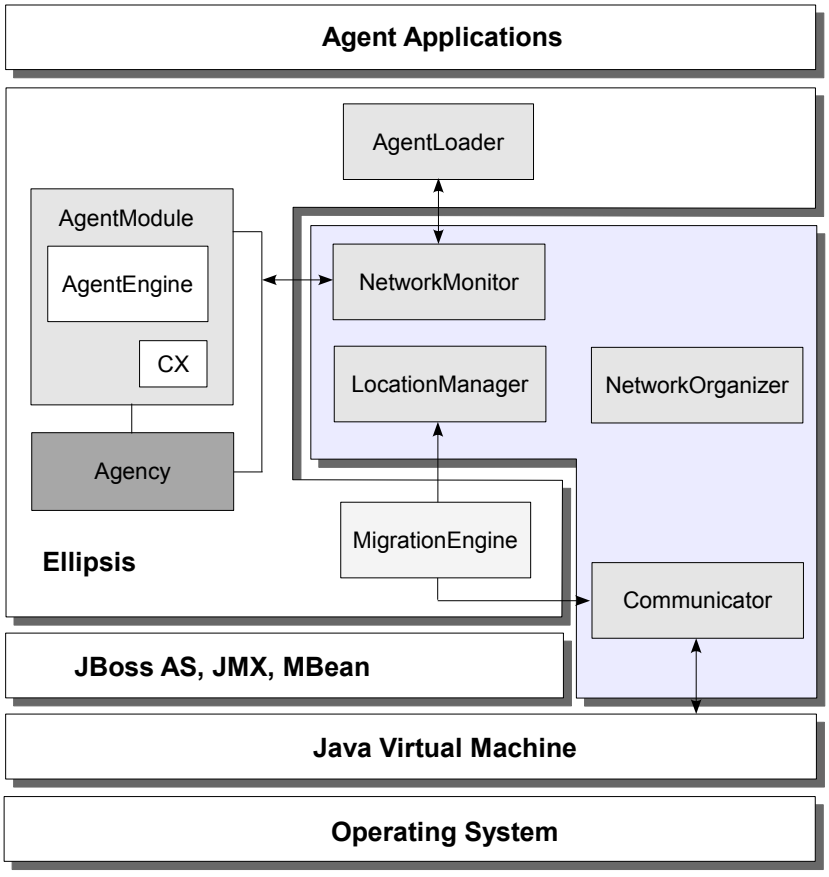

Figure 7.4: Relation to Ellipsis

The software prototype works in conjunction with Ellipsis. Figure 7.4 depicts the interface and relation of the software prototype to the MAS. The gray jigsaw puzzle represents the software prototype in relation with Ellipsis's existing software components. Essentially, the software prototype is a complement to the current Ellipsis architecture, it assists the system to work in dynamic networks. It does not interface directly to applications; all function calls are handled by Ellipsis.

The Euler diagram (HBF+07) in Figure 7.5 shows the software components and their corresponding functionalities. The self-organizing tasks are performed by NetworkMonitor, NetworkOrganizer and LocationManager. LocationManager and Communicator work together to ensure reliability in transferring agent messages. Fault tolerance is guaranteed by NetworkMonitor and NetworkOrganizer.

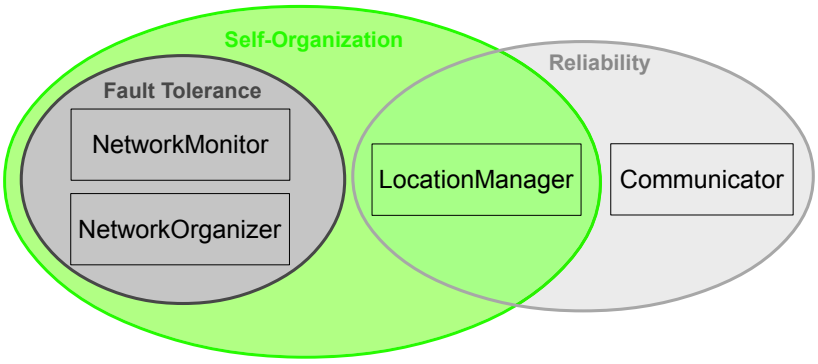

Figure 7.5: System's features and corresponding software components

The internal structure of the components and their mutual relationships are illustrated in Figure 7.6. NetworkMonitor gets input data from the surrounding environment. It manages a list of nodes working in its region. NetworkOrganizer makes decision based on the data processed by NetworkMonitor. It has interface to other NetworkOrganizers of the neighbouring platforms. Communicator encodes, sends and receives agent messages. It maintains a cache to store messages. LocationManager receives, answers and looks up mobile agent's location information.

We are going to introduce the design of the components in more detail in the next sections.

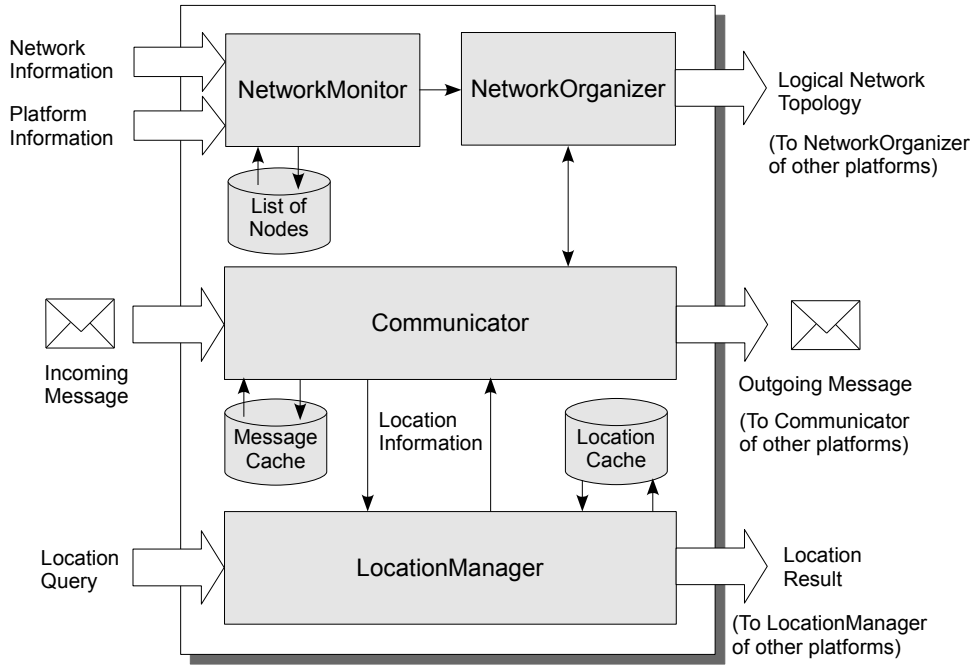

Figure 7.6: Functional structure of the constituent software components

7.5 NetworkMonitor

7.5.1 Functions

The software entity NetworkMonitor observes the agent platforms of a region. It collects connectivity information between a platform and its neighbours and information of the platforms. NetworkMonitor undertakes the following tasks:

- Managing a dynamic list of agent platforms.
- Deploying scout agents periodically to get update on the network situation.
- Receiving and replying ping messages.
- Calculating the values τ_{avg} and $split$.
- Making a decision of splitting region by counting the boolean values $split$.

7.5.2 Sub-Components and Relationships

NetworkMonitor consists of four sub-components. The sub-components and their mutual relationships are displayed in Figure 7.7.

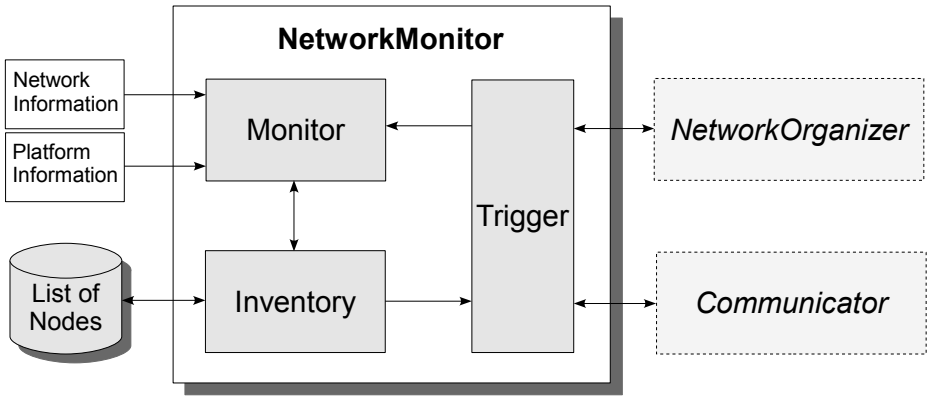

Figure 7.7: NetworkMonitor: Sub-components and mutual relationships

- Inventory handles the cache for storing the list of nodes in a region.
- Monitor deploys scout agents and waits for ping messages. It sends ping messages back to the senders.
- Trigger works as a port for exchanging information with NetworkOrganizer and Communicator.

7.5.3 Processing

The state diagram in Figure 7.8 illustrates the behaviour of Monitor. The sub-component runs on every platform, but only the one that exists on the RegionMaster is responsible for deploying scout agents. Monitor remains dormant until the corresponding platform becomes a RegionMaster. A platform starts to work as a RegionMaster either by being configured from the beginning or being voted and promoted by other platforms once Inventory receives a list of agent platforms.

The process of monitoring a region begins at the RegionMaster. Based on the scale of the region, Monitor of the RegionMaster creates an appropriate nummber

of scout agents. Monitor assigns each scout agent a list of nodes that are going to be visited by that agent.

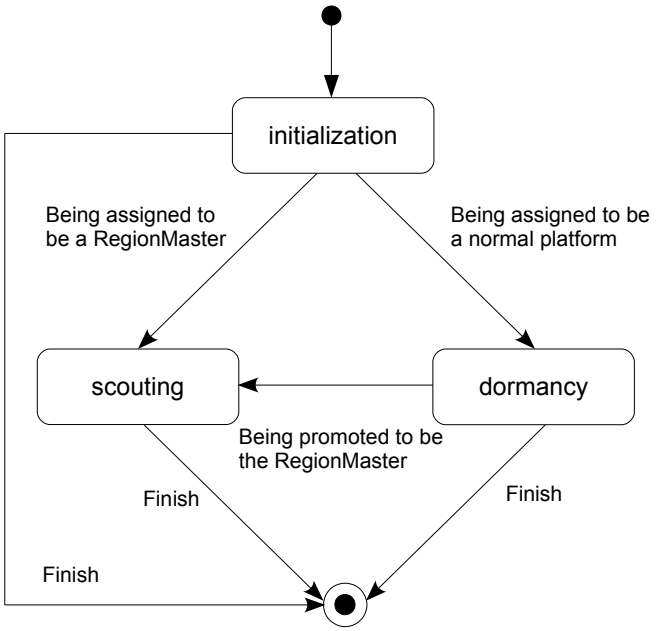

Figure 7.8: State diagram for Monitor

At a platform, when a scout agent arrives, the platform sends ping messages to all nodes of the region. Monitor listens on a TCP port for incoming ping messages. When Monitor receives a ping message, it sends the message back to the sender. Monitor waits for a specific time until all ping messages come back.

The scout agent uploads performance information of the previous node to Monitor. Once all ping messages from the neighbours arrive, Monitor calculates the average time τ_{avg} and $split$ as specified in Section 6.3.4. Monitor compares the performance of the previous platform of the itinerary with the performance of the current platform and nominates the platform with the higher performance as a candidate for a new RegionMaster.

Monitor hands over the following information to the scout:

- The values τ_{avg} and $split$.
- Performance information of the platform.

- The candidate for a new RegionMaster.

Afterwards, Monitor sends the scout to the next node on its itinerary. At the new node, the same routines repeat. After the final node on its itinerary has been investigated, the scout migrates back to the RegionMaster.

Once all scout agents have completed their tasks and come back to the RegionMaster, they submit information collected to Monitor of the RegionMaster. Monitor counts the number of platforms that expect the region to be bisected, based on the values $split$. If the poll constitutes a quorum, Trigger sends the list of platforms to NetworkOrganizer. Otherwise, Monitor continues to deploy scout agents to perform a new round of surveillance.

7.5.4 Data

A scout agent brings data with it when it migrates. The data consists of records, each stores information of an agent platform in the region. The structure of a record is shown in Figure 7.9.

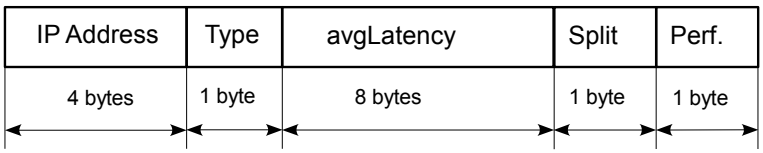

Figure 7.9: The data structure represents a platform's information

where

- $IPAddress$ is a 4-byte field and stores the agent platform's address.
- $Type$ is a 1-byte field, it defines the type of agent platform, specified as follows:
  - 0: Normal platform.
  - 1: RegionMaster.
  - 2: Reserve RegionMaster.
  - 3: Lightweight platform.

- $avgLatency$ is 8-byte field and holds the average latency from the current platform to the remaining platforms.
- $Split$ is a 1-byte boolean value and indicates if the node wants the region to be splitted or not (0: No; 1: Yes).
- $Performance$ is 1-byte field; it represents the computing performance of the platform.

The structure of the list of nodes in a region is depicted in Figure 7.10.

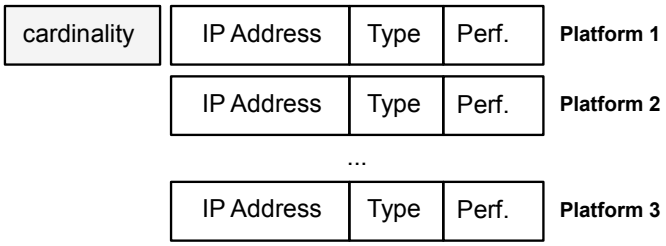

Figure 7.10: The list of nodes in a region

- $cardinality$ stores the number of records in the list.
- The fields that have the same name have the same meaning as defined aboved.

7.6 NetworkOrganizer

7.6.1 Functions

NetworkOrganizer re-organizes the logical network based on the conditions percepted by NetworkMonitor. This component is present at every platform of a region. However its role is dependent on the type of the platform. In a RegionMaster, NetworkOrganizer has the following functions:

- NetworkOrganizer receives the list of platforms from NetworkMonitor.

- NetworkOrganizer selects a new RegionMaster from the candidates nominated by NetworkMonitor. It broadcasts the ID of the new RegionMaster to all platforms of the region to conduct a poll.

- NetworkOrganizer activates a platform when the platform has been promoted to the post of a RegionMaster.

- NetworkOrganizer notifies other NetworkOrganizers of the changes by sending broadcast messages.

In a normal platform, NetworkOrganizer performs the succeeding tasks:

- It receives a proposal for a node to be the new RegionMaster from the RegionMaster.

- It chooses which region the corresponding platform should belong to by measuring the latencies to the proposed platform and to the RegionMaster.

- It sends the ID representing the chosen platform back to the RegionMaster.

7.6.2 Sub-Components and Relationships

- Organizer communicates with NetworkMonitor to obtain the list of platforms. It returns the new list to the NetworkMonitor and sends the list to the new NetworkMonitor and activates this platform. Organizer votes for node to be the new RegionMaster. It informs all nodes of the changes and waits for replies from the platforms.

- Observer is present at a reserve RegionMaster. It regularly senses the presence of the RegionMaster.

- Broker provides Communicator with necessary information for transferring agent messages.

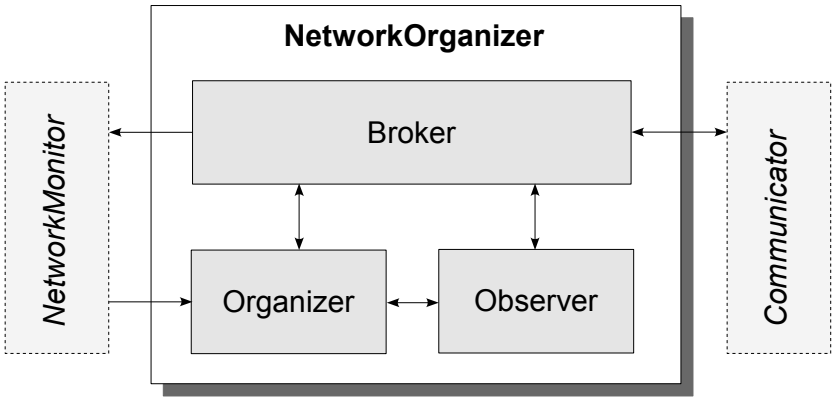

Figure 7.11: NetworkOrganizer: Sub-components and mutual relationships

7.6.3 Processing

Figure 7.12 illustrates the functions of NetworkOrganizer in different roles. The upper part depicts NetworkOrganizer in a RegionMaster and the lower part depicts NetworkOrganizer in a normal platform.

At a RegionMaster, NetworkOrganizer takes up its job once Organizer receives the list of agent platforms from NetworkMonitor. From the platforms nominated by the scout agents, QueenBee selects the candidate that meets the criteria described in Section 6.4.2 in Page 68 to be the new RegionMaster. Organizer sends the ID of the proposed platform to all nodes of its region and waits for replies. This aims to poll every platform of the region which region it prefers to belong to.

At a normal platform, when receiving the ID representing the proposed platform, Organizer sends one ping message to the proposed platform and one ping message to the current RegionMaster to measure the corresponding latencies. Organizer selects the platform with the lower latency to be its new RegionMaster. That means a platform can select only one of the two proposed platforms as the RegionMaster. The ID of the chosen platform will be sent back to the RegionMaster.

After Organizer of the RegionMaster gets the decisions from the platforms, it has two lists of platforms. The first list contains the platforms that choose to stay in the old region, the second list stores the platforms that join the new region. Organizer sends the first list back to Monitor of the RegionMaster, causing this

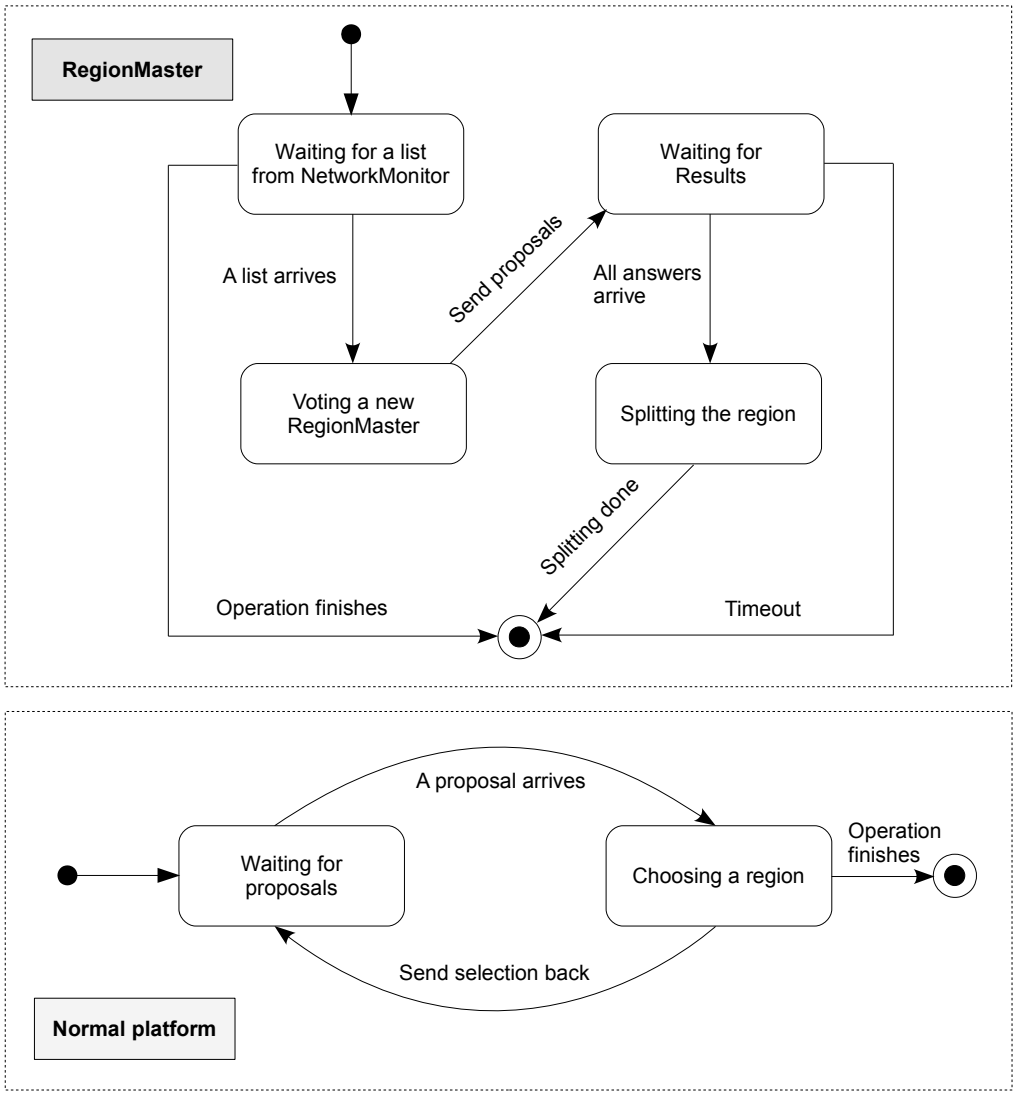

Figure 7.12: State diagram for the behaviours of different types of Organizer

platform to continue scouting but with a new set of platforms. It sends the second list to the new RegionMaster. By sending the list, it triggers the new RegionMaster to commence its operations as a RegionMaster.

The flowchart in Figure 7.13 represents the processing of a reserve RegionMaster. Its Observer checks the availability of the RegionMaster by sending an inquiry periodically to this platform. If it receives the message, Observer of the RegionMaster sends a heartbeat message back to acknowledge its presence. In

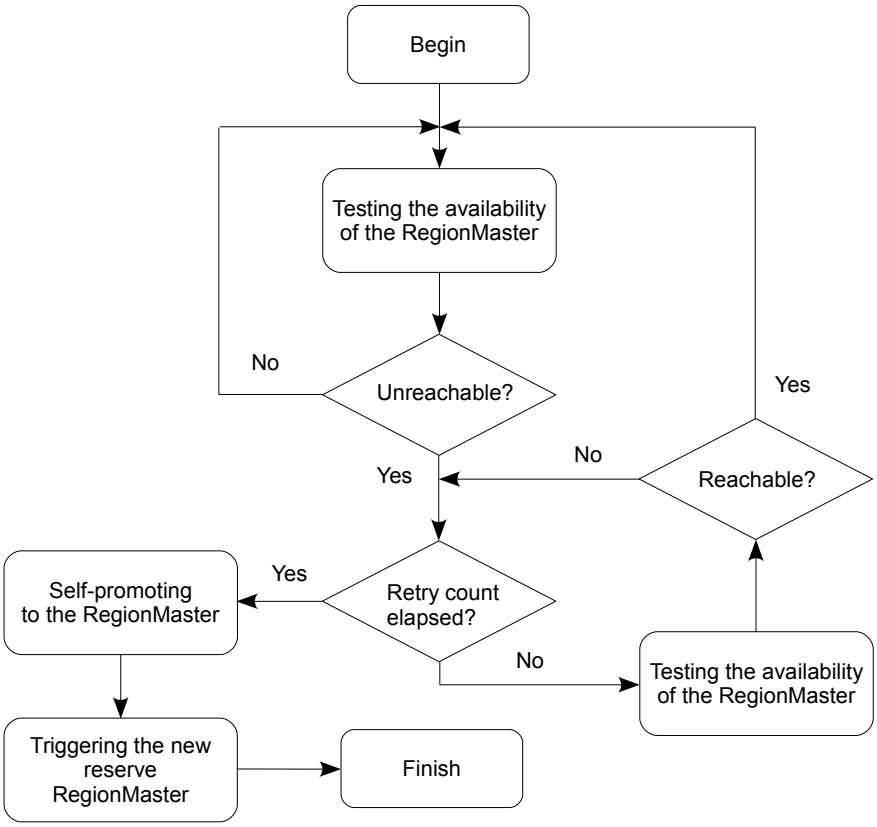

Figure 7.13: The nature of a reserve RegionMaster

this process, timeout is handled in order to ensure reliability. If the acting RegionMaster stalls, the reserve RegionMaster receives no heartbeat message. Given the circumstances Organizer of the reserve RegionMaster tries to send a message again. After a specific number of attempts, if the platform is still unreachable then Observer nominates the reserve RegionMaster to be the RegionMaster of the region. This platform begins the tasks of a RegionMaster.

7.6.4 Data

The format of the list sent by NetworkMonitor to NetworkOrganizer is depicted in Figure 7.14.

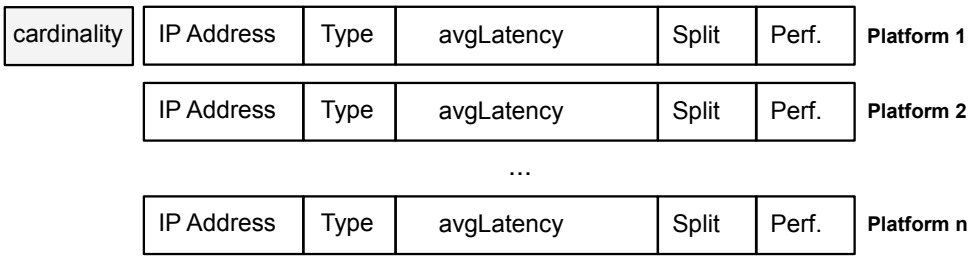

Figure 7.14: The list sent by NetworkMonitor to NetworkOrganizer

7.7 LocationManager

7.7.1 Functions

LocationManager manages location information of mobile agents, it updates location of mobile agents when they migrate, answers location query and queries agent's location information. LocationManager holds a cache for storing location information. Essentially, the component performs its function to answer the question: *Which agent locates where?*

The following tasks are undertaken by LocationManager:

- Interfacing to Ellipsis to manage a dynamic list of location information for mobile agents.
- Receiving and answering location query from other nodes as well as from agents.
- Sending location information query to other LocationManagers.

7.7.2 Sub-Components and Relationships

- Inventory handles the cache that stores location information of mobile agents.
- QueryManager receives and answers location information query.

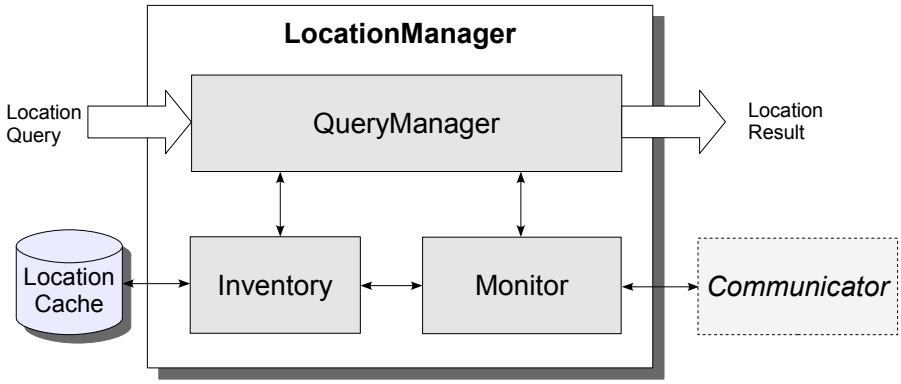

Figure 7.15: LocationManager: Sub-components and mutual relationships

7.7.3 Processing

Manager listens on a TCP port for location queries. Manager in a normal platform can query Manager of the RegionMaster for location of mobile agents. Manager in a RegionMaster can accept and answer location queries from its platforms or other RegionMasters. When a mobile agent arrives at a platform, Manager inserts the agent's ID into its cache. It sends an update messages to Manager of the RegionMaster to reveal the presence of the agent.

When a mobile agent leaves a platform, Manager of the platform removes the corresponding entry in its cache. If Manager receives a look up query, it checks the cache to see if the agent's address is in there. If the address is found, Manager sends it back to the sending agent. Otherwise it sends a query to its RegionMaster and waits for reply.

7.7.4 Data

The location cache of a normal platform has the format depicted in Figure 7.16. The field $cardinality$ stores the number of entries in the cache. Each AgentID represents the identity of an agent currently working on the platform.

Figure 7.17 depicts a location cache in a RegionMaster. The cache saves location information of all agents working in the platforms of its region. A mobile agent can be located by a tuple of two fields, AgentID and the IP address of the platform where the agent is working in.

Figure 7.16: Format of the location cache in a normal platform

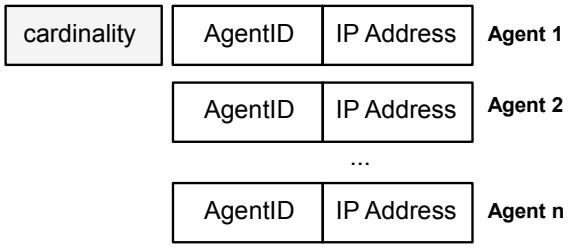

Figure 7.17: Format of the location cache in a RegionMaster

7.8 Communicator

7.8.1 Functions

The software entity Communicator is responsible for initializing connection between platforms and transferring messages between agents. It encodes, decodes, and transfers agent messages over the network. This component maintains two caches of agent messages at every platform, one for incoming messages and one for outgoing messages.

Communicator has the following functions:

- Establishing connection between two agent platforms.
- Encoding the envelope and the payload of an ACL message at the sender side.
- Sending byte streams containing ACL messages over the network.
- Decoding the envelope and the payload of ACL messages at the receiver side.

7.8.2 Sub-Components and Relationships

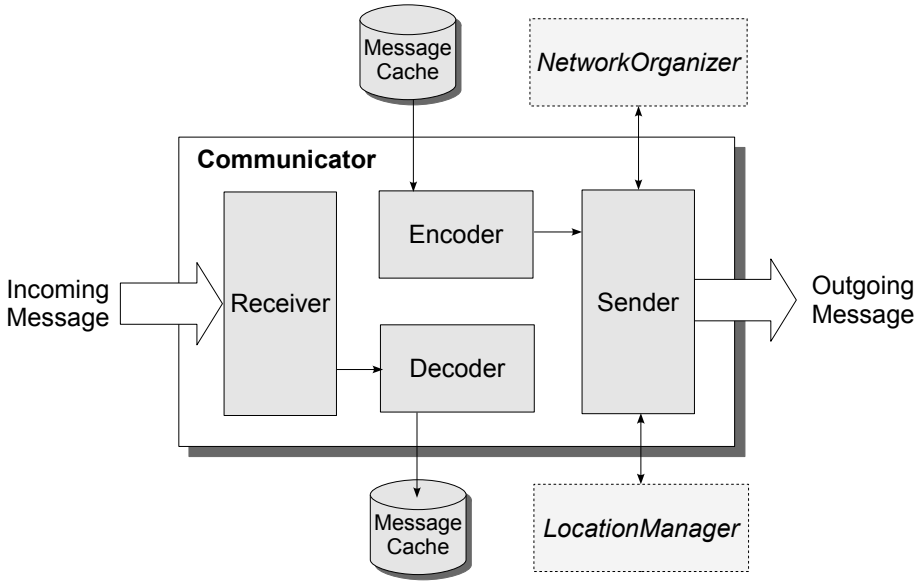

Figure 7.18: Communicator: Sub-components and mutual relationships

- Encoder encodes agent messages in two levels: envelope and payload.
- Sender delivers messages to the recipient.
- Receiver listens on a port and waits for incoming messages.
- Decoder decodes agent messages in two levels: envelope and payload.

7.8.3 Processing

Figure 7.19 illustrates the process of delivering an ACL message from a sender to a receiver. At the sender side, Sender checks the cache periodically to get outgoing messages. If there is any message that needs to be sent, Encoder converts the message into payload using the Bit-efficient ACL message encoding algorithm (FIP02a). Encoder also encodes the message's envelope using XML Codec. Sender establishes a TCP communication channel to the destination. The message then is encapsulated in an Ethernet packet and transferred over the network.

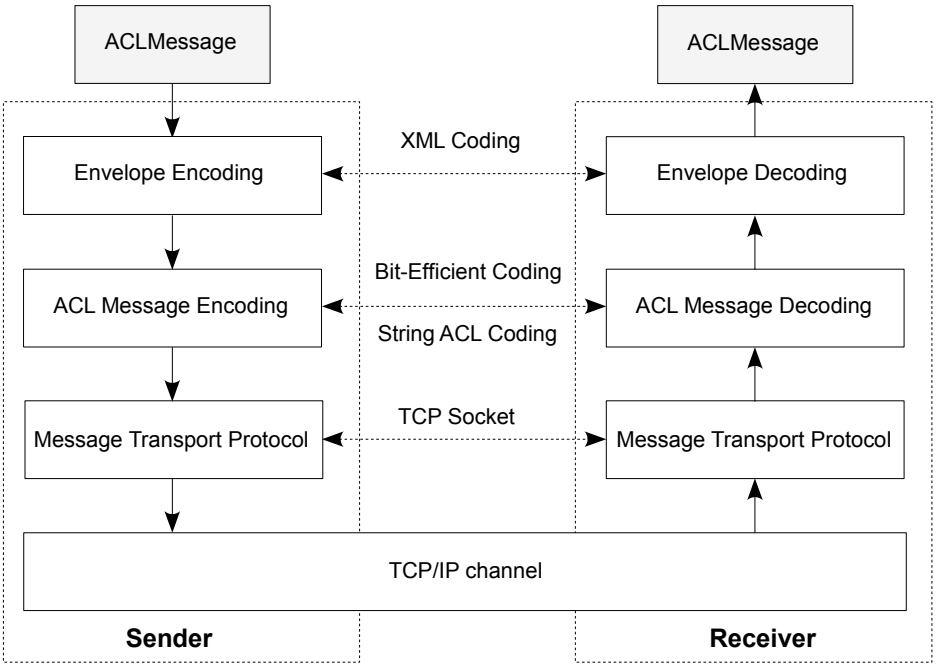

Figure 7.19: ACL Communication

At the receiver side, Receiver waits for incoming connection on a TCP port. It receives incoming messages using the Message Transport Protocol interface. The received data is then decoded using the Bit-efficient ACL encoding. Decoder decodes the envelope of the message. The message will be saved into the incoming message cache. Agent checks new messages by accessing the cache. This is done by application programmers.

7.9 Scout Agents

Scout agents both form an important part of the software prototype as a whole and have much to contribute to achieving fault tolerance and reliability in message transferring. The above mentioned software components are able to work efficiently only when scout agents supply them with comprehensive information of the surrounding environment.

Ellipsis supports three types of agents: standard agent, service agent and system agent. To meet the requirements of a scout agent, however, a standard agent

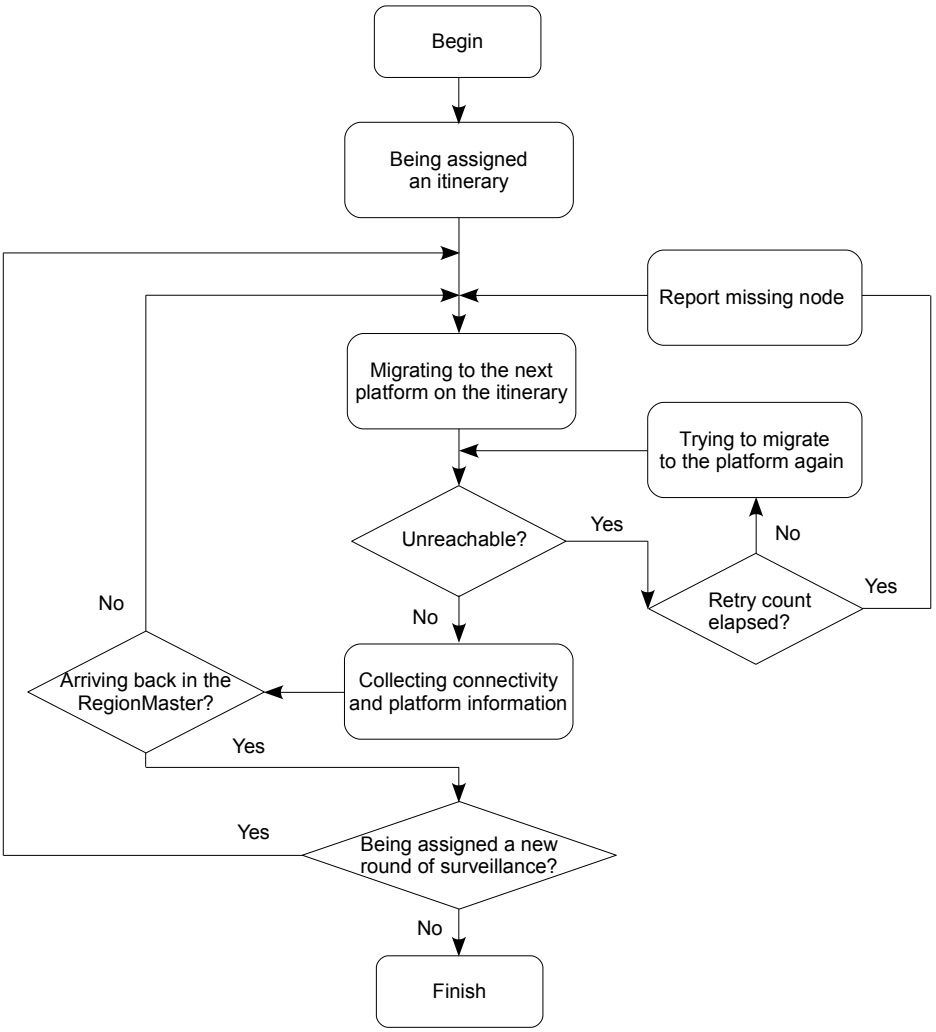

Figure 7.20: The behaviour of a scout agent

needs to be customized. A scout agent has common features of a conventional mobile agent, in addition it has other distinctive features. The flowchart in Figure 7.20 depicts the nature of a scout agent. At the creation time, a scout agent is assigned an itinerary - the list of agent platforms that the agent is going to visit. On each platform of its itinerary, the scout agent performs its routines to collect necessary information. Based on timeout handling, it can report a broken link and the existence of an agent platform. Once it finishes a round of surveillance and returns to the RegionMaster, the scout agent interfaces to NetworkMonitor to

submit information it has fetched. It continues to travel around the region until the region is splitted. Under the circumstances, the scout agent will be assigned a new route and it starts its work from the start.

7.10 Design Summary

This chapter provided the design of the software prototype. The design helps to turn concept into realization and develop a plan for the implementation. With the aims of attaining fault tolerance and reliability in transferring agent messages, the design has been based on the honey bee inspired approach described in Chapter 6.

Being interfaced to the multi-agent system Ellipsis, the software prototype consists of four software components: NetworkMonitor, NetworkOrganizer, LocationManager, and Communicator. Each of the components has its own features and functions. NetworkMonitor employs mobile agents to chart the changes of the network environment. NetworkOrganizer re-organizes a region of agent platforms based on the information acquired by NetworkMonitor. Communicator is responsible for storing, encoding, and transferring agent messages over networks. LocationManager receives, answers and looks up mobile agent's location information. The components interface to the existing components of Ellipsis and cooperate with each other in obtaining the targeted qualities. NetworkMonitor and NetworkOrganizer provide fault tolerance. The two components also work in conjuntion with LocationManager to bring in self-organization. LocationManager and Communicator work together to ensure reliability in transferring agent messages.

A scout agent is built based on a standard agent supported by Ellipsis with some customization. Scout agents work as information feeder for the software components. They contribute to the overall performance of the approach, thereby being deemed to be of importance to the prototype.

Chapter 8

Implementation

8.1 Connection Management in Java

For distributed applications working in dynamic networks, it is essential to manage network connection and data transmission adequately. A sudden broken network connection causes a data transfer to be interrupted and forces the corresponding thread to wait endlessly. Hanging threads both waste resources and reduce system's performance. In the worst case, a server becomes inoperative and cannot respond to legitimate traffic if it is saturated with unreleased resources.

Efficient network connection establishment and management contribute to achieving fault tolerance and reliability. Timeout handling is essential, especially in dynamic networks. Good timeout handling helps detect broken links as well as transfer failures, thereby facilitating the removal of wild threads. Timeout handling can be considered as an integral part of the programming logic for distributed applications.

Java supports TCP and UDP sockets. UDP is faster than TCP but not reliable since it transfers data packets but does not guarantee that they arrive at the destination. TCP sockets provide us with a convenient way to reliably transfer data over the network. In return, TCP has a slower speed compared to that by UDP. For a TCP socket, there are two communication channels, an input channel for getting the incoming data stream and an output channel for delivering the outgoing data stream. Timeouts can be handled by working with the two channels using excep-

tion in socket programming. Specifically, timeout needs to be controlled in three phases:

- Connection establishing.
- Data sending.
- Data receiving.

A timeout handling at the connection establishment phase is depicted in Listing 8.1. At the first attempt, if there is no connection then the client retries to connect after the duration of MAX_DELAY (ms). After a number of tries, defined by MAX_RETRY, either a connection is established or the connecting attempt is aborted.

Listing 8.1: Timeout handling at connection establishment

```
public void handleTimeoutAtConnection() {
  connection = new TCPConnection(destIPAddress, destPort);
  int retryCount = 0;
  int retVal = -1;
  try {
    retVal = connection.connect();
    while((retVal==NO_CONNECTION) && (retryCount<MAX_RETRY)){
      retVal = connection.connect();
      retryCount += 1;
      Thread.sleep(MAX_DELAY);
    }
  } catch (UnknownHostException uhe) {
    uhe.printStackTrace();
  } catch (IOException ioe) {
    ioe.printStackTrace();
  }
}
```

The code excerpt in Listing 8.2 illustrates the delivering of a data buffer over a TCP socket's output channel and the corresponding failure handle. If the connection is broken while sending is taking place, the sender can catch errors, thereby helping to avoid producing a hanging thread.

Listing 8.2: Timeout handling at sending

```
public void handleTimeoutAtSending() {
  connection = new TCPConnection(destIPAddress, destPort);
  try {
    outstream = new DataOutputStream(connection.getOutputStream());
    outstream.write(buffer);
    outstream.flush();
  } catch (InterruptedIOException iioe) {
    iioe.printStackTrace();
  } catch (IOException ioe) {
    logger.info("The connection is lost");
  }
}
```

Listing 8.3 shows the management of data transfer at the recipient side. By using this code, the receiver is able to detect tranmission errors once they occured and try to resume the receiving process. In extreme cases, after a number of failed attemps, the receiver aborts the transmision gracefully.

Listing 8.3: Timeout handling at receiving

```
public void handleTimeoutAtReceiving() {
  connection = new TCPConnection(destIPAddress, destPort);
  InputStream inputStream = connection.getInputStream();
  int retryCount = 0;
  try {
    while (retryCount<MAX_RETRY) {
      read = inputStream.read(buffer, bytesRead, length-bytesRead);
      bytesRead += (read > 0) ? read : 0;
      if (bytesRead >= buffer.length) break;
      retryCount += 1;
    }
  } catch (IOException ioe) {
    ioe.printStackTrace();
  }
}
```

Throughout the implementation of the software prototype, timeout handling is going to be employed in the following activities:

- Migrating scout agents.
- Measuring network latency.
- Detecting the availability of a RegionMaster.

8.2 NetworkMonitor

8.2.1 Classes

NetworkMonitor is implemented as a Java package and encapsulated in an executable *.jar file, named $agent.networkmonitor.jar$. Figure 8.1 depicts a simplified class diagram for the NetworkMonitor package. The classes and their main functions are explained as follows:

- MigrationEngine interfaces to the MigrationServiceAgent of Ellipsis. It is an enhanced version of the migration service of Ellipsis as it facilitates the management of scout agents as well as supports connection handling. It can provide scout agents with their own way of migrating.
- MigrationIn and MigrationOut are responsible for the management of incoming migration and outgoing migration, respectively.
- AgentHandler is the interface for reconstructing an agent from an incoming data stream.
- TCPConnection is the pipeline for transferring native byte stream over the network. It supports timeout handling.
- Trigger is used to activate the component Monitor of a platform when the platform is configured as a RegionMaster.
- Receiver is an abstract class, it provides interface for receiving incoming byte streams. Monitor, AgentReceiver and MessageReceiver are instances of this class.
- Monitor is normally on standby. When it is activated by Trigger, it deploys and manages scout agents.

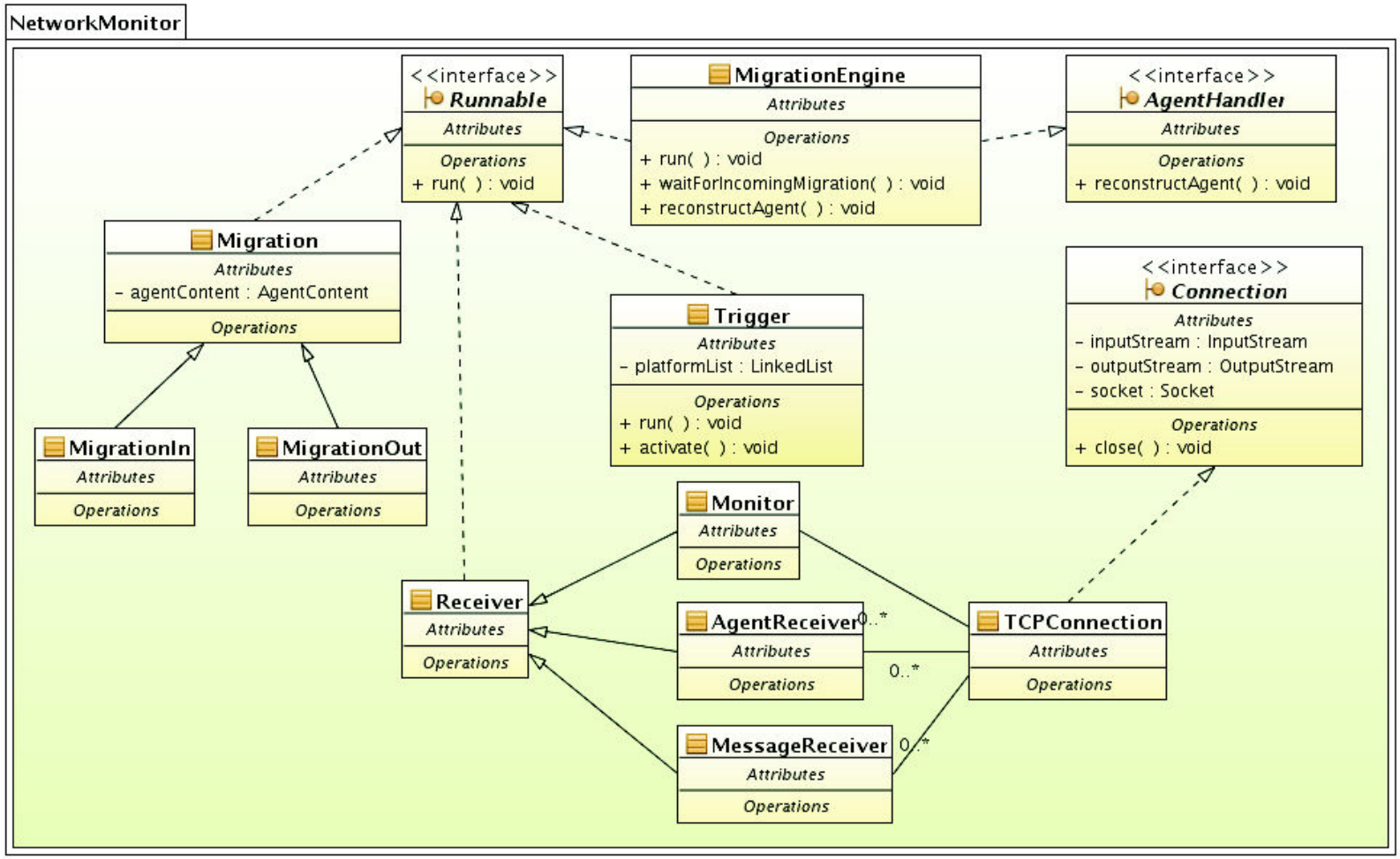

Figure 8.1: A simplified class diagram for NetworkMonitor

8.2.2 Realization

Monitor manages scout agents. Initially, it is idle and will be activated only when the corresponding platform receives a list of platforms. Pseudo code 1 illustrates the behaviour of Monitor on its activation. Scout agents are created and each of them is assigned a list of platforms. The number of agents is dependent of the scale of a region. The behaviour of a scout agent when it arrives at an agent platform is going to be specified later in Section 8.6.

Pseudo code 1 Activating Monitor

```
1: procedure ONACTIVATION(platformList)
2:     scoutAgent ← createScoutAgent(platformlist)
3:     scoutAgent.initialize()
4: end procedure
```

In Pseudo code 2, an agent platform measures the average latency by sending ping messages to all other platform of the same region. The values are then averaged out to get the final result as in Line 5.

Pseudo code 2 Measuring the average latency

```
1: function MEASUREAVERAGELATENCY(thisPlatform,allOtherPlatforms)
2:     for every platform do
3:         latency ← sendPingMessage(message,platform)
4:     end for
5:     return average(latency)
6: end function
```

Pseudo code 3 shows how Monitor reacts when it receives a ping message from an other platform. Whenever a ping message arrives, Monitor sends the message back to the sender. Timeout is handled in this phase to prevent a hanging connection from happening. The platform that sent the message when receiving the message back will calculate the time that the message needs to circumnavigate, resulting in the corresponding latency.

In the beginning, a platform is configured as the RegionMaster of a region. Pseudo code 4 describes how Trigger is used to activate a platform to undertake

Pseudo code 3 Receiving a ping message

```
1: procedure ONMESSAGEARRIVAL(message)
2:     sendMessageBackToSender(message)
3: end procedure
```

the tasks of a RegionMaster. A list of platforms in a region is sent to Monitor in Line 2, causing the platform to work as RegionMaster.

Pseudo code 4 Triggering a platform

```
1: procedure TRIGGERING
2:     sendPlatformList(platformList)
3: end procedure
```

As soon as a RegionMaster starts to work, it uses Trigger to assign the tasks of a reserve RegionMaster to a second platform. The second platform will test the availability of the RegionMaster by periodically sending inquiry messages. The behaviour of a reserve RegionMaster is going to be described in the next section.

8.3 NetworkOrganizer

8.3.1 Classes

NetworkOrganizer is built in $agent.networkorganizer.jar$. This package is automatically launched every time JBoss AS is invoked. Figure 8.2 represents a simplified class diagram of NetworkOrganizer.

- Organizer is responsible for performing organizing tasks for a region.
- Observer acknowledges the presence of RegionMaster by sending ping messages.
- Trigger has the same functionality with the component with the same name of NetworkMonitor.
- Listener listens for incoming messags for Observer and Organizer. It is automatically invoked when JBoss AS starts.

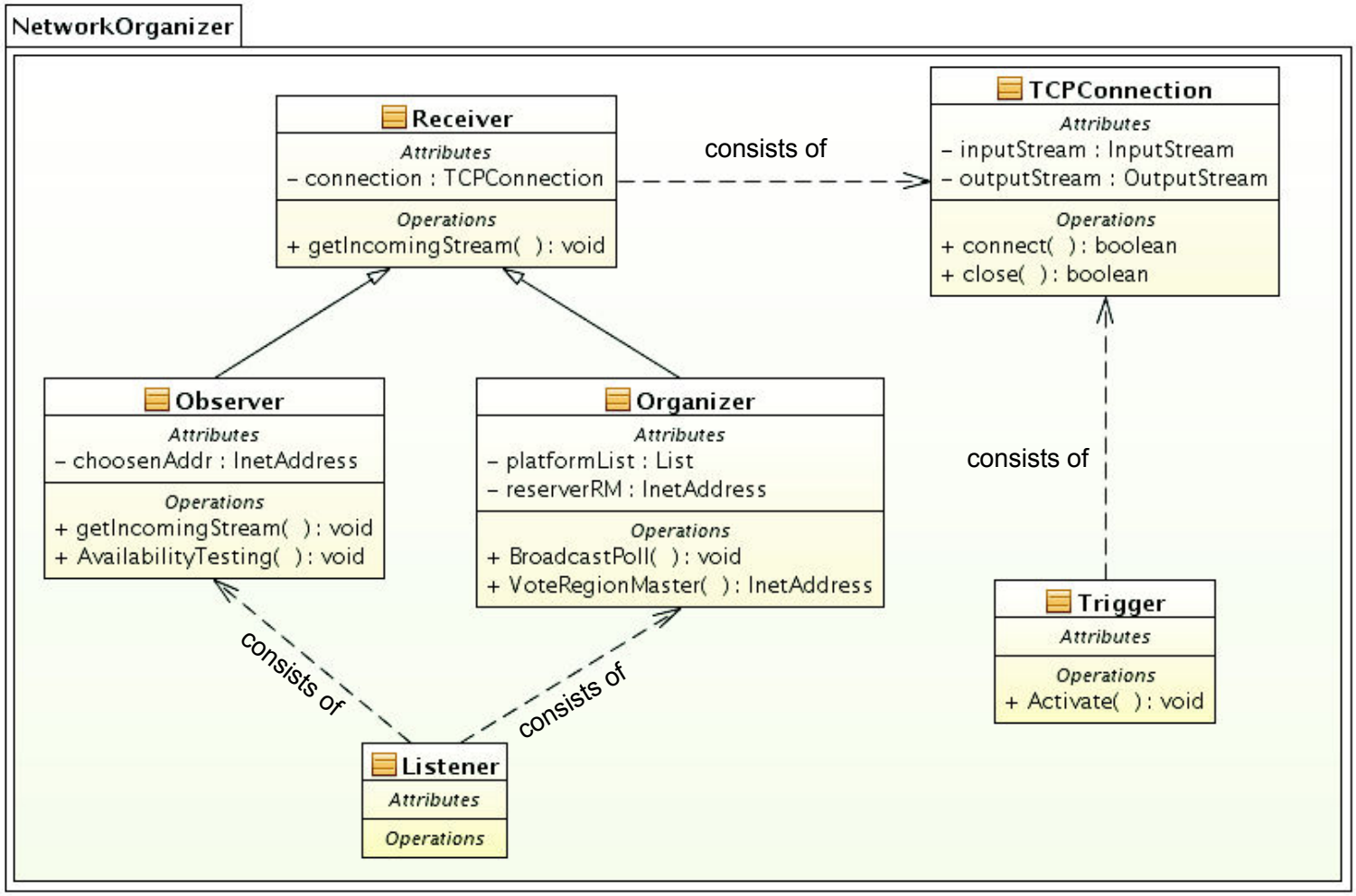

Figure 8.2: A simplified class diagram for NetworkOrganizer

8.3.2 Realization

NetworkMonitor assigns an observer whenever it starts to work as the RegionMaster by sending an activation message to NetworkOrganizer. When NetworOrganizer receives the message, it deploys Observer to look after the RegionMaster so that when the platform fails a substitution can be made. Pseudo code 5 illustrates how Observer works.

Pseudo code 5 Testing the availability of the RegionMaster

```
 1: procedure AVAILABILITYTESTING(RegionMaster)
 2:     testing ← true
 3:     while testing do
 4:         connection ← establishConnection(RegionMaster)
 5:         handleTimeoutAtConnection()
 6:         sendInquiryToTheRegionMaster(connection)
 7:         handleTimeoutAtSending()
 8:         waitForReply()
 9:         handleTimeoutAtReceiving()
10:     end while
11:     selfPromoteToRegionMaster()
12: end procedure
```

In this function, timeout is handled in three phases: at connection establishment, during sending and during receiving. The codes for handling timeout in these phases have been already specified in listings 8.1, 8.2 and 8.3 in Section 8.1. Line 5 handles timeout at connection establishment. Line 7 manages timeout during data sending. Line 9 detects errors while data receiving is taking place. Whenever error occurs in one of the three phases, the variable $testing$ is set to $false$ and the program exists the $while$ loop and executes the code in Line 11, thereby promoting the reserve RegionMaster to the RegionMaster.

Another component which is also important by NetworkOrganizer is Organizer. There may be multiple scout agents and each of them nominates its own candidate for a new RegionMaster. When the region needs to be splitted, the RegionMaster selects one from the candidates to be the new RegionMaster. Organizer broadcasts

the identity of the RegionMaster and the proposed platform to all other platforms to conduct a poll. The behaviour is illustrated in Pseudo code 6.

Pseudo code 6 Broadcasting a poll for new RegionMaster

```
1: procedure BROADCASTPOLL(RegionMaster,proposedPlatform)
2:     for every platform do
3:         proposeRegionMaster(RegionMaster,proposedPlatform)
4:     end for
5: end procedure
```

Each platform should choose which region it belongs to by measuring the latencies to the RegionMaster and the selected platform. Pseudo code 7 demonstrates what a platform does when it receives a proposal for the RegionMaster. It measures the latency to the RegionMaster (Line 2) and to the proposed platform (Line 3). The platform with a lower latency will be selected as the new RegionMaster (Line 5 and Line 7). In our implementation, we employ a simplified model for selecting a RegionMaster that uses latencies as the only criterion. However, we recommend that the parameters related to the computing power of a candidate should be considered. Interested readers are referred to the work of Arndt Döhler (DÖ8).

Pseudo code 7 Voting a new RegionMaster

```
1: function VOTEREGIONMASTER(RegionMaster,proposedPlatform)
2:     latency1 ← measureLatency(RegionMaster)
3:     latency2 ← measureLatency(proposedPlatform)
4:     if latency1 ≤ latency2 then
5:         return(RegionMaster)
6:     else
7:         return(proposedPlatform)
8:     end if
9: end function
```

8.4 LocationManager

8.4.1 Classes

LocationManager is encapsulated in the Java package $agent.locationmanager.jar$. The package should be called by JBoss AS right from the start.

- QueryManager listens on a TCP port for incoming location information queries. It can look for mobile agent address by communicating with other platforms.
- Whenever a mobile agent arrives at a platform, Monitor inserts the ID of the agent into its location cache.

8.4.2 Realization

Pseudo code 8 describes how the location of a mobile agent is updated. When a mobile agent migrates to an agent platform, the platform inserts the ID of the agent in its location cache in Line 2. The RegionMaster also updates the new location of the mobile agent in Line 3. Finally, the plaform where the agent came from deletes the entry that stores the location of the mobile agent.

Pseudo code 8 Updating location information

```
1: procedure LOCATIONUPDATE(agent)
2:     updateCurrentPlatform(agent.address)
3:     updateRegionMaster(agent.address)
4:     acknowledgeLastPlatform(agent.address)
5: end procedure
```

The process of searching for the location of an agent is illustrated in Pseudo code 9. A mobile agent looks up the address of an other agent by first querying the current platform in Line 2. If the platform has no information about the agent then the query will be sent to the RegionMaster in Line 6. This platform can look for address of any mobile agent.

Pseudo code 9 Querying location information

```
1: function LOCATIONQUERY(agent)
2:     address ← queryCurrentPlatform(agent)
3:     if found then
4:         return(address)
5:     else
6:         return queryRegionMaster(agent)
7:     end if
8: end function
```

8.5 Communicator

8.5.1 Classes

Communicator is encapsulated in the Java package $agent.communicator.jar$. Figure 8.3 represents a simplified class diagram for Communicator.

- ACLMessageCoder is responsible for encoding and decoding ACL message content.
- MessageReceiver listens on a TCP port for incoming byte stream.
- Envelope is used for managing ACL message envelope.
- EnvelopeCoder encodes and decodes ACL message envelope.
- ACLMessage is the class for manipulating ACL messages.
- TransportMessage serializes agent messages into byte stream to be transferred over networks.
- TCPConnection handles a connection between a sender platform and a receiver platform.
- MessageTransportProtocol is a simplified transfer protocol that utilizes a TCPConnection for delivering byte stream representing agent messages over networks.

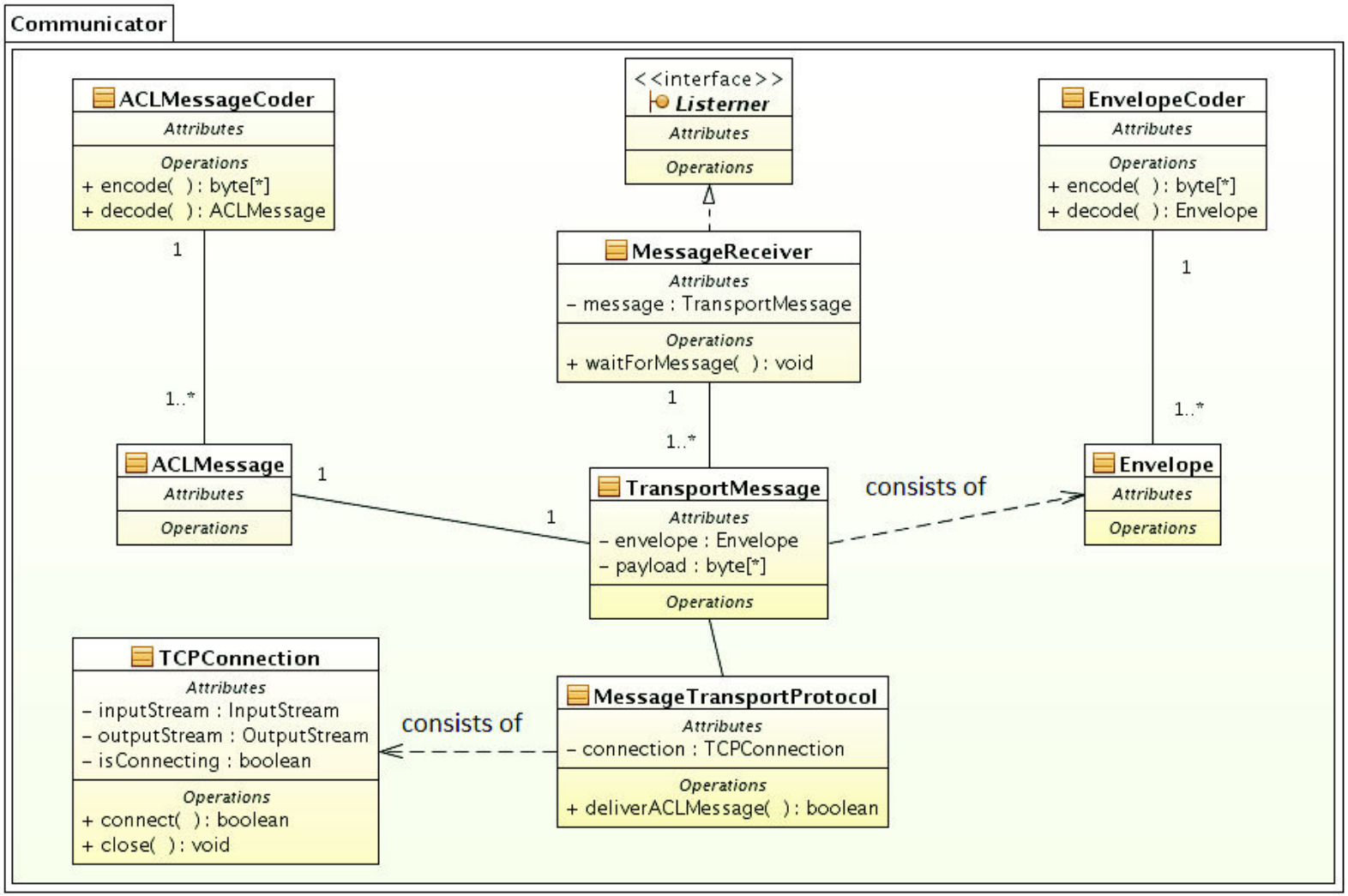

Figure 8.3: A simplified class diagram for Communicator

8.5.2 Realization

There are already fully implemented libraries for representing, encoding and decoding ACL message content as well as for encoding and decoding ACL message envelope. We adopt libraries XMLCodec, String ACL Message Coding and Bit-efficent ACL Message Coding to our implementation. The libraries are utilized under the GNU Lesser General Public License (GNU99), they can be found at (TIL12). In addition, we implement a simplified message transport protocol (MTP) for transferring agent messages.

The process of delivering an ACL message is illustrated in Pseudo code 10. The ACL message consisting of an envelope and a payload, they are encoded (Line 4 and Line 5) and then serialized into a byte stream (Line 6). A TCP socket connection is established between the sending and the receiving platforms. Finally, MTP delivers the byte stream to the destination as in Line 7.

Pseudo code 10 ACL message coding and transferring

```
1: procedure ACLMessageTransferring(destination,ACLMessage)
2:     envelope ← ACLMessage.getEnvelope()
3:     payload ← ACLMessage.getPayload()
4:     encodedEnvelope ← EnvelopeCoder.encode(envelope)
5:     encodedPayload ← ACLMessageCoder.encode(payload,BitEffACLCodec)
6:     byteStream ← serializeByteStream(encodedEnvelope,encodedPayload)
7:     mtp.deliverACLMessage(destination,byteStream)
8: end procedure
```

Pseudo code 11 illustrates the reverse process of message delivering. It shows the process of receiving and decoding an ACL message from a byte stream received by MTP. In Line 2 and Line 3, the encoded enveloped and the encoded payload are extracted from the byte stream. Each of them is then decoded to the original format as in Line 4 and Line 5. Finally, an ACL message is created from the envelope and the payload in Line 6.

Pseudo code 11 ACL message receiving and decoding

```
1: function ACLMessageReceiving(byteStream)
2:     encodedEnvelope ← getEncodedEnvelope(byteStream)
3:     encodedPayload ← getEncodedPayload(byteStream)
4:     envelope ← EnvelopeCoder.decode(encodedEnvelope)
5:     payload ← ACLMessageCoder.decode(encodedPayload,BitEffACLCodec)
6:     return new ACLMessage(envelope,payload)
7: end function
```

8.6 Scout Agents

A scout agent is built based on a standard agent supported by Ellipsis. It works as an information provider for NetworkMonitor by collecting necessary information about network status and platforms. The core function of a scout agent deals with the event happening when the scout agent arrives at an agent platform on its itinerary. This function is explained in Pseudo code 12.

Pseudo code 12 The behaviours of a scout agent on its arrival

```
1: procedure onScoutArrival
2:     measureAverageLatency(thisPlatform,allOtherPlatforms)
3:     updatePlatformInformation()
4:     if arrive at the RegionMaster then
5:         Count(split)
6:         if a quorum is reached then
7:             sendPlatformListToOrganizer()
8:         else
9:             nextnode ← getNextNodeOnItinerary()
10:            migrate(nextnode)
11:        end if
12:    else
13:        nextnode ← getNextNodeOnItinerary()
14:        migrate(nextnode)
15:    end if
16: end procedure
```

On its arrival, a scout agent measures the average latency and updates the platform's information as in Line 2 and Line 3. The agent checks the type of the platform in Line 4. If it is a normal platform then the scout migrates to the next platform on its itinerary in Line 10. Otherwise, if it finishes one round of exploration and comes back to the RegionMaster, it reads the information collected in the list and counts the number of platforms that set $split = true$ in Line 5. If a quorum is reached, it sends the list of platforms to Organizer which in turn re-organizes the region in Line 7. If not, the scout agent continues with a new round of surveillance as in Line 14.

Part III

Evaluation

Chapter 9

Evaluation of Functionality

The software prototype presented in Chapter 8 is the realization of the proposals in Chapter 6. The implementation of the prototype is built based on the design in Chapter 7. To validate the proposed thesis, it is necessary to evaluate the software prototype in accordance with the criteria mentioned in Chapter 5. For this purpose, different evaluation tests are going to be conducted to examine the practicality and efficiency of the approach as well as to assess system qualities.

Testing strategies have a significant impact on the efficiency of the evaluation process. To evaluate such type of software, the most common sense approach should be conducting evaluation in a real network scenario where network dynamics are naturally present. However, due to limitations of resources, building a testbed resembling a real scenario is an arduous task. In addition, maintaining and operating this type of system require a lot of personnel. Therefore, the evaluation will be conducted as simulation in a laboratory scale where conditions of a real network will be imitated, without a real scenario being present. To construct such a test environment, characteristics of a dynamic network are going to be simulated with assisstance of other softwares. The laboratory scale tests are expected to simulate probable occurrences and encompass a wide range of possibilities of execution in real dynamic networks. The test's outcomes will then be used to evaluate functionality as well as system qualities.

In this chapter we are going to evaluate the software functionality. The aim is to examine whether the software framework is able to fulfil the essential requirements regarding feasibility, being self-adaptive, and self-organizing as expected.

9.1 System Overview

We deploy a laboratory scale test network with the presence of eight computers connected through a local Gigabit Ethernet LAN 1000 Mbps network. Each computer corresponds to an agent platform and is equipped with the software framework as well as other essential softwares. The system configuration is specified in Table 9.1. The agent platforms representing by their alias are going to be used throughout the succeeding test scenarios.

Computer	Alias	OS	Kernel	RAM	Processor
Desktop	D1	Fedora 12	2.6.31	2.0 GB	AMD Athlon 2.2 GHz
Desktop	D2	Debian 6.0.4	2.6.32	4.0 GB	Intel Xeon 2*2.0 GHz
Desktop	D3	Debian 6.0.4	2.6.32	4.0 GB	Intel Xeon 2*2.0 GHz
Desktop	D4	Debian 6.0.4	2.6.32	4.0 GB	Intel Xeon 2*2.0 GHz
Fujitsu	FJ	Fedora 12	2.6.31	3.0 GB	Intel 2*2.8 GHz
Portégé	PT	Windows 7	N/A	4.0 GB	Intel 2*2.4 GHz
IBM Thinkpad	T1	Fedora 12	2.6.31	2.4 GB	Intel 2*2.4 GHz
IBM Thinkpad	T2	Windows XP	N/A	1.0 GB	Intel 1.7 GHz

Table 9.1: Hardware configuration of the experiments

9.2 Test Scenario 1: The Feasibility of Network Monitoring

9.2.1 Preamble

Network monitoring supplies the software components with up-to-date information about network connectivity as well as agent platform's status. It accounts for an integral part of the software prototype. Factors pertaining to the efficiency of network monitoring contribute to the benefits of the software framework.

In a dynamic network environment the cost for monitoring network might be considerably high. Monitoring activities may place a burden on the network traffic, thereby reducing overall system's performance. The implementation is beni-

ficial only if network monitoring gains a good performance whilst maintaining a reasonable running cost. The issues need to be examined throughtfully.

This test scenario aims to evaluate the practicality as well as the efficiency of the network monitoring functionality. In the first place, it is necessary to investigate whether the approach is applicable to fulfil expected requirements, i.e. monitoring network and supplying information. Afterwards, metrics related to cost of the monitoring activities are going to be measured to verify the feasibility of the approach.

9.2.2 Experiment Setup

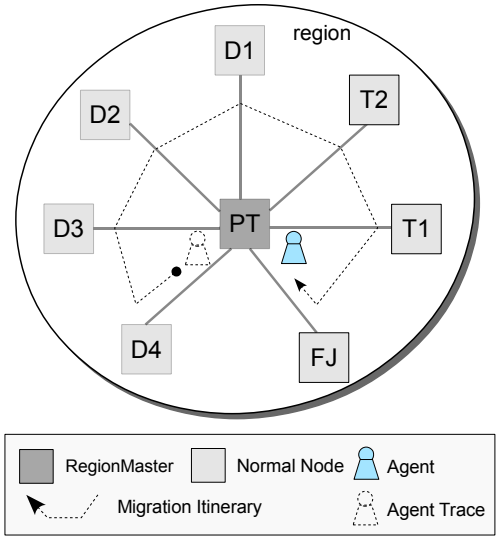

Figure 9.1: Network layout for the practicality test

Figure 9.1 shows the logical connection for the first experiment. The scenario simulates a region with $n = 8$ agent platforms. One scout agent is created by the RegionMaster PT. The RegionMaster feeds the scout agent with the list of agent platforms $\{D1, D2, D3, D4, FJ, PT, T1, T2\}$. The scout agent is deployed around the region to gather information. One round of exploration happens when the agent starts at the RegionMaster, travels through all nodes of the region, performs its routine and migrates back to the RegionMaster. A round consists of $h = 8$ hops.

9.2.3 Metrics

The following parameters are going to be measured:

- The time needed for the scout agent to accomplish its tasks at a platform: $t_{processing}$.
- The time needed for the agent to perform one round of exploration: $t_{exploration}$.

The processing time at an agent platform is the duration from when the agent arrives until it completes its tasks and leaves for the next node. We presume that the time delay for creating and processing a message is small and can be ignored. As a result, the value is calculated as follows:

$$t_{processing} = t_l - t_a = t_{RTT} + t_d + t_s \tag{9.1}$$

in which

- t_a is the time when the scout arrives at a platform.
- t_l is the time when it leaves for the next node of its itinerary.
- t_{RTT} is the period of time from the first ping message sent until the last response received.
- t_d is the time to deserialize a scout agent from an incoming stream of byte.
- t_s is the time to serialize a scout agent into a byte stream to be sent over the network.

By measuring the processing time, we examine how fast the scout agent performs its tasks at every platform.

The exploration time is given by:

$$t_{exploration} = t_{end} - t_{begin} = \sum_{i=1}^{n} t_{processing} + \sum_{j=1}^{h} t_{migration} \tag{9.2}$$

where t_{begin} and t_{end} are the time when a scout starts and completes one round of exploration, respectively; $\sum t_{migration}$ is the total time that the agent migrates through all the platforms.

9.2.4 Experimental Results

The scout agent is deployed to visit the platforms one by one. At each agent platform the duration from the agent arrives until it finishes its work and leaves for the next platform is measured. This aims to analyse the processing speed of the scout agent at a platform. By every platform, the measurement is done for 80 times. Figure 9.2 displays the experimental results for $t_{processing}$ of the platforms. The measurement results are shown in more detail in Appendix B.

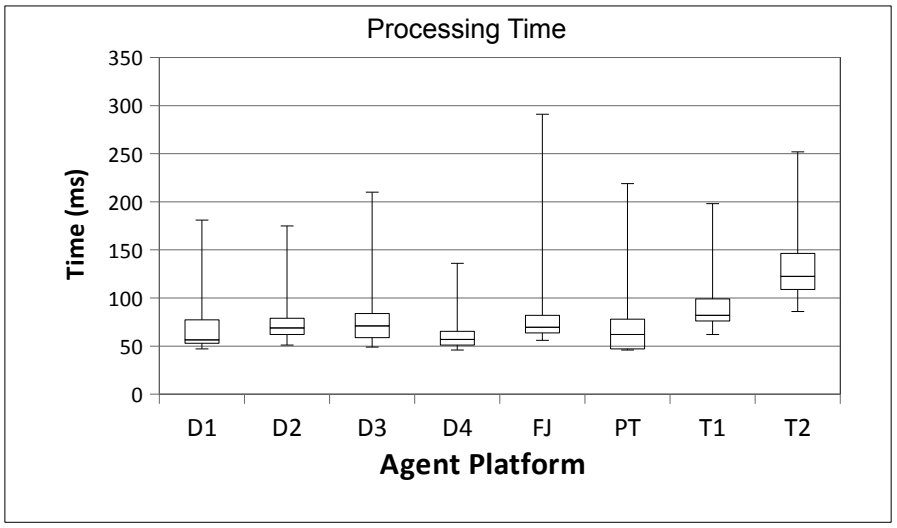

Figure 9.2: Processing time

The processing time represents the time that the agent needs to perform its tasks at a platform following Equation 9.1. It is dependent on the processing power of agent platforms as well as the latencies to the other platforms, which are in turn dependent on the network connection speed. It can be seen that, except platform $T2$ that has a higher processing time because of its limited processing power (as specified in Table 9.1), the processing time for the other platforms is considerably low. It guarantees that the processing activities place comparatively little burden on the system performance.

To measure the average exploration time $t_{exploration}$, the scout agent is sent around the network for a specific number of rounds r. Figure 9.3 presents the measurement results for $r = \{100, 200, 300, 500, 1000, 2000, 3000, 5000\}$ rounds. According to Equation 9.2 the exploration time is the sum of the processing time at

all agent platforms and the migration time. This parameter demonstrates how fast the scout agent can supply the RegionMaster with up-to-date information about the network. If the agent needs a long time to perform its tasks at the platforms or to hop from platform to platform, the information submitted to the RegionMaster might be out-of-date. As a consequence, the reactions produced by RegionMaster would not be adequate.

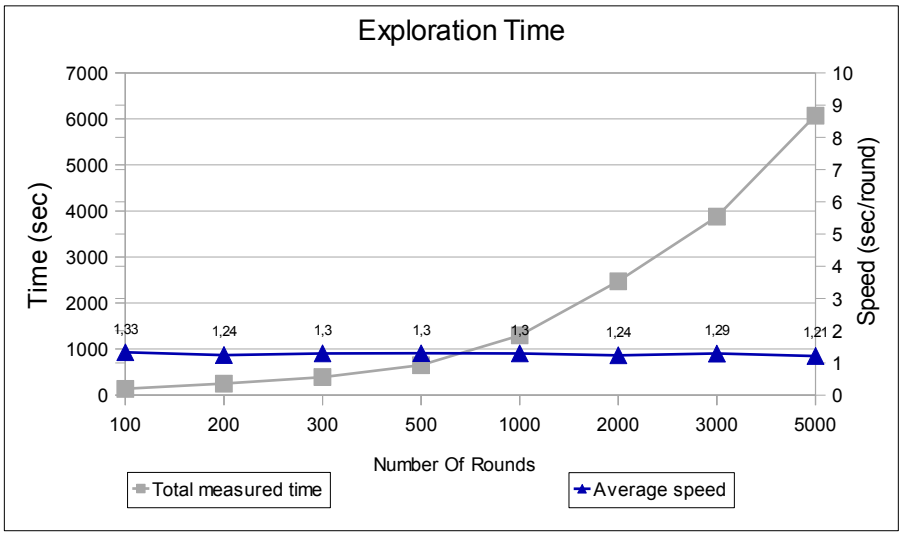

Figure 9.3: Exploration time

However, the measurement results show that this is not the case. In the diagram in Figure 9.3, the curve represents the accumulated exploration time. The straight line which depicts the average time for finishing one round of surveillance (second/round) provides evidence that the parameter is stable, no matter how many rounds the agent has migrated. This indicates that the scout agent produces no overhead when it works in the long run.

The experimental results imply that the scout agent requires a reasonable processing time. In addition, it has an acceptable migration speed so that it can finish one round of exploration in considerable short time, thereby supplying the RegionMaster with informative network's status. This test scenario confirms that the monitoring activities are operationally feasible in the given context. The framework provides a practicable means for monitoring network. From our viewpoint, the software framework meets the feasibility requirement.

9.3 Test Scenario 2: Being Self-Adaptive

9.3.1 Preamble

One of the main objectives of the software prototype is to help a system working in dynamic networks react adequately to environmental stimuli. In a region the RegionMaster plays a central role and any deterioration in its quality may have a negative impact on the overall system's performance. Given that the RegionMaster suffers a degradation that makes it no longer suitable to accomplish the tasks of a central node, it is necessary to find a substitution for it. Being self-adaptive is an important feature of the software framework, the platform colony should be able to take appropriate measures to counteract adverse effects happening to the RegionMaster.

9.3.2 Experiment Setup

The logical network layout for this experiment is depicted in Figure 9.4. A scout agent is deployed to survey the region.

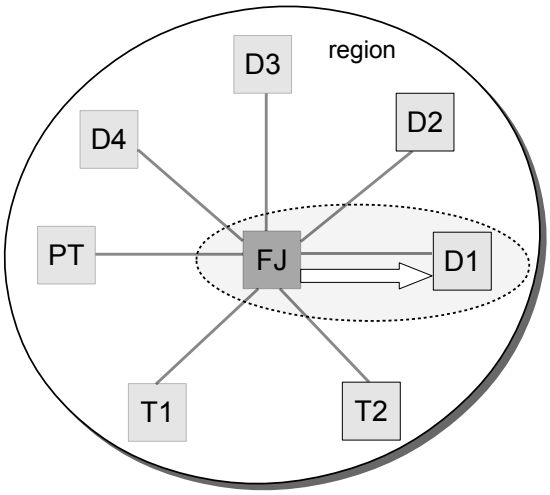

Figure 9.4: Network layout for the self-adaptive test

In this test scenario, the connectivity between the RegionMaster and the remaining platforms is going to be degraded using software. This aims to investi-

gate the countermeasures of the system given that the quality of the connection to the RegionMaster has declined. Since the RegionMaster is responsible the management of the whole region, the deterioration adversely affects system performance, processing speed may considerably slow down. Given the circumstances, it is expected that the platform colony is able to re-organize and recover from the degradation.

In this experiment, simulation of the occurrence of connectivity degradation is required. A certain network traffic between the RegionMaster and the other platforms will be created, resulting in a smaller bandwidth left to the remaining platforms. For the purpose of creating network traffic, we utilize the open source software Iperf (NLA12). This tool is used for measuring throughtput and performance of a network. It can also be used to produce both TCP and UDP data streams over networks; data sent by the client will be received and eventually discarded by the server.

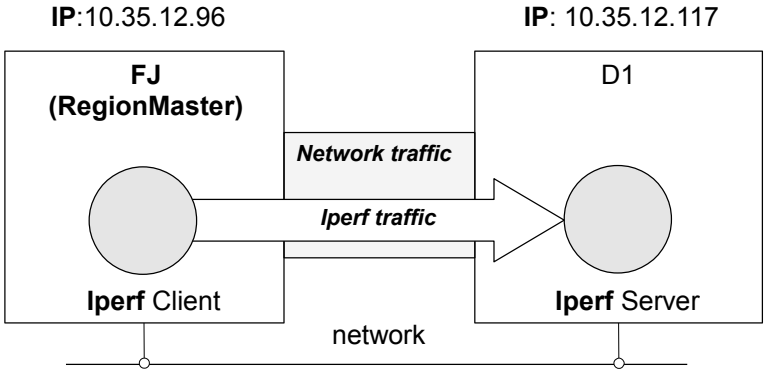

Figure 9.5: The employment of the traffic generator Iperf

Figure 9.5 is an expansion of the gray ellipse depicted in Figure 9.4. The RegionMaster plays the role of an Iperf client and agent platform $D1$ is the Iperf server. This figure illustrates how Iperf is employed to generate network traffic between the RegionMaster FJ and $D1$. The bandwidth created by Iperf will occupy a certain amount of the whole bandwidth. As a result, the bandwidth available to other applications will be decreased proportionally.

The command in Listing 9.1 is executed at the Iperf server to instruct it to listen for incoming traffic on TCP port 5001.

Listing 9.1: Iperf command at the server side

```
$ iperf -s
```

where

- $iperf$ is the command name.
- $-s$ stands for ”server”.

Listing 9.2 shows the command at the client side and its corresponding effect. This command causes the client to connect to the Iperf server IP address $10.35.12.117$ on port 5001 and send data to the server for a duration of 5 seconds.

Listing 9.2: Iperf command at the client side

```
$ iperf -c 10.35.12.117 -i 1 -t 5
------------------------------------------------------------
Client connecting to 10.35.12.117, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 10.35.12.96 port 44162 connected with 10.35.12.117 port
   5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   108 MBytes   902 Mbits/sec
[  3]  1.0- 2.0 sec   109 MBytes   914 Mbits/sec
[  3]  2.0- 3.0 sec   109 MBytes   915 Mbits/sec
[  3]  3.0- 4.0 sec   108 MBytes   908 Mbits/sec
[  3]  4.0- 5.0 sec   109 MBytes   918 Mbits/sec
[  3]  0.0- 5.0 sec   543 MBytes   911 Mbits/sec
```

where:

- $iperf$ is the command name.
- $-c$ indicates client, $10.35.12.117$ is the IP address of the Iperf server.
- $-i$ is the interval that Iperf prints out the metrics.
- $-t$ defines the time in seconds that the test executes.

Figure 9.6 shows the traffic and bandwidth produced by Iperf for different time durations $d = \{10, 20, 30, 40, 50, 60\}$ seconds.

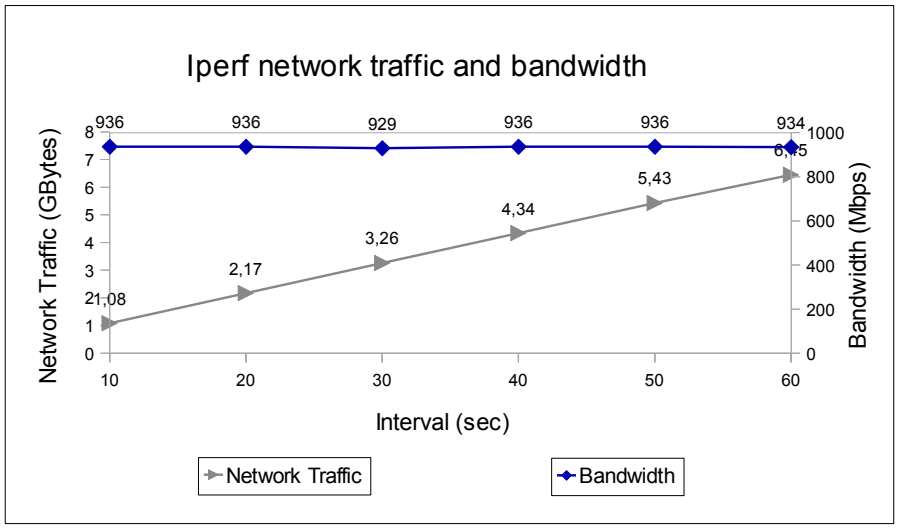

Figure 9.6: Network traffic and bandwidth produced by Iperf

9.3.3 Experimental Results

Since Iperf consumes a certain bandwidth on the connection between the RegionMaster and the rest of the region, there is smaller bandwidth left for other applications. As a result, each agent platform will experience effects from the traffic generator. The latencies between the platforms and the RegionMaster grow sharply. These changes are sensed by the scout agent every time it visits the platforms.

Every platform sets the value $split$ based on the ratio τ_{avg}/τ_{RM}; where τ_{avg} is the average latency and τ_{RM} is the latency to the RegionMaster, respectively. In this experiment the splitting threshold is set to $\tau_{avg}/\tau_{RM} < 50\%$. That means:

$$split = \begin{cases} \text{true, if } \tau_{avg}/\tau_{RM} < 50\% \\ \text{false, otherwise} \end{cases} \tag{9.3}$$

In a real network scenario this parameter, however, needs to be fitted with environmental characteristics. Table 9.2 shows the ratio and the corresponding value $split$ for every platform. At the end of a round of exploration, the scout agent goes back to the RegionMaster and submits information it collected. Based on this information, the RegionMaster makes a decision. Since most of the platforms set the value $split$ to $true$ the region is splitted.

Platform	D1	**D2**	D3	D4	FJ	PT	T1	T2
τ_{avg}/τ_{RM} **(%)**	47,8	47,6	37,5	27,5	—	37,5	44,4	29,7
Split	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*true*

Table 9.2: Metrics measured at the time of self-organizing

The new region structure is depicted in the central part of Figure 9.7. There is only one platform staying at the old region, this is the RegionMaster. The remaining nodes join the new region with $D2$ promoted to be the new RegionMaster. In this case, an adaptation has been made, the old RegionMaster relinquishes its leadership in the only-one platform region and becomes an inferior node of the new region. Eventually, a new region emerges from the original region. The layout of the new region is depicted at the right-most part of Figure 9.7.

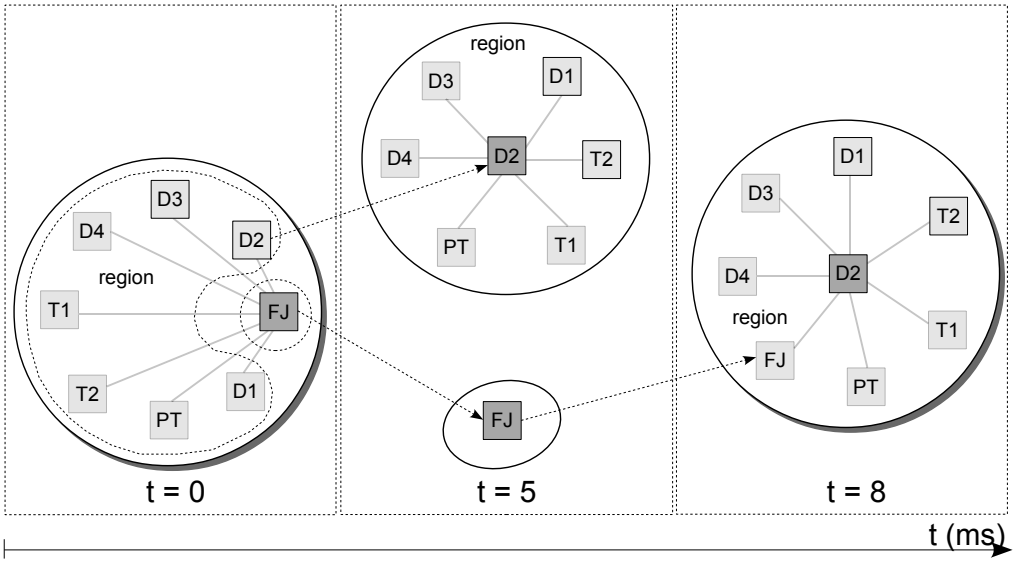

Figure 9.7: Exchange in role of the command node

Figure 9.8 displays the latency to the RegionMaster τ_{RM} of every platform in three stages. For a platform, the left column is the latency before Iperf is activated; the middle column represents the time while Iperf is operating and the right column is the latency of the platform after the self-adapting process has occurred.

In the beginning and while Iperf was working FJ was the RegionMaster so

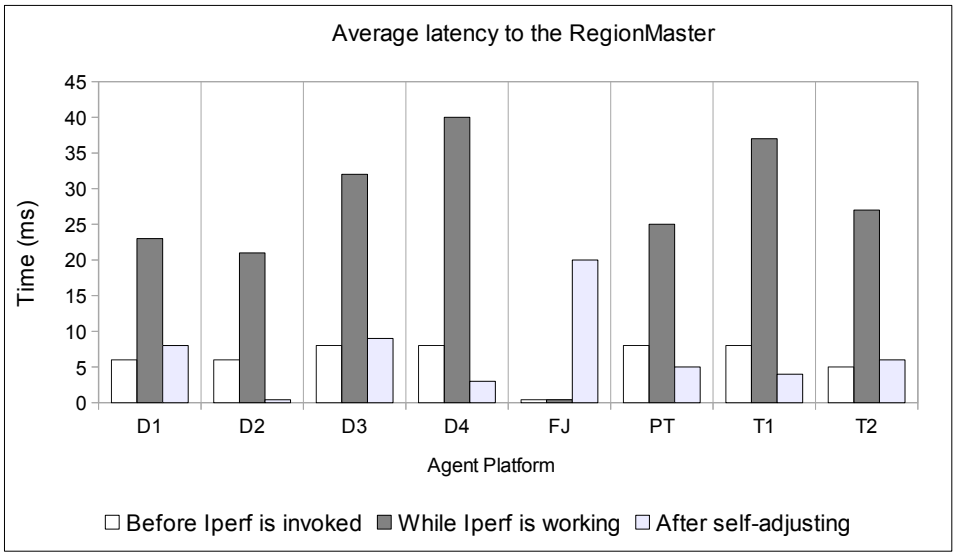

Figure 9.8: Latencies to the RegionMaster in different points in time

$\tau_{RM}(FJ) = 0$. Similarly, after $D2$ has taken its job as the RegionMaster $\tau_{RM}(D2) = 0$. As usual expected, while Iperf is being executed, the latencies to the RegionMaster measured for every platform increase significantly. However, once $D2$ takes over as the RegionMaster, the latencies decrease proportionally.

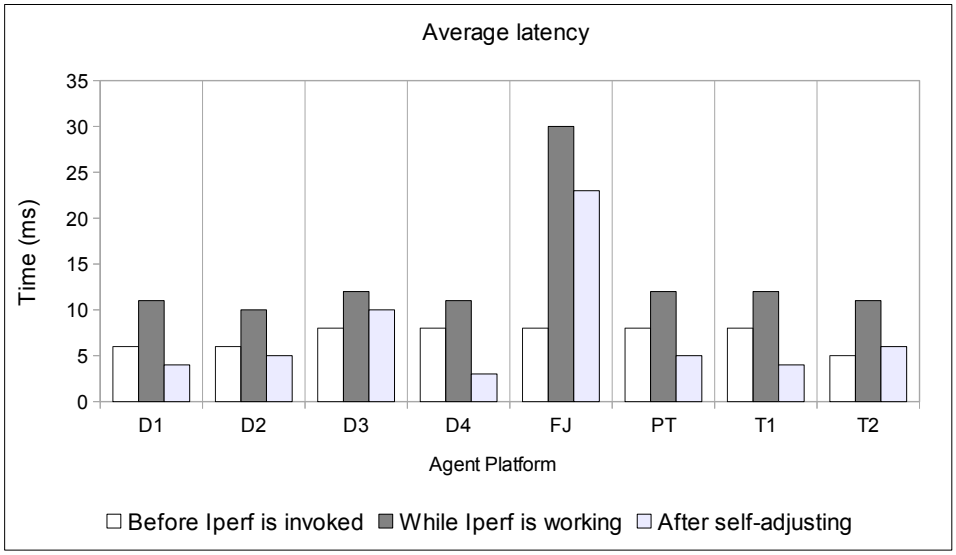

Figure 9.9: Average latencies in different points in time

Figure 9.9 shows the average latencies τ_{avg} of every platform in three stages: Before Iperf is executed, while Iperf is being executed and after the adaptation has been made. These latencies also shift in the same pattern as by τ_{RM}. Before additional bandwidth was produced, the average latencies had been at a normal level. While Iperf was operating the latencies rose markedly. After the region has been restructured, these values resume to a normal level. It can be seen that the network monitoring activities supply the framework with up-to-date information about network situation, thereby facilitating the decision making process.

The metrics measured after the splitting took place are shown in Table 9.3. The ratio τ_{avg}/τ_{RM} of every platform has increased and is well above the threshold 50%; all the platforms set $split = false$. The metrics suggest that the swap in role of the RegionMaster from FJ to $D2$ brings a more stable connectivity to every platform compared to that of the old arrangement, right after Iperf started producing bandwidth.

Platform	D1	**D2**	D3	D4	FJ	PT	T1	T2
τ_{avg}/τ_{RM} **(%)**	150	—	133	83,3	83,3	100	88,9	83,3
Split	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*

Table 9.3: Metrics measured after the adaptation has been made

This test scenario demonstrates that the framework is able to detect degradations in connectivity of the RegionMaster once they occurred. Based on the information gathered by the scout agent, the framework provides the system with a measure to adequately overcome the problem that adversely affects the RegionMaster. As shown in the experimental results, the countermeasures appear to be effective since they help the platform colony to promote a new equilibrium in connectivity between the platforms. The connection qualities from a platform to the RegionMaster as well as from a platform to the others have been improved. From our perspective, this test validates that the software framework principally fulfils the self-apdative requirement.

9.4 Test Scenario 3: Self-Organizing

9.4.1 Preamble

In dynamic networks of the SpeedUp-Type, it is a common occurrence that a group of agent platforms leaves other platforms of the same region and swarms onto a new location, causing the connectivity to change correspondingly. Under the circumstances, maintaining a fixed logical connection between the abandoning platforms and the stationary platforms would not be an optimal solution. We are convinced that it is more sensible to re-organize the colony to adapt to the new structure. The abandoning platforms should form a new region while the stationary platforms group into a second region. The ability of a platform colony to re-organize given that the correlation between the members has changed, is essential. A reasonable re-organizing manoeuvre helps facilitate a harmonious correlation between the platforms, thereby balancing network load and saving processing time.

9.4.2 Bandwidth Shaping

In Test Scenario 2, Iperf has been used to generate traffic over networks. The created traffic occupies a portion of the whole bandwidth. It is not possible to use Iperf to limit bandwidth for a specific service in a server whilst leaving other services unaffected. Such a task, however, can be performed by traffic shaping.

In network applications, a service is identified by the combination of an IP address and a network port. A traffic shaper controls network bandwidth by delaying traffic to meet pre-defined requirements. A traffic can be precisely allotted to a service by referring to the service's identification. By traffic shaping, two important factors for a service are determined: a minimum usable bandwidth ($MiUB$) and a maximum usable bandwidth ($MxUB$). The $MiUB$ is the guaranteed bandwidth that the service can consume. The $MxUB$ is the bandwith that the service can reach as long as there is free bandwidth available (BHea05).

9.4.3 Experiment Setup

The logical network layout for this test scenario is depicted in Figure 9.10. In the beginning, platform $D2$ with the IP address $10.35.14.185$ is configured as the RegionMaster. A scout agent is deployed to survey the region.

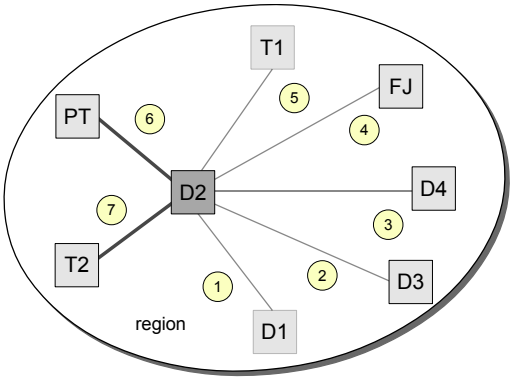

Figure 9.10: The network layout and bandwidth shaping imposed on ping services

The traffic shaper which is comprised of the software packages tc (traffic control), $iptables$ and $iproute$ in Linux distributions provides us with a convenient way to throttle bandwidth. In this test scenario, this traffic shaper is used to regulate the traffic of the ping messages exchanging service between the RegionMaster and the platforms $\{D1, FJ, PT, T1, T2\}$ on TCP port 3242. This aims to simulate the occurrence when the platforms, together, move far away from the RegionMaster and swarm onto a new location, resulting in a decrease in the corresponding bandwidths. It is expected that the platforms with a large latency to the RegionMaster will group to form a new region.

Listing 9.3: Parameters for bandwidth shaping

```
Destination IP Address: 10.35.14.185
Destination Port: 3242
Rate: MiUB
Ceil: MxUB
```

Listing 9.3 presents the parameters and values assigned in an executable script at every platform of the set $\{D1, FJ, PT, T1, T2\}$ to shape the outgoing traffic to

the RegionMaster. A full script for bandwidth shaping is given in Appendix A.

Table 9.4 lists the corresponding values $MiUB$ and $MxUB$ for every connection representing by the circled numbers depicted in Figure 9.10. Only the connections from $\{PT, T2\}$ to the RegionMaster $D2$ are allocated the maximum bandwidth, i.e. 1000 Mbps.

Connection	(1)	(2)	(3)	(4)	(5)	(6)	(7)
$MiUB$ (Mbps)	0,512	1	2	20	54	1000	1000
$MxUB$ (Mbps)	1	2	3	30	60	1000	1000

Table 9.4: Bandwidth allotted to the ping service of the platforms to the RegionMaster

It should be noted that the change in bandwidth defined in Table 9.4 is exclusive to the connections on TCP port 3242 from the platforms to the RegionMaster, IP address $10.35.14.185$. Other services whose bandwidth is not throttled can use the normal bandwidth, i.e. 1000 Mbps.

9.4.4 Experimental Results

While bandwidth shaping is being activated, there are changes in network connectivity within the region. Table 9.5 displays the metrics recorded by the scout agent before the splitting occurs.

Platform	D1	**D2**	D3	D4	FJ	PT	T1	T2
τ_{avg}/τ_{RM} **(%)**	14,8	—	16,8	15,7	38,9	75,0	29,4	120,0
Split	*true*	*false*	*true*	*true*	*true*	*false*	*true*	*false*

Table 9.5: Metrics measured before the self-organizing takes place

The platform colony re-organizes to react to the changes in network bandwidth. Since more than a half of the agent plaforms set $split = true$, the region is bisected. Two regions emerge from the original region. Figure 9.11 depicts the splitting process, those platforms with a ratio $\tau_{avg}/\tau_{RM} \geq 50\%$ namely $\{D2, PT, T2\}$ group into Region 1; Region 2 is made of the platforms with $\tau_{avg}/\tau_{RM} < 50\%$ i.e. $\{D1, D3, D4, FJ, T1\}$ and $D1$ is appointed to the RegionMaster.

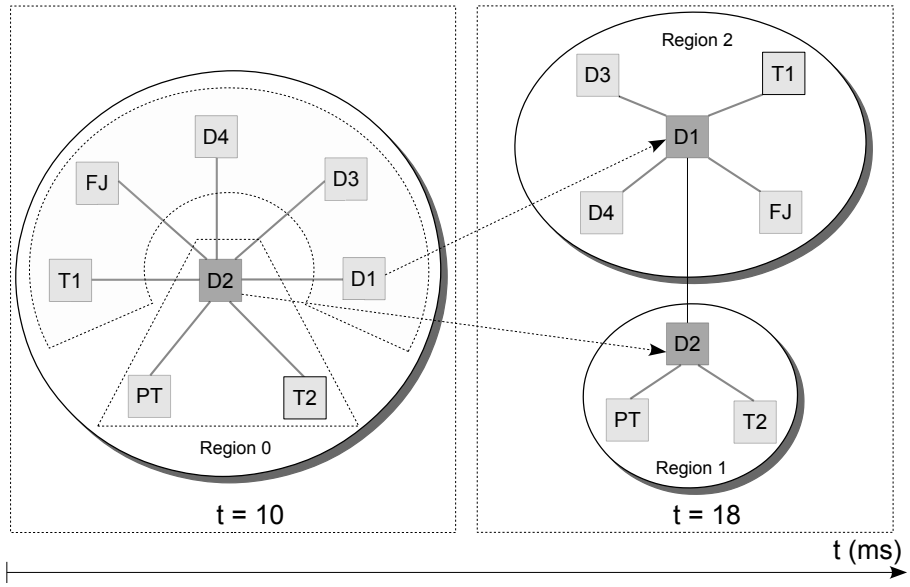

Figure 9.11: The self-organizing process of the platform colony

Table 9.6 and Table 9.7 display the metrics measured for Region 1 and Region 2 after splitting, respectively. According to the metrics shown in the tables, both Region 1 and Region 2 benefit from the splitting, the ratio τ_{avg}/τ_{RM} grows markedly and exceeds the threshold 50%. As a result, all the platforms reset $split$ to $false$.

Platform	**D2**	PT	T2
τ_{avg}/τ_{RM} **(%)**	—	75,0	83,3
Split	*false*	*false*	*false*

Table 9.6: Metrics measured after splitting in Region 1

Platform	**D1**	D3	D4	FJ	T1
τ_{avg}/τ_{RM} **(%)**	—	75,0	100	87,5	83,3
Split	*false*	*false*	*false*	*false*	*false*

Table 9.7: Metrics measured after splitting in Region 2

The average latencies measured for every platform are shown in Figure 9.12.

For each agent platform, the first column represents the average latency before the bandwidth for ping service is throttled as specified in Table 9.4. The second column is the average latency measured while bandwidth shaping is working. The third column is the average latency for the corresponding platform after the self-organization has been conducted. Since the connection between $D1$ and the RegionMaster $D2$ has a small bandwidth i.e. $MiUB = 512Kbps$ and $MxUB = 1Mpbs$, the average latency during bandwidth shaping increases considerably compared to that when bandwith was not yet throttled. However, after the self-organizing has taken place, the corresponding latency resumes to a normal state. The latencies of the remaining platforms change likewise.

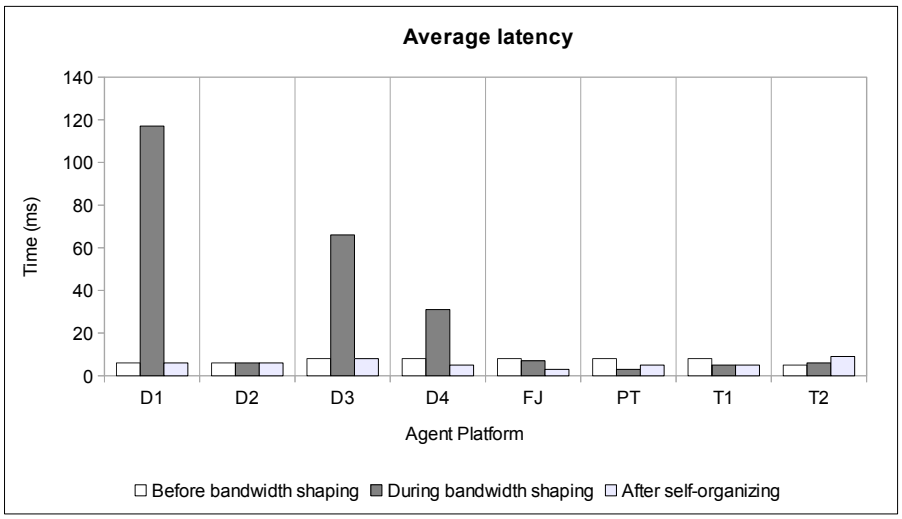

Figure 9.12: Average latencies

The measured metrics show that after the network connectivities degrade, the self-organizing process helps the platform colony improve connection quality significantly. This test scenario along with the test presented in Section 9.3 supports the hypothesis that the software framework is able to assist the system to react appropriately to changes happen in the surrounding environment; the deployment of the honey bee inspired mechanism in the network brings benefit to the platform colony. In our judgement, the features demonstrate that the software framework complies with one of the essential requirements that we have been aiming for: the ability to self-organize in a changing environment.

Chapter 10

Evaluation of System Quality

This chapter presents the experimental results for the evaluation of system qualities regarding fault tolerance and reliability. Fault tolerance and reliability are the key facets of the software prototype. Fault tolerance means the whole system is able to operate even some of the member agent platforms disconnect. It avoids a single point of failure and prevents defects from spreading out. In the worst case, when the disconnection escalates, the system is able to degrade gracefully. Reliability means the location of a mobile agent is always found, the sender agent knows where the receiver agent stays, even when the RegionMaster has failed. However, reliability does not mean that an ACL message is always able to reach its target. An ACL message can be delivered to its recipient only if there exists a connection between the sender and the receiver.

10.1 Test Scenario 4: Fault Tolerance in Network Management

10.1.1 Preamble

Fault tolerance accounts for a crucial part of the overall reliability of systems working in dynamic networks. In a dynamic environment, every agent platform is susceptible to network failure and a master node is no exception. A region of agent platforms relies much on its RegionMaster to operate. A failure of the Region-

Master may lead to a breakdown of the whole region. The ability of the software framework to provide up-to-date information about the status of the incumbent master node is intrinsic to a good performance. To avoid a single point of failure, redundancy has been employed; a second platform is used to test the availability of the RegionMaster. Whenever the RegionMaster goes haywire, the observing platform can detect the failure and takes over as the RegionMaster, so that system's operation is not interrupted.

In this test scenario, we are going to investigate how efficient is the measure in achieving system tolerance.

10.1.2 Experiment Setup

Figure 10.1 depicts the logical network layout for this test scenario. There are eight agent platforms, i.e. $\{D1, D2, D3, D4, FJ, PT, T1, T2\}$. Initially platform $D2$ works as the RegionMaster and platform $D3$ is configured as the reserve RegionMaster. The reserve RegionMaster detects the presence of the RegionMaster by regularly sending inquiry messages. A scout agent is used to survey the region.

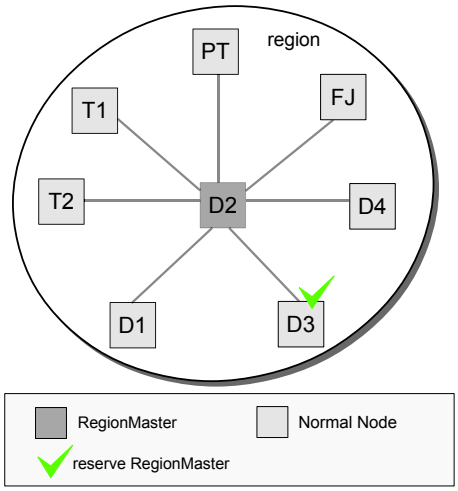

Figure 10.1: Network layout for the fault tolerance test

A failure of the RegionMaster is simulated by removing it from the network. This can be done either by turning off its JBoss AS or by blocking TCP port 2205 for exchanging inquiry messages with the reserve RegionMaster.

It is expected that the reserve RegionMaster will take over as the RegionMaster once it discovers that the platform is no longer available. The duration from when the RegionMaster is out of service until the reserve RegionMaster becomes the RegionMaster and deploys scout agents, thereby resuming the whole region to a normal state, will be measured.

10.1.3 Experimental Results

We investigate the reaction of the system by gradually shutting down the incumbent master node. There are four occurrences as follows:

Occurrence 1: The RegionMaster $D2$ is manually disconnected from the network by blocking TCP port 2205 as depicted on the left part of Figure 10.2. The reserve RegionMaster $D3$ takes over as the RegionMaster. Once $D3$ starts to work as a RegionMaster, it deploys a scout agent to survey the region and appoints $D4$ as the new reserve RegionMaster. Platform $D4$ starts to send inquiry messages to $D3$ to detect its presence. The new network layout is shown on the right part of Figure 10.2.

Occurrence 2: $D3$ is removed from the network, resulting in the promotion of $D4$ to the post of the master node as illustrated in Figure 10.3. $D4$ sends a scout agent to survey the region which now has six platforms, i.e. $\{D1, D4, FJ, PT, T1, T2\}$. It also assigns the task of a reserve RegionMaster to $D1$. The new network layout is depicted on the right part of Figure 10.3.

Occurrence 3: $D4$ is deactivated and $D1$ takes over as the RegionMaster. There are five platforms in the region $\{D1, FJ, PT, T1, T2\}$ as shown on the right part of Figure 10.4. $T1$ is selected as the new reserve RegionMaster.

Occurrence 4: Figure 10.5 depicts the last occurrence in this experiment. The RegionMaster $D1$ is removed from the network and $T1$ promotes itself to the new RegionMaster. Four platforms are now present in the region, they are $\{FJ, PT, T1, T2\}$.

The occurrences signify that the framework is able to help the whole system recover once failure has happened. Right after the incumbent master node fails, the reserve node undertakes its tasks, deploys a new scout agent, votes a reserve RegionMaster, thereby resuming region's normal routines.

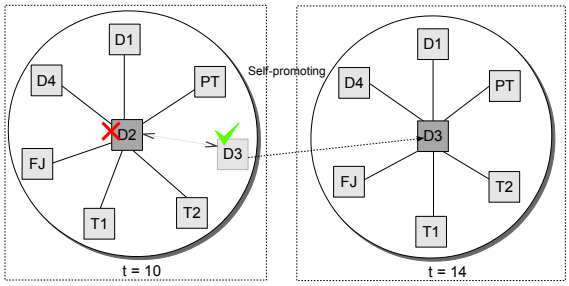

Figure 10.2: Occurrence 1: D2 stalls, D3 becomes the RM

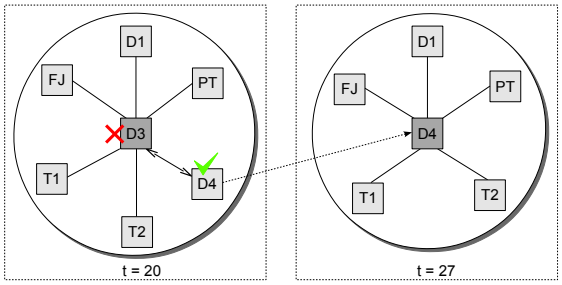

Figure 10.3: Occurrence 2: D3 stalls, D4 becomes the RM

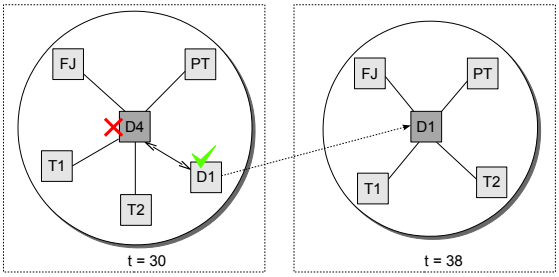

Figure 10.4: Occurrence 3: D4 stalls, D1 becomes the RM

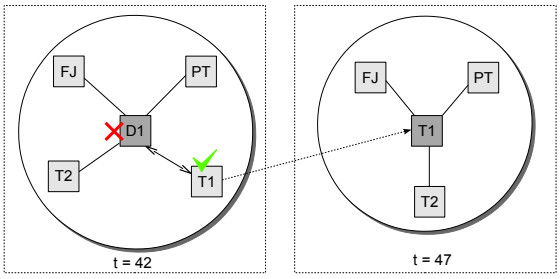

Figure 10.5: Occurrence 4: D1 stalls, T1 becomes the RM

To examine the efficiency of the measure, it is necessary to explore the correlation between the platforms once a change comes into effect. For an occurrence, the average latencies and the latencies to the RegionMaster after the takeover occurs are measured. To get a more reliable result, a time measurement test is executed 15 times for every platform.

In the beginning, there are eight agent platforms working and the region is in a normal state. The metrics measured for Occurrence 0 which corresponds to this initial state of the region can be considered as a basis for comparison for the succeeding occurrences.

Table 10.1 and Table 10.2 present the metrics for the platforms in all occurrences. Table 10.1 shows the latencies to the RegionMaster. In this table, the first column represents the mean value of the latencies to the RegionMaster t_{RM}, the second column is the standard deviation σ of the sampled data. A cell with the content "*" indicates that the corresponding platform is the RegionMaster and a cell with the content "—" means the platform is no longer available.

	Occurrence									
	(0)		(1)		(2)		(3)		(4)	
Platform	t_{RM}	σ	t_{RM}	σ	t_{RM}	σ	t_{RM}	σ	t_{RM}	σ
$D1$	7,71	2,37	4,85	2,03	6,57	3,15	*	*	—	—
$D2$	*	*	—	—	—	—	—	—	—	—
$D3$	8,21	2,22	*	*	—	—	—	—	—	—
$D4$	3,64	0,84	5,71	3,17	*	*	—	—	—	—
FJ	4,21	0,89	3,92	0,91	4,50	0,65	4,85	0,77	5,07	0,73
PT	3,57	1,08	4,00	0,78	6,57	3,25	2,64	0,74	4,14	0,53
$T1$	4,50	0,94	5,21	1,31	6,00	2,50	2,64	0,49	*	*
$T2$	4,71	0,91	5,92	1,54	7,00	3,18	4,07	1,63	4,64	0,49

Table 10.1: Latencies to the RegionMaster

The measurement results presented in Table 10.1 imply that by each occurrence, after a new master node takes up its job, every platform can gain a normal connectivity to the RegionMaster.

Table 10.2 displays the average latencies for the agent platforms. At a platform, the measurement is done likewise in 15 times. Like in Table 10.1, for an occurrence the first column is the mean value of the average latencies of a platform to the other platforms t_{avg}; the second column σ represents the standard deviation of the measured values. The content "—" signifies that the corresponding platform already failed.

	Occurrence									
	(0)		(1)		(2)		(3)		(4)	
Platform	t_{avg}	σ	t_{avg}	σ	t_{avg}	σ	t_{avg}	σ	t_{avg}	σ
$D1$	3,71	0,61	4,64	1,44	3,71	0,61	4,42	0,64	—	—
$D2$	9,50	2,95	—	—	—	—	—	—	—	—
$D3$	8,42	1,34	6,85	1,40	—	—	—	—	—	—
$D4$	2,64	0,63	5,00	1,61	8,92	1,63	—	—	—	—
FJ	3,57	0,75	4,64	0,84	3,42	0,64	4,64	0,49	3,50	0,75
PT	3,50	0,65	5,57	1,34	3,28	0,46	3,21	0,42	4,21	0,80
$T1$	4,21	0,70	4,07	0,61	3,78	0,42	3,42	0,51	3,57	0,51
$T2$	5,14	0,77	4,50	0,94	4,57	0,85	5,42	0,85	5,85	0,86

Table 10.2: Average latencies

The results shown in this table support the argument that after a new master node takes up its job, the region can reach a new equilibrium, the member platforms have a stable mutual connectivity.

The duration from the master node fails until the reserve node resumes region's operation is called the recovery time. Table 10.3 presents the recovery time measured for the occurrences.

Occurrence	(1)	(2)	(3)	(4)
RegionMaster	D2	D3	D4	D1
reserve RM	D3	D4	D1	T1
Recovery time (ms)	4	7	8	5

Table 10.3: Recovery time by the occurrences

For every occurrence the reserve RegionMaster can detect the disappearance of the incumbent RegionMaster and swiftly take up its job. This guarantees that the system's operation is not interrupted once failure has occurred. There is always a node that undertakes the network management tasks for the whole region. This attribute is particularly useful for agent platforms working in dynamic networks.

This test scenario shows that the software framework can assist the platform colony to maintain normal operation in an unstable network environment. Based on the occurrences as well as their corresponding metrics regarding connection quality, it is our view that the software framework generally conforms to the fault tolerance requirement.

10.2 Test Scenario 5: Reliability and Efficiency in Message Transferring

10.2.1 Preamble

It has been shown that the delivery of an ACL message from a mobile agent to another mobile agent consists of two phases: looking for the agent's location and sending the byte stream containing the message over networks. To find the targeted mobile agent, the platform hosting the sender agent queries the RegionMaster. In our approach, with the employment of redundancy a substitution for a RegionMaster can be made whenever it becomes inoperative as already demonstrated in Test Scenario 4. This guarantees that the location of a mobile agent is always found no matter whether the RegionMaster fails or not. Nevertheless, an ACL message can be delivered to its destination only if there exists a connection between the sender and the receiver platforms. That means, if either of the two platforms loses connection to the network then the transmission cannot be done.

As illustrated in Figure 7.19, before being transmissed, an ACL message is encoded in two levels: envelope encoding and ACL content encoding. The two parts are then serialized into a byte stream and the byte stream is transmissed over the network. At the recipient side, the two parts are then extracted from the received byte stream. They are then decoded to the original format. Finally,

the message is saved into the message cache of the recipient platform. Message coding and transferring are expected to have a reasonable running cost, thereby facilitating reliability in message transferring. To validate the suitability of the transmission model for working in dynamic networks, it is necessary to evaluate the performance of message coding and transferring.

10.2.2 Experiment Setup

The network layout for this experiment is illustrated in Figure 10.6. ACL messages are sent to random mobile agents. The message envelope of an ACL message is encoded and decoded using XMLCodec. The content of an ACL message is encoded and decoded using two ACL message coding techniques: String ACL Message Encoding (FIP02b) and Bit-efficient ACL Message Encoding (FIP02a). Since the same coding method is used for encoding message envelopes, we can use the measurement results to evaluate the performance of the two ACL message content coding techniques. Two independent tests are conducted to measure the performance of the two ACL message encoding techniques.

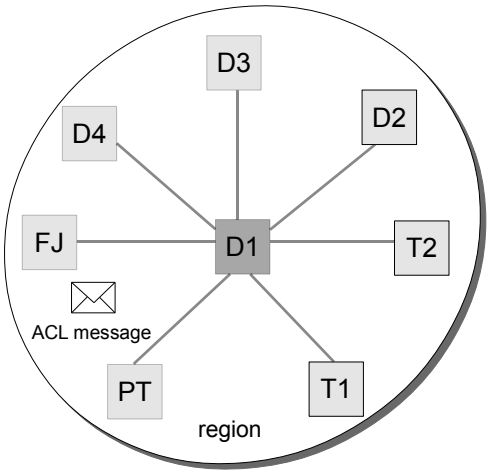

Figure 10.6: Network layout for transferring messages

To use as a basis for comparison, an additional test is conducted. A byte array stream (BAS) of the same size with the ACL message used in the two tests for evaluating message encoding techniques is sent from the same sender to the

same destination platform. We perform evaluation for different message sizes $s = \{10000, 20000, 50000, 100000, 200000\}$ bytes. For each message size category, the content of the message and that of the byte stream are identical. The time from when an ACL message is encapsulated at the sender, transmissed and until it is received at the recipient is measured. Similarly, the time for transferring BASs is also measured for comparison.

The time for transferring an ACL message is calculated as follows:

$$t_{ACLtransfer} = t_{location\_query} + t_{encode} + t_{transfer} + t_{decode} \tag{10.1}$$

The time for transferring a BAS is solely the time the byte stream needs to reach the destination.

$$t_{BAStransfer} = t_{transfer} \tag{10.2}$$

10.2.3 Experimental Results

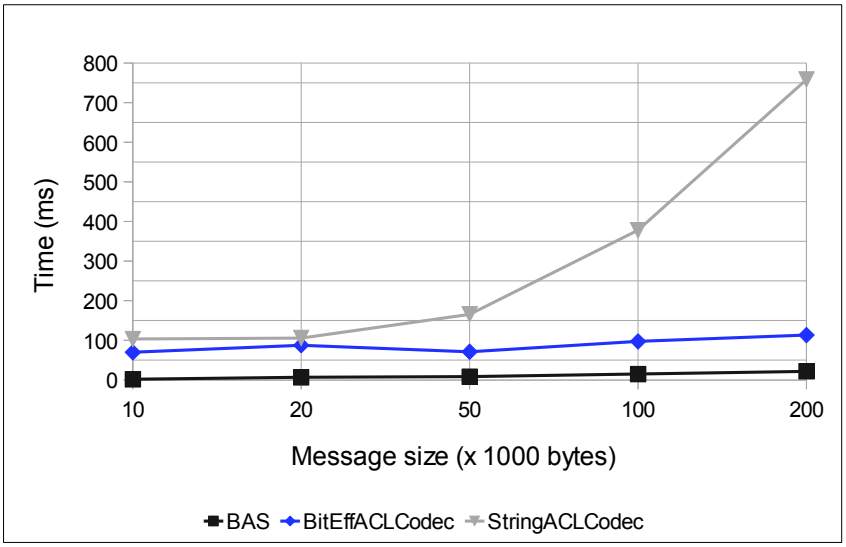

Figure 10.7: Time measurement for transferring messages

Figure 10.7 shows the measurement results for the three performance tests. As usual expected the transmission of a native byte stream BAS takes the shortest

time to finish. The transmission of ACL messages using String ACL Message Encoding takes the longest time. By this transmission, the transferring time grows exponentially when message size increases. By contrast, the transmission of ACL messages that uses Bit-efficient ACL Coding takes a much shorter amount of time. In addition, the transferring time is less affected by message size. Compared with the transmission of a BAS, the gap between the two lines which is the total time for searching location, encoding and decoding is small. The measurement results suggest that Bit-efficient ACL Message Encoding gains a good computing performance as it needs a low cost for encoding and decoding ACL message content.

Along with Test Scenario 4, in this test scenario, we prove that the software framework provides a means to search mobile agent's location as well as to deliver ACL messages at an acceptable cost. The feature is intrinsic to successful transmission of agent messages in dynamic networks. We come to the conclusion that the software satisfies the basic requirements concerning reliability and efficiency in message transferring.

Chapter 11

Conclusions and Outlook

Communication has been identified as an essential element in mobile agent systems. Our research was motivated by a mobile agent system working in networks for rescue forces in a mass casualty incident. In this type of networks, connection instability militates against successful transmission of agent messages. To facilitate message transmission, it is necessary to organize the logical connection among agent platforms in an operationally feasible manner. Our work aims to develop a communication model for mobile agent systems that is able to deal with the inherent dynamics of modern networks. We suggested applying self-organization to the community of agent platforms.

A look at nature provided us with a way to solve the problem. We found inspiration from the honey bee colony where honey bees have a special mechanism to reach self-organization. The organization of the whole colony is done through the interaction among thousands of bees. The behaviours of honey bees inspire us to employ the self-organizing model on the colony of agent platforms. We proposed a solution to a network management mechanism where network organizing tasks are performed in an autonomous manner.

Like in the honey bee colony where honey bees fly and search for food sources, mobile agents are sent over the network to gather network and platform information. Information collected by mobile agents gives a base for managing the network. Each agent platform plays a contributory role in organizing the region. A final decision to re-organize the network is made based on a consensus of the platforms. In a platform colony, though the master node plays an important role,

it is replaceable. Thanks to a failure detector, whenever it fails a second platform can substitute for it and system's operation is not interrupted. Based on the existing mobile agent system Ellipsis, the software prototype for an adaptive model for mobile agent communication has been designed and implemented.

To validate the proposed thesis, the software prototype was deployed to run in a system of a laboratory scale where conditions of a real dynamic network had been simulated using various software tools. The first three tests demonstrated that the software prototype provides the system with the ability of being self-adaptive. It helps the system to react adequately to environmental stimuli as well as to take suitable measures to counteract unexpected network degradation. By the last two test scenarios, given that network failure happened, the system can achieve fault tolerance and maintain a reasonable processing cost for message coding and transferring. From our perspective, the software framework meets the basic goals.

In the test scenarios, some parameters have been defined and they may be valid only for a laboratory scale system, e.g. $\tau_{avg}/\tau_{RM} < 50\%$ for the splitting threshold. Nevertheless, in a real network environment, these parameters might no longer be applicable and need to be adapted to suit the characteristics of the environment. We recommend performing a learning phase in which all environmental factors are thoughtfully analysed to produce appropriate thresholds. In addition, though the software framework basically complies with the principal objectives, obviously there is a room for improvement. The failure detecting model used in the implementation works well in the given test scenarios. However in a real scenario it needs to be enhanced. There is also another aspect that is important, given that the platforms fail en masse, the whole platform colony needs to find a consensus and eventually votes a new master node. Finally, security shall also be taken into account, since in our implementation this issue has been left open.

It is our firm belief that the honey bee inspired model cannot only be utilized for mobile agent systems. It can also be applied in other types of P2P network, where it is necessary to network member nodes in a flexible manner. The model can be used to solve other types of problems in network management, such as for discovering resources in P2P networks.

Bibliography

[ACK+97] Marcos Kawazoe Aguilera, Wei Chen, Marcos Kawazoe, Aguilera Wei, and Sam Toueg. Heartbeat: A timeout-free failure detector for quiescent reliable communication, 1997. 73

[Ahn04] JinHo Ahn. An adaptive communication mechanism for highly mobile agents. In *International Conference on Computational Science*, pages 192–199, 2004. 26

[Ahn10] Jinho Ahn. Atomic mobile agent group communication. In *CCNC'10: Proceedings of the 7th IEEE conference on Consumer communications and networking conference*, pages 797–801, Piscataway, NJ, USA, 2010. IEEE Press. 26

[AP95] Baruch Awerbuch and David Peleg. Online tracking of mobile users. *J. ACM*, 42(5):1021–1058, 1995. 3

[Aus75] J. L. Austin. *How to do things with words*. Harvard University Press, Cambridge, Mass., 1975. 9

[Bag05] Faruk Bagci. *Reflektive mobile Agenten in ubiquitären Systemen*. PhD thesis, Universität Augsburg, 2005. 33

[Bau00] Joachim Baumann. *Control algorithms for mobile agents*. PhD thesis, University of Stuttgart, 2000. v, vii, 3

[BCPR03] Fabio Bellifemine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. Jade: A white paper. 3:6–19, 2003. `http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf`. 36

[BGP05] Dimitrios A. Baltatzis, Christos K. Georgiadis, and Ion G. Pagkalos. Mobile agents in e-commerce environments: Supporting collaborative activities. *International Conference on Computational Intelligence for Modelling, Control and Automation*, 1:205–210, 2005. v, vii

[BHea05] Bert Hubert et al. Linux Advanced Routing and Traffic Control HOWTO. 2005. `http://lartc.org/lartc.pdf`. 138

[BHR+97] Joachim Baumann, Fritz Hohl, Nikolaos Radouniklis, Kurt Rothermel, and Markus Strasser. Communication concepts for mobile agent systems. *Proceedings of the First International Workshop on Mobile Agents*, pages 123–135, 1997. v, vii, 3

[BJL06] John F. Buford, Gabriel Jakobson, and Lundy Lewis. Multi-agent situation management for supporting large-scale disaster relief operations. *International Journal of Intelligent Control and Systems*, 11(4):284–295, 2006. 41, 48, 49

[BP01] Federico Bergenti and Agostino Poggi. Leap: A fipa platform for handheld and mobile devices. In *ATAL*, pages 436–446, 2001. 38

[BPR01] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi-agent systems with jade. In *ATAL '00: Proceedings of the 7th International Workshop on Intelligent Agents VII. Agent Theories Architectures and Languages*, pages 89–103, London, UK, 2001. Springer-Verlag. xix, 37, 38

[BPTU03] Faruk Bagci, Jan Petzold, Wolfgang Trumler, and Theo Ungerer. Ubiquitous mobile agent system in a p2p- network. *IEEE Communications Magazine*, pages 12–15, 2003. 34

[BR04] Peter Braun and Wilhelm Rossak. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. xix, 21, 22, 23

[Bra97] Jeffrey M. Bradshaw, editor. *Software Agents*. AAAI Press, Menlo Park, CA, 1997. 7

[Bra03] Peter Braun. *The Migration Process of Mobile Agents*. PhD thesis, Friedrich-Schiller-Universität Jena, 2003. v, vii, 3, 84

[BSP+05] Faruk Bagci, Holger Schick, Jan Petzold, Wolfgang Trumler, and Theo Ungerer. Communication and security extensions for a ubiquitous mobile agent system (ubimas). In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 246–251, New York, NY, USA, 2005. ACM. xix, 25, 33, 34

[BWSU10] Faruk Bagci, Julian Wolf, Benjamin Satzger, and Theo Ungerer. Ubimass - ubiquitous mobile agent system for wireless sensor networks. *Sensor Networks, Ubiquitous, and Trustworthy Computing, International Conference on*, 0:245–252, 2010. 33

[BWUB09] Faruk Bagci, Julian Wolf, Theo Ungerer, and Nader Bagherzadeh. Mobile agents for wireless sensor networks. In *ICWN*, pages 502–508, 2009. 34

[Cal02] Monique Calisti. Abstracting communication in distributed agent-based systems. In *Proceedings of the 16th European Conference on Object-Oriented Programming*, 2002. 7, 14, 16

[CBK+10] SungJin Choi, MaengSoon Baik, HongSoo Kim, EunJoung Byun, and Hyunseung Choo. A reliable communication protocol for multiregion mobile agent environments. *IEEE Trans. Parallel Distrib. Syst.*, 21(1):72–85, 2010. 30, 31

[CCB+09] SungJin Choi, Hyunseung Choo, MaengSoon Baik, HongSoo Kim, and EunJoung Byun. Oddugi: Ubiquitous mobile agent system. 5593:393–407, 2009. 30

[CFLD02] Jiannong Cao, Xinyu Feng, Jian Lu, and Sajal K. Das. Design of adaptive and reliable mobile agent communication protocols. *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 471, 2002. 25, 29

[CGL07] Min Chen, Sergio Gonzalez, and Victor C.M. Leung. Applications and design issues for mobile agents in wireless sensor networks. *IEEE Wireless Communications*, 14:20–26, 2007. 3

[CKB+06] SungJin Choi, HongSoo Kim, EunJoung Byun, ChongSun Hwang, and MaengSoon Baik. Reliable asynchronous message delivery for mobile agents. *IEEE Internet Computing*, 10:16–25, 2006. 25, 31

[CLC08] Bo Chen, David D. Linz, and Harry H. Cheng. Xml-based agent communication, migration and computation in mobile agent systems. *Journal of Systems Software*, 81:1364–1376, August 2008. 18

[CLM+08] Tiziana Catarci, Massimiliano de Leoni, Andrea Marrella, Massimo Mecella, Berardino Salvatore, Guido Vetere, Schahram Dustdar, Lukasz Juszczyk, Atif Manzoor, and Hong-Linh Truong. Pervasive software environments for supporting disaster responses. *IEEE Internet Computing*, 12:26–37, January 2008. 43

[CMCV02] L. M. Camarinha-Matos, Octavio Castolo, and Walter Vieira. A mobile agents approach to virtual laboratories and remote supervision. *J. Intell. Robotics Syst.*, 35:1–22, September 2002. v, vii

[CTE07] Judith Hooper Corinna Thom, David C Gilley and Harald E Esch. The scent of the waggle dance. *PLoS Biology*, 2007. `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1994260/?tool=pmcentrez`. 63

[DÖ8] Arndt Döhler. *Hochdynamische logische Netzwerke als Infrastruktur mobiler Agentensysteme*. PhD thesis, Friedrich-Schiller-Universität Jena, 2008. 4, 116

[DER04] Arndt Döhler, Christian Erfurth, and Wilhelm Rossak. An infrastructure-based approach to support dynamic networks with mobile agents. In Michael Smirnov, editor, *WAC*, volume 3457 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2004. 4

[DER05] Arndt Döhler, Christian Erfurth, and Wilhelm Rossak. A framework of autonomous and self-adaptable middleware services to support mobile agents in dynamic networks. *GESTS Int'l Trans. Computer Science and Engr.*, Vol.19, No.1:109–122, 2005. 4

[ea02] Arne Grimstrup et al. Toward dynamic interoperability of mobile agent systems. In *Proceedings of the Sixth IEEE International Conference on Mobile Agents*, pages 106–220, 2002. v, vii

[ea08] Lass et al. Coordination of first responders under communication and resource constraints (short paper). In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Padgham, Parkes, Mueller and Parsons (eds.), May, 12-16., 2008,Estoril,Portugal*, pages 1409–1412, 2008. 43

[ED04] Christian Erfurth and Arndt Döhler. A first look at the performance of autonomous mobile agents in dynamic networks. In *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS04) - Track 9, Big Island*. IEEE Computer Society, 2004. 4

[Erf04] Christian Erfurth. *Proaktive autonome Navigation für mobile Agenten*. PhD thesis, Friedrich-Schiller-Universität Jena, 2004. v, vii, 3, 4, 48

[FIP00a] FIPA. Fipa abstract architecture specification, 2000. `http://www.fipa.org/specs/fipa00001/`. xix, 12

[FIP00b] FIPA. Fipa agent message transport service, 2000. `http://www.fipa.org/specs/fipa00067/`. xix, 13, 14

[FIP00c] FIPA. Fipa interaction protocol specifications, 2000. `http://www.fipa.org/repository/ips.php3`. 17

[FIP00d] FIPA. Fipa request interaction protocol specification, 2000. `http://www.fipa.org/specs/fipa00026/SC00026H.pdf`. xix, 17

[FIP01] FIPA. Fipa ontology service specification. 2001. http://www.fipa.org/specs/fipa00086/XC00086D.pdf/. 13

[FIP02a] FIPA. Fipa acl message representation in bit-efficient specification. 2002. 18, 102, 150

[FIP02b] FIPA. Fipa acl message representation in string specification. 2002. 18, 150

[FIP02c] FIPA. Fipa acl message representation in xml specification. 2002. 18

[FIP02d] FIPA. Fipa communicative act library specification, 2002. http://www.fipa.org/specs/fipa00037/SC00037J.pdf. 8, 11

[FLP98] Tim Finin, Yannis Labrou, and Yun Peng. Mobile agents can benefit from standards efforts in inter-agent communication. *IEEE Communications Magazine*, 36:50–56, 1998. v, vii, 7, 10, 13

[FSS05] Kai Fischbach, Christian Schmitt, and Detlef Schoder. Core concepts in peer-to-peer networking. In *Peer to Peer Computing: The Evolution of a Disruptive Technology*, chapter 1. Idea Group Inc., 2005. 47

[FSU11] FSU. The SpeedUp Project. 2011. http://www.speedup-projekt.de. v, viii, 5, 39

[Gad96] Raghavendra Gadagkar. The honeybee dance-language controversy. *Resonance: Journal of Science Education*, 1(1):63–70, 1996. 64

[GKRM+11] Aygul Gabdulkhakova, Birgitta König-Ries, Mareike Mähler, Yeliz Yildirim-Krannig, and Fabian Wucholt. Identifying and supporting information needs in mass casualty incidents an interdisciplinary approach. In *Proceedings of the 8th International ISCRAM Conference*, 2011. 43, 44

[GL99] Benjamin N. Grosof and Yannis Labrou. An approach to using xml and a rule-based content language with an agent communication language. In *Communication Language, IJCAI-99 Workshop on Agent Communication Languages*. Springer-Verlag, 1999. 18

[GNU99] GNU. GNU Lesser General Public License. 1999. `http://www.gnu.org/licenses/lgpl-2.1.html`. 120

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *KNOWLEDGE ACQUISITION*, 5:199–220, 1993. 13

[Gru95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43:907–928, December 1995. 13

[GWZ02] Joachim Giesen, Roger Wattenhofer, and Aaron Zollinger. Towards a theory of peer-to-peer computability. In *Proceedings of the 9th International Colloquium on Structural Information and Communication (SIROCCO)*, pages 115–132, 2002. 47

[Hay07] Jacqui Hayes. Pleasure chemical controls bee dance. *Cosmos Online*, 2007. `http://www.cosmosmagazine.com/node/969`. 63

[HBF+07] John Howse, Richard Bosworth, Andrew Fish, Gem Stapleton, John Taylor, Peter Rodgers, and Simon Thompson. Euler diagram-based notations, 2007. 19, 89

[Hel03] Heikke Helin. *Supporting Nomadic Agent-based Applications in the FIPA Agent Architecture*. PhD thesis, University of Helsinki, Finland, 2003. vii, xix, 3, 14, 15, 16, 38, 87

[Hey99] Francis Heylighen. The science of self-organization and adaptivity. In *in: Knowledge Management, Organizational Intelligence and Learning, and Complexity, in: The Encyclopedia of Life Support Systems, EOLSS*, pages 253–280. Publishers Co. Ltd, 1999. 60, 61

[HG03] Francis Heylighen and Carlos Gershenson. The meaning of self-organization in computing. *IEEE Intelligent Systems*, 18:4:72–75, 2003. 60, 61

[Hur95] Gurdeep S. Hura. Client-server computing architecture: An efficient paradigm for project management. In *Proceedings of the 1995 IEEE Annual International on Engineering Management Conference*, pages 146–152, 1995. 46

[Jen11] Uni Jena. Multi Agent System Ellipsis, 2011. `http://sourceforge.net/projects/masellipsis/`. 80

[JYBz07] Hojjat Jafarpour, Nasser Yazdani, and Navid Bazzaz-zadeh. A scalable group communication mechanism for mobile agents. *J. Netw. Comput. Appl.*, 30(1):186–208, 2007. 21

[KA09] Dervis Karaboga and Bahriye Akay. A survey: algorithms simulating bee swarm intelligence. *Artif. Intell. Rev.*, 31:61–85, June 2009. 62

[Kim10] John W. Kimball. Honeybee communication. 2010. `http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/B/BeeDances.html`. 62, 63

[LA05] Xining Li and Guillaume Autran. Inter-agent communication in imago prolog. *Lecture Notes in Computer Science*, Volume 3346/2005:163–180, 2005. 27

[LF98] Yannis Labrou and Tim Finin. Semantics and conversations for an agent communication language. pages 235–242, 1998. 10

[LHL02] Mikko Laukkanen, Heikki Helin, and Heimo Laamanen. Supporting nomadic agent-based applications in the fipa agent architecture. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1348–1355, New York, NY, USA, 2002. ACM. v, vii

[Li01] Xining Li. Imago: A prolog-based system for intelligent mobile agents. In *MATA '01: Proceedings of the Third International Workshop on Mobile Agents for Telecommunication Applications*, pages 21–30, London, UK, 2001. Springer-Verlag. 27

[Mor06] Antonio Moreno. Guest editor's introduction: On the evolution of applying agent technology to healthcare. *IEEE Intelligent Systems*, 21(6):8–10, 2006. v, vii

[MT99] Hongsong Ma and Sun Teck Tan. Dynamic mobile agent based distributed network management using internet technology and corba. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Oct, 12-15., 1999, Tokyo, Japan*, pages 101–106, 1999. v, vii

[NLA12] NLANR/DAST. Network Performance Measurement, 2012. `http://sourceforge.net/projects/iperf/`. 132

[NSR11a] Phuong Nguyen, Volkmar Schau, and Wilhelm Rossak. Performance comparison of some message transport protocol implementations for agent community communication. In *Proceedings of the 11th International Conference on Innovative Internet Community Systems*, volume 186 of *Lecture Notes in Informatics (LNI-P)*, pages 193–204, Bonn, Germany, June 2011. GI-Edition, Bonner Köllen Verlag. 87

[NSR11b] Phuong Nguyen, Volkmar Schau, and Wilhelm Rossak. Towards an adaptive communication model for mobile agents in highly dynamic networks based on swarming behaviour. In *Proceedings of the 9th European Workshop on Multi-agent Systems*, Maastricht, The Netherlands, November 2011. 58

[NT04] Sunil Nakrani and Craig Tovey. On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 12:223–240, September 2004. 62

[Obi07] Marek Obitko. Communication between agents. 2007. `http://www.obitko.com/tutorials/ontologies-semantic-web/communication-between-agents.html`. 8

[PAK07] Truong Pham, Ashraf Afify, and Ebubeki Koc. Manufacturing cell information using the bees algorithm. In *in Proceedings IPROMS 2007 Innovative Production Machines and Systems Virtual Conference, Cardiff, UK.*, 2007. 62

[Pau02] Herv Paulino. A mobile agent systems' overview. Technical report, Departamento de Informtica, Faculdade de Cincias e Tecnologia. Universidade Nova de lisboa, February 2002. v, vii, 19

[PB02] Hardy Pundt and Yeaser Bishr. Domain ontologies for data sharing-an example from environmental monitoring using field gis. *Computers & Geosciences*, 28:95–102, 2002. 13

[PC00] Stefan Poslad and Monique Calisti. Towards improved trust and security in fipa agent platforms. In *Proceedings of Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies*, 2000. 11

[PS] Evaggelia Pitoura and George Samaras. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering*, 13:571–592. 22

[PSV07] Kevin M Passino, Thomas D Seeley, and P Kirk Visscher. Swarm cognition in honey bees. *Behavioral Ecology and Sociobiology*, 62(3):401–414, 2007. 65

[SB01] Thomas D. Seeley and Susannah C. Buhrman. Nest-site selection in honey bees: how well do swarms implement the best-of-n decision rule? *Behavioral Ecology and Sociobiology*, 49:416–427, 2001. 63, 65

[Sch12] Volkmar Schau. *Ellipsis - Konzeption und Implementierung einer vereinheitlichten und leistungsfähigen Architektur fr Multi-Agenten Systeme (MAS) auf Basis frei verfügbarer IT-Infrastrukturkomponenten unter besonderer Bercksichtigung internationaler Standards*. PhD thesis, Friedrich-Schiller-Universität Jena, 2012. 5, 79, 82

[See03] Thomas D Seeley. Consensus building during nest-site selection in honey bee swarms : the expiration of dissent. *New York*, 53(6):417–424, 2003. 64, 65

[Sim07] Paul Simoneau. The osi model: Understanding the seven layers of computer networks, 2007. 16

[SKEE10] Volkmar Schau, Kathrin Kirchner, Christian Erfurth, and Gerald Eichler. Ad-hoc community composition of rescue forces in action situations. In *10th International Conference on Innovative Internet Community Services*, pages 41–52, 2010. 41, 43, 49

[SKV04] Thomas D Seeley and P Kirk Visscher. Group decision making in nest-site selection by honey bees. *Apidologie*, 35(2):101–116, 2004. 65

[SS97] Markus Straßer and Markus Schwehm. A performance model for mobile agent systems. In *LAS VEGAS*, pages 1132–1140. CSREA, 1997. v, vii, 3

[SSB02] Maja Stula, Darko Stipanicev, and Mirjana Bonkovic. Feasible agent communication architecture. In *International Conference on Software, Telecommunications and Computer Networks*, 2002. 16

[SV03] Thomas D. Seeley and P. Kirk Visscher. Choosing a home: how the scouts in a honey bee swarm perceive the completion of their group decision making. *Behavioral Ecology and Sociobiology*, 54(5):511–520, 2003. 64, 65

[SV04] Thomas D. Seeley and P. Kirk Visscher. Group decision making in nest-site selection by honey bees. *Apidologie*, 35(2):101–116, 2004. 63, 65

[SWU10] Peter Sewell, Paweł T. Wojciechowski, and Asis Unyapoth. Nomadic pict: Programming languages, communication infrastructure overlays, and semantics for mobile computation. *ACM Trans. Program. Lang. Syst.*, 32(4):1–63, 2010. 21, 23

[TIL12] TILab. Java Agent DEvelopment Framework. 2012. `http://jade.tilab.com/`. 120

[TYI99] Shinji Tanaka, Hirofumi Yamaki, and Toru Ishida. Mobile-agents for distributed market computing. In *Proceedings of the 1999 International Conference on Parallel Processing*, ICPP '99, pages 472–, Washington, DC, USA, 1999. IEEE Computer Society. v, vii

[VMWB07] Renata Vieira, Álvaro Moreira, Michael Wooldridge, and Rafael H. Bordini. On the formal semantics of speech-act based communication in an agent-oriented programming language. *Journal of Artificial Intelligence Research*, 29:221–267, June 2007. 7, 10

[Woj01] Pawel T. Wojciechowski. Algorithms for Location-Independent Communication between Mobile Agents. Technical report, 2001. 22, 23, 24

[Yeo10] Kiwon Yeom. Bio-inspired self-organization for supporting dynamic reconfiguration of modular agents. *Mathematical and Computer Modelling*, 52(11-12):2097–2117, 2010. 61

[ZK02] Konstantinos G. Zerfiridis and Helen D. Karatza. Mobile agents as a middleware for data dissemination. *Neural, Parallel and Scientific Computations*, 10:313–324, September 2002. v, vii

Appendix A

Bandwidth Shaping Script

The script in Listing A.1 is executed in platform $D3$ to shape bandwidth on TCP 3242 to the RegionMaster $D2$ (IP $10.35.14.185$) as illustrated in Figure 9.10.

Listing A.1: Bandwidth shaping script

```
#!/bin/bash
# Name of the traffic control command.
TC=/sbin/tc
IPTABLES=/sbin/iptables
SERVICE=/sbin/service

MiUB=1mbit    # Minimum Usable Bandwidth
MxUB=2mbit    # Maximum Usable Bandwidth
PORT=3242     # TCP Port 3242

# The network interface on which we are going to control bandwidth
IF=eth0             # Interface

# IP address of the machine we are controlling
IP=10.35.14.185     # The IP address of the targeted host

start() {
# Hierarchical Token Bucket (HTB) is used to shape bandwidth.
$TC qdisc add dev $IF root handle 1: htb
$TC class add dev $IF parent 1: classid 1:1 htb rate $MiUB ceil
   $MxUB prio 0
$TC filter add dev $IF parent 1: protocol ip u32 match ip dport
   $PORT 0xffff match ip dst $IP classid 1:1
```

```
$IPTABLES -t mangle -A POSTROUTING -o $IF -p tcp --dport $PORT -j
   MARK --set-mark 6
}

# Stop bandwidth shaping.
stop() {
$TC qdisc del dev $IF root
$IPTABLES -t mangle -D POSTROUTING -o $IF -p tcp --dport $PORT -j
   MARK --set-mark 6
}
exit 0
```

In Appendix B, tables B.1, B.2, B.3, B.4, and B.5 show the processing time measured for the platforms displayed in Figure 9.2. The measurement test has been executed in 80 times for every agent platform. For each table, the first row - printed in bold - represents the sequence test number. The next rows represent the corresponding measurement for each of the platforms $\{D1, D2, D3, D4, FJ, PT, T1, T2\}$.

Appendix B

Processing Time

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
$D1$	181	143	102	54	66	72	67	52	54	79	56	52	81	52	77	53
$D2$	144	86	94	87	73	62	77	78	60	51	63	91	80	66	66	60
$D3$	115	87	120	101	91	68	111	96	58	77	70	125	69	59	118	58
$D4$	111	136	124	77	121	56	49	80	56	112	53	50	63	57	51	74
FJ	91	76	69	188	60	80	69	83	69	82	71	66	65	98	90	68
PT	62	47	47	47	94	47	219	62	78	47	47	78	63	62	63	47
$T1$	119	98	84	92	82	85	161	118	71	81	110	192	70	88	120	80
$T2$	141	133	134	152	138	99	252	102	110	165	103	159	157	146	137	110

Table B.1: Measurement results sequences from 1 to 16

	17	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
$D1$	56	57	60	53	52	74	56	58	54	55	48	80	87	52	53	55
$D2$	69	76	68	73	55	60	59	52	57	83	85	101	63	74	175	52
$D3$	67	92	73	53	66	70	74	72	75	86	95	68	69	55	52	82
$D4$	47	105	48	53	50	56	46	63	60	59	57	94	53	58	59	65
FJ	59	70	64	63	74	68	291	68	59	69	93	97	104	63	61	62
PT	78	62	47	47	47	63	47	47	62	63	47	47	47	47	47	93
$T1$	83	71	82	120	76	71	69	84	93	78	98	84	179	91	75	67
$T2$	133	120	128	118	170	144	141	107	100	107	125	128	158	181	115	194

Table B.2: Measurement results sequences from 17 to 32

	33	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**
$D1$	48	51	51	53	89	60	57	59	64	59	64	81	78	91	49	54
$D2$	93	57	79	55	83	70	87	65	69	65	56	71	75	71	79	69
$D3$	71	50	71	137	103	73	73	51	210	69	51	51	87	72	56	75
$D4$	54	73	47	96	55	69	54	54	88	60	54	57	59	93	54	50
FJ	62	66	99	61	65	71	70	61	91	63	64	60	100	56	73	80
PT	93	94	78	94	62	47	47	62	93	46	47	110	47	94	47	47
$T1$	62	76	73	173	166	72	76	66	78	176	78	87	115	108	198	80
$T2$	146	99	116	224	129	110	104	121	123	109	155	98	96	105	141	239

Table B.3: Measurement results sequences from 33 to 48

	49	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**	**64**
$D1$	96	54	56	52	54	67	170	55	97	73	86	56	93	52	50	86
$D2$	62	66	62	69	77	153	69	61	60	67	64	73	61	73	69	84
$D3$	68	63	55	71	53	57	71	92	58	50	83	54	54	82	61	93
$D4$	48	51	63	74	52	64	51	57	48	51	83	51	62	48	64	53
FJ	63	72	69	64	63	73	73	66	89	72	72	63	86	101	75	101
PT	47	47	63	62	47	47	109	47	62	63	63	94	187	78	47	124
$T1$	76	76	103	80	72	78	79	70	77	116	106	88	79	96	67	86
$T2$	152	159	137	121	159	153	122	106	130	86	128	96	219	145	94	111

Table B.4: Measurement results sequences from 49 to 64

	65	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**
$D1$	59	64	54	47	59	180	52	55	56	50	52	89	48	86	55	74
$D2$	55	175	81	69	67	66	98	63	53	66	64	82	60	65	68	65
$D3$	75	60	59	49	75	102	103	57	73	80	61	74	65	74	55	62
$D4$	48	57	50	59	61	48	77	49	50	55	73	46	66	51	64	57
FJ	222	69	69	79	72	100	98	65	63	60	82	71	62	70	62	66
PT	156	47	47	93	62	62	94	47	47	46	78	187	63	47	47	187
$T1$	102	89	81	87	81	72	80	120	79	71	109	74	98	90	75	68
$T2$	118	147	111	116	109	150	109	110	109	119	113	108	105	95	107	249

Table B.5: Measurement results sequences from 65 to 80

Ehrenwörtliche Erklärung

Hiermit erkläre ich,

- dass mir die Promotionsordnung der Fakultät bekannt ist,
- dass ich die Dissertation selbst angefertigt habe, keine Textabschnitte oder Ergebnisse eines Dritten oder eigenen Prüfungsarbeiten ohne Kennzeichnung übernommen und alle von mir benutzten Hilfsmittel, persönliche Mitteilungen und Quellen in meiner Arbeit angegeben habe,
- dass ich die Hilfe eines Promotionsberaters nicht in Anspruch genommen habe und dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen,
- dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe,
- dass ich weder die gleiche, noch eine in wesentlichen Teilen ähnliche, noch irgendeineandere Abhandlung bei einer anderen Hochschule als Dissertation eingereicht habe.

Jena, den 20. Juni 2012

Phuong T. Nguyen

Lebenslauf

Persönliche Daten

Name	Phuong Thanh Nguyen
Geburtsdatum	10. September 1979
Geburtsort	Quangninh, Vietnam
Familienstand	verheiratet, 1 Tochter

Schulische und berufliche Ausbildung

1985 - 1997	Grundschule und Gymnasium in Quangninh und in Hanoi Abschluss: 12. Klasse
1997 - 2002	Studium der Informatik an der Technischen Universität Hanoi Abschluss: Engineer of Information Technology
2002 - 2004	Studium der Informatik an der Technischen Universität Hanoi Abschluss: Master of Science in Information Technology
2009 - Präsens	Studium der Informatik an der Friedrich-Schiller-Universität Jena, Doktorand Stipendiat der vietnamesischen Regierung und des DAADs

Wissenchaftliche Tätigkeit

seit 2002	Wissenschaftlicher Mitarbeiter an der Fakultät für Informationstechnik der Technischen Universität Hanoi

Jena, den 20. Juni 2012

Phuong T. Nguyen