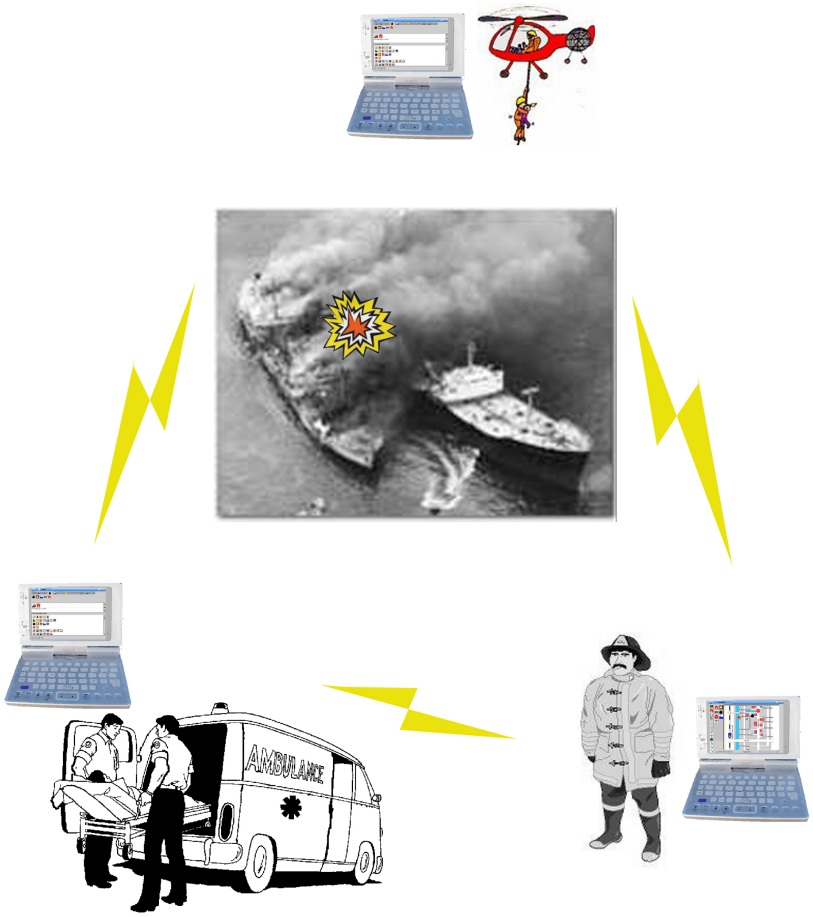
Thesis report:

TICS:

A Topology based Infrastructure for Crisis Situations



Paul Klapwijk, 1047507

Thesis report
August 22, 2005

Man-Machine Interaction Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology, The Netherlands

Delft University of Technology

Paul Klapwijk (p.klapwijk@gmail.com)

TICS: A Topology based Infrastructure for Crisis Situations

Thesis report
August 22, 2005

Man-Machine Interaction Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology, The Netherlands
http://www.kbs.twi.tudelft.nl

Graduation committee:

Dr. Drs. L.J.M. Rothkrantz
Dr. S. Oomes
Drs. J.M.V. Misker
Ir. F. Ververs

Abstract

Recent disasters have not only pointed out to humanity how small we are compared to nature, but also showed us that terrorism has become a factor that we cannot ignore. We are being forced to handle crisis situations in the most efficient way we can. Especially the first hours after a disaster can be of vital interest to the victims and an efficient and well thought approach can greatly reduce the number of victims.

After the event of a crisis, rescue workers therefore have to be able to do their work in a dynamic and dangerous environment, with all means available. In these situations it has turned out that on a local and global level, the communication between rescue workers is sometimes a problem. Recently more interest has been going out to assisting rescue workers in their work by offering them solutions to problems or extra tools and services to simplify their tasks while improving the results.

The approach described in this thesis work aims at facilitating the communication for the rescue workers and offering them services if needed. This communication is provided by an ad-hoc network of mobile devices, PDAs, that offer services to rescue workers, while at the same time providing an infrastructure for processing of data from the rescue workers. The communication primarily takes place via a blackboard structure, that is distributed over the nodes in the network.

Besides offering communication to the actual rescue workers that have the task of helping victims and reporting about the situation, another challenge is to collect all the knowledge that is available, to get to an overview of the situation. For this purpose, we propose introducing a structure in the topology of the network that is most applicable to a crisis situation. Our approach consists of introducing mobile crisis centers, with around them rings of nodes that take care of the processing of the data that comes in from the outer ring, that consists of the sensor nodes. These sensor nodes can be rescue workers with PDAs, but also sensors on robots or fixed sensors (e.g. chemical gas sensors). This approach is inspired by current procedures of crisis teams.

The approach aims at providing services in the network, by dynamically assigning tasks to nodes in the network. Besides the place in the network, the (distributed) decision where a tasks is to be executed, is based on performance measures of the nodes. These are integrated in one value, that is used as a decision criteria. Besides assigning tasks to the best node at hand, the system offers more functionalities with respect to fault tolerance, in the form of replication and monitoring of other nodes.

Keywords: Mobile ad-hoc networks, communication, coordination, dynamic role assignment, distributed blackboards, crisis/disaster management, fittest node.

Preface

Project

This thesis work is part of the Intelligent Systems project of the faculty EWI of the Delft University of Technology. One of the main themes of the project is Crisis Management. The more specific topic is *to facilitate communication during crisis response* ([Ww05]). The idea of the project is to facilitate communication between agents and to execute tasks with combined forces.
Due to an insufficient and inadequate communication, wrong decisions were made in recent crisis situations, decisions that in some cases turned out to be fatal. The reason that communication was lacking, was that the standard communication systems completely or partially failed at disasters like the ones on September 11th 2001 in New York and the fireworks disaster in Enschede, the Netherlands.
The communication that is to be facilitated, is to take place in an environment that is characterized as very unreliable. The overall ambition of the project is to define and implement an architecture that is capable of dealing with this dynamic and chaotic situation ([Ww05]). According to this whitepaper, eventually a demonstration is desired of a system that takes care that the distributed agents have complete information about the crisis situation and about their package of tasks. The focus of the whole project is defined as follows:

> *"How can the facilities in the 'personal environment' be used to realize a better response to crises? The development of techniques in the field of 'distributed intelligent agents', specifically the development of a virtual blackboard over the agents, new architectures for chaotic environments of interacting agents that are robust to the insertion or leaving of individual agents without a central controller; efficient use of 'network information theory"* (see [Ww05]).

This project is related to the COMBINED project of the DECIS lab. DECIS is the abbreviation for **De**lft **C**ooperation on **I**ntelligent **S**ystems. The description of the COMBINED project that can be found on the website of the project (see [com]) is as follows:

> *"We choose the domain of crisis response in which multiple parties (police, firebrigade, health services, governmental agencies, etcetera) work together to contain the crisis. Collaboratively they aim to save lives, stabilize the cause of the incident, and conserve the surrounding infrastructure.*
> *We intend to come up with new concepts and systems that will be used by the crisis response organization of the future. Our Combined System will make the performance more effective, efficient, and robust".*

Another project that is currently working on solutions for crisis situations is the recent RESCUE (Responding to Crisis and Unexpected Events) project (see [res]). The project focuses on responding to man-made and natural catastrophes. The project aims at data collection, information analysis, information sharing and information dissemination. The overall goal of

the project is to *"radically transform the ability of responding organizations to gather, manage, use, and disseminate information within emergency response networks and to the general public"* [res]. The CAMAS (Crisis Assessment, Mitigation and Analysis System) is the test-bed within this project (see [MBK+04]).

Much inspiration for the work described in this thesis report is also taken from the [Rot04] and [BR05]. In the former a system is described that is envisioned in a crisis situation. This thesis can be considered as a part of a prototype for a system as described. On some points the system will be different. As a proof of concept some parts will be implemented in a simple form. In the latter report, the common/current procedures of the emergency center, police and firemen are described.

Acknowledgements

First of all I would like to thank my thesis supervisor, Leon Rothkrantz, for all the inspiration he gave during my thesis work. Not only the weekly (group) meetings were very helpful, but also the "offline" discussions gave me many new ideas, many of which became part of my thesis work.

Secondly I want to express my deepest, deepest gratitude to Wim Tiwon, for helping me with many of the technical parts of my thesis work, for all the time (and all the blood, the sweat and the tears) while getting the hardware and operating system part of the system running. By this I mean the help with configuring the Zaurus devices, my own laptop and, of course, with porting/hacking/debugging and testing of the (M)AODV algorithms, which all have been of great importance to my work. Without this help and without the help with "fighting Murphy" this project would have meant months and months of extra work and would also have made the development much more boring.

Furthermore I want to thank the other (MSc. and PhD.) students and other people from the university who helped me during the project and gave me useful input during my project. These are (in random order): Jan Misker, Florian Haradji, Herve Ducourtieux, Filip Miletic, Stefan Strijdhaftig, Tom Benjamins, Vincent de Lange, Bogdan Tatomir, Marcel van Velden, Jan Chau, and Paul Schooneman.
I would also like to thank Ruben Stranders, for the help during my thesis project and also for all the interesting on and off topic discussions during the period that I was working on my thesis.

Two other people that I want to thank are Amy Murphy and Lorenzo Bellini. I want to thank them for letting me use (the source code of) Lime II even before it was officially released, so that I could use it in my thesis work. Furthermore I want to thank them both for their feedback on my questions.

Last, but certainly not least, I would like to thank my parents and my sister for supporting me, not only during this project, but during my entire life.

August 22, 2005
Paul Klapwijk

Contents

Abstract **i**

Preface **iii**
Project . . . iii
Acknowledgements . . . iv

1 Introduction **1**
1.1 Problem definition . . . 1
1.2 Research topic . . . 4
1.3 Structure of the document . . . 5

2 Related work **7**
2.1 Introduction . . . 7
2.2 Crisis management . . . 7
2.3 Mobile Ad-hoc Networks . . . 8
2.3.1 Mobile Ad-hoc Networks in general . . . 8
2.3.2 Routing in MANETs . . . 9
2.4 Coordination . . . 10
2.5 Agent systems . . . 11
2.5.1 JADE . . . 12
2.5.2 Cougaar . . . 13
2.6 Communication . . . 14
2.6.1 Communication in general . . . 14
2.6.2 C2000 . . . 14
2.6.3 Blackboards . . . 14

3 System overview **17**
3.1 Assumptions . . . 17
3.2 Proposed solution . . . 19
3.2.1 Network structure . . . 19
3.2.2 Communication using blackboards . . . 21
3.2.3 Introducing a topology . . . 21
3.2.4 Communication flow in the system . . . 23
3.2.5 Knowledge flow . . . 24
3.2.6 Scope . . . 26
3.2.7 Simulation vs. Real life system . . . 28
3.3 Extra requirements and constraints . . . 28
3.4 Development process . . . 29

4 Global design **31**
4.1 Introduction . . . 31
4.2 Overview of the prototype . . . 31
4.3 Order in chaos: Setting up a crisis center . . . 33
4.3.1 Humans and/or systems . . . 33
4.3.2 The design . . . 34
4.3.3 Role assignment . . . 34
4.3.4 Position of roles . . . 35
4.4 Communication . . . 35
4.4.1 Design with one blackboard . . . 35
4.4.2 Design of hierarchical blackboards . . . 37
4.4.3 Preprocessing . . . 38
4.5 Dynamic role assignment . . . 38
4.5.1 Use of roles . . . 38
4.5.2 Use of monitors . . . 39
4.5.3 Role assignment . . . 40
4.5.4 Duplicate services . . . 41
4.5.5 Replication . . . 41
4.6 The fitness of a node . . . 42
4.6.1 Reason for use of fitness . . . 42
4.6.2 The components . . . 42
4.6.3 The integrated value . . . 43
4.6.4 Observing own fitness . . . 46
4.7 Startup of services . . . 46

5 Setup low-level infrastructure **47**
5.1 Introduction . . . 47
5.2 Lime . . . 47
5.3 Setup of a wireless network . . . 49
5.4 Multihop routing part I . . . 49
5.4.1 MAODV . . . 49
5.4.2 More limitations . . . 51
5.5 Network cards and kernels . . . 51
5.6 Multihop routing part II . . . 52

6 Implementation **55**
6.1 Introduction . . . 55
6.2 Overall implementation . . . 55
6.3 Used software & tools . . . 55
6.3.1 Eclipse & Ant . . . 55
6.3.2 JUDE . . . 56
6.3.3 Test software . . . 56
6.4 Used hardware & OS . . . 59
6.4.1 The Sharp Zaurus . . . 59
6.4.2 Other hardware & OS . . . 60
6.5 The implemented framework . . . 60
6.5.1 Introduction . . . 60
6.5.2 General implementation notes . . . 60
6.5.3 Crisis center . . . 61
6.5.4 Position of roles . . . 62
6.5.5 Communication using blackboards . . . 62

6.5.6 Role assignment . . . 65
6.5.7 Fusion . . . 67
6.5.8 Monitors . . . 68
6.5.9 Replication . . . 69
6.5.10 Solving duplicates . . . 71
6.5.11 The fitness of a node . . . 72
6.5.12 Panic services . . . 75
6.5.13 Applying some functionalities to the test programs . . . 77

7 Experiments & results **79**
7.1 Introduction . . . 79
7.2 General test setup . . . 79
7.3 The fitness function . . . 80
7.3.1 Determining the fitness parameters . . . 80
7.3.2 Comparison with linear combination . . . 80
7.3.3 Result . . . 81
7.4 Test cases . . . 81
7.4.1 Case 1: Testing communication . . . 81
7.4.2 Case 2: Testing role assignment . . . 84
7.4.3 Case 3: Duplicate services . . . 85
7.4.4 Case 4: Panic / group partition . . . 85

8 Conclusions & further research **87**
8.1 Conclusions . . . 87
8.1.1 General conclusions . . . 87
8.1.2 Literature survey . . . 87
8.1.3 Model . . . 87
8.1.4 The used design methodology . . . 88
8.1.5 The implemented solution . . . 88
8.2 Further research & possible improvements . . . 90
8.2.1 Further research . . . 90
8.2.2 Possible improvements . . . 92

Bibliography **99**

Glossary **101**

Appendices **103**
Appendix A: C program for JNI . . . 103
Appendix B: Communication flow . . . 104
Appendix C: Paper . . . 105

List of Figures

1.1 September 11th 2001 . . . 1

2.1 The CAMAS system (from [MBK+04]) . . . 8
2.2 2 possibilities for LEAP (taken from [Cai03b]) . . . 12
2.3 Agent internal structure (taken from [cou04]) . . . 13

3.1 A network with a stable core . . . 18
3.2 The PIRA system (taken from [RDFT05]) . . . 19
3.3 Approach in this thesis work . . . 20
3.4 An overview of the topology in the total system . . . 22
3.5 Overview of the communication flow . . . 24
3.6 The knowledge flow in the system . . . 25
3.7 The layers in the proposed system . . . 27
3.8 A schematic overview of the development process . . . 29

4.1 Overview of the architecture of the system . . . 32
4.2 An ad-hoc network of PDAs using one blackboard . . . 36
4.3 An ad-hoc network of PDAs using a hierarchy of blackboards . . . 37
4.4 A schematic picture of a node . . . 38
4.5 A schematic picture of an agent . . . 39
4.6 The use of monitors . . . 40
4.7 The general relation between fitness and a component . . . 44
4.8 The general relation between fitness and a component . . . 45

5.1 The low-level infrastructure . . . 47
5.2 Schematic overview of overall functioning of Lime . . . 48
5.3 Route establishment ([aod]) . . . 50
5.4 Route establishment ([KC01]) . . . 51

6.1 The components of the prototype . . . 56
6.2 Screenshot of the Eclipse development environment . . . 57
6.3 Screenshot of JUDE . . . 58
6.4 Screenshots of the test applications . . . 59
6.5 The Sharp Zaurus including specifications . . . 59
6.6 Overview of the packages in the system . . . 61
6.7 Class diagram of communication package (BBCommunicator is displayed somewhat simpler for readability) . . . 65
6.8 Communication flow from the outer to the inner part of the network . . . 66
6.9 A simplified sequence diagram of the role assignment procedure . . . 68
6.10 Class diagram of a fusion role . . . 71
6.11 Class diagram of ReplicaKeeper . . . 72

6.12 The Java code for getting a Linux process id 73
6.13 Class diagram of the package for maintaining routing information 75
6.14 Class diagram for the classes concerning the calculation of the fitness 76

7.1 The initial test setup 79
7.2 Test setup for multihop testing 82

8.1 The C code for getting a Linux process id 103
8.2 Communication flow from the outer to the inner part of the network 104

Chapter 1

Introduction

1.1 Problem definition

September 11th 2001. *Terrorists hijack airplanes and fly into the Twin Towers in New York and the Pentagon building in Washington D.C. After the collapse of the Twin Towers, most of the communication equipment, that was in the Verizon building at 140 West street, is destroyed ([Wat04]). From that moment on, many people at "Ground Zero" lost the capability to communicate. By 10 A.M. the nearby police could not use the phone anymore, along with e-mail, cellular and pager services ([Wat04]). Because of this and because of the fact that the communication between rescue services was not effective, many people lost their lives, because they were in a building that was collapsing and did not know about it (see for example [sen], [wir] and [voi]).*

December 2002. *A "simple" bombalert at the "Ikea" shop in Delft. Immediately there was a huge amount of GSM traffic, which made the mobile phone network useless for communication, since it was (simply) overloaded. This is a problem that occurs at many crises/disasters.*

December 26th 2004. *The sea flood, called "the Tsunami" makes over 200.000 victims in the shores of Sri Lanka, Indonesia, India, Thailand, Myanmar, Bangladesh, Maleisia, the Maldives, the Seychellen and the Andaman-islands ([tsu]). A large part of the coastal area is devastated. Among the problems that occurred immediately after the disaster, was that the power and communication lines were down at some places [pow]. This made it very difficult to contact parts of the stricken areas.*

March 28th 2005. *A flood makes approximately 1000 victims on islands at the shore of Sumatra, Indonesia. One of the main problems is that the islands are not easy to reach and there is no communication. The initial amount of victims is estimated on 2000, but turned out to be*

Figure 1.1: September 11th 2001

approximately 1000 (see [zeea] and [zeeb]). The area was only reachable with small material (transported by helicopters).

April 6th 2005. *A large disaster, played in the surroundings of the Amsterdam Arena. The scenario that is played, is that there is a bomb attack, there are rumours about anthrax and the result is a choas, in which rescue workers should execute their task. The reason for the crisis simulation is that the danger of natural disasters and terrorist attacks are considered as non negligible.*

Perhaps not the most pleasant way to begin a Master thesis work, but unfortunately these are not ficticious stories, but, as the dates already imply, it concerns events that really happened. Not only were we reminded how tiny we are in comparison to nature and that certain terrorist organizations do not shy away from causing large disasters, we were also forced to deal with these situations. Of course we want to deal with these situations as good and effective as possible, resulting in a minimum of losses. However, in dealing with these situations, the above examples show that the right information is not always at the right place at the right time. More examples of this can be found at [bru]. Problems with communication also became clear with the recent release of the chaotic and failing communication between the rescue workers at the Twin Towers during the 9-11 attacks.

The need to deal with this type of chaotic crisis situations is also recognized in other projects, such as the COMBINED project of the DECIS Lab in Delft and the American RESCUE project. The latter projects focuses on responding to man-made and natural catastrophes (see [res]). The CAMAS (Crisis Assessment, Mitigation and Analysis System) is the test-bed within this project (see [MBK+04]).

In both cases, natural disasters and terrorist attacks, we are dealing with disasters that will create a chaotic situation, especially in the first hours after the disaster has happened. In these situations it is necessary that as many victims as possible are rescued and that the chaotic situation is controlled. Concerning the controlling of the situation, some research is done on the current procedures (see [dec]). These procedures tell little about the immediate actions that should be taken just after the disaster has happened (perhaps this information is considered to be somewhat too "sensitive" for the general public). Especially the first hour after the disaster has happened, the chaos can be large and there is, first on local level, a strong need for control of the situation and some coordinated actions. In these cases we believe it is very useful to have one or more mobile crisis centers. This is now a very common procedure for firemen / policemen[1]. The goal of such a center is first to get an overview of what is going on and next to provide appropriate help. This implies collecting witness reports from the crisis domain, setup an infrastructure to communicate and provide access to information systems / expert systems to organize and mobilize help.

Especially in case of larger disasters, setting up a fixed crisis center with a wired infrastructure, can be impossible due to lack of a suitable infrastructure in the area where the disaster has happened. For this reason it seems suitable to use a more mobile setup of the crisis center and use a way of communication that is independent of the local infrastructure. The above example about the bomb alert is a nice illustration of this. If the people that were to control the situation would have had a communication network based on the GSM network, this would immediately have become useless. Thus, for a reliable and always applicable way of handling

[1]These are called "COmmando Plaats Incident centers" or "Co-Pi centers" in the Netherlands, according to the Environmental monitoring service in Rijnmond, DCMR. ([BR05])

crisis situations, the communication between the rescue workers and the crisis center should be facilitated without the use of the fixed infrastructure. One possible way to build a system in these situations is by making them distributed over multiple (simple) devices, such as PDAs. This topic will be central in this thesis work. This idea, using multiple (smaller) devices to setup a communication infrastructure in crisis environments is not a completely new idea. Many scientists mention this as the classic example of the use of **M**obile **A**d-hoc **Net**works (abbreviated with MANETs from now on). Several definitions of these type of networks exist, but they all come down to the same type of networks. For example, [PB94] defines MANETs as "...the cooperative engagement of a collection of Mobile Hosts without the required intervention of any centralized Access Point". For more information about these networks, see chapter 2 and [Kla04].

Making the structure of a network ad-hoc, makes the network flexible in the sense that if nodes are (almost) equal, the lack of one or more nodes does not always have to be a problem. Because of the multiple nodes in the network, communication can be handled in a multihop manner, which makes the lack of an infrastructure less fatal. In the situation of September 11th, one of the problems was that not al rescue workers got the message that the building was going to collapse, for some rescue workers unfortunately with fatal consequences. Furthermore, a distributed approach can make a system more fault tolerant for the case that part of the system goes down (in case of the example, a PDA can be broken e.g. if the rescue worker that is carrying it, dies) and communication is not dependent on a single node (i.e. no single point of failure).

Besides the communication, there is also the problem that there is no overview information about the crisis situation. This knowledge is initially not available. However, for this reason, rescue workers are usually sent into the area to explore this area. If this is done, the knowledge about the situation is distributed in the network. This knowledge is only useful when it can be interpreted by a human. Therefore, this knowledge is to be gathered and combined into one (shared) view of the world. For this to be possible, the information from the different observers is to be sent to other locations. This comes down to the need of a communication infrastructure again.

The problems we are dealing with can be broadly formulated as the lack of a communication infrastructure in a crisis situation, a lack of knowledge about the situation and the chaos that usually occurs in a these situations. These problems can, more specifically, be formulated as a question, "how can we setup a communication network in a crisis situation that can assist in the managing of the chaotic situation that occurs after a disaster". If this problem is seen in the light of the use of MANETs, the problem goes to some more technical aspects: how do we let people that walk more or less randomly through an area communicate with each other, how do we handle the not so imaginary situation that part of the network goes down? Despite their suitability for being applied in crisis situations, MANETs also have some disadvantages, such as the always changing topology, which makes routing and storage of data complex problems (see chapter 2).
In this thesis work, as opposed too most current approaches, we will not limit the scope to the network layer (i.e. optimizing multihop routing), but also consider the application layer, to come to an approach that handles some problems in a more intelligent manner. Furthermore, the application of a specific communication paradigm, blackboards, will be applied, to simplify the communication, while keeping it reliable.

1.2 Research topic

The problem described in the previous section results in the general research topic / assignment of this thesis. From the above description is it clear that there is a need for a communication and coordination infrastructure that can function in an environment that is chaotic and unreliable with respect to the components from which the system is built up. The general research topic can therefore be summarized as:

The design of a reliable ad-hoc communication and coordination infrastructure that can operate in a (chaotic) crisis situation.

The communication includes the storage and processing of data / information and access to information / knowledge. The above (high level) topic can be split up in some subtopics, that will be discussed throughout this thesis. First of all, there was a need to get more familiar with the environment and with certain techniques already available. This was done in a literature survey that primarily focused on the distribution of data, decisions and roles in wireless ad-hoc networks (see [Kla04]). Furthermore, a model / design of the situation had to be developed (see chapter 3 and 4), based on which a prototype of the solution was to be implemented. Finally, as a part of the implementation, experimenting was necessary to determine whether and how the solution worked.

The design and implementation of the solution to the problem (see chapter 4 and 6) can be divided into the hardware and network level on the one hand and into the application level on the other hand. Since the idea is to build a real life system, as opposed to a simulation, the functionalities on the lower levels will also have to be set up. The topic concerning the design and implementation of the solution described in this thesis can be split up in some more detailed subtopics which are discussed in this thesis. The research assignment is therefore the following:

- On the hardware and network level:
  - Designing and setting up an ad-hoc network of mobile devices.
- On the application level
  - Designing a structure in the topology of a MANET that is most appropriate in the situation at hand, a crisis situation. To realize this, a structure is to be designed that offers the facilities for an optimal communication infrastructure to *emerge*. Besides communication, the structure should offer facilities to run tasks in the appropriate place in the topology.
  - Designing a communication layer, based on a blackboard structure.
  - Design an agent network which provides a communication flow from the lower level (sensor) observations, up to high level decisions making in the control room.
  - Design a structure that takes into account the fact that nodes can go down, for which some measures have to be taken to prevent from data loss and to prevent that certain tasks/services stop executing while they are not finished/still necessary.
  - Design a structure that supports decentralized decision making.
  - Implement, as a proof of concept, a prototype that provides some of the necessary functionalities of the solution that is designed.
  - Test this prototype and experiment with the implementation to see whether the system and the chosen approach work.

The system to be developed will be a proof of concept and therefore some parts of the system will be implemented in a simple form. Furthermore, some applications that can be useful in crisis situations have already been developed in the past and these application will, as will be clear later, be used as test applications for the infrastructure. These applications will be described in section 6.3.3.

1.3 Structure of the document

In chapter 2 an overview of some related work will be given. For more related work, see [Kla04]. Chapter 3 gives a more detailed overview of the (crisis) situation that we take under consideration and the assumptions that are taken, to limit the problem, since we cannot solve all problems related to the problem definition at once. Furthermore, this chapter gives an overview of the solution to the problem defined in 1.1. In chapter 4 the design of the solution / implemented application is discussed. In the following chapter, chapter 5, the setup of the low level infrastructure is discussed. After discussing the implementation of the designed system in chapter 6, some experiments and tests are discussed in chapter 7. Finally in chapter 8 the conclusions, possible improvements and some ideas for further research are discussed.

Chapter 2

Related work

2.1 Introduction

In this chapter a short overview will be given of work related to the topics that will be discussed in this thesis. After a short discussion of crisis management, some literature will be discussed related to MANETs, routing, coordination, agent systems and communication. The discussion here will be kept somewhat concise. More about these topics can be found in [Kla04], where a more elaborate overview of literature related to these topics is given. In this literature survey, the topics are also discussed in the light of the problem domain, crisis situations, although there is less emphasis on this domain than there will be in the remainder of this thesis work.

2.2 Crisis management

As indicated in the project description, this thesis work is part of the Intelligent Systems project. Before the start of this thesis project this project was already started. Within this project research is being done on intelligent systems to help rescue workers. Some of the topics that have been studied or are currently topics of research projects are e.g. offering services based on icon communication (see [Sch05]), research on the workings in crisis situations(see [Cha05]), dynamic traffic routing, escape routing[1], dynamic creation of maps (see [vV05]). Besides this project, more research is done in the RESCUE project (see [res]), which also focuses on responding to crisis situations, and in the COMBINED project of the DECIS Lab (see [com]).
Both projects focus on managing the crisis, which is characterized by chaos. The focus of the COMBINED project is to assist rescue workers by coming up *"...with new concepts and systems that will be used by the crisis response organization of the future"*. The RESCUE project puts more emphasis on transforming *"...the ability of responding organizations to gather, manage, use, and disseminate information within emergency response networks and to the general public"*. The testbed within this project is called CAMAS. The idea of this system is visualized in figure 2.1.
On the left of the figure the input is described. This can be multimodal input from humans that, in this approach, serve as sensors. Based on this input, some events are extracted that are used in the analysis. Based on the analysis, that also uses GIS information (i.e. information from geographical information systems), a decision support system can be used for taking decisions. The result of this can be disseminated.
More work is also being done in other projects around the world, as can be concluded from the recent ISCRAM conference 2005 (International Community on information systems for crisis

[1] S. Strijdhaftig

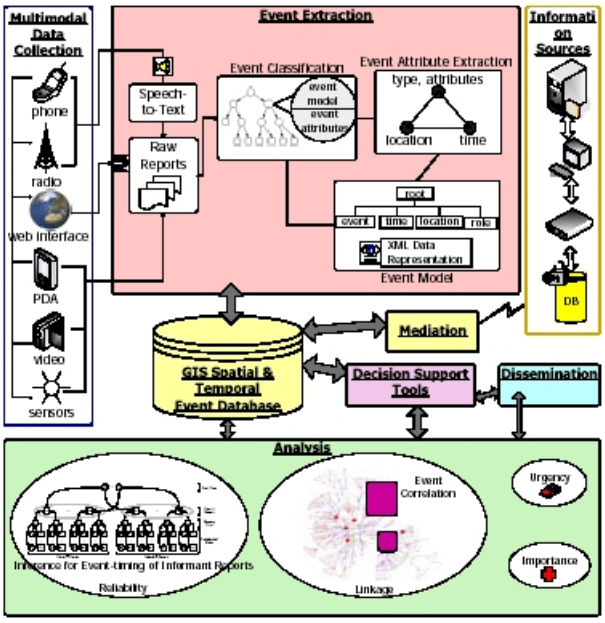


Figure 2.1: The CAMAS system (from [MBK+04])

response and crisis management). On this conference some work was discussed on knowlegde engineering aspects in crisis situations (i.e. how to get the right information at the right place). This aspect is, although on a somewhat lower level, also discussed in this thesis work.
Furthermore E.L. Quarantelli, from the University of Delaware, presents an overview of 50 years of research in the crisis domain in [Qua99].

2.3 Mobile Ad-hoc Networks

2.3.1 Mobile Ad-hoc Networks in general

As already indicated in the introduction, this thesis will search the solution for dealing with crisis situations using of Mobile Ad-hoc Networks, or, abbreviated, MANETs. In section 1.1 already an example of a definition of MANETs (from [PB94]) was mentioned. Let's recall it here for clarity:

> *"...the cooperative engagement of a collection of Mobile Hosts without the required intervention of any centralized Access Point".*

This definition puts the accent on two very important aspects of this type of network. First of all the networks have no central point through which all communication takes place, or in other words, the networks are distributed. The second property that is of importance concerns the "cooperation". This indicated that the hosts in the network should not only fulfil their own task but also cooperate, which implies that there should be some form of coordination, which in turn implies some form of communication between the different nodes.
A "collection of Mobile Hosts" can for example be a collection of laptops or **P**ersonal **D**ata **A**ssistants (PDAs). This latter type of devices has some constraints. They have, for example, limited battery power, limited antenna range and are less powerful than "normal" computers. To overcome these limitations, research has been done and some interesting directions for finding solutions have been opened. For example, if we have many of these devices to form a network, the limitations in the antenna range can be handled with special routing algorithms

that use closer nodes to pass messages to nodes that are not in direct range. See section 2.3.2 for more information about routing in MANETs.

Furthermore, if a network is built up of a collection of mobile devices, this has consequences for the reliability of the network. Not only can nodes go out of range of the network, which makes them unreachable, they can also go "down" unexpectedly (e.g. out of battery) and the network can split up in multiple parts (usually referred to as "group partition"). These problems have consequences such as the fact that data is not always accessible. Thus one of the problems is how to store and distribute data in a MANET. All these problems should be taken into account when dealing with MANETs, although usually (i.e. in most approaches) the focus is only on some of the problems, to limit the specific problem at hand.

Problems and challenges in MANETs are to guarantee Quality of Service (QoS), offer the possibility for internetworking (letting multiple different networks, including non-ad-hoc networks, cooperate), security, and, very importantly, reducing power consumption and (as indicated above) routing (see for example [Sun02]). The latter of these challenges, facilitating routing in MANETs, will be discussed in the next section. The reduction of power consumption is important since the devices from which a MANET is usually built up, are mobile and are thus using a battery. Due to the limited battery power of a device it can only be used for a limited amount of time. Much effort is put in research to come to solutions for MANETs that use less battery power (e.g. [GJG02]). Especially the use of the wireless network costs a lot of energy. Therefore many approaches focus on sending as few messages as possible, which is also necessary because of the limited bandwidth that mobile devices have.

2.3.2 Routing in MANETs

One of the main topics in the field of MANETs is routing, since facilitating reliable and stable communication is one of the main functionalities that is essential to the use of MANETs. Routing in MANETs is different from routing in wired networks. The major difference is that the range of the antennas of the nodes is limited. As a consequence of this, routing is usually done without *fixed* routers, but the different nodes serve as routers themselves. This means that packets/messages are sent through other nodes. This is called multihop routing. From this point in this report, ad-hoc routing will be considered as multihop routing, since it is the way it is almost always done. One of the problems of multihop routing is the fact that the network structure can change dynamically, or as it is put in [Tan03], connections can disappear suddenly, the topology can change and obstacles can block a connection. Furthermore, in [MMDM04] some of the routing problems in MANETs are discussed: The authors indicate that normal routing algorithms are not suitable for MANETs, since they cost too much bandwidth, they are not good for energy saving, not scalable, risk of redundancy and they cannot cope with the dynamics of MANETs.

For the problem of ad-hoc routing, several specific algorithms are developed, that, as indicated, use nodes as routers. Besides dealing with the dynamics, that are inherent to MANETs, trying to make these algorithms energy efficiency is an important issue in MANETs in general and especially in routing much effort is put in making the routing more energy efficient.
It is important to make a distinction between two types of routing algorithms. In [BB03] a distinction is made between *proactive* and *reactive* algorithms. The former are also called *table-driven* algorithms, the latter *on-demand* algorithms. Proactive algorithms are basically adjustments of algorithms for wired networks, with some enhancements, such as extra topology information to prevent loops and make the routes stable in a faster way, control flooding (do not flood to everybody if not necessary), combine features of the two classical algorithms and

varying the frequency and size of the update messages ([BB03]).
Reactive algorithms concern an approach in which there is no routing overhead if no data is sent. Paths are calculated only if there is a need for them, so the main difference is that nodes do *not* send their routing tables into the network periodically, as opposed to routing algorithms for "fixed" networks.

Some examples of routing algorithms are the Destination-Sequenced Distance-Vector routing (DSDV), Ad-hoc On-demand Distance Vector routing (AODV) , Dynamic Source Routing (DSR) and Ant Based Control routing (ABC). DSDV (see [PB94]) is an adjustment of the existing distance vector routing algorithm. Adjustments have been made to make it loopfree and to prevent the count-to-infinity problem. It uses sequence numbers to distinguish between older and newer messages. Furthermore, updates of the routing tables, the distance vectors, are only sent on important events (new routes found or old routes gone). This is thus clearly a proactive routing algorithm.
DSR (see [JM96])and AODV (see [PR99]) are both routing algorithms that are working on demand. The ABC routing algorithm is based on the emerging behaviour that ants exhibit when searching for food. For more information about this algorithm, please see ([GSB02], [Dib03] and [Kla04]). Since the AODV algorithm is quite popular and for the reason that this algorithm is also used in the implementation of the solution discussed in the remainder of this thesis works, it seems useful to highlight some of the properties of this algorithm.

The idea of the AODV algorithm is that nodes that are not on a path have no routing information. This remains the case until a node has to communicate with other nodes or it is on the route to other nodes. "HELLO" messages are used for becoming aware of the neighbor nodes.
The algorithm's primary objectives are ([PR99]):

- *To broadcast discovery packets only when necessary;*
- *To distinguish between local connectivity management (neighborhood detection) and general topology maintenance;*
- *To disseminate information about changes in local connectivity to those neighboring mobile nodes that are likely to need the information.*

Some of the advantages of the AODV routing algorithm are that it is scalable to many nodes, can respond to changes in topology, minimizes bandwidth use for control and guarantees loop free routing [PR99]. A somewhat more detailed overview of the algorithm will be given in section 5.4.1.

As indicated, this field of research is very large and many algorithms and improvements are available. Therefore there will not be given an overview of all these algorithms. For a more elaborate overview of routing algorithms, see for example [BB03].

2.4 Coordination

As indicated in chapter 1, the approach discussed in this thesis, focuses not primarily on the lower levels of the OSI model, i.e. routing, but more on the higher levels. One of the issues that is to be handled in the crisis situations, is that tasks are to be executed in a MANET. Furthermore, part of the research assignment is to "design a structure that supports decentralized decision making." For this reason coordination and cooperation are necessary in the solution. These issues is also research topics that have gained some more attention in the last

years. One of the main fields where these research topics are present is in robotics, where the issue of role assignment sometimes plays an important role. In [OPBS03] an overview is given of some important design aspects. The authors give the following definition of a role:
In an agent-based system, we define role as a class that defines a normative behavioral repertoire of an agent [OPBS03].

They continue their discussion with the concept of role change, of which an overview is given in table 2.1.

Table 2.1: Schematic overview of role changes (taken from [OPBS03])

Operation	Pre-state	Post-state
Classify	A and not B	A and B
Declassify	A and B	A and not B
Reclassify	A	B

This idea is further used in other approaches, of which a popular example is the soccer competition for robots. In [INPS03] the assignment of roles to robots in a robot competition is based on domain specific measures, such as distance to the ball etc. Based on these characteristics the utility is calculated for a role, e.g. the role of running to the ball. All nodes execute this calculation and send this information to all other nodes. This way they come to a decision what robot will take the role under consideration.

Another possible way to assign roles, is to use an auctioning mechanism, as in [HG00]. In this approach the idea is that the individual agent decide whether or not to participate in a certain group activity. In this approach, not only is taken into account whether the agent can give a good contribution, another aspect that is taken into account, is whether the roles of the agent can be taken over by other agents in case it has to give up it's old role.

A more theoretic approach that was also applied in the Robocup soccer domain is the use of coordination graphs. This approach divides a global payoff function in local payoff functions if this is possible. This converts the global problem of taking a decision to a number of local decision problems. The approach costs a lot of computational power. The situation is to be converted to a graph after which the problem is converted to propositional logic before it can be solved.

2.5 Agent systems

The solution to the problem defined in chapter 1 will be handled by a distributed approach, more specifically, by an agent oriented approach. By agents is meant (see [RN95]) anything that can perceive its environment through sensors and can act upon that through actuators. For implementing systems based on agents, special agent systems are available, two of which will be discussed in this section. The choice for these agent systems has been made based on the fact that the platforms are up to date (latest version from 2004/5), the platforms are free (unlike, for example, ADK (Tryllian)), the security is "in place" and the platforms are suitable for mobile devices [Str04].

2.5.1 JADE

JADE (Java Agent DEvelopment framework) is middleware for making agent based systems that can work both in wired as in wireless networks [BCPR03]. It provides the basics for building distributed agent systems that can communicate and dynamically discover other agents. It supports asynchronous messaging.

JADE takes care of security and provides abstract classes to deal with the implementation of complex conversations[2]. Furthermore, *"the platform also includes a naming service (ensuring each agent has a unique name) and a yellow pages service"* [BCPR03]. In [Cai03a] is stated that the system should run on a main container. All agents have to register with this main container. This container provides the yellow pages service to the other agents. In the latest version of JADE it is also possible to replicate the main container over multiple nodes, in case the node with the main container crashes.

For mobile environments, JADE has an extension called LEAP. This enables the use of JADE on mobile devices that have limited memory and limited storage capabilities. It allows developers to split the container of an agent in a front end and a back end. This front end can then be on a mobile device, and the back end can be on a part of the fixed infrastructure. Of course the container of an agent can also be completely on the mobile device if the mobile device has enough storage space and processing capabilities. See figure 2.2 for an illustration of this.

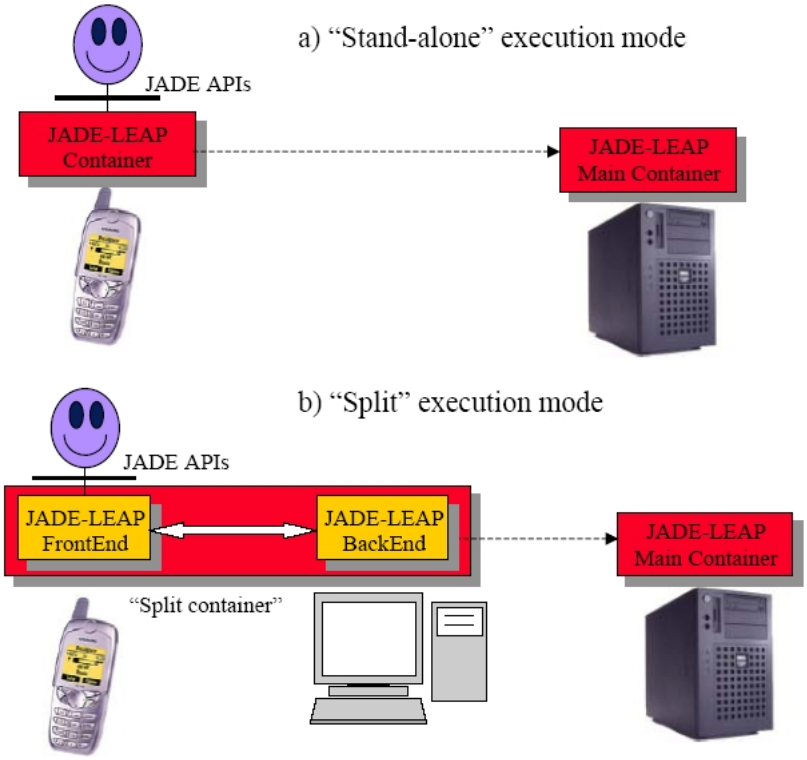


Figure 2.2: 2 possibilities for LEAP (taken from [Cai03b])

[2] *"JADE provides a set of skeletons of typical interaction patterns to perform specific tasks, such as negotiations, auctions and task delegation."* [BCPR03]

2.5.2 Cougaar

Cougaar is the abbreviation for **Cog**nitive **A**gent **Ar**chitecture. It is a software architecture for developing distributed agent based systems. The ideas come from the military field. It was made for DARPA[3] and the requirements for it were to be robust, secure and scalable. By robustness is meant that the platform should be able to "survive" if some of the nodes are lost.

The agents in Cougaar communicate with each other via an asynchronous, built in message passing system. *"Cougaar agents cooperate with one another to solve a particular problem, storing the shared solution in a distributed fashion across the agents"* [cou04]. A group of agents that interact to solve a problem together is called a society. A society can consist of one or more communities. Communities are groups of agents that work together on particular task or goal (a sort of "sub-society"). Societies have a DNS-like namespace. Agents in communities speak a "common dialect" [cou04] which means that only the agents of a community can understand each other.

Agents in Cougaar have plugins that provide the "behavior and business logic" of the agent and a "partitioned distributed" blackboard [cou04]. Schematically an agent looks as in figure 2.3.

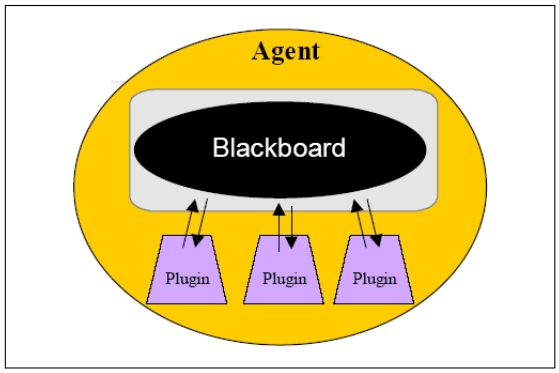


Figure 2.3: Agent internal structure (taken from [cou04])

Every agent has its own local blackboard which it can share with other agents if it wants to. Plugins can communicate with the agent and with other plugins using a publish/subscribe mechanism. Data can be published on the blackboard by a plugin and other plugins can subscribe to this data. Removing and updating of messages on the blackboard also has to be done by the plugins. The communication between agents is done point to point via message transfer facilities that are offered by Cougaar.

Objects that are stored in the network can be represented in multiple agents at a time, but only at one place does the actual managing of this object take place. Cougaar works in a distributed way and asynchronously. This means that consistency is not guaranteed by the platform (but must be implemented by developers if desired).

For communication Cougaar has different mechanisms, including the White pages (similar to DNS), Yellow pages (attribute based searching for other agents), local discovery (discovering nearby agent) and peer to peer search (search adjacent agents). Cougaar requires Java version 1.4.2.

[3]Defense Advanced Research Projects Agency

2.6 Communication

2.6.1 Communication in general

In the field of communication in MANETs a lot of research is ongoing. Many research approaches focus on the lower levels of the OSI model, of which the network level, i.e. the routing, is already discussed in section 2.3.2. Besides this, on the MAC layer there is also ongoing research considering the handling of messages from different nodes on antenna level. On the higher levels, i.e. on the level of the application and of the way of communicating in MANETs also some research is done. After a short discussion of the new communication system that is recently introduced in the Netherlands, the C2000 system, some attention will be given to blackboards and of the use of blackboards to actually solve problems.

2.6.2 C2000

The C2000 system is a communication system that is developed for the rescue services in the Netherlands. This communication system, which was to be introduced in 2000, but was delayed until 2003/2004, provides communication to the users, which are the police, the fire department, ambulance service, and military police. The system offers these services over a radio network that is supported by a fixed backbone (which can make the system unusable as was seen in February 2004, when the system was not used for a week during a failure of this backbone). The main components from which the system exists are walkie-talkies, car phones and the system provides gateways to link it to e.g. the phone network. The system provides coverage in the whole country, but the communication is not guaranteed to work indoors (see [c20]).

2.6.3 Blackboards

In the field of blackboard communication, some work is already done. The initial idea of a shared piece of memory that was visible to multiple entities is relatively old. In [Sch95] the discussion of blackboards is already elaborated to the concepts of virtual and distributed blackboards. A useful definition of blackboards can for example already be found in work from 1992. The definition of blackboards from [McM92] is the following:

> *"A Blackboard System can be viewed as a collection of intelligent agents who are gathered around a blackboard, looking at pieces of information written on it, thinking about the current state of the solution, and writing their conclusions on the blackboard as they generate them."*[McM92].

According to [McM92] *"...the major issue in designing a distributed blackboard system is deciding what information to communicate, where to store it, and when and where to send it."*.

In [McM92] a distinction is made between distributed blackboards with a centralized and a distributed blackboard data structure. In the latter case, parts of the blackboard are located at the different nodes. Some of the problems that this introduces are the large number of messages that have to be sent over the network if the systems are tightly coupled and the extra trouble in keeping the system consistent.
Concerning the implementation, [Sch95] discerns *four* ways to implement blackboards:

1. Using shared memory (The blackboard can become a bottleneck, serializing is necessary if locking is used).

2. Distributed approach (Every node has a part of the blackboard → much communication, can be solved by partial overlap, but this spoils the blackboard concept.).

3. Blackboard server (One central node has all info, looks like 1 blackboard.).

4. Virtual blackboards (Combination of 2 and 3, virtual area controlled by some of the nodes.).

Recently the application of blackboards in mobile environments has gained more and more attention. Especially because of the mobile property of the nodes in wireless networks, virtual and distributed blackboards are becoming more and more popular. However, many existing blackboard systems were not developed for wireless and mobile environments, i.e. environments in which the nodes are changing location. Some implementations of systems that do provide blackboard functionalities in mobile and wireless environments are for example Lime (see [lim] and [Bel04]) and a variant of this software, Limone (see [NFM04]). Both systems are made for mobile environments, although the systems were not developed specifically for MANETs. The use of the software is primarily limited to use on laptops that are in each others range and that can, in this way communicate. As indicated in 2.3.2 communication in MANETs is usually done multihop. This is also the case in this thesis work and this is thus a way to test the performance of virtual/distributed blackboards in MANETs. Other tools and systems that can help in the development of virtual/distributed blackboards are e.g. Cougaar (see [cou04]) and JXTA (see [jxt04]).

Another approach that uses the idea of blackboards can be found in [BMB+05]. The authors point out that the discussion of these kinds of systems often remains limited to the design of effective communication and the concept of blackboards is not seen in the broader context of an infrastructure. The latter observation will also be central in this thesis, since the solution built in this thesis also focuses on a broader context.
The approach in [BMB+05] deals with a blackboard approach to support multiple users collaborating around certain places and exchanging information. The purpose is to share information related to a specific place. For this purpose users can use laptops and tablet PCs in a peer-to-peer network. To provide the user with location specific information, their location can be determined with GPS or users have to manually enter their location information. Despite the similar architecture, i.e. a peer-to-peer network with the blackboard paradigm to communicate, the system is quite different from the systems necessary as a solution to the problem discussed in section 1.1. The main difference is the fact that the approach discussed in this thesis, requires a data / knowledge *flow* toward a location, instead of providing location based information to different locations. This will become clear in the following two chapters.

Chapter 3

System overview

In this chapter a global overview is given of the solution for the problem described in section 1.1. First a short overview is given of some assumptions that have been taken to limit the problem at hand. Next a global overview of the solution will be given, after which the communication flow and the knowledge flow will be discussed in more detail. The chapter will continue with the scope to which this thesis work will be limited. Finally, after presenting some extra requirements and constraints, a short overview of the development process will be given.

3.1 Assumptions

In this section and section 3.2 we give our view of the working and properties of a crisis situation. It is good to have this in mind. This view can help in finding reasonable requirements and constraints for systems that can be used to solve the problems at hand. For the problem domain, there was not much information available. The techniques that are considered are relatively new and due to recent terrorist attacks on e.g. New York, Bali, London and the recent "Tsunami" only recently more attention has gone out to handling these kinds of situations. Although there are many scientific papers about some of the techniques, actually implementing certain systems for these situations is not very common. Furthermore, simply asking the stakeholders what kind of applications they need, leads to some good ideas and advice, but they (fortunately) have little experience with these kinds of situations. Besides this, since the terrorist attacks on the World Trade Center in New York in 2001, people concerning the managing of these situations, are not very helpful anymore.
Therefore, this model is partly based on other research from the DECIS Lab (see [dec]), on ongoing research from the Delft University of Technology (see [Cha05]), on common sense and partly on the things we have learned from (small) crisis situations in the past. Besides an overview of the assumptions, this section will also give a short motivation to explain why we choose for these assumptions.

To build a system that can be helpful in crisis situations, we have made some assumptions which limit the problem to manage and which characterize the situation that is imagined, for which a solution is thus to be found. These assumptions are the following (based on [Rot04]):

- The existing communication infrastructure is down. The rescue workers only have their PDAs to communicate.
- The group of rescue people has no central control or command center, which they can directly reach.
- The decision making is decentralized.

- The environment is changing continuously.
- Detailed location information is *not* available (so no GPS).
- The communication network is *to some extent* ad-hoc. Some parts of the network, the core(s), are relatively stable, other parts around it are highly dynamic.

The first assumption seems reasonable, since in case of earthquakes or other natural disasters, we cannot count on the fixed/wired infrastructure. This might be destroyed. For example, we do not know in advance whether the GSM network will still work, since an earthquake can cause the GSM antennas to break. Furthermore, even if the GSM network would still work, it is not sure whether it can be used, since this might be overloaded (which was, for example, the case at the "simple" bombalert at Ikea in December 2002). Similar reasons can cause the UMTS network not to work anymore when a disaster has happened. Both examples have the assumption that these networks are available, which in many countries is not the case, of course.

As opposed to a central control center, a local control center seems logically, since someone has to tell where the rescue people should go (see also next section). Furthermore, as already indicated in chapter 1, this has become a standard procedure for policemen / firemen. The decentralized decisions are necessary, since there should be no (or at least as little as possible) centralized (and because of that usually weak) parts in the network. Centralized systems are usually better and more efficient, but in our situation a centralized system is not an option, because we cannot guarantee an optimal communication network, since in a crisis situation the environment is too dynamic. This means that sometimes / usually nodes are not reachable. This should be taken into account when designing applications that rescue workers should be able to use in a real crisis situation.

The last assumption, that an ad-hoc network is the structure that is appropriate for the situation, is an idea that is agreed upon by many scientists, who agree that these types of networks are useful in crisis situations (see for example [GJG02], [HHL02] and [KMP02]). Some people even consider crisis situations (together with military applications) even as "classical" examples of applications of MANETs ([GSB02]). The motivation is further explained in section 3.2.1.

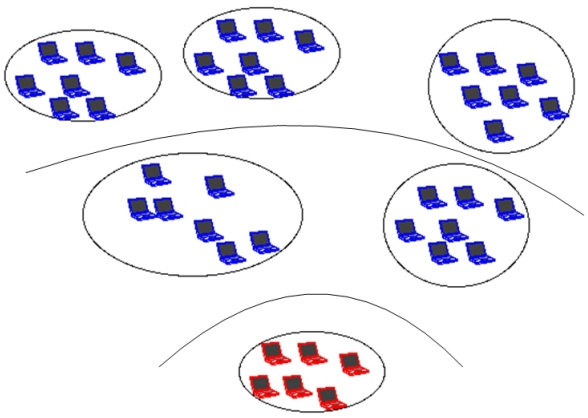


Figure 3.1: A network with a stable core

The situation that is imagined when rescue workers are in the field with their PDAs, is schematically illustrated in figure 3.1. The red nodes represent the local command center. This is the

stable part of the ad-hoc network. Nodes can leave this part, but it is assumed that this does not happen suddenly, but is announced or happens gradually. The blue nodes in the figure represent the groups of nodes that form the dynamic part of the network. The nodes move through the area and the people carrying the PDAs execute different tasks, like exploring, locating and helping victims, extinguishing fires, setting up roadblock etc. A more elaborate description of how the situation actually works is indicated in the next section.

3.2 Proposed solution

The solution to the communication problem in crisis situations and the fact that data can get lost due to nodes that can go down, will be handled, as indicated, by a MANET. In the field of MANETs, much research is done on the lower layers of the OSI reference model (see [Tan03] for an explanation of the OSI model). Especially on the network layer (in the field of multihop routing), and on the MAC level, much research has already been done. On application level, less research results are available. Although this thesis mostly focuses on the application level, the system is based on other levels, from which certain information is also used. Therefore this can be called a cross layered approach. The communication will be done via a virtual blackboard structure.

3.2.1 Network structure

The chosen network structure, a MANET, is chosen for multiple reasons. The primary reason is that it is a structure that is completely ad-hoc. This means that it does not depend on any underlying infrastructure. This means that it is really ad-hoc and that it is not vulnerable to e.g. the case of a power break down or a directed terrorist attack on the infrastructure. Furthermore, as indicated in section 3.1, in a centralized system we cannot guarantee an optimal communication network, since in a crisis situation the environment is too dynamic. In [RDFT05] the PIRA model is described, which makes use of an ad-hoc network of PDAs that are connected to a server. This connection is to be established via GSM. The model is summarized in figure 3.2.

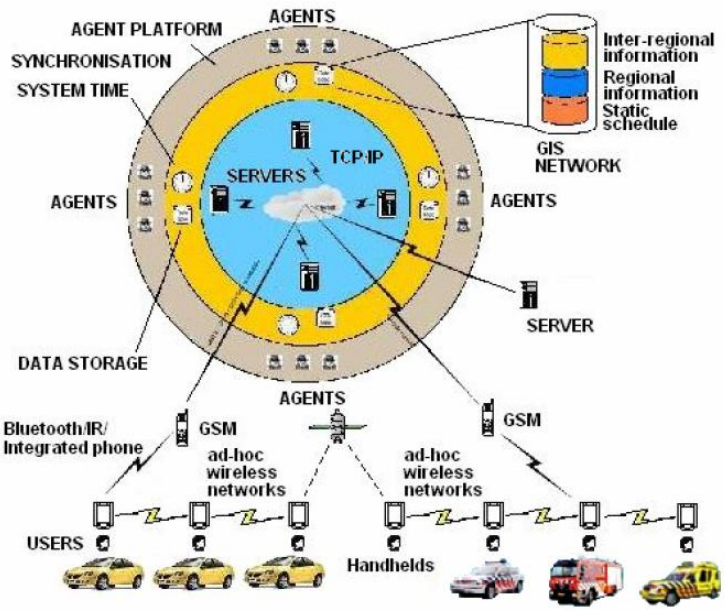


Figure 3.2: The PIRA system (taken from [RDFT05])

The approach described in this thesis is different than the approach described in [RDFT05]. The main differences are that there is no wired "backend" consisting of one of more servers and the GSM network is not used. It thus does not make use of the wired infrastructure such as the servers in the PIRA model, which means that there is no dependency on any form of existing infrastructure. Furthermore, the PIRA approach depends on the use of the GSM network that might not always be available. This can be overloaded, or this can be (partially) destroyed in the case of an earthquake or directed terrorist attack. Similar objections apply for the wired infrastructure at a crisis scene: this might not be available for the same reason. Therefore a completely distributed ad-hoc structure was chosen. This structure is illustrated in figure 3.3. This figure will be discussed in more detail in section 3.2.4.

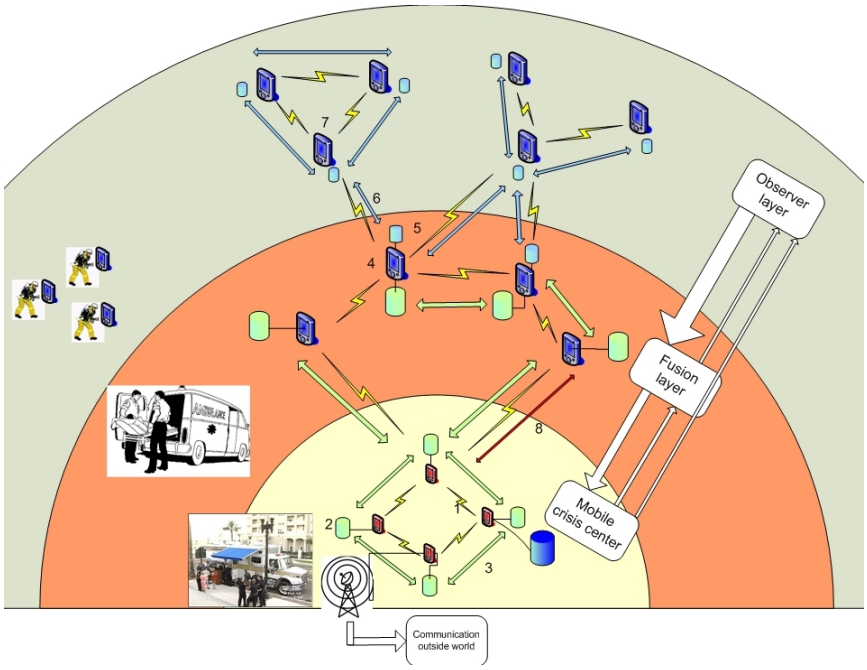


Figure 3.3: Approach in this thesis work

For enabling communication in these types of networks, standard agent frameworks are not suitable. Most existing agent systems are not specifically developed for wireless environments, but primarily for wired networks, although some wireless extensions are available. Furthermore, these agent frameworks are not distributed, but "star"-like systems. Certain functionalities are centralized, such as the services for finding other agents, usually called "white pages" and "yellow pages" (the former is for lookup of agent by name, the latter for lookup by offered service). In the case of JADE (Leap), there should, for example, always be a so called "main container". Even though this "main container" can, in the newest version (which does not run on the target devices, the Sharp Zaurus), be replicated, this remains an approach that is a star network, something that is not suitable for the dynamic part of the network. Loosing one of the nodes that runs these services, can be catastrophic for the network. In the chosen structure this way of communicating could be used in the stable core of the network. Existing agent systems (e.g. JADE (Leap) or Cougaar (see section 2.5, [Kla04] and [Str04] for a summary of some of the properties of these systems)) could be used for this, although this is not done in the developed system, since it seems better to make the whole system using the same underlying structure since in this way the crisis center(s) are "really" mobile (i.e. the structure of the network can change, including the crisis center).

Furthermore, some existing agent systems, like Cougaar, only run on Java version 1.4, which is

not available for the target devices, the Sharp Zaurus. Therefore the solution will not use one of these existing, mostly "star"-like agent frameworks, but use, also in the part of the network that is assumed to be more stable, a truly distributed approach to facilitate communication for services in the crisis network. To facilitate this communication, the system must have a multi-hop routing algorithm, to make the system suitable for a MANET. The services in the network will not be implemented from scratch, but some existing services from the Delft University were used and adjusted to run in the proposed network structure (see 6.3.3).

3.2.2 Communication using blackboards

The system will be implemented using communication and coordination via *blackboards*. This mechanism is chosen because of the fact that it makes sending, reading and processing of messages independent of time and place. Furthermore, this data-driven approach is considered to be effective when there is no strong coupling between the producers of information and the consumers of the information in the network ([Mil05]).

Especially when networks are dynamically changing and nodes can go down, a centralized system is not desirable as already mentioned in section 3.1. Storing all the data at one place is not desirable, since it can mean that if this central point goes down, all data is lost. Furthermore, this central point might not be reachable at all times and if it is, this would have to handle a huge amount of information from all the reporters in the network. The other extreme is that all the data is stored at all nodes. This is not only a lot of redundancy, this would also lead to too much network traffic, which would overload the whole network. Another limitation is that the nodes in the network are devices with limited processing capability. If all nodes were to act as a replica of every node, this would be too much for the limited storage and processing capability of the nodes.

For the reasons mentioned in section 3.1 and 3.2.1, the choice was made for making the system distributed instead of centralized. Since the structure should be distributed and at the same time, the communication takes place via a blackboard-like structure, we are dealing with a virtual/distributed blackboard structure. This virtual / distributed blackboard structure allows the nodes in the system to stay in touch, despite changes in the connections between nodes due to mobility. Furthermore, in the light of the problem domain, it makes it easy to get information about the situation for people in a mobile crisis center. They just need to have access to the blackboard and do not have to know exactly where the different pieces of data have to be found. This can be quite useful in cases where there are multiple sources. All they then need to know is how to get the information from the blackboard. A more detailed overview of the communication in the system will be discussed in 3.2.4.

3.2.3 Introducing a topology

As already indicated in section 1.2, a structure will be designed in the topology of the MANET that is most appropriate in the situation at hand, a crisis situation. Based on earlier research in the Intelligent systems project and the DECIS project (see e.g. [dec] and [Cha05]), we have come to a topology in which we discern three layers of nodes, as illustrated in figure 3.4.
The three layers consist of:

- Observer nodes
- Preprocessing / fusion nodes
- Mobile crisis center nodes

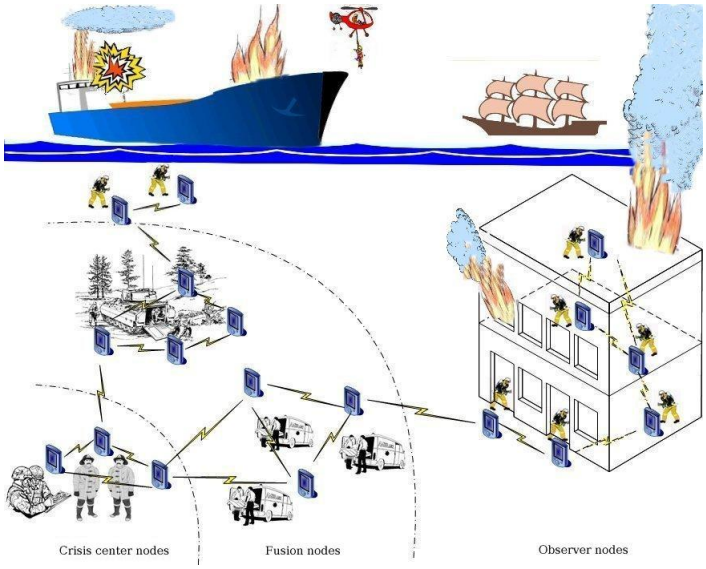


Figure 3.4: An overview of the topology in the total system

The first layer of the network concerns the outer part of the network. This concerns the sensor nodes. By this we mean the field workers that are doing the actual rescue work in buildings, tunnels etc. These are the observers that should report about the crisis situation. This layer of the network contains human observers that can report about the situation, but can also consist of, for example, chemical sensors that can detect gas and smoke[1]. Another way to report about the situation is by using a camera. The images of the camera can be made available as well, which can give detailed information about the situation.

Eventually, all this sensor information is to be sent to the mobile crisis center, which is assumed to be the most stable part in the network. This is the part of the network where the people that are locally in charge should be, to get an overview of the situation and manage the situation. From the overview information in the network the people in this local crisis center should be able to define more/new actions for the rescue workers, that can be sent into the network again to be delivered to the rescue workers.

The third layer, the middle layer, that we discern, involves the part of the network that is somewhat stable, although nodes are assumed to disappear more often than the nodes in the mobile crisis center. The assumption is that this does not happen as often as in the case of the outer ring of the network that is assumed to be very dynamic. However, as in the case of the mobile crisis center, the possibility cannot be ruled out. The function of this layer is to preprocess the data that comes in from the observer nodes. This data can come from multiple sources, as indicated above. The combining of this information into one (local) world model, which we will call *fusion* in this thesis, is one of the main functions of the middle layer. The idea is to combine information from the nodes in the outer layer to come to a local world model.

Furthermore, the nodes in the middle ring also take care of the "messaging" task. By this we mean that in the case of a MANET no fixed infrastructure is available. Since wireless connections only have limited range, the distances between the nodes may not be too large. If the middle ring is missing, wireless connections might not be sufficient to offer the desired connectivity of the nodes in the network to be able to communicate. Therefore the middle ring

[1]Currently some research is done in our project concerning chemical gas detection and spreading ([Ben05]).

is essential to the functioning of the network. Furthermore, considering the tasks of people that are supposed to carry these nodes, the nodes can be very useful in the network. If we are, for example, dealing with a crisis situation in a building (similar to the 9-11 attacks), a certain group of rescue workers is constantly getting into the building to get wounded people out of the building. When they are inside they are likely to be in wireless range of some of the nodes inside the building. They can then fulfil their role as fusion node. When they are taking the wounded people outside, they are closer to the mobile crisis center and the fused data can be delivered there.

Besides providing the communication infrastructure for crisis networks, the solution will also concern the assignment of tasks within the network. Some tasks are specifically requested by the user, and should thus be executed on the node under consideration. However, some tasks, including some tasks for keeping the services in the network "up", can be executed at different places, although the place in the topology might be important. This assignment of tasks to nodes, will therefore also be taken into account in the system. This will all be done with the model discussed above (and in section 3.1) in mind. Eventually the idea is that the data that is generated by the users (via the services) will safely get to the location where it is requested.

Now that a short overview of the system is given, the next two sections, section 3.2.4 and 3.2.5, will give a more detailed overview of the communication flow and the knowledge flow in the proposed system.

3.2.4 Communication flow in the system

In figure 3.5 an overview of the communication in the designed system is given. This picture is divided in the same three layers as in section 3.2.3. The lower part of the picture indicates the mobile crisis center. The connections between these nodes, which are assumed to be reasonably stable, are indicated in yellow (e.g. number 1 in the picture). The PDAs of the mobile crisis center (indicated in red) do not only serve as a place to send all the data, these are also the nodes that have connections with the outside world. The blackboard communication between the nodes is indicated by the green lines in the figure (e.g. 3).

The nodes in the crisis center are (parts of the time) connected with the nodes in the fusion layer. These nodes (e.g. 4) are also looking at the blackboard (if reachable). Since they are closer to the nodes in the observer layer, they can also (part of the time) communicate with the nodes in the observer layer (e.g. 7). The communication with these nodes is done via local blackboards, of which the parts are indicated in light blue (e.g. 5). Again the communication is done via the blackboard (e.g. 6). Besides the communication via the blackboard, there is also a possibility to contact nodes directly (if desired) as indicated by arrow 8. The communication flow in the system can then be visualized as in the right part of the figure. The communication flow from the observer layer is drawn with a thick arrow to indicate that a lot of data messages are flowing from this layer to the fusion layer. After preprocessing the amount of data has decreased. Therefore the arrow from the fusion layer to the mobile crisis center is drawn less thick. The arrows back into the network are even thinner, since part of the data is stored in the crisis center and is not relevant to the observers. Besides part of the relevant information, the (human) decisions taken in the crisis center are to be sent to the nodes in the fusion layer and the observer layer. This is captured by the arrows from the crisis center to the fusion nodes and the observer nodes.

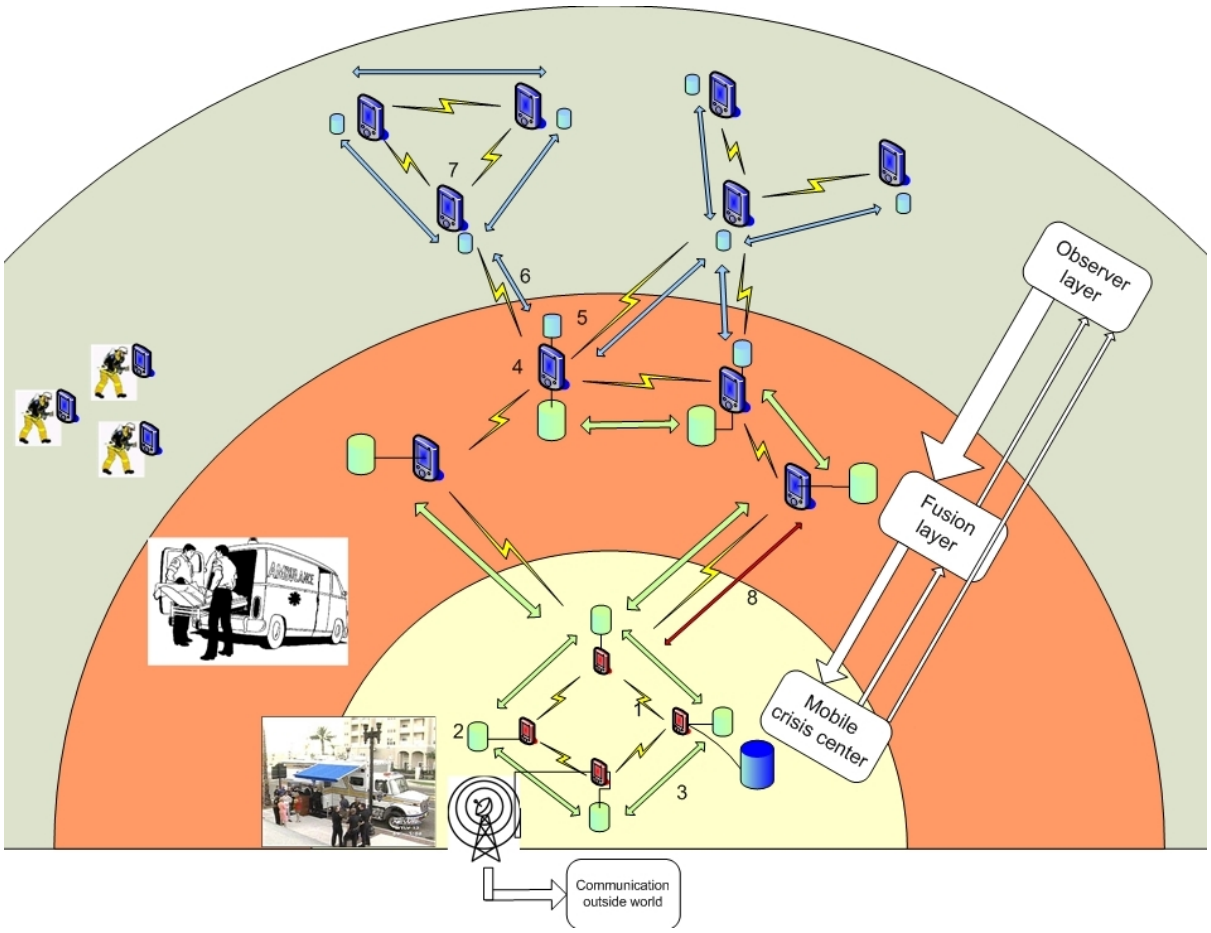


Figure 3.5: Overview of the communication flow

3.2.5 Knowledge flow

Now that the communication is explained, this section will give a more detailed overview of the flow of the knowledge and the distributed information in the network. The knowlegde is distributed via a distributed blackboard. This structure requires a reasonable amount of message passing and data storage. To take care of the processing of information the design, which is already introduced in section 3.2.3, is visualized in figure 3.6.
The three layers consist of systems that are all in some form intelligent. On top of this picture, above layer A, there is the “world”. In the case of a crisis situations there is thus a crisis in this world. This can be seen as an event. This event is observed by the people at the location of the crisis and by e.g. cameras and sensors etc. The people that observe the crisis have their own view of the world in their minds, their world model. This can loosely be defined as ”their best estimate of the world”. This world model is different for the different people, such as the policemen, the firemen and “civilians”.

This world model is to be translated by using concepts and relations. These should be limited, since computers cannot reason with as many and as complex concepts as humans can. Therefore, the world model has to be translated with a number of concepts and relations, which is done by the applications. For example, in the Lingua application (an application that allows the user to make sentences with the use of icons, see 6.3.3), the concepts are the icons that the user can input. The grammar that is included in the application, to form sentences, can be considered as the relations that can be defined between the concepts in the form of a grammar

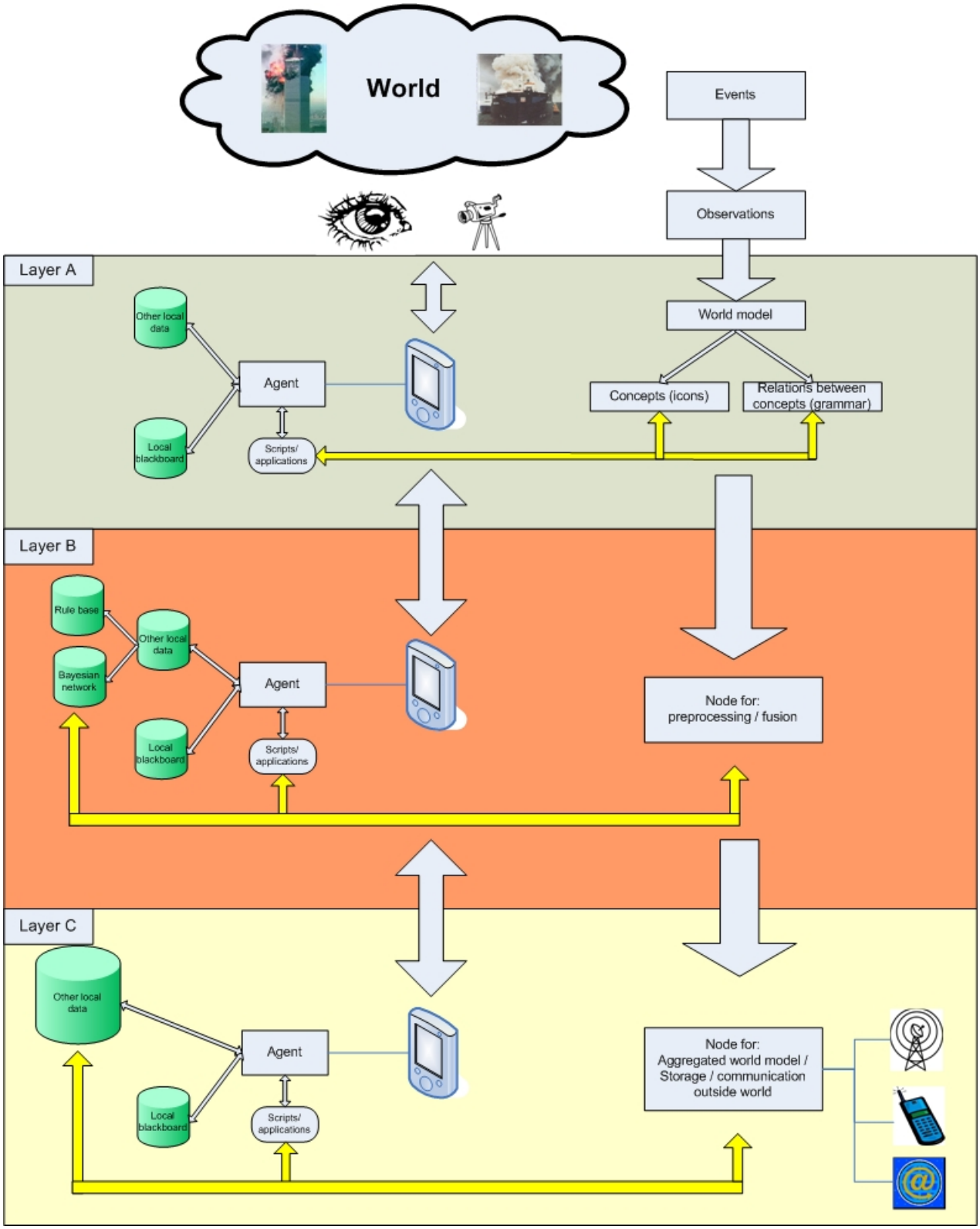


Figure 3.6: The knowledge flow in the system

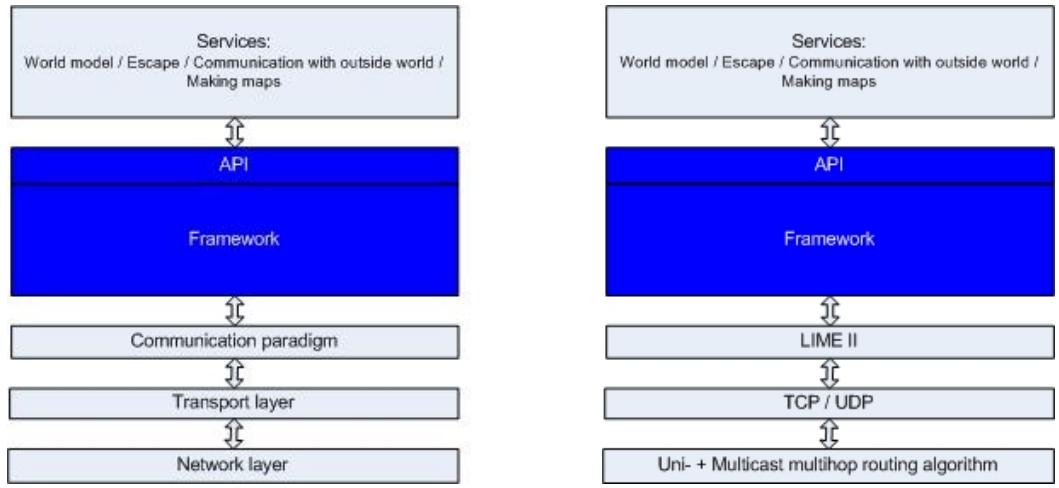
(e.g. a house cannot be in a bomb, but there can be a bomb in a house). This world model is thus converted to concepts and relations that other parts of the computer system can "understand". The available concepts and relations in the system can be made dependent of the user (e.g. different icons for policemen and firemen). The agents in this layer of the system have, besides a local part of the blackboard and other data to fulfil their task, a number of scripts/ applications that can be executed. These thus limit the number of concepts and relations between those concepts, which can be used as input of knowledge into the system. This input is then sent to the next layer.

In this next layer, layer B, the preprocessing or fusion of data takes place. This layer can, for example, have nodes that try to solve conflicts in the different input that is received from different agents in layer A. Data from different observers, different locations and from different times, are to be fused to come to the most probable (local) world model (e.g. in case of icons: detect duplicates, solve conflicts, complete information etc.). This should be done somewhat close to layer A, since these nodes have information about the local situation. The nodes in the crisis center usually do not have a good idea what is happening in the outer part of the network, since it can be too far away. Therefore, part of the processing should be done closer to the nodes that are actually observing the environment. One of the forms of preprocessing, fusion, can be based on, for example, expert system rules or Bayesian networks, which have to be locally stored on the nodes in e.g. XML files. Since the local storage capacity is limited, in case of applications that use much storage room, older messages are, in case that this is necessary, to be discarded (or instead of basing the decision on time, this can also be done on e.g. priority).

The lowest layer in the figure, layer C, concerns the mobile crisis centers. In this layer the data from layer B is received and further processed to come to an aggregated world model. This world model should (again) be the "best estimate" of the world, i.e. the (whole) crisis situation. Furthermore, this layer is to store the data that comes in from the outer part of the network. The data that is to be collected in the crisis center, is stored distributed over the nodes. Each node has its own part of the blackboard and data can be stored there. The nodes also have background knowledge to process data and provide additional information to nodes from the other layers. Storing information is necessary since, the information can then later be re-used if necessary (e.g. information about the dynamic floor plan can be used for escape routing). Furthermore, the mobile crisis center should answer requests from the observer layer. This can be, for example, be done by providing information about how to get out of a building/the area. This is an important functionality in a crisis network, since one of the main tasks of the rescue workers is to get an idea what the situation is (based on reports) and how to save as many people by getting them out of the area. Besides gathering information in layer C, it is also the task of this layer to communicate this with the outside world, e.g. to inform people that have the overall lead (ministers etc.).

3.2.6 Scope

The total system that is envisioned in the project should offer multiple functionalities. These functionalities are located at different levels. In figure 3.7 an overview of the different layers in the system is given. All these layers are (partially) running on all the nodes in the MANET.
The top layer concerns the services that the rescue workers have running on their PDAs (every PDA has a subset of the total services running).



(a) General layers

(b) More specific

Figure 3.7: The layers in the proposed system

These services concern for example:

- Provide information about the local situation (see for example [Sch05]).
- Updating and reasoning about the world model[2].
- Escape routing[3].
- Making maps dynamically from local information (see for example [vV05]).
- Communication with outside world.

The communication paradigm is mentioned separately, since it makes quite a difference how the communication takes place. In the implemented system, the communication will take place via the blackboard paradigm for which the system Lime (middleware that enables making a blackboard system) will be used (see section 5.2). At first sight the transport and network layer seem to be two layers that are not really important to the system, since they are normally provided by an operating system. However, in the case of MANETs we want to have some more functionalities concerning the lower layers, namely multihop routing, as discussed in section 2.3.2 and also in, among others, [Kla04]. Because of this, these lower layers will also be addressed in this thesis. Since much research has already been done on these algorithms, an existing implementation will be used (see section 5.4.1). It might be possible to improve parts of existing routing algorithms, but this is not the focus of this research.

The layer in blue, the framework (and API) layer, concerns the scope of the solution discussed in this chapter. This will take care of part of the communication and the processing in the network. As indicated above, much research is already done on the lower levels. On the application layer some systems are available, but not many take into account the dynamic nature of MANETs. This blue layer should be the layer that connects them and provide a fault tolerant platform on which the services can run. One possible way this layer can be implemented is discussed in the rest of this thesis report.

[2] J. Chau
[3] B. Tatomir

3.2.7 Simulation vs. Real life system

One of the first choices that was to be made, was whether to build a simulation or a real life system. Simulations are very handy to test algorithms, such as routing algorithms. However, when dealing with larger systems or algorithms, simulations can sometimes not handle larger datasets. Furthermore, simplified simulations for wireless environments can give inaccurate results as is shown in [CBH+04]. Besides the fact that simulations of wireless environments can be inaccurate, the solution that is built was also not really suitable for a simulation (because of, for example, fitness calculations based on local performance information and assigning roles in MANETs, as will become clear later). If this solution was to be simulated, this would mean that an enormous amount of factors should have been simulated/guessed. It was expected that this would lead to overall inaccurate results. For this reason, many of the experiments were executed by walking around with actual PDAs and by testing the performance and working of the system in this manner.

Furthermore, from the side of the TU Delft, the applications ISME en Lingua were build for the disaster management project, but there was a need to integrate them in one (distributed) network, that should be useful in crisis situations. Since these applications were both only running on one computer (instead of distributed), it was also a chance to find out how the applications work in practice. Especially the chosen way of communication, using blackboards, is not very suitable for simulation environments, since the fact that it is distributed has implications for the consistency and reliability. Using a simulation would therefore not give a good idea of the actual performance of the system. Besides consistency, factors as speed of the blackboard, speed of the applications on the Zaurus devices and timing problems could all be determined during experiments. This, in combination with the chosen solution, led to the choice for a real system. It will, however, not be an attempt to build a complete and fully functional agent system, but will be a "proof of concept" to show that the chosen communication paradigm works, the way of role assignment works and that a structure can be made into the network to control the chaos that is inherent to crisis situations.

3.3 Extra requirements and constraints

Now that an overview of the solution is given, some more technical requirements to which the system should comply, have to be determined. This can be considered as an extension to the requirements mentioned in section 1.2. These extra requirements and contraints can mostly be determined from the overview described in section 3.1 and 3.2:

- The system should be suitable for a MANET. Since one of the main ideas of the system is that it should be distributed, this means that there should be no central server that is a bottleneck and a central point of failure.
- The mobile devices, the PDAs, communicate via WiFi connections (802.11b network).
- The applications/services that are running in the network should not go down if one node would go down. (This situation is not imaginary since if people are carrying PDAs around in a crisis situation, people can drop these or be killed themselves.)
- If possible, measurements should be taken to prevent services from going down.
- The communication overhead should be low, since it costs a lot of battery power to send messages.

- The systems should also run on the Sharp Zaurus PDAs (type SL-C760 and SL-C860), running a Linux OS with kernel version 2.4.18.
- Due to this previous constraint, the system should be implemented in Java 1.3.1. Unfortunately no higher version of Java runs on the Zaurus.

3.4 Development process

One of the parts of the assignment was to build a prototype. As indicated, a real life system was the target of the implementation. Due to the fact that the design of the system is, besides a solution to the problem mentioned in section 1.1, a *blueprint* for the whole crisis management project within the Intelligent Systems project, the design and implementation enclosed a lot of different layers as indicated in section 3.2.6 and as will be explained in the next chapters. Due to time limitations it is not possible to build everything from scratch, something that would, of course, be preferred. One of approaches that could have been chosen was to develop only a small part of the overall solution and build this from scratch. However, it was considered a greater challenge to try and develop a more broad prototype. This was to be built using other software, which therefore had the extra challenge of "gluing" and fitting all loose components together. This has of course a disadvantage since it implies a dependency on other peoples work and, as will be clear, this also has the disadvantage that some parts are incompatible, can contain bugs etc. The advantage is that this enables experimenting with a more complete system, which can be used to draw conclusions about the whole system and therefore about the whole blueprint for the project. For this reason the choice was made for a broader prototype.

The development of the implemented system has been done using an incremental approach. After making an overview of the requirements of the desired system, the different components were designed, implemented and tested separately.

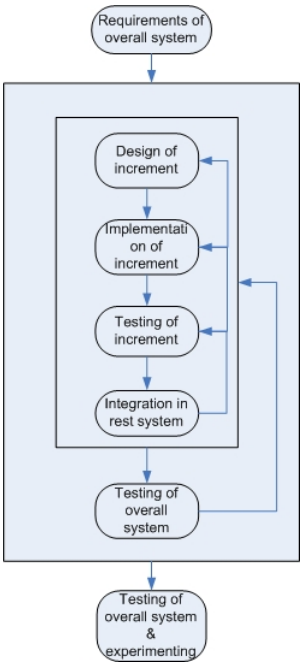


Figure 3.8: A schematic overview of the development process

Due to the research character of the whole project, many iterations were made during these phases, not only *within* these phases, but also *between* these phases. Thus the Waterfall model

(see [BD00]) that is often used, was not really applicable in this situation. The implementation of the increments, which were kept as separated from each other as possible, has a closer resemblance to the V-model. This model puts more emphasis on the dependency between the development and verification activities. Verification was necessary, since new functionalities can always, even when kept as separated as possible from the rest of the system, introduce bugs or errors in the part that was considered as ready. Furthermore, the verification was necessary to see whether the new increment had the desired effect with respect to the eventual target of the project. Therefore testing and verification got quite some attention during the project. Figure 3.8 gives a schematic overview of the development process.

Chapter 4

Global design

4.1 Introduction

This chapter gives an overview of the global design of the developed system. Furthermore, this chapter will also give some rationale about the decisions made during the design. The focus of this chapter will primarily concern the design of the prototype, although here and there some remarks will be made concerning the total design. The design in this chapter will be global. More details about the design / implementation, such as class diagrams and pseudocode will be deferred to chapter 5 and 6.

Some parts of the design were improved after a first implementation and some initial testing. This will also be explained in this chapter. As indicated in section 3.4, the system was built using an incremental approach. The different increments/functionalities were kept separately as much as possible and therefore the design is also explained in different, separated parts (to the extent that this is possible, of course). However, this must all be read with the "total picture" of the system in mind, discussed in section 3.2.

This chapter will start with a global overview of the design of the solution discussed in chapter 3. The chapter will continue with the design of how to bring the structure in the topology of the network. This section will be followed by an explanation of the design of the communication. After this the use of roles within the agents is explained followed by the explanation of the assignment of tasks/roles to nodes in the system. Besides the topology of the network, other measures are taken into account when assigning a role, measures that are integrated into a "fitness"-measure that will be explained in section 4.6.

4.2 Overview of the prototype

The total architecture of the envisioned system is visualized in figure 4.1.
The lowest level part of the system concern the wireless network, the operating system that is present at the nodes and the hardware. Above this level, the network infrastructure is responsible for the routing of messages in the network. The sending and receiving of the messages is handled by an agent system or something similar. In this thesis, the choice was made for Lime. Both of the blue boxes are parts of the system that were not completely made from scratch, but to get the prototype of the system running, it was necessary that this part of the system was also setup. This will be discussed in chapter 5.

The red part of the system is the part of the system that is discussed in the remainder of this chapter. This part of the system concerns the storage of the data locally at the nodes and

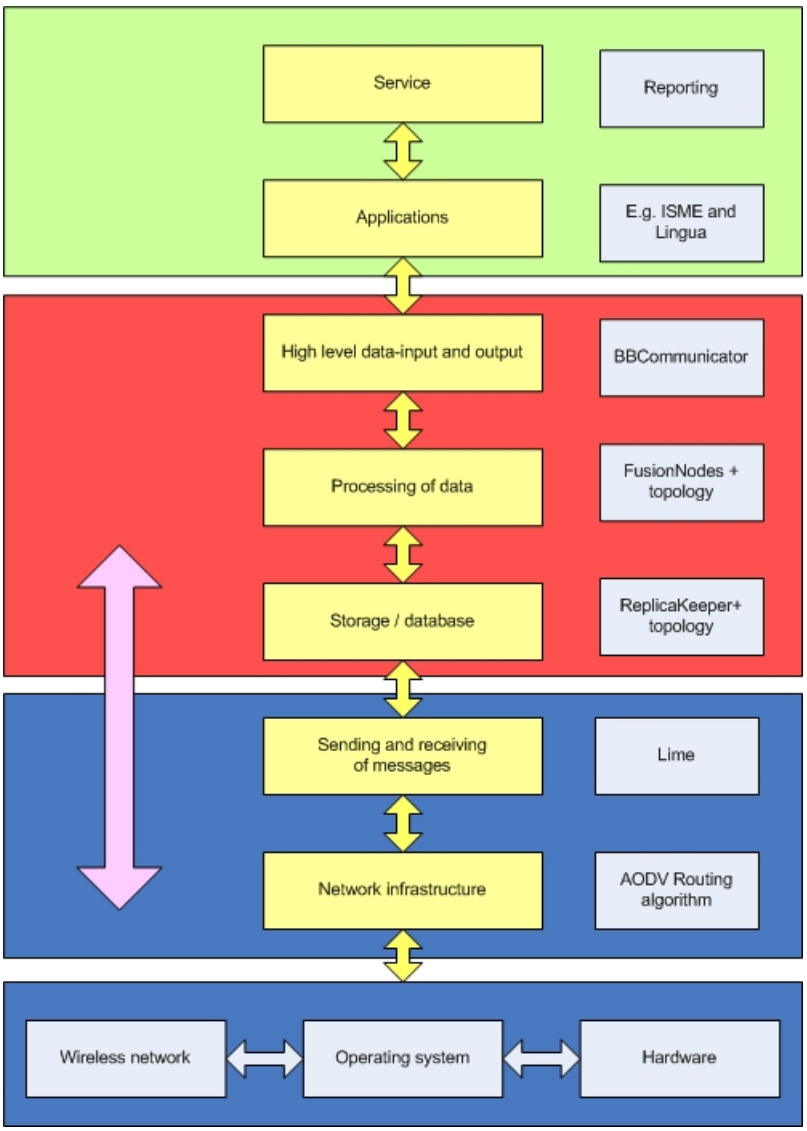


Figure 4.1: Overview of the architecture of the system

also the more persistent storage that takes place in the mobile crisis center (see 4.3) after the preprocessing of the data (see 4.4.3). The data that is stored in the network can be replicated as will be discussed in section 4.5.5. The processing of data takes place by the fusion nodes as described in 3.2. To deal with the topology, and to keep the topology as it is even when nodes change location, the tasks (or roles) that nodes are executing are to be assigned in the network dynamically, i.e. they should be assigned based on the places of the nodes in the topology and should be re-assigned if this topology changes (section 4.5). For this reason, the topology is mentioned at the level of the processing of data and the storage of data. The topology information which is required, is available in the lower layers of the network, which is indicated with the left arrow. The high level input and output is the level that takes care of handling the data that an application puts on the blackboard. This data is to arrive at the right destination and therefore there should be a (simple) protocol for exchanging data. This is captured and abstracted away by the extension to the blackboard system that will be primarily discussed in section 6.5.5.

The highest level indicated in green concerns the services that are to be offered and the applications that are to support this. One of the services that is important in crisis situations,

is the reporting about the situation. This is one of main services that is required when the rescue workers are to explore the place where a disaster has happened. Another important aspect is routing people out of the disaster area. Thus escape routing is also an important service that is to be provided. However, this layer is beyond the scope of this thesis and for experimenting with the system, two applications (for reporting) were used, that are based on icon communication. These are the Lingua and ISME application see 6.3.3.

4.3 Order in chaos: Setting up a crisis center

When the system starts, we assume that a disaster has just happened. In this case, the system should be booted. In the model, described in chapter 3.1, we assumed a stable core. In experiments, but also at real disasters, it turns out that one of the problems is that there should be some people that are in charge and that take the decisions, but that in practice, not everybody is willing to do this. In the model described, that seems to be close to reality, the leaders should be located in the stable core, that serves as a sort of local crisis center.

One of the steps taken was to assist in the setup of these crisis center(s). From this local crisis center, the actions should be coordinated. This is to bring some order in the chaotic situation that is characteristic for situations in which a disaster has just happened.

4.3.1 Humans and/or systems

The nodes in the network can execute different roles. Some of the roles are more suitable to be executed in the crisis center (like the collection of information about the situation) and others are more suitable for people in the field (like reporting). In the ideal situation, all these roles are executed in the right place and the system takes care of this. However, the people are the ones taking decisions about certain things.

It is important to make a distinction between the different types of communication between the agents in the system (humans or systems). First of all there is communication between humans, i.e. human-human interaction. Secondly there is communication between the humans and the computer, human-computer interaction and thirdly there is computer-comuter interaction. Humans take decisions themselves and they base those decisions on many factors. Only a few of these factors are known to the system. The network cannot take a decision based only on technical characteristics of the system that are known (or could be made available). The system that is discussed therefore decides how to distribute and store data in the network, without human intervention. Services thus run on the background automatically. Humans only have control over the applications / services that require human-computer interaction.

One of the decisions that is to be taken in the system is that a crisis center is to be setup. This is a human decision, for which many factors, unknown to the system, are important. Therefore the decision whether or not to setup is left to the users of the system. In the implemented only an *advise* is given based on what the knows, namely topology information and strength of the node. The user can then take the decision to follow this advise or to ignore it, even though this leads to situations where users can do what is suboptimal from the point of view of the system. This seems to be a better choice than forcing the users. This latter choice could lead to situations where rescue workers will stop using the PDA at all, which is something that is worse than ignoring the advise.

4.3.2 The design

The actual setup of the crisis center is not something that can be easily modeled in a system. However, when some assumption are made, a simple setup can be implemented. The first assumption is that when an agent tells the system that he wants to join the crisis center, that this person will stay in this crisis center until it is setup. Furthermore, there will be only one crisis center in the area. If there are physically more than one, these will be considered as one crisis center.
The crisis center will have a status that is kept on the blackboard. This status can be read by any node. This way, the nodes can see whether there is a crisis center (if it is in reach of the wireless network). This status must also contain the hosts that are also member of the crisis center. This information can be used to determine whether the crisis center is large enough and who is a member of the crisis center.

As indicated, the status of the crisis center contains information about the members of the crisis center. To join the crisis center, this status must be updated. This will be done by the joining members themselves. To make sure that the status will not be edited by more than one agent at the same time, there is a simple locking scheme used. To prevent deadlock, some measures are taken for the case that the node that has the lock goes down (see section 6.5.3).

The status of the crisis center is kept at one node, that should be in the stable part of the network, as already indicated. Furthermore some measures are taken the make sure that the information is kept available in the network even if this node would go down. How this is done, is further explained in section 4.5.2.

4.3.3 Role assignment

The information about the members of the crisis center, as described in the previous section, can also be a useful factor when assigning a role to another node. In the system, this information is therefore taken into account during the assignment of roles to node. This is done in the following way:
Roles are given a certain type. This type can be “core task”, “field tasks” and “middle tasks” (again based on the model described in 3.1). The core tasks concerns the roles that should be executed in the (stable) core of the network, “field tasks” are roles that should be executed by the actual rescue workers in the field, “middle tasks” are roles that should, in the ideal case, be executed somewhere in the middle. Furthermore, some roles can have the type “neutral” if it does not matter where the role is executed. This last type is thus more or less a wildcard for the place where it should be executed. These types are taken into account when a role is assigned. When the role to assign should be located in the core, the assignment procedure will first try to assign the role to a node that is known to be in the crisis center. This way, the roles that should be located in the core, will be primarily be executed there (if possible). This dealing with the place where the role should be executed is done for all the roles that are to be assigned.

The assignment of roles that is mentioned here is thus based on the position in the network. This position is determined using the hop count of the nodes to the crisis center. In the system there are ranges defined for the different ranges. However, when a role is to be assigned, in the usual situation there are many nodes in the right ring that are able to fulfil the role. Therefore a (decentralized) decision is to be made in the network where to assign the role that is to be assigned. How this is done will be described in section 4.5.3.

4.3.4 Position of roles

As indicated in section 4.3.3 the position relative to the crisis center is taken into account. This position is determined using the number of hops, since the actual distance is difficult to determine under the assumption that there is no GPS (not accurate indoors) and no reference points that are known in advance (such as in other approaches like [pla]). Furthermore, determining the distance between nodes based on the strength of the wireless connection is problematic since indoors the strength of the connections can vary a lot due to reflections of the radio waves against object (doors, walls etc.)[1].

Due to the dynamics in the network, the probability that the structure of the network changes is quite large. Therefore on every node there is a service that once in a while checks to see whether all the services that are present on a node should be executed there, considering the position of the node relative to the crisis center. The ranges belong to the types of tasks, i.e. core tasks for in the crisis center, tasks for the middle layer and field tasks. The ranges are, of course, adjustable and the right values for these parameters have to be determined in combination with the total amount of nodes.

When testing the re-assignment procedure, some adjustments were made to the initial values of the ranges. To make the system more stable, the decision was made to let the hop ranges for the different "rings" overlap with one hop. This would prevent "border-cases" from being re-assigned too often. In the initial tests, these roles were re-assigned too fast, which made the system spend too much time on re-assignment and less time on executing the actual role.

4.4 Communication

This section concerns the communication between the agents. As already indicated in the model, the communication will be done using a virtual blackboard. Because the focus of our research is on the application level, an existing (pre-release of a) system is used. This system, called Lime (II), will be described further in section 5.2. All that has to be known to understand this chapter, is that Lime provides the functionality to share tuples (which can contain Java objects) on a shared virtual blackboard. This way, the focus of this project could be on the higher layers of the system, instead of on the design of a virtual blackboard itself.

4.4.1 Design with one blackboard

The first attempt to communicate via a blackboard was to use only one blackboard that was accessible by all the agents. This can be visualized as in figure 4.2.

In this picture the green blocks represent the part of the blackboard that is located at a node. All these little parts form the total blackboard. This is thus truly distributed over all the nodes. All the agent have access to the blackboard and can thus read from it and write to it. If a message has to be sent from A to B, this is done in the following way:
The first field of a message indicates the type of the message. The second field contains the data. This way A can put a message on the blackboard with a certain type. B should then be waiting for messages of this type. As soon as one of these messages appears on the blackboard, B can read the message and handle it. If the message is only meant for B it can also remove the

[1] To determine the strengths of the connections some model has to be used to take reflection into account. Some research has already been done and some mathematical models are constructed for this. However, most of these are complex, too "heavy" for Zaurusses and not (yet) suitable for systems that should run in real time. However, research in this area is still ongoing.

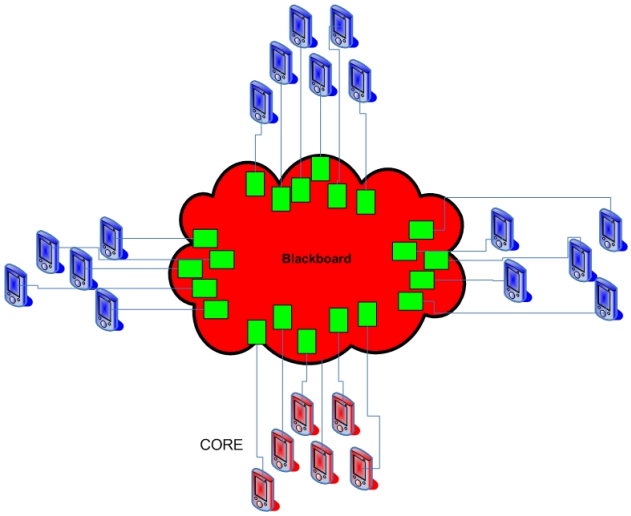


Figure 4.2: An ad-hoc network of PDAs using one blackboard

message from the blackboard. Waiting for a message can be done in two ways. One way is by polling the blackboard every x seconds, another way is to add a subscription to the blackboard. The latter mechanism means that if a message appears, it will automatically delivered at a node.

After testing this approach, this approach was extended. When a message only has to be passed from A to B, it is not always preferable to put this message on the blackboard and let the other node read it. In case the message is part of a small one to one protocol, it is sometimes easier to use a direct connection. The reason is that exchanging messages via the blackboard is asynchronous. Since it is not always known how fast nodes respond, a node might have to poll the blackboard multiple times, before the message is on the blackboard. This can slow down the communication. Furthermore, in case of communication via the blackboard it is, due to the asynchronous character, not always possible to determine whether a message has arrived (or that the answer to the message is just delayed). To overcome this, an extra functionality was offered to the nodes in the form of a possibility to send messages directly. The idea is the following:
When an agent, e.g. agent B, that has a certain service to offer is started, it puts a message on the blackboard. This message is called a registration message. When another agent, e.g. agent A, wants to send a message only to this agent, it does not put the message on the blackboard as in the approach in the previous paragraph. Instead, it takes the registration (that contains the IP address of agent B) from the blackboard. Now that agent A has the IP address of agent B it can send a message directly.
The advantage of this approach is that an agent does not have to know in advance which agent offers a certain service in the network. The idea behind this approach is often used in systems. A well known example in real life are the yellow pages. In many agent systems a service similar to this yellow pages service is available. This means that agents can lookup who is providing a certain service. However, this central point is also a central point of failure. Instead of having a central point to get this information, using the blackboard is thus a more distributed approach. These registrations are not only used for the purpose of efficient one-to-one communication, but, as will be clear in section 4.5.2, it also has another function. More about the implementation can be found in section 5.2 and 6.5.5.

4.4.2 Design of hierarchical blackboards

After trying the approach with one blackboard, another approach was implemented. This approach came forth from the idea described in the model. The idea indicated there was that there was a stable core of nodes in the network with the local command center. Around this core groups are roaming around that do not necessarily have a continuous and stable connection with the core. Furthermore, if all agents "in the field" start sending all kinds of messages directly to the core nodes, this could mean that these nodes recieve very few messages at one moment and a huge amount of messages at the next moment.
Therefore, the decision was made to introduce multiple blackboards, instead of just one. This idea is visualized in figure 4.3.

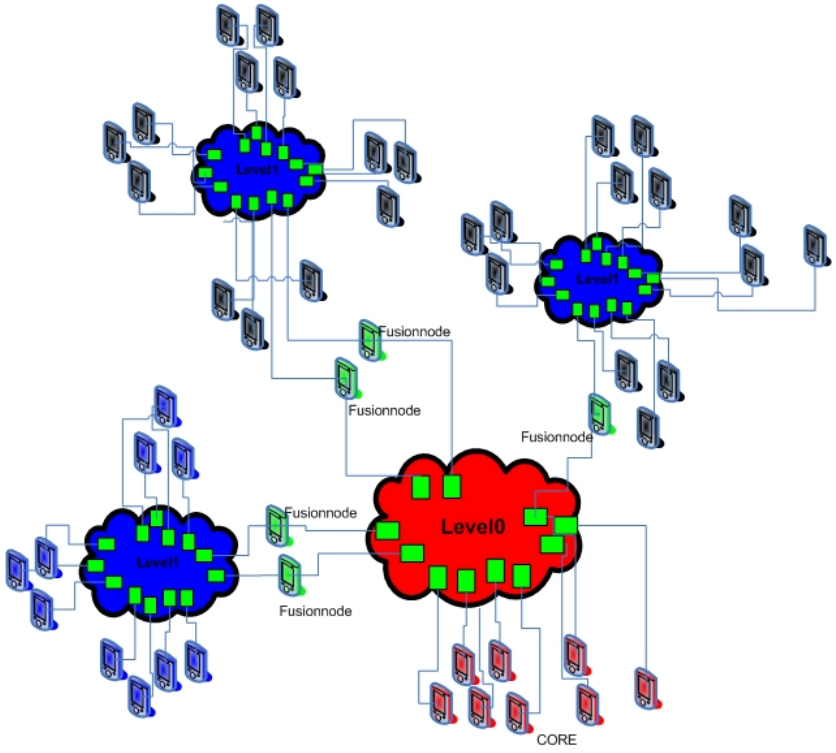


Figure 4.3: An ad-hoc network of PDAs using a hierarchy of blackboards

The idea is to put the most important information on the blackboard that can be reached by the nodes in the core. This is indicated with the red blackboard in figure 4.3. The nodes around the core (that are thus not necessarily connected) form their own blackboards (indicated in blue in the figure). These blackboards are available for all the sensor nodes, so that they can put their data on it. This is where the fusion nodes (indicated in green) have to get their input data. After fusing this data from the sensor nodes, the result of the fusion of the data can be put on the blackboard for the important information (the red blackboard). This way, the nodes in the mobile crisis center have only the preprocessed information at their disposal instead of the "raw data" that comes in from the sensor nodes.
Each of the blackboards has (at least) one node that is responsible for getting information from the local (blue) blackboard to the blackboard that is also accessible by the core of (stronger) nodes. See section 6.5.5 for more information about the implementation.

4.4.3 Preprocessing

The nodes that fulfil the function of putting information from one blackboard on another blackboard, take care of preprocessing of the data. Besides removing old messages from the blackboard or update messages on the blackboard, fusion of information is one of main functionalities that is to be offered. The fusion of data can be done in multiple ways. In case that it concerns reasoning about the data, the most common way to do this is by using expert system rules or by using Bayesian belief networks. The latter can also be adusted to be able to handle the time aspect (*Dynamic* Bayesian Networks).

This means, in the case of the ISME application, that the information that comes in from the sensor nodes (that have an graphical user interface (GUI) with the map to which icons can be added) send their information to a fusion node, that can, for example, filter out duplicates and messages that claim the opposite. It seems likely that in some cases conflicts between the content of different messages can be resolved easier on a local level than in the core. However, this is the decision of the node that executes the fusion. These nodes can make the decision whether to solve the conflict locally or whether to let the core take care of it.

Fusion is one of the functionalities that is be provided by the preprocessing nodes in the system, although the actual implementation of the fusion algorithm will differ depending on the data that is to be fused. One can for example imagine that, in the case of the ISME application (see section 6.3.3), besides the icons with their attributes, more information is provided. Currently, research is being done on how to combine this information with simple human input of geometric shapes and certain symbols (like question marks and exclamation marks). Other input from (other) applications can also be added there, to improve the local world model that has to be sent to the mobile crisis center.

4.5 Dynamic role assignment

4.5.1 Use of roles

In the system, all functionalities that can be executed on a node are considered to be roles that a node might or might not execute. A node looks like as in figure 4.4.

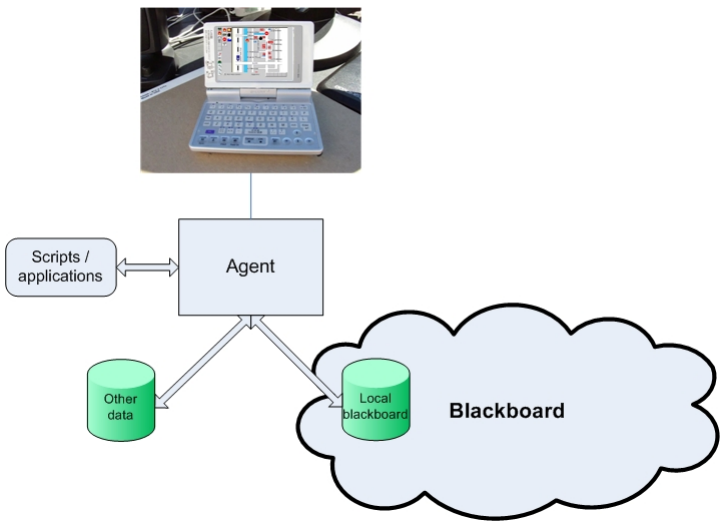


Figure 4.4: A schematic picture of a node

A node has an agent that can have a number of scripts / applications running. Furthermore, the agent has some local data and data that is to be shared and that is thus put on the blackboard. In the system the applications are called roles. These roles are made runnable independent of the rest of the system as much as possible. This way, it was possible to run a special agent on every node, that could start roles separately and shut them down separately. This agent, which is always running and controlling the tasks that the node is executing (called `HomeAgent`), can then act as a manager / scheduler of tasks. The communication with the blackboard is handled by the classes implementing the roles. Schematically an agent is build up as indicated in figure 4.5.

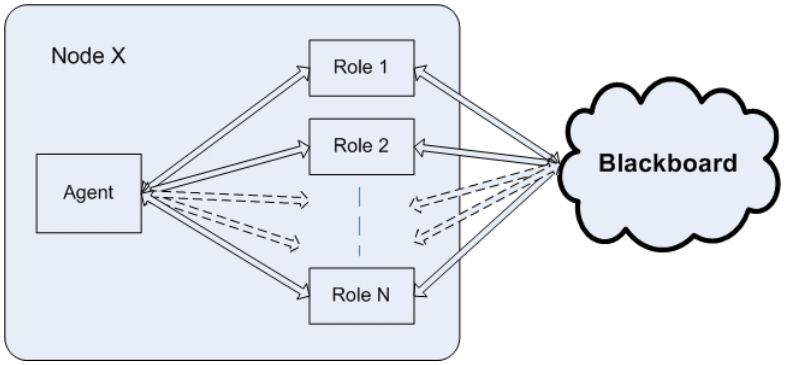


Figure 4.5: A schematic picture of an agent

4.5.2 Use of monitors

Besides nodes that are executing applications that are needed in the network, the system also introduces nodes that "keep an eye" on other nodes. These are called "monitors" within the system. The idea is that every node that executes a task that is important for other nodes, will be monitored by another node. If the node under consideration disappears from the network, the monitor will notice this. The monitor can then try to find a node that can restart the role that it was monitoring. When a node goes down, the data that it has put on the blackboard, is gone as well[2]. Because of this, a monitor only has to check every now and then whether the registration of the node it is monitoring is still on the blackboard. If not, it is time to undertake some action. To prevent from data loss, it is wise to replicate data that is important. For replication a separate role has been designed, as will be explained in section 4.5.5. In the developed system, replication is provided with the monitors, since they are suitable for this purpose. This way the monitor of a service can also keep a history of messages that is put on the blackboard by the other node. The information is then kept in the network, even if the monitored node goes down.

Of course, the next question is, what if the monitor goes down? Who is monitoring the monitor? It is obvious that this cannot be done by assigning another random node the task of monitoring the monitor, since this can lead to an "infinite" number of nodes monitoring each other. This is of course not useful and efficient. Another choice could be to let n nodes monitor each other in a circle. Since this will give a large amount of overhead, the choice was made to let nodes monitor each other in pairs. This means that a service X, that is monitored by a monitor M at another node B, also has to monitor service X at node A. Thus the idea is that service A not only starts a monitor at node B, but also starts a MonitorMonitor locally to

[2]This is the way that the underlying blackboard system, Lime, was implemented. More about this can be found in 5.2

monitor the monitor at node B. This perhaps sounds somewhat confusing, but this will be clear when looking at figure 4.6. This figure represents the situation described above. The arrows indicate the "x monitors y" relation.

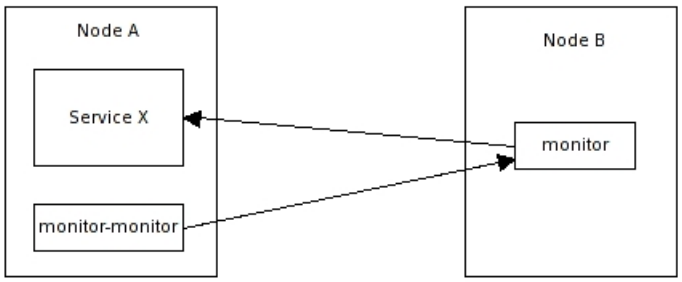


Figure 4.6: The use of monitors

4.5.3 Role assignment

As already indicated, the assignment of roles is also taken into account in the system. The idea is not to appoint a role to a random node that is willing to do execute a role. This could for example mean that a role is assigned to a node that might be at the point of going down. If this is the case, it seems not very intelligent of a system to assign an important task to this node. To prevent this from happening some things are taken into account in the system.

First of all the roles in the system have priorities. For example, in the case of the ISME application, the role of the reasoning engine is more important than the role of client, since if one client goes down, we miss only a small amount of information, while when the reasoning engine goes down, no information is fused anymore, which also means that the ISME clients do not receive any updates of the world model anymore. The priorities are used to determine which roles a node can stop in case a more important role is to be executed.

The second factor that is taken into account is, as already indicated in 3.2.4, the topology. To be precise the distance to the crisis center counted in the number of hops is taken into account. Based on the distance to the crisis center, a node is considered as being primarily suitable for observing, preprocessing or being in the crisis center. Roles that are to be assigned have a type. Based on this type a decision is made where the role is best executed and the role will be assigned in that part of the network (if possible).

The third factor that is taken into account is whether a node is "strong" enough or not. By strong is meant that it is likely that it has enough resources to complete the task that it has to execute and that it is not likely that it will leave the network soon. Both events can never be predicted with 100% certainty, but some measures can be taken into account. Examples of these measures are the amount of CPU usage, the memory usage, the storage usage, and the battery power. How these measures are combined to make a useful comparison, will be further explained in section 4.6.

When the system determines that a role is not executed (anymore) and this role *should* be executed by a node in the system, the system starts a protocol to start up the role again using

an auctioning mechanism. The global idea is that one node tries to assign a role to a node via the blackboard. A request is put on the blackboard. Based on the responses on this request, the role will be assigned to one of the nodes. In this assignment, the above mentioned factors are taken into account. How this is exactly done will be further explained in 6.5.6.

4.5.4 Duplicate services

Some of the roles in the network should only be executed once. An example is the reasoning engine of the ISME application. This reasoning engine should get messages, which are preprocessed in the fusion nodes, from the blackboard and reason about these incoming messages and build from these messages a shared view of the world. This should only be done at one place. Doing the same reasoning in multiple places would be unnecessary redundancy and for other services duplicates might even conflict. For this reason the system has a mechanism for shutting down duplicate services that should only run once in the network. Of course it is not difficult to extend this idea to a maximum of n service instead of 1 service that should remain running. One of the "tricky" side effects is that shutting down a service is not always enough. This has to do with the mechanisme of monitoring discussed in section 4.5.2. This has to be taken into account, since if we only shutdown a service, the monitor of this service will restart the service. Thus, if a duplicate service is detected, the monitor of this service is to be stopped as well.

Duplicates are solved by a special role. This role constantly watches whether there are two or more services that should be unique. If it finds out (by the registrations on the blackboard) that there are too many instances of a service in the network, it will shut them down except for one. This decision is based on the "fitness" of the node (see section 4.6). The "fittest" node that runs the service under consideration will be left running, the rest is shut down. The shutdown is not done externally, but should be done by the applications themselves. This is done because in this case the services that are shutdown still have enough time to upload their data so it is not suddenly lost.

4.5.5 Replication

The system also offers the possibility to replicate messages on the blackboard. The replication strategy for services is dependent on the service at hand. In the eventual system some services have a random subset that is keeping replicas. Finding the best distribution of the replicas with respect to possible group partition is done in for example [Har01] and [Har03] and is considered as a further improvement (see section 8.2.2). The implemented system thus offers the possibility for replication, primarily for the monitors. Since the idea is that these restart services if they go down, they also store some necessary data that is needed to continue operation after a service is restarted. In the case of the reasoning engine of the ISME application, this data concerns the last update that was sent to the clients.
Furthermore it is possible to store a history of multiple messages in case this is desired. Only the last piece of data is replicated on the blackboard. If older messages are necessary, a service has to ask for these separately. Due to the system of registrations that include an IP address, this should not be a problem. This way a history is kept and this makes it possible for the nodes in the network to get this information and reason about the messages (see section 4.4.3).

4.6 The fitness of a node

As already indicated in section 4.5.3 the system will take into account that some nodes are stronger than other nodes. In [CN01], where an approach concerning storage of data in MANETs is discussed, a linear combination is used of *"...free hosting space, remaining energy life, and processor idle percentage"* as the capability of nodes. In this approach the emphasis is on storage of data and not on the execution of tasks or the collection of data at the location that it is requested. Probably for this reason the problem of group partition is more emphasized and the capability is only used as a decision criteria in certain circumstances.

In the designed framework the "strength" of a node is also taken into account. This strength is based on a number of performance factors that are combined to give one value, which we will call the "fitness" of a node. This fitness value will be described globally in this section. In section 6.5.11 we will go into more details about how the different factors were measured.

4.6.1 Reason for use of fitness

The reason that a fitness value is used, is that we think that assigning a role "blindly" to a random node is not a very "intelligent" thing to do for a system. Especially when more information about the nodes and the network is available, it seems logically that this information is taken into account. Furthermore, by making these values of the fitness dynamic, this can distribute the load equally over the different nodes. By dynamically we mean that the values change during runtime. This way, when a node has much to do, the performance of the node will be lower and therefore, the fitness will go down. Newer roles will then be assigned to other nodes that are less busy. This way, in the long term, the load in the network will be distributed more evenly over the nodes. This way the situations in which a single node is overloaded is less likely to occur, although this situation can never be 100% prevented.

At the same time, the fitness can also give an indication about the chance that a node will go down. This information can be used to rescue important information. This information must be sent to other nodes in the network, when a node that "owns" the information "thinks" that it is going down soon. Going down in this case means that the battery is empty or that a node is overloaded. See 4.6.2 for a definition of what is considered to be overloaded in the system. Of course it is not always possible to predict what exactly is going to happen in the future based on information from the past, but it does give an indication (e.g. battery power very low and not loading means that the node will go down soon).

4.6.2 The components

The fitness value is composed of some performance components. These components are:

- the battery power
- the memory usage
- the storage usage
- the CPU usage
- the number of (direct) neighbours

The *battery power* is taken into account, since if the battery power is too low, the node will go down completely. The percentage of battery power that is left is called *bf*.

The *memory usage* considers whether the main memory is almost full. If so, the node should not accept more tasks, since this will slow down the node too much (swapping), which can make a node useless. The percentage of memory that is in use, will be indicated with *mf*.
The *storage usage* considers whether the persistent memory is full or not. Some services that run within the framework may want to use persistent data, for example for logging or storing important information. The percentage of storage that is in use, will be indicated with *sf*.
The *CPU usage* is the percentage of CPU power used. If this is near 100% the node will become slow if more tasks are accepted. Furthermore, a higher use of CPU power usually means that the battery will be empty sooner. The percentage of CPU usage, will be indicated with *cf*.
The *number of (direct) neighbours* is also taken into account. The idea is that when a node has many direct neighbours, it is probably more "in the middle of the network". This is an indication that it is (in most cases) less likely that the node will be completely disconnected from the rest of the network. This is a first order approach to have an idea of how central a node is. Another approach (that is not implemented) could be based on the ant based control algorithm, in which all routes in the network are taken into account to calculate the *connectivity* of a node (see section 8.2).

Now we have determined what components are taken into account, we can indicate what we mean by "overloaded". In this thesis, a node will be considered overloaded when the CPU usage is approximately 100%, the memory usage is approximately 100% or the storage usage is approximately 100%. The amount of network traffic of a node is thus not taken into account, although this could also be a bottleneck. This is only done for practical reasons (not easy to measure), but this could also be taken into account and can be considered as a future improvement.

4.6.3 The integrated value

The values of the different performance measurements should, in some way, be used to decide if and where to (re-)assign a role. This is done by integrating the different measures in one value. Another option would have been to use an expert system. However, it seems quit difficult to find rules that can be used to determine when a node is "fit" or not. Since the question whether a node is "fit" or not has many aspects, this would lead to a rule base that would quickly become very large, untransparent and very hard to adjust in case an extra measure should be added. Furthermore, this could lead to a rather large rule base, with many fuzzy rules for which the fuzzy parameters have to be determined, making the systems much more complex, slower and more power consuming than a number of "simpler" calculations. Furthermore, it is hard to determine the fuzzy parameters. Therefore, the choice was made to integrate the measures in one value, using a mathematical calculation. This is somewhat similar to earlier work as in [ADA04] and [CSN02]. However, in these publications the values, that can be considered as attempts to define a simple fitness, are simple and not used to make important decisions. In the approach described in this thesis work, the fitness value will have a more prominent role and therefore more care will be put in the calculation of this value. Integrating the performance measures in one value can thus be done more easily, since the behaviour of the fitness of the system can be determined relatively easy for the separate measures. These can then be integrated in one value in different ways, of which one is shown in the following.

The next question is, of course, *how* to integrate all these values into one useful value. The first decision concerns what values the fitness should be able to take. The choice was made for values between zero and one, zero meaning that the node is not "fit" and cannot take other tasks (and should even try to loose some tasks) and one meaning that the node is able to take new tasks (perhaps even more important tasks).

Since some factors can be of great importance, the choice was eventually made for a multiplication of functions of the measurements. At first (and as a comparison to the final function) another, linear combination was also tried. As will be clear in the chapter about the tests and experiments, a linear combination of the factors did not provide the result that was desired. By using a multiplication, it turned out to be possible to give certain values the desirable impact on the fitness. For example, if the battery power approaches zero and it is not recharging, this means that the node will probably go down within a short time. For this situation, it seems fair that the fitness should be very close to zero. This is not easy to achieve with a linear combination of factors, because if one of the factors (in this case the battery power) becomes very important in the formula, this usually means that the other factors have hardly any influence anymore. This is of course not desirable. Therefore, the choice was made for a multiplication.

The next problem is of course, how the fitness should respond to the changes in the different values. The basic idea was to make the dependency of fitness with respect to the individual componenents of the shape as in figure 4.7.

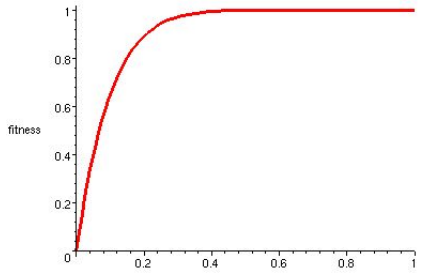


Figure 4.7: The general relation between fitness and a component

In figure 4.7 the dependency for, for example, the battery power component is shown. If we want to add this as a part of a multiplication, the component of the total multiplication that belongs to the battery power should have the same shape as in this figure.
The decision for this shape is made for the reason that if the value of the battery power is low, the fitness should be low. Furthermore, reason for the sharp increase in the graph is that above a certain battery level, there is no reason to decrease the fitness by a large amount. If for example the battery level is 50%, there is no reason to decrease the fitness by 50%. For this reason for all components a function of the value of the component is used. For the different components similar functions were used. The function that was chosen as an initial approach for the fitness components was based on the shape and on some initial experiments. As will be clear later, experiments showed that this equation led to good results, although the function that leads to the theoretical best results was not possible to determine with a real system. Detailed simulations are necessary for this.
For the battery power level a function of the form

$$1.0 - (1.0 - bf)^{ps} \tag{4.1}$$

was used, where bf is the fraction of battery power that is left and ps is the sensitivity to the battery power. See section 6.5.11 for more information about determining the value of ps.

For some components, for example the memory usage, the shape should look as in figure 4.8. Again the idea is that the value should not influence the fitness too much in a certain area, in this case, when the memory usage is low. Only when the memory usage is approaching 100%, the fitness should be influenced, as can be seen in the figure.

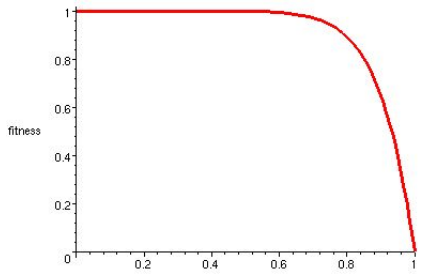


Figure 4.8: The general relation between fitness and a component

The function that belongs to this shape is of the form

$$1.0 - mf^{ms} \tag{4.2}$$

where mf is the fraction of memory that is in use and ms is the sensitivity of the fitness with respect to the memory usage.
The storage usage and the CPU usage components will be integrated in the total fitness by using a similar function. For the number of neighbours, a separate function is used to calculate this component. This function is very low when the node has no neighbours, increases linear until it reaches one. The value of one is reached if the node has a certain number of direct neighbours. If the node has more neighbours, the value remains one.

The total fitness is, as already indicated, a multiplication of the different components. The complete value is computed with equation

$$(1.0 - (1.0 - bf)^{ps}) \cdot (1.0 - cf^{cs}) \cdot (1.0 - mf^{ms}) \cdot (1.0 - sf^{ss}) \cdot nonp(x) \tag{4.3}$$

where:

- ps is the sensitivity with respect to the percentage of battery power that is left,
- cs is the sensitivity with respect to the percentage of CPU power that is used,
- ms is the sensitivity with respect to the percentage of memory that is in use, and
- ss is the sensitivity with respect to the percentage of storage that is in use.

and $bf, cf, mf, sf \in [0, 1]$ and $ss, ps, cs, ms > 1$

The function $nonp(x)$ is defined as:

$$nonp(x) = \begin{cases} \frac{1}{2} \cdot ns & \text{if } x = 0 \\ 1.0 & \text{if } x \geq \frac{1}{ns} \\ x \cdot ns & \text{otherwise} \end{cases} \tag{4.4}$$

where x is the number of neighbours ($x \in N$), and ns is the sensitivity with respect to the number of nodes ($ns \in \langle 0, 1]$).

See section 6.5.11 and 7.3 to see how the values of the variables bf, cf, mf, sf, x and the parameters ss, ps, cs, ms, ns are determined.

4.6.4 Observing own fitness

Another functionality that is included in the system concerns a sort of "panic-alarm". Every node has a role that calculates the fitness every x seconds. If the fitness of a node suddenly goes down, this role takes action. What it does is that it tells all the roles that are running on the node under consideration, that there is a reason to believe that the node is going down/ will be overloaded. The roles can then take the decision themselves what to do. In some cases this is doing nothing, but some services will try to put their most recent data on the blackboard so it can be saved by other nodes etc.

4.7 Startup of services

Starting a service in the network is done by the `HomeAgent` located on a node. This can be considered as the manager of the tasks that are executed on a node. If a service is to be started, such as the ISME service or Lingua service (see section 6.3.3), the startup of the GUI's of the services, that should be present at the observer nodes, has to be done manually. The reasoning engines and the fusion tasks can be started up manually, but this can also be done automatically. The latter approach will take more time, since the system first has to determine that a certain service is not running. Furthermore, the monitor roles cannot be started by hand. This is done automatically by the system, since this is something that the users are assumed not to be interested in.

Chapter 5

Setup low-level infrastructure

5.1 Introduction

In this chapter the setup of the low-level infrastructure is discussed. The desired infrastructure that was to be setup before (and eventually parallel to) the implementation looks as in figure 5.1.

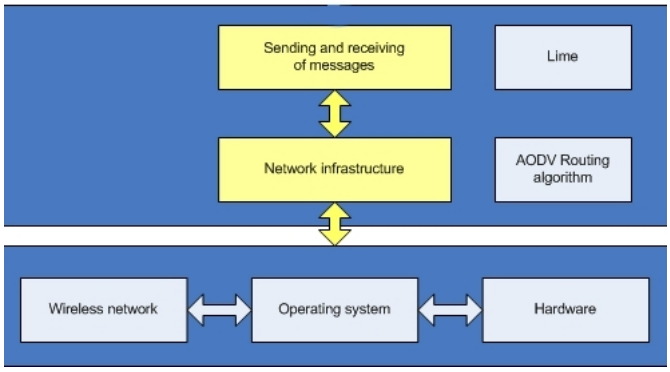


Figure 5.1: The low-level infrastructure

For the reason that the application was to run in a MANET and because of the design of the rest of the system, the wireless communication was to take place via multihop routing. After the discussion of Lime will be clear that there was a need for multicasting, since the Lime system depends on this. The setup of the low-level infrastructure thus required, wireless, multihop, unicast and multicast routing. This setup, in combination with Lime was, never tried. Furthermore, most approaches that concern MANETs concern simulations and not real life systems. Therefore we ran into a number of problems that were to be solved. This chapter will, besides giving an overview of the setup of the infrastructure, also give an overview of the problems encountered during the process and the (sometimes long) road to an eventual running system.

5.2 Lime

As already mentioned in section 3.2, the communication and coordination is done via a blackboard. Another requirement was that the application was distributed and (to some extent) fault tolerant. Therefore, the choice was made for a distributed/virtual blackboard system. Because the focus of this thesis was not on *making* blackboards, but on the *use of* blackboards, the choice was made not to implement a blackboard structure from scratch, but to use an existing system.

However, there were not many blackboard like systems available in which the blackboard was distributed. Centralized systems were available, but this was not suitable for the environment in which it was to be used, a MANET. The only freely available system that was found that provided the necessary functionality was Lime (see [lim]). To be precise, a *pre-release* of the newer version of Lime was used, called Lime II. As in the first version of Lime, all information was distributed over all nodes in the network. Information that is put on the blackboard, is shared, but the original piece of information remains at the node that put the information on the blackboard. As a consequence of this, if the node would disappear from the network, the information is lost. In this version of Lime it was not possible to let the system store a replica at another node. This is the task of the programmer and it seems obvious that the proposed fitness values can be used to decide at which node replicas are expected to be stored safely. This is thus exactly one of the functionalities that is offered by the eventual system. Furthermore, in the eventual system, data does not remain at one node, but is processed/merged/fused and the result of this is sent to the stable part of the network as explained in section 3.2.

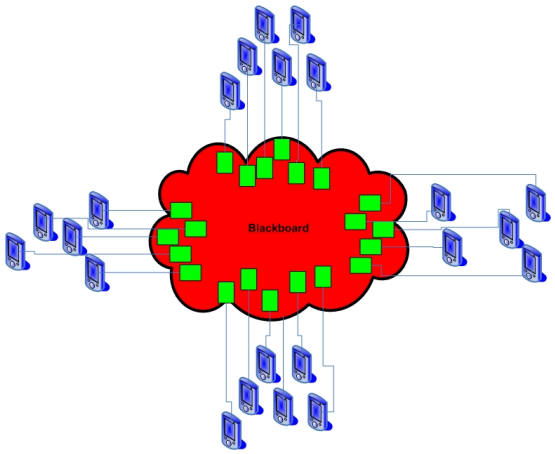


Figure 5.2: Schematic overview of overall functioning of Lime

The internals of Lime are reasonably complex. However, due to the fact that at some points the system "did not work", some knowledge was necessary for understanding the problems encountered and solving those problems. Lime makes use of Java tuples. For the developed system, the LighTS [lig] was used, for its simplicity and for the reason that Lime was already successfully tested with this library. Operations that are executed on the blackboard, are put in a queue and are handled (in parallel) by a pool of threads. Since certain operations can lead to inconsistency problems (e.g. removing a message), the system strongly depends on internal synchronization. The idea of the system is that every node has its own local piece of the blackboard. If a message is added to the blackboard, it is added to this local piece. If a node wants to get a message from the blackboard, a read/remove operation is available (blocking and non-blocking). After checking the local piece of the blackboard and in case the required message is not found, these operations send a request for the message to the other members of the blackboard. This is done by using multicast messages. As soon as the node has an answer (or more in case this is requested), the operation returns the result to the user. Another service that is provided by the Lime systems, is that the user can subscribe to a certain type of message. Is this type of message is the put on the blackboard, the message will automatically be delivered to the subscriber by means of a Java "event". The global behaviour of the system can be described as in figure 5.2. The green rectangles indicate the local part of the blackboard, the red cloud indicated that the user does not see what is happening inside the system, i.e. it only sees one blackboard.

Discovery of other members of the blackboard is done by means of "HELLO" messages. These are also multicast messages. The communication that the system provides is for TCP connections and for UDP connections. The most important property of the pre-release of the software was that the system could handle unexpected disconnections unlike the previous version, which would make all connected nodes in the network crash (leading to an oil-stain effect).

More information about the new version of Lime can be found in [Bel04], which also gives an interesting overview of existing blackboard systems.

5.3 Setup of a wireless network

After testing, debugging and implementing some extensions to Lime, the next step was to be taken, i.e. taking it to a wireless environment on the Zaurus devices and the "development laptop", referred to as "the laptop" in the remainder of this chapter.

The setup of the wireless network started with the setup of an ad-hoc network. After getting a wireless card running on the laptop (for which we had to compile and install a new OS kernel), the *iwconfig* tool could be used to setup the wireless network on the laptop. The Zaurus offered another tool for the setup.
Now that this was running, the multicasting was to be setup. This is where we encountered a problem, since it turned out, after some debugging, that some drivers for WiFi-cards do not support ad-hoc mode and others do not support multicasting, both properties that were necessary in the eventual system. Luckily there were enough wireless cards available at the University that did support ad-hoc mode and multicasting.

5.4 Multihop routing part I

The next property that was necessary, was that the messages were to be sent multihop. For this reason we had to use an implementation of a multihop routing algorithm. Since it was already known in advance that multicasting was necessary, an implementation of an algorithm was used that should support both. Furthermore, it was also necessary that the algorithm should run on the Zaurus, which, due to the fact that it has an ARM-processor, limited the number of algorithms that were available. Eventually was chosen to use an implementation of the Multicast Ad-hoc On-Demand Distance Vector (MAODV) routing algorithm made by the Uppsala University and the University of Maryland. The former University was responsible for the unicast part of the algorithm, the latter made a patch that should add the code for multicasting. The word "should" is used here on purpose, as will become clear. The reason for this choice was that there was a reasonable amount of documentation, the algorithm was tested for Zaurus devices and a there was even a paper about some experiments available.

5.4.1 MAODV

AODV

Although the AODV routing algorithm is one among many attempts to route packets in MANETs, this is one of the few that has withstood the ravages of time along with e.g. Dynamic Source Routing (see [Kla04] and [dsr]). Perhaps for this reason there was a version available that could run on the Zaurus devices.
The AODV routing algorithm (AODV) is, as already indicated in section 2.3.2, discussed in [PR99]. The algorithm works as follow (based on [PR99]): Path discovery is done by sending a

route request packet to the neighbors. Those neighbors can forward the packet to others and keep some information to prevent forwarding duplicates and to be able to construct the reverse path. Once arrived at the destination, a Route Reply message is sent back to the source, using the shortest path.

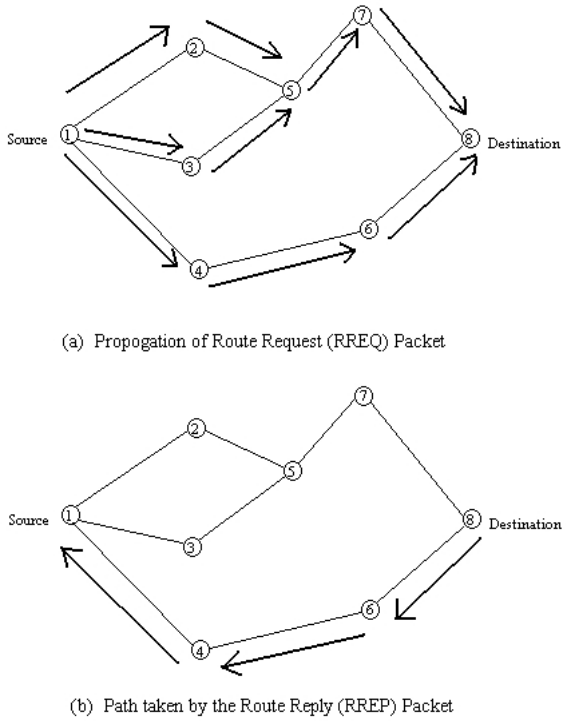


Figure 5.3: Route establishment ([aod])

It uses sequence numbers as a check to see whether a route is still up to date (if a path is invalid, the source can send a route request again with a higher sequence number, so it will not receive data about the old route). Furthermore, paths can also “time out” due to the fact that after a certain time nodes consider routes to be invalid. To keep local connectivity, “HELLO” messages are periodically broadcasted (time to live = 1). If a node is not reachable anymore, the upstream node send a route reply message back through the route with a hop count of infinity.
An optimization for AODV routing that is mentioned in [SLL03] is the expanding ring search for the route request messages, which means that the messaged are forwarded to more nodes consecutively (using a larger ”time to live” every iteration of the search algorithm). This feature was not implemented in the used implementations.

MAODV

The MAODV algorithm makes multicast groups in the form of trees. One of the nodes is the group leader. If a node wants to send data to the multicast group and it has no route to the tree, it starts a discovery protocol similar to the AODV algorithm, by broadcasting a route request message. It can then recieve multiple replies. It chooses the node with the smallest hop count to a member of the multicast group. The route to this node is enabled in the routing table and an activation message is sent to this node. If this node is also a member of the group, this node will add the node to the routing table and not forward the activation message. Otherwise, if the node is only on the route to the group, the activation message is forwarded (to the next

hop).

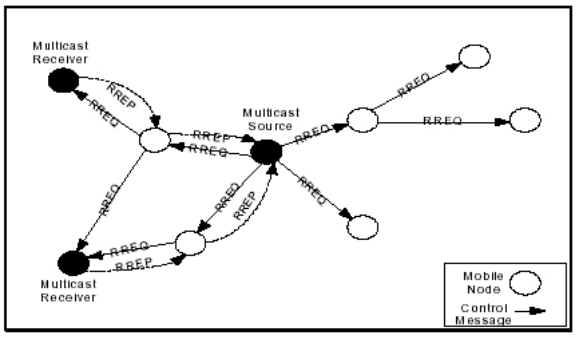


Figure 5.4: Route establishment ([KC01])

This is done until the activation message reaches a node that is in the multicast group. This activation message takes care that messages are only sent via the route that it has traveled to the group. This prevents duplicates in case that multiple routes exist to a node.
The group leader is the first node that starts the group. This node is responsible for sending the "group sequence number" to the group by means of multicast "HELLO" messages. These messages contain information about the group leader address, the group sequence number and hopcount. The algorithm takes care of a disappearing group leader itself (by choosing a new leader).

5.4.2 More limitations

When the algorithm was to be compiled, we ran into some problems. There were multiple versions, and the one that was suitable for the Zaurus did not "just" compile. First the algorithm was to be ported to the kernel of the Zaurus and the laptop (2.4.18 and 2.6.10, respectively). Furthermore, the Zaurus devices needed 2 more modules, that were to be compiled as well (ip_queue and ip_tables). The algorithm consists of a kernel module. Due to this fact and due to changes in the Linux kernel from 2.4 to 2.6, a choice was to be made for 1 kernel version. Since there were no (stable) 2.6 kernels for the Zaurus and due to the fact that the algorithm was tested with 2.4.18 kernels, the choice was made to change the kernel of the laptop to a 2.4.18 kernel. However, now there was a new problem: The wireless devices of the laptop did not work on the old kernel.

5.5 Network cards and kernels

To get a wireless device running on the laptop, we needed a higher kernel version, but not to high, since the routing algorithm would/might not work anymore on the Zaurus. For this reason we started a quest for the lowest kernel version $\geq$2.4.18 that had support for the PC-card interface on the laptop (, which at that time, only worked on kernels $\geq$2.6). We tried to compile and install a number of kernels, among which 2.4.18, 19, 22, 23, 24, 26 and 29. By trial and error we found that kernels upto 2.4.29 have problems with IRQ handling/routing/sharing. The reason that the wireless card did not work was because of an IRQ conflict with other important devices which the (old) needed kernel could not resolve (the combination does work on newer kernels). Kernel parameter tweaking (like pci=biosirq) did not help as well. Therefore we searched for another solution to get a wireless device running. For this reason, a change was made to a wireless card that works via USB (one of the few cards that works on old 2.4.18

kernels). After compiling and installing a driver for this card and after some experimenting, this turned out to work.

5.6 Multihop routing part II

Now that we had the wireless network running we did some initial testing with the AODV algorithm. It did not seem to work on our initial attempts. The reason for this turned out that the setup of the wireless network channel was not always correct. The channel determines the frequency on which the radio waves are sent. The IEEE 802.11b standard defines channels from 1 to 13 with frequency ranges from 2412 MHz to 2472 MHz. On the level of antennas research is ongoing, since interference is to be avoided and since multihop routing requires multiple message, from different "directions" to be handled by one antenna, i.e. the one in the wireless card.

Some drivers, among those of the Zaurus, automatically set the channel of the network to the channel of the first accesspoint encountered, instead of first checking the ESSID (i.e. the name given to the ad-hoc network). For this reason, the routing did not seem to work on some attempts. Furthermore, due to the fact that the wireless accesspoints are extensively used within the University building, using the channel of these access points led to interference. Using some private commands of the wireless cards, it was possible to let it use the right channel.

The next step was to get the multicasting part running. After some experimenting, it turned out that this part of the algorithm did not work. This requires debugging on multiple levels. After debugging the routing algorithm by "sniffing" packets with the tcpdump-tool, it turned out that multicast messages were not forwarded. One possible reason that was found, was that the forwarding of multicast messages is an option in the Linux kernel. It turned out that the Zaurus kernel, a kernel specifically made for PDAs, did not (!) have this option turned on. After replacing the kernel the algorithm still did not work. More debugging of the code and contact with the author of the algorithm did not lead to a working version. For some reason the activation of the routes did not succeed. The activation messages were sent, but did not arrive at the right node. The exact reason for this was not found.

As a replacement of the multicast part, another solution was searched. Eventually an implementation of a routing algorithm was found that can not send unicast messages, but can only send multicast messages. The name of this algorithm is the On Demand Multicast Routing Protocol (ODMRP) (see [odm]). A partial implementation of this algorithm for Java was available, named JOMP (see [jom]). This was an application level implementation, which meant that it was to be inserted in the code of Lime. This led to some problems (due to some problems with sequence numbers), but eventually this was included successfully. For this, the internals of Lime had to be adjusted. This was to be done with some care, since the whole system rests upon the blackboard. Introducing bugs here, could ruin the whole system.
The Java multicasting part used in the JOMP algorithm uses multicasting as well. Therefore, the AODV kernel hook, i.e. the part of the algorithm that was to handle the messages instead of passing it to the standard procedures of the kernel, was also adjusted to ignore the (multicast) packets of this algorithm.

Finally, when the multicast part of the algorithm was no longer necessary, the unicast part of the algorithm was the only part that needed to work. However, the used implementation of the AODV algorithm, of which the unicast part was still working, was not very stable and routes would brake and be established very unexpectedly. Sometimes the algorithm failed com-

pletely. However, since only the unicasting part was necessary due to the use of JOMP, the choice for available routing algorithms became somewhat larger. Therefore, a switch was made to the AODV implementation from the Wireless Communications Technologies Group from the NIST (see [ker]). After changing the source code to be able to work with the Java Multicasting part, this was the last functionality of the low level infrastructure that was to work. As will be described in section 7.4, (partially due to some of the mentioned problems) some experimenting was also done with the low-level infrastructure, to see how the performance of this layer was.

Chapter 6

Implementation

6.1 Introduction

This chapter gives an overview of the implementation of the developed system. The implementation is based on the design in in chapter 4. After discussing shortly the overall implementation, the software and hardware that were used during the implementation are discussed in section 6.3 and 6.4. This section is followed by the discussion of the implementation of the system. In the section about the implementation, the different parts of the systems will be discussed in approximately the same order as in chapter 4.

6.2 Overall implementation

As already indicated in the previous chapter, the overall implementation was somewhat hampered due to some restrictions that followed from the assignment and the requirements and constraints (see section 1.2 and 3.3). The available PDAs had, as already indicated, some constraints. To be exact, the operating system version was old and there were no newer versions available. Furthermore, due to the processor in the PDAs, the Java Runtime Environment that was running on these devices, was also older, which meant that an older version of Java (1.3.1) was to be used for the entire system.

Furthermore, the software used in this project was not "off the shelf" software that was bug free, as already indicated in the previous chapter, but experimental software from other research projects. The software that was available sometimes contained bugs, did not do what it promised, was incompatible with the other software and was not (well) documented. In the eventual prototype, all components were glued together to come to a running system. A total overview of the components of the system is given in figure 6.1.
The setup of the underlying low level infrastructure has been discussed in the previous chapter. The remainder of this chapter will therefore focus on the higher part of the system, i.e. the top two layers.

6.3 Used software & tools

6.3.1 Eclipse & Ant

For the implementation, the programming language used is Java, as already indicated. As a development environment the choice was made for Eclipse. This is a reasonably complete development environment that can speed up the development process a lot. It provides many functionalities for Java programming. The program has a code editor that offers code assistance

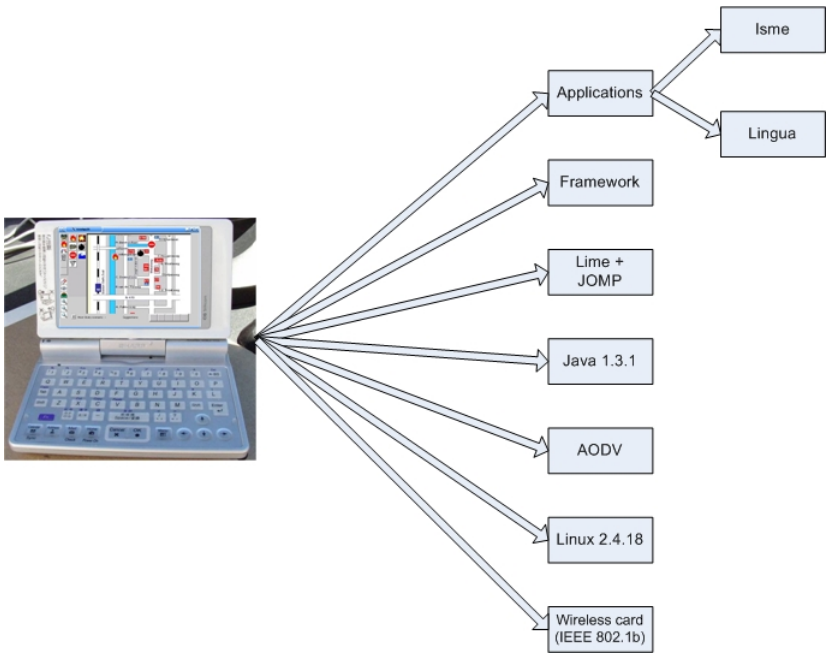


Figure 6.1: The components of the prototype

(e.g. making default "getters and "setters" and offering code completion). Furthermore, a debug perspective is provided to assist in the debugging of the application. Besides this it provides multiple overviews of the code, which makes it easy to jump to specific parts of the code. Eclipse also has a plugin that enables the use of the build tool Ant. Like Eclipse, this is free and open source software. Ant is a tool to automate the compiling, javadoccing and running of a Java project. In this project this tool was used extensively, not only to build the project, but also for tasks like emptying files and copying files to other devices (to the Zaurus in most cases). For more information about Eclipse and Ant, see [ecl] and [ant].

6.3.2 JUDE

For making diagrams, the JUDE UML modeling tool was used. This tool, which is freely available, enables making UML diagrams and exporting the classes including the methods to Java code. This tool was very useful to support the design of the system.

6.3.3 Test software

For testing the framework, some applications were used. This concerns applications that need (some) communication. Both programs were built for a crisis situation and therefore these were chosen for testing. This framework could integrate the programs into one system. However, some adjustments had to be made, since during the development of the applications the emphasis was not on the distributed character of the applications, but more on the usability aspect. These applications were used, but were only used for testing. The question whether or not these specific applications are useful for a crisis situation is considered not relevant in this thesis.

ISME

One of the test programs is the Icon based System for Managing Emergencies (ISME), written by P. Schooneman. This is an application for reporting about disasters using icons. The idea is that when a disaster has just happened, there is a need to find out how the situation looks like

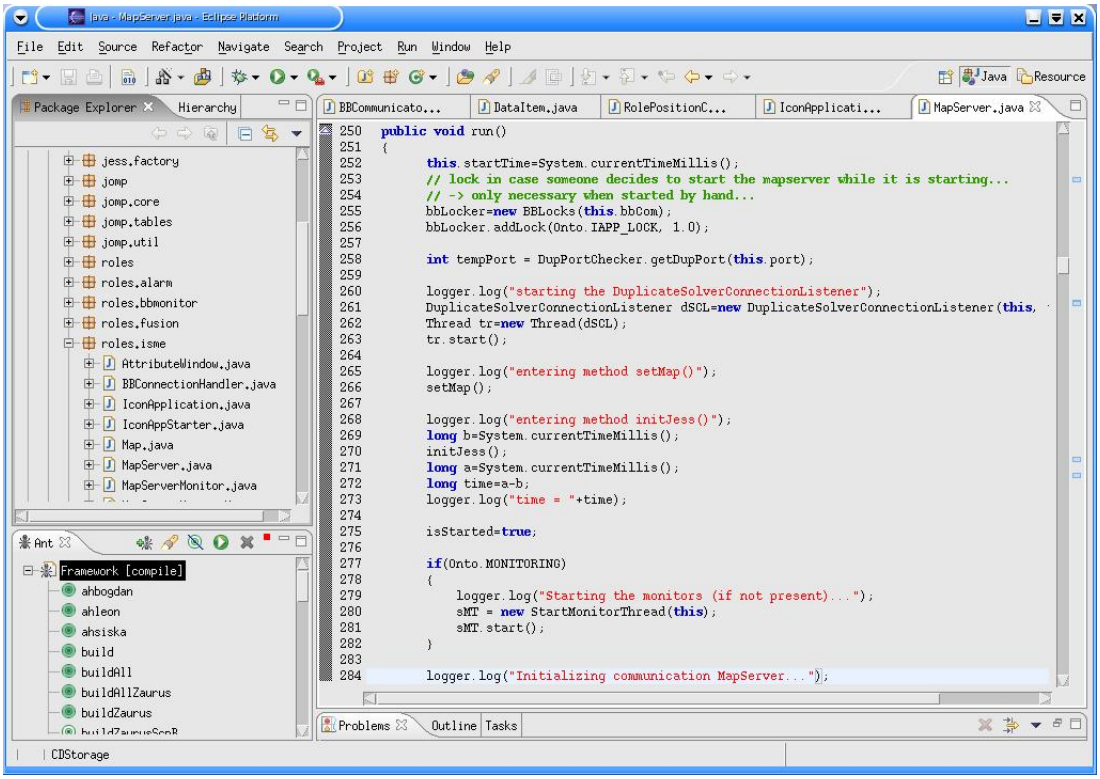


Figure 6.2: Screenshot of the Eclipse development environment

in the area where the disaster has happened. The most logical way to find out is to send people in the area and let them report about the situation. In the ISME application the reporting is done by using icons.
The application is divided in two parts, a client and server part. The latter part, the server, was adjusted. Because we use a decentralized approach, this has become just another node, which was renamed to "reasoning engine". The reasoning about the world and the creation of a shared view of the world is done at the side of this reasoning engine. The reporting is done with the clients, with provide a graphical user interface on which icons can be put on a map of the area (see figure 6.4(a)). The clients send information to the reasoning engine that collects and processes the information. Clearly this is an application in which the client part of the application is the part to be used in the outer ring by the observer nodes and the reasoning engine is suitable for the stable part of the network. Preprocessing can be done in the middle layer. For more information about the ISME application see [Sch05].

Lingua

The second application that was used for testing was the Lingua application. This application is for communication with strings of icons. In the case of crises, this application can be used for reporting about the situation. As with the ISME application, the benefit of this application will not be discussed here.

The Lingua application offers an icon based communication interface that allows user to use icons as input to make sentences (see figure 6.4(b)). These can be spoken by a text-to-speech application. For test purposes, this application was split in two parts, namely the text processing and the text-to-speech part. The sentences that are formed (by using icons) are sent over the network to the text-to-speech part. There they are spoken and the text is put on the screen

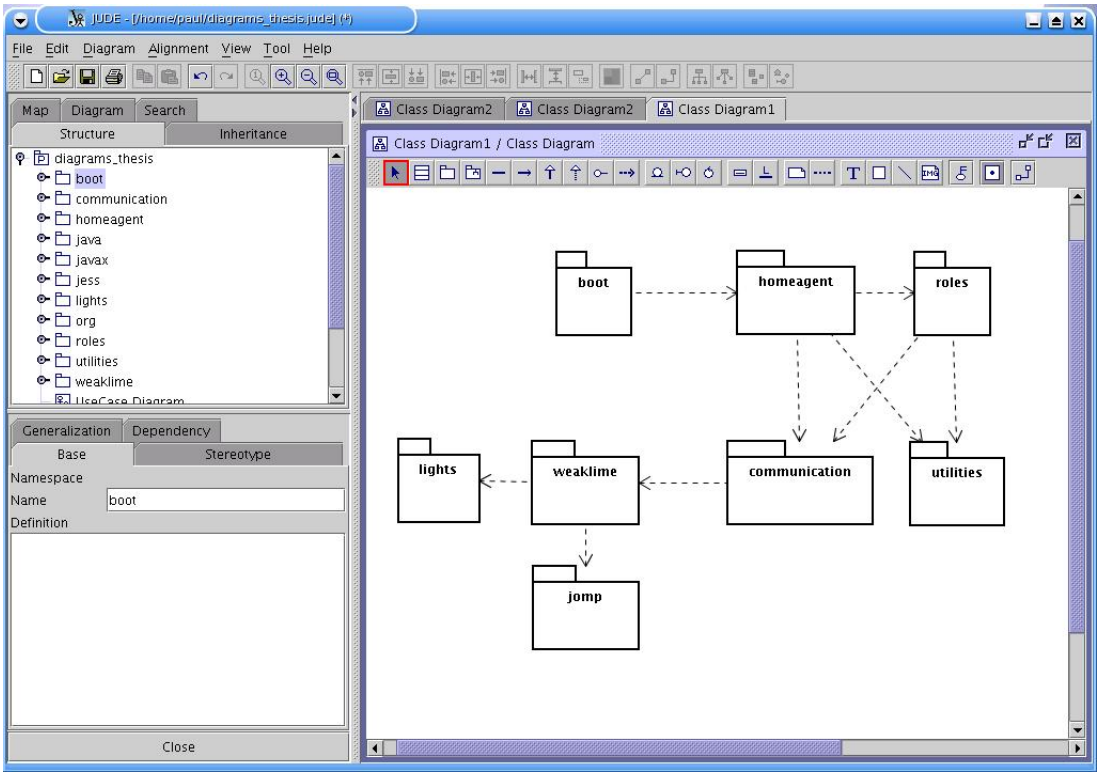


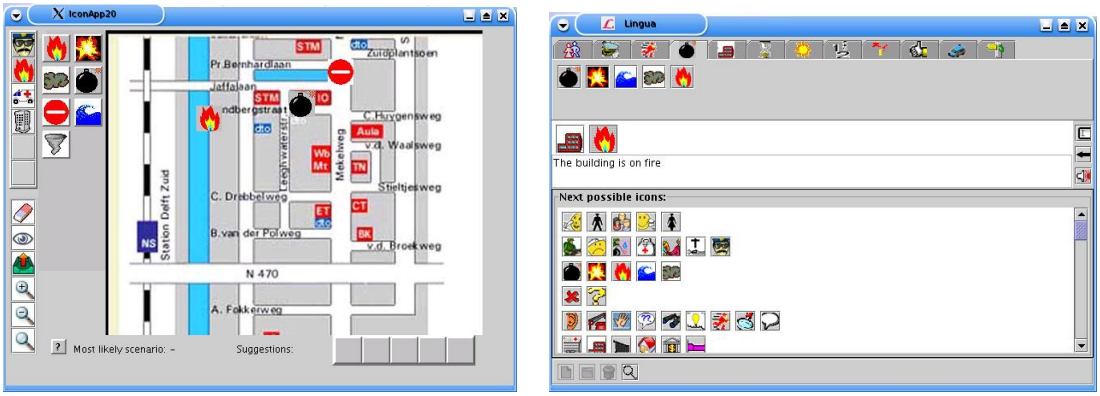
Figure 6.3: Screenshot of JUDE

as well. In the chosen model of the situation, it seems obvious that the text-to-speech part is used in the more stable part of the network. There the information can be interpreted and be used to make the view of the world more complete. For more information about the Lingua application, see [Fit04].

Adjustments

Since both test applications were primarily developed as a proof of concept of communication using icons, the communication in the original programs was kept simple. Since a different sort of communication was used in the system, the test applications had to be adjusted to fit in the system. Furthermore, the ISME application was not running on Java 1.3, since it used some libraries that were not available in Java 1.3. Therefore, the application first had to be ported to the right version of Java, before it could be run on the Zaurus devices. The difference in Java-versions mainly concerned the reading of XML files into the application. The solution to this was to use another library that was available from the Apache website (`http://apache.org`), which could (more or less) replace the libraries used in the Java 1.4 version, besides some extra parsing. Furthermore, the Java 1.4 code used the == operator to compare strings and this did not work for Java 1.3 (i.e. the result was always false), so this was to be adjusted throughout the whole program. Besides these adjustments, some measures were also taken to speed up the application, since it was very slow when run on the Zaurus.

Furthermore, using the functionalities of the developed system, the applications were adjusted so that they could be run in the system, roles could be assigned, monitored etc. This will not be described here any further. See section 6.5 for some more details.



(a) ISME (b) Lingua

Figure 6.4: Screenshots of the test applications

6.4 Used hardware & OS

6.4.1 The Sharp Zaurus

As already indicated in section 3.3, the application should run on wireless devices. The devices available at the University, are Sharp Zaurus devices. To be more precise, it concerns the Sharp Zaurus SL-C760 and SL-C860. At this moment probably not the most advanced devices anymore, since they were bought in 2003 and the developments go quite fast in this area. However, these were the devices available from the Unversity and these were to be used for any real life testing. In the picture below the Zaurus is shown. The devices have a touch screen as most PDAs and the screen can be turned around 180 degrees if the keyboard is not needed. The screen resolution is 640x480 pixels.

Specs:

Display: 3.7" TFT CG Silicon display color LCD, 64,000 colors. Semi-transmissive, backlit. VGA Resolution: 640 x 480 pixels.

Battery 1700mAh Lithium Ion rechargeable. Battery is user replaceable. This battery is the extended battery, Sharp also makes a smaller capacity battery.

Performance: Intel PXA255 400 MHz XScale processor. 128 megs of RAM for storage, ~ 64 of which is available to the user and 64 MB program memory.

Size: 4.25" x 3.25" x 1.0" at thickest point of hinge, .8" for the rest of the unit. Weight Approximately 9.5 ounces (120 x 83 x 23.6mm. Weight: 250 grams).

Software: Linux-based operating system (OpenPDA). Calendar, Address Book, To-Do, and Memo apps, Hancom Office suite: Word processor compatible with Word docs, spreadsheet app compatible with Excel files, NetFront v.3 web browser, E-mail program supporting POP3, SMTP, IMAP4 protocols, ImagePad image viewer and editor, Video Player (plays one flavor of MPEG4), Music Player for MP3s, Voice Recorder, Text Editor, Calculator, Clock, City Time, backup app and more. Java runtime included.

Audio: Built in speaker, and 3.5mm stereo headphone jack that can also accommodate a microphone.

Expansion: 1 SD (Secure Digital) slot (not SDIO). 1 CF type II slot. IR port.

(a) Zaurus (b) Specifications

Figure 6.5: The Sharp Zaurus including specifications

The devices run the Linux Operating System with kernel version 2.4.18. The devices are equipped with an ARM-processor. This had some restrictions as a consequence, since the

devices can therefore not always run software made for an i386 architecture. There is a cross-compiler available for the Zaurus, but despite this (and because of the older kernel version), not all software can run on the Zaurus. This is one of the reasons that the search for a routing algorithm was somewhat troubled, as indicated in chapter 5. The devices have 64 Megabytes of internal memory that is used for storage of the Operating System as well as for the main memory. Other data is to be stored using persistent storage such as SD memory cards etc.

6.4.2 Other hardware & OS

Other hardware that was used during the development, was in the first place a laptop. This was the device that was used for the development of the application and also took part as one of the nodes in the ad-hoc network. The network thus consisted of Zaurus devices and a laptop. All these devices were equipped with Wifi (802.11b) network cards. Furthermore, the routing algorithm used was made for Linux as a kernel module. For this reason the same kernel version (2.4.18) that was running on the Zaurus devices was also used on the laptop.
The choice for the Linux operating system was because of the fact that the Zaurus devices already ran this operating and it turned out to be somewhat easier to do the performance measurements that were necessary for determining the fitness value.

6.5 The implemented framework

6.5.1 Introduction

This section discusses the implementation of the system. The section will start with some general remarks about the development. After this, the section will continue with the implementation of the approach to bring the topology in the network. This will be followed by a section about the communication in the network. After discussing the roleassignment procedure on implementation level and discussing the monitoring mechanism, one section will be devoted to the implementation of the fitness value.

6.5.2 General implementation notes

As already mentioned in section 4.5.1, every node in the system has an agent running that controls the managing (stopping and resuming) of tasks. This agent, called `HomeAgent` in the system, thus has a number of roles running. Some of these roles are vital to the agent's functioning and can therefore not be stopped (e.g. the roles for observing the fitness and the routing table). To be able to fit a service or an application into the system, this has to be able to be started and stopped. For this reason, every class has to implement a general class, called `NamedThread`. Implementing this class entails implementing the starting and stopping of the service, providing the process id and the start time (both will become clear in section 6.5.11), returning a name and a method for handling panic. The latter method should define what action to take when the node is expected to go down or leave the system soon (according to the system).
Besides implementing the interface `NamedThread` the nodes can also extend the class `Role`, that already defines some default behaviour for the methods. These can be overridden if this is desired.

The functionalities described in the rest of this section are all fitted into the system like described above. Adding more applications can be done in the same manner, so it is possible to add more services in the future, with perhaps other functionalities than ISME and Lingua offer. The system consists of the packages shown in figure 6.6.

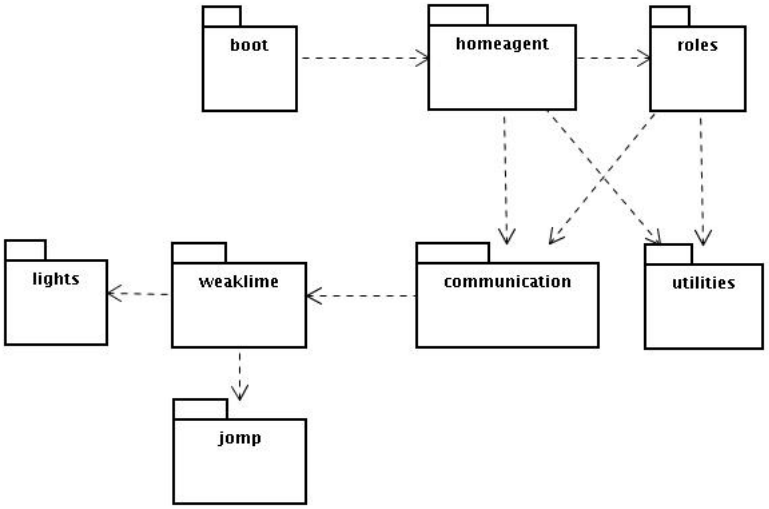


Figure 6.6: Overview of the packages in the system

The package "utilities" contains classes for the locking mechanism, the logging in the system, a package that offers some services concerning the networking (e.g. finding a port that is not yet in use), the ontology file that is used by the entire system, wrapper objects, a package that maintains general information about tasks etc. Some of the classes from this package will be mentioned and shortly discussed in this chapter. Some of these (simpler) classes are only helpful "on the background" and are therefore not discussed any further.
The package "roles" contains the services that are to run in the network. This package contains the (sub)packages and classes for assigning role, monitoring roles, fusion, maintaining the position of roles, parsing of the route information etc. The content of these packages will be the main focus of the remainder of this chapter.

6.5.3 Crisis center

The implementation of the crisis center in the network, was done by making a special role that took care of the crisis center. As already indicated in section 4.3, the crisis center is used as a reference point in the network. Therefore it is one of the functionalities that is monitored. The crisis center message contains the leader of the crisis center (which is the owner of the message), that the other nodes can then use to lookup in their routing table for the amount of hops. For more information on how the roles are exactly assigned to the nodes, see section 6.5.4 and 6.5.6.

To make sure that there is only one crisis center at a time, there is a simple locking scheme used for locking the blackboard. Since the developers of Lime found that it is impossible to guarantee that an action is atomic on the whole blackboard and at the same time, keep the speed of the blackboard high enough, this feature was removed from Lime. For this reason, no new mechanism was added to Lime to make actions atomic anyway. The implemented scheme therefore uses timing. Perhaps a protocol for leader election can serve as an improvement, but since the application was only used as a proof of concept, no time was invested in implementing an algorithm to handle this[1]. If a node wants to join the crisis center, it has to get the lock first. This is to make sure that the crisis center message that is on the blackboard will be modified by only one node at a time. If multiple nodes want to get the lock at the same time, the fitness

[1]These algorithms are actually quite complex due to the fact that they have to deal with the fact that at every moment a node can go down. This is usually not taken into account in election algorithms such as in Peterson's election algorithm.

of a node is used as a decision criteria to decide which node gets the lock. If a node does not get the lock, because another node already has it, the agent just backs off and waits for a few seconds (actually a random number of seconds), before a retry.
The locking scheme also takes care that the lock is not held too long by a node (due to a crash or something similar). Therefore, if a node sees that the lock is already held by a node in the network, it first checks to see whether the node is still "alive". If not, the lock will be released and the node can try to get it again.

6.5.4 Position of roles

As already indicated before, the roles are assigned in the network to the fittest node among the nodes that are in the right part of the network. For this reason, a specific package named `taskcontrol` maintains a list of roles that can run in the network. Based on this list, that is known to all nodes, the nodes know where a role should, in the ideal case, be executed. Besides this information, this package also provides information about the priority of the roles, i.e. how important it is that this role is to be executed. This information is also used during the role assignment, just like the minimum fitness that a node should have to execute the role. By having this overview, every node in the system is able to assign a role to another node, since it has all the necesary information to do this. This thus enables the assignment of roles in a distributed manner (see section 6.5.6).

One of the functionalities in the system was also to reassign roles to the right part of the network if this is considered as "better", with regard to the topology of the network. The nodes can move through the network freely, since they are carried by people. Therefore one of the roles of the agents takes care of the re-assignment of roles if this is necessary. Algorithm 1 shows the implementation of this idea in pseudocode. The role switch notification is to notify interested nodes that the service has changed location.

This re-assignment of roles, based on the place relative to the crisis center, is thus an attempt to preserve the structure that is brought into the topology of network, as indicated in chapter 3. This is thus an attempt to repair the desired topology if it is "broken", the topology that facilitates the emergence of an optimal communication infrastructure.

6.5.5 Communication using blackboards

The implementation of Lime was made in Java and could thus be used on the Zaurus devices. Before this could be used, it had to be made suitable for the Zaurus devices, i.e. it was to be converted to Java version 1.3. After some initial experimenting with the software, it turned out that The Java Virtual Machine (v. 1.3) for Linux has somewhat primitive behaviour when it comes to the handling of IP addresses when multiple network interfaces are present on a system. It turned out that in order to use the "right" IP address (i.e. the right IP address for the right wireless network interface), the file "/etc/hosts" was to be edited in a way that the current hostname was associated with the IP address of the wireless network interface to use. In the eventual system this problem is solved in the startup script, so the programmer does not have to worry about it anymore.

After some more experimenting it turned out relatively easy to use. Basically every node has to start up a sort of local Lime service that could[2] then share the local blackboards (or

[2]in fact the term *"could"* is not entirely correct here, since in the new version of Lime it was no longer possile to have "local" tuple spaces. All tuple spaces were *always* shared.

Algorithm 1: Handling the roles concerning their position in the network

```
1: while running do
2:   numberOfHops ← Homeagent.getNumberOfHopsToCrisisCenter()
3:   if numberOfHops ≠ −1 then
4:     threads ← HomeAgent.getRunningProcesses()
5:     for all elements in threads do
6:       if ! isUserProcess() && ! isInRightRange(taskType, numberOfHops) then
7:         put a lock on the bb
8:         attempt to assign role elsewhere in the network
9:         if successfully assigned then
10:          stop the local process
11:          put a RoleSwitch notification on the BB
12:        end if
13:        release lock
14:      end if
15:    end for
16:  end if
17:  pause
18: end while
```

tuple spaces[3]) that can be created by an agent in the system. However, the Lime system concerned a pre-release of the eventual software. Because of this, the software was not entirely bug free. A specifically nasty bug that was found, concerned a bug in a `ThreadPool` class. This bug, which would "sometimes" make the whole system hang, did not always occur, which made it particularly difficult to find it. Because of the fact that the person that had built Lime, was no longer doing development on the software, the debugging of the Lime system had to be done by ourselves.

For the sake of simplicity a special class was added to the communication package in the framework. This class was basically a wrapper for the calls to the Java code of Lime. This was to make the communication more easy to make it less likely that mistakes are made when implementing services. Because of this wrapper class, that is called `BBCommunicator`, the services that are running within the system seldom have to use actual, sometimes somewhat difficult, method calls directly to Lime. This is all taken care of by the communication package. More importantly, the communication package also took care of queueing messages that come in. The reason for this queueing was that the Lime system would wait until a message was handled. If a lot of work was to be done, no access to the blackboard would be possible on the node anymore. Furthermore, this also restricted responding on a message by accessing the blackboard. The reason is that the blackboard is blocked for everyone, including the running thread, when handling messages. Therefore two queues were made for registration messages and for other messages. This way, the incoming messages are put in the queue, after which the blackboard is accessible again. The message can then be handled by a different thread.

Communication with the blackboard can be done in multiple ways. For reading there is a choice between a blocking and non-blocking read of the blackboard. Since we are dealing with a MANET in a crisis situation, the chance that nodes go down is relatively high in comparison to "normal" situations. Therefore blocking methods are avoided. This can easily lead to dead-

[3] "Tuple space" is the term used in Lime, but for clarity we will use the term blackboards from now on.

locks, which is something that is to be prevented at all times, certainly when dealing with a shared blackboard. A deadlock could make the whole blackboard communication stop, which could eventually stop all communication in the network. For this reason, only non-blocking methods were made available directly. Some blocking methods are also provided, but these cannot block the rest of the system. Waiting on a certain piece of information could be done by using reactions. This feature of Lime works as follows: A node can register for one or more types of messages. If a message of a registered type is put on the blackboard, a Java event is fired. This event also indicates what message triggered the event. The communication package then takes care of this message by adding the message to the local queue.

To make a distinction between messages of different services, all messages that are sent have a message type as indicated in section 4.4.1. The system keeps a file with all these possible message types, called `Onto`. This way it is possible for services to easily identify messages that should be read and handled and messages that should not. The use of these message types is not compulsary in Lime, but is automatically done correctly if the blackboard communication class in the system is used. Furthermore, if the messages are sent in this way, the sent messages are wrapped in a wrapper object. This can add some information to the messages such as source IP address and time of sending and arriving. The latter information can be used for sorting of messages and can (later) also be used for actions that require timestamping (e.g. Dynamic Bayesian Networks).

As already indicated in section 4.5.2, nodes that have to be monitored, put a registration on the blackboard. If the service that puts the registration on the blackboard should be contacted, it should include it's own IP address in the message including the port where a serversocket is listening. This way it is possible for services to open a one to one connection to send messages from one node to another node that runs a specific service, without the need to know the name or address in advance. These direct connections are also handled by a class from the system, called `DirectMessageSender`. This abstracts away the communication for the largest part from the programmer. Another part of the communication concerned the communication within a node. Since the applications that are to run are sometimes relatively "separate" from the rest of the system, some pieces of information were made available in the `HomeAgent`. This was done by using the Remote Method Invocation that is part of Java. This was for example used for making the fitness of the node available to the different services in the system.

Besides the use of the blackboard in the stable core, the system offers the possibility to use more than one blackboard. The implementation of the blackboards uses multicasting to send messages to the nodes that are part of the blackboard. Using multiple blackboards means that there are multiple groups in the system. To get the communication fitting for the situation as modeled in chapter 3, the services that are to be running in the outer layer are using the so called "level1" blackboard(s) as they are called in figure 4.3. The nodes in the stable part of the network primarily use the "level0" blackboard to get their messages from. The fusion nodes, can then take messages from the "level1" blackboard, process it and put it on the "level0" blackboard. The class `BBCommunicator` offers the choice for a blackboard in the constructor. In figure 6.8 a simplified sequencediagram is given of the communication flow from the outer to the inner part of the network (see appendix B for a somewhat more readable version of this diagram).
In figure 6.7 a class diagram of the communication package in the system is given. For more details, please refer to the (Javadoc) documentation that is provided with the code.

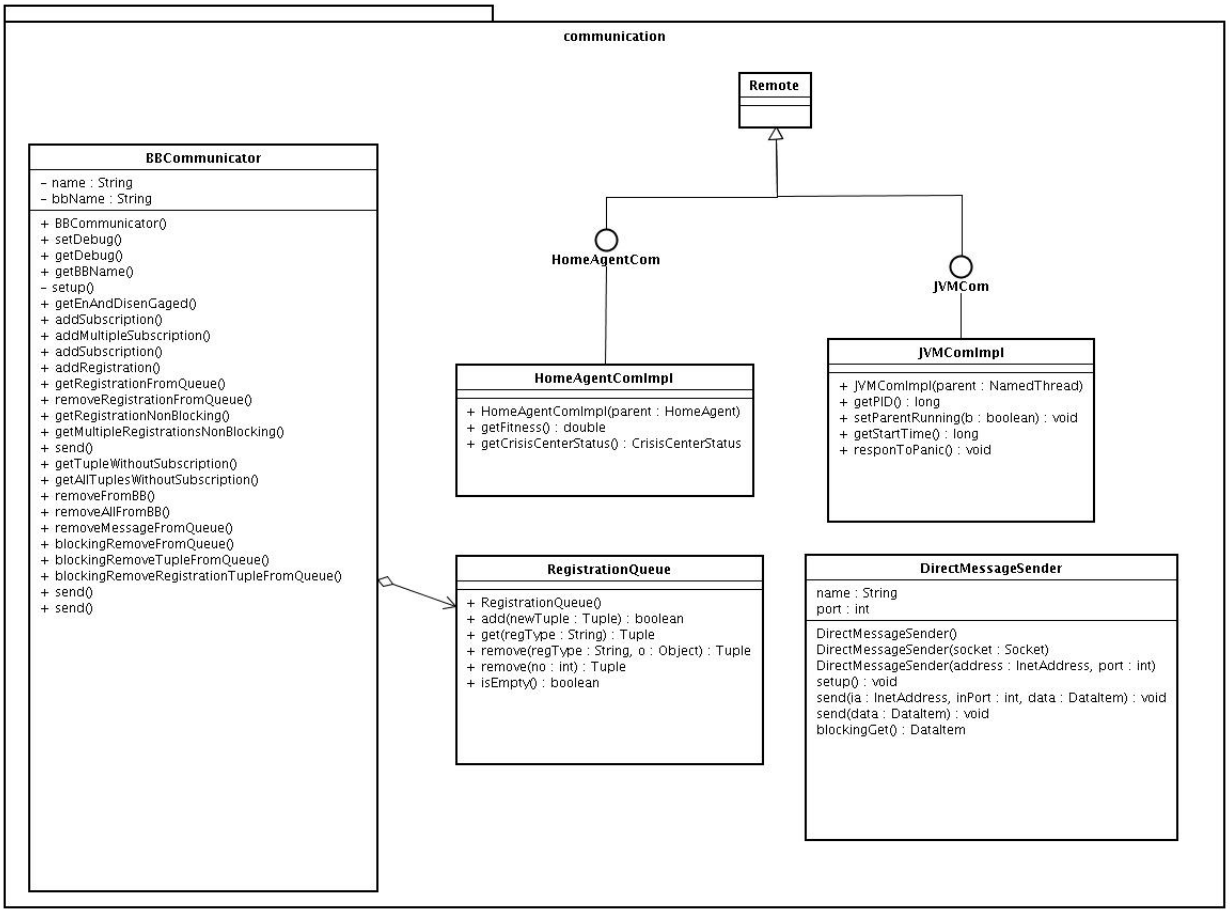


Figure 6.7: Class diagram of communication package (BBCommunicator is displayed somewhat simpler for readability)

6.5.6 Role assignment

The role assignment procedure uses an auctioning mechanism as indicated in section 4.5.3. Another approach that could have been used, is to send around a list with tasks and fitness values of the nodes to come to a decision. Based on this list that every node should receive from every other node, a distributed decision could then be taken. However, this approach seems not suitable for a MANET, since messages do not always arrive (on time). This means that the views that the nodes have, can be inconsistent, which can lead to the situation in which the nodes assign roles multiple times, or a role is not assigned at all. Besides being decentralized, an auctioning mechanism also offers the possibility to reason about the decision.
In most cases, the monitor of a role will locally start a `RoleAssignmentRole` that handles the assignment of the role to another node. However, other nodes are also free to assign roles in the network (e.g. keeper of replicas, but also missing services). The role assignment in the system is based on communication via the blackboard. The node that wants to assign the role to another node takes the role of auctioneer, which is thus called `RoleAssignmentRole` in the system. The first thing it does is to put a request on the blackboard that contains the role to be executed, the address of itself and the minimum fitness that is necessary to be able to execute a role[4]. Then it starts waiting for nodes that offer to fulfil the role. It sets a timer and the nodes that respond before the timer goes off will be taken into account.

[4]To prevent all nodes, including the worst nodes, from responding

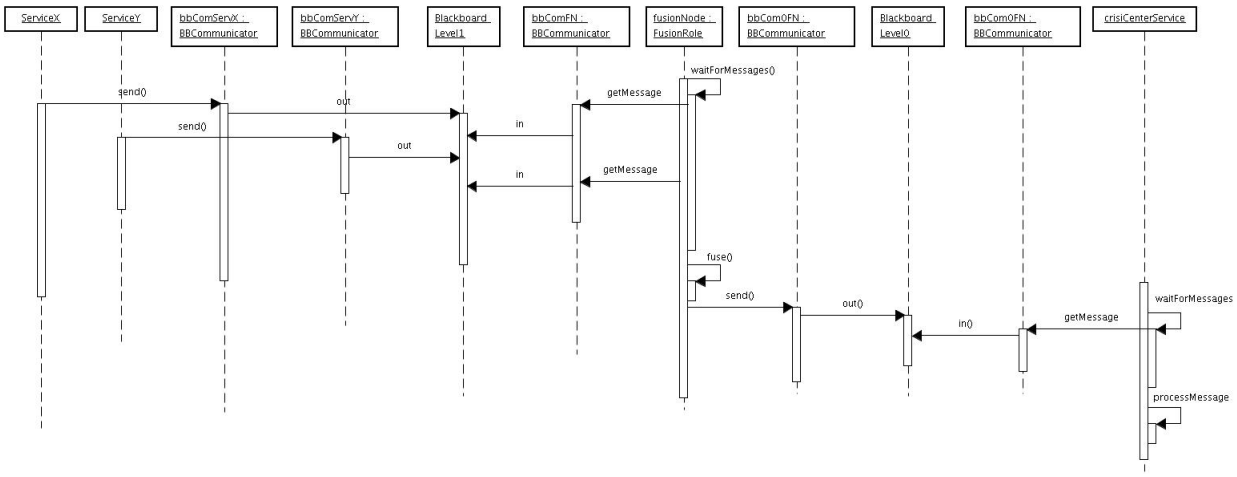


Figure 6.8: Communication flow from the outer to the inner part of the network

The nodes that want to be able to respond to requests for fulfilling roles, must have a `RoleAcceptor` running (which is of course a role that a node can or cannot have). If a node has one, it can check its own fitness and compare it with the minimum fitness that is needed to fulfil the role. If it has a fitness that is higher than the one requested, it can offer to fulfil the role. If it has a fitness value that is less than the one requested, it can first have a look what other roles it has running. It can check how heavy these roles (approximately) are in terms of memory and CPU usage. Based on this it can calculate how "fit" the node could be when it would shutdown some of the other roles. Of course it should only consider roles that have a priority that is lower than the role for which the request for fulfilment is placed on the blackboard. Furthermore, certain tasks that are started by the user are also not taken into account, since it is not very user friendly to shutdown a role that the user has specifically asked for.

Once the `RoleAcceptor` has decided that it will offer to fulfil the role, it puts an offer on the blackboard with which it thus indicates that it can and is willing to fulfil the role. This offer that is put on the blackboard should contain the address of the node (since it has to be contacted if it is assigned the role), the fitness of itself, whether or not it has to shutdown other (less important) roles in order to fulfil the role and for this prototype was also taken into account what kind of device we were dealing with (a laptop is usually has more resources available than a Zaurus so takes less time to start). Furthermore, the distance to the crisis center (number of hops) is also to be included. Based on this information, the auctioneer can then decide who will be assigned the role.

When the timer of the `RoleAssignmentRole` goes off, it stops reading the offers from the blackboard and chooses a node to execute the role, based on the offers that it has received in the time that it was waiting. The decision is firstly based on the location of the node in the network. The nodes have to indicate how many hops they are away from the crisis center. Based on this information will be attempted to assign the role to the node that is at the right place in the network. From the subset of nodes that have the "right" hopcount to the crisis center (i.e. hopcount in the right range), the role will be assigned to the fittest node. If no crisis center is seen/available in the network, only the fitness of all nodes will be taken into account. It chooses the node that has the highest fitness from among the nodes that do not have to shutdown other roles in order to execute the role. If it finds no roles that can do this, than the second choice is to assign the role to a node that must shut other (less important) services down. The pseudocode for this algorithm can be found in algorithm 2. This is the

version that assigns the role to any node if there is no node in the right range. There is also an option to stop the procedure after line 13, i.e. only assigning in the right range if possible, failing otherwise.

Algorithm 2: Getting the best offer

```
1: if crisis center is available then
2:   taskType ← getTaskTypeForRole
3:   range ← getRangeForRole(taskType)
4:   bestOffer ← find the best offer in range (without stopping other roles)
5:   if bestOffer! = null then
6:     return bestOffer
7:   else
8:     bestOffer ← find the best offer in range (with stopping other roles)
9:     if bestOffer! = null then
10:      return bestOffer
11:    end if
12:  end if
13: end if
14: bestOffer ← find the best offer in all ranges (without stopping other roles)
15: if bestOffer! = null then
16:   return bestOffer
17: else
18:   bestOffer ← find the best offer in all ranges (with stopping other roles)
19:   if bestOffer! = null then
20:     return bestOffer
21:   else
22:     return null
23:   end if
24: end if
```

When the auctioneer has chosen a node that should execute the role, it tells the node to start the role. If this node then replies on this message with a confirmation, it is considered to be started. This confirmation is needed to find out whether the node is still alive. If the assignment to the chosen node fails, the node with the second best offer will get the role. This is repeated until there is a node that has successfully started the role (or all nodes fail). Algorithm 3 shows the pseudocode for the algorithm.
In figure 6.9 a (simplified) sequence diagram gives an overview of the total role assignment procedure. This diagram shows only one agent that responds (and gets the role as well).

6.5.7 Fusion

As already indicated, the fusion of the data from the observers takes place in the middle layer of the network. Now that we have seen how roles are assigned in the network and how the communication flow is to take place in the network, the fusion nodes are relatively easy to understand. As suggested by figure 6.8 these nodes are continuously waiting for messages that are put on the blackboard. The system provides the class `FusionNode` for this. If this class is extended, i.e. a subclass is made, with a minimum effort it is possible to start a `FusionNode` for a service. The class has a number of associations between messages and the input and output blackboards of this message type. These have to be given to the role, so it knows what messages it should get from which blackboard and where to output the result. It thus makes

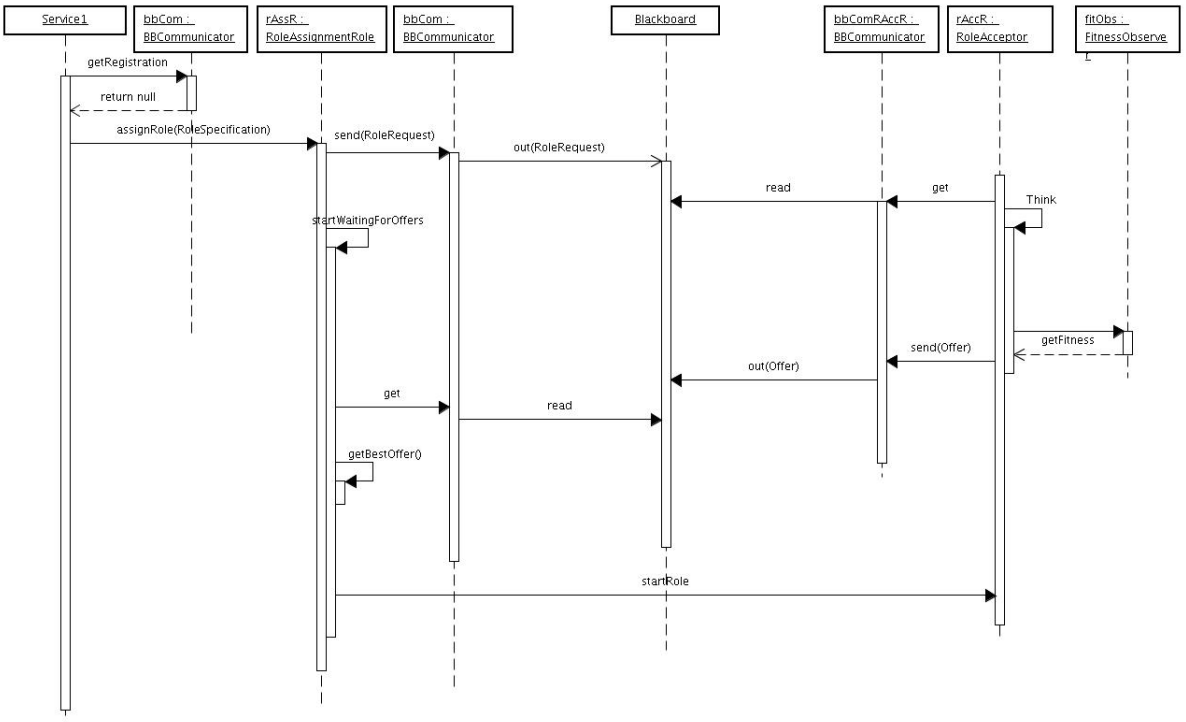


Figure 6.9: A simplified sequence diagram of the role assignment procedure

a selection of messages that it gets from the blackboard (which is done by subscribing to the types of messages). Furthermore a subclass of the abstract class `FusionQueue` is to be provided. This class will then be used to store and fuse the incoming data. This queue thus has a method `fuse()` that can take care of the fusion.

The subclass of the `FusionNode` inherits the behaviour of its parent, which entails that it automatically gets all the specified messages from the blackboard, adds them to the right queue and calls the method `fuse()` on all queues every x seconds. It is thus done using a time frame. The default behavior of the nodes is that the messages that are fused are discarded. However, this "historic" information can also be saved for future use for reasoning in time. Furthermore, the fusion nodes can then also remove the messages that are "out of date", which can become important when the system is running for a longer period of time. After executing the `fuse` method, the result of the fusion is then automatically put on the output blackboard. Figure 6.10 gives a class diagram for the relations of the classes discussed.

6.5.8 Monitors

The mechanism of monitoring is implemented in a way that a service is monitored by a monitor role on another node. Monitoring the same node is useless, since this would mean that if the node goes down, the monitor is gone as well. The monitoring is implemented by a class `Monitor`, which should be extended (i.e. by making a subclass) to make a monitor for a service. This class checks for the registration every x seconds. If the registration of the service to monitor is gone, this is considered as a service that is gone from the network. To deal with the fact that messages in a MANET can get lost sometimes and that sometimes a new route is to be established to a node, the monitor only decides after a specific (adjustable) number of failures that a service is down. After checking whether the service is not being re-assigned to another node (which is possible as discussed in section 4.5.2), the role assignment procedure is started

Algorithm 3: The auctioning mechanism

```
1: make RoleSpecification
2: put request message on the BB including RoleSpecification
3: while waiting do
4:    get RoleOffer from the blackboard
5:    add RoleOffer to offerVector
6: end while
7: if offerVector.isEmpty() then
8:    return failure-message
9: else
10:   while !offerVector.isEmpty() do
11:      bestOffer ← getBestOffer(offerVector)
12:      succeeded ← assign role to owner of bestOffer
13:      if succeeded then
14:         return success-message
15:      else
16:         remove offer from the offerVector
17:      end if
18:   end while
19:   return failure-message
20: end if
```

to re-assign the role to another node. Monitors offer the possibility to monitor a service at *a* node, or at a *specific* node. For the latter case the monitor also checks to see if the service is not being reassigned due to the changing topology. See algorithm 4 for the pseudocode for the monitoring algorithm.
To use the monitors, first a service (important enough to be monitored) is started. This service then assigns a role to a node in the network, to monitor this service. This is thus implemented using the role assignment procedure discussed in section 6.5.6. After starting the monitor, the node locally starts a MonitorMonitor, a class to monitor the monitor, as explained in section 4.5.2. The MonitorMonitor has the same functionality as a monitor and therefore, in order to implement a monitor or a monitor-monitor, extending the class `Monitor` provides the monitoring functionality. For this reason, no specific class `MonitorMonitor` is provided, although the subclasses preferably carry this name to indicate the exact function of the class.

6.5.9 Replication

Replication i implemented by a specific class called `ReplicaKeeper`. This class provides the functionality to keep replicas of a certain type of messages. It keeps a queue for every message type it is to replicate. For accessing the right queue, a `Hashtable` is used to map the types to queues. Furthermore, since it is a role, it can be assigned to other nodes, so that replicas are kept from other nodes. This assignment can take place anywhere in the network. In principle this class can also be used to replicate one message, but due to the nature of most messages this was not necessary. Furthermore, the `ReplicaKeeper`s are offered as a standard functionality of the monitors. As already explained, this way, when a service has disappeared, the service can be restarted and the last message can be placed back on the blackboard. The amount of replicas that is kept in the `ReplicaKeeper` is adjustable, so it is possible to keep a history of messages of a certain type, so that it can, for example, be used to redo the reasoning in a service (e.g. one of the future plans within the Intelligent Systems project is to replace the reasoning

Algorithm 4: Monitoring

```
1: missingCounter ← 0
2: while running do
3:    found ← checkForRegistration()
4:    if found then
5:       missingCounter ← 0
6:       pause
7:    else
8:       reassigning ← checkForReassignLock()
9:       if reassigning then
10:         missingCounter ← 0
11:      else
12:         missingCounter + +
13:      end if
14:      if missingCounter == missing then
15:         if !isLocked then
16:            Put lock on BB
17:            succeeded ← re-assign role to fittest node
18:            if succeeded then
19:               missingCounter ← 0
20:            else
21:               missingCounter − −
22:            end if
23:            Remove lock from BB
24:         else
25:            missingCounter ← 0
26:         end if
27:      end if
28:   end if
29:   pause
30: end while
```

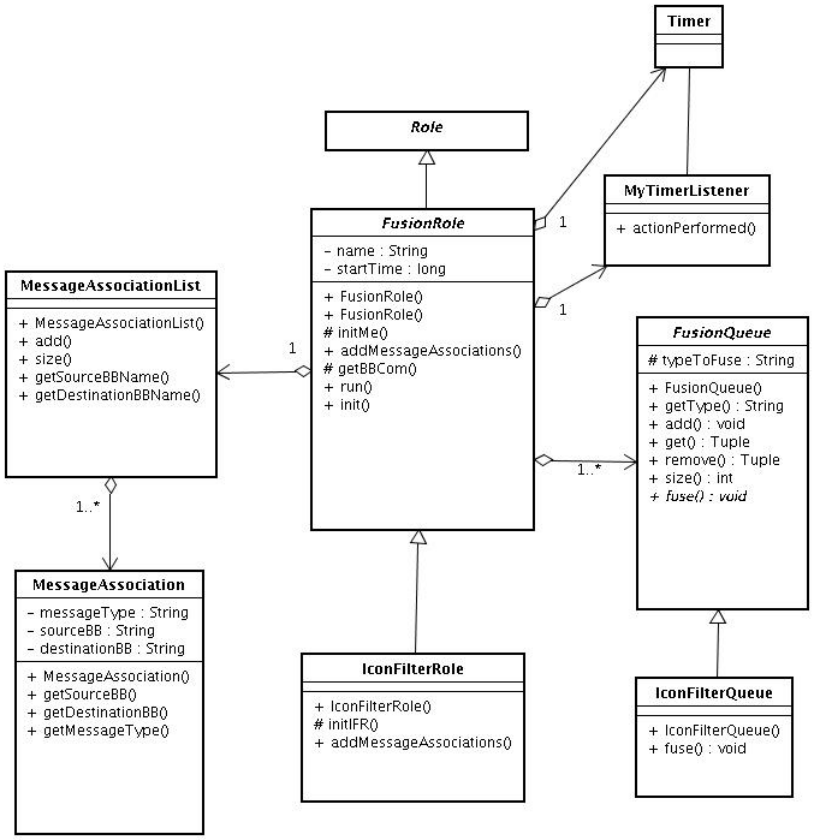


Figure 6.10: Class diagram of a fusion role

in the ISME application with rules from an expert system by Bayesian reasoning). This way the `ReplicaKeepers` can take care of the storage of information in the network. Of course the replication nodes have the possibility to remove part of the data if this is thought to be wise.

6.5.10 Solving duplicates

As indicated in section 4.5.4, there is also a class available that takes care of solving duplicate services. This is primarily used for the problem of group partition. If a service is started in a separate group of the network and later the groups merge again, this could mean that some services are running twice. For the implementation a class `DuplicateSolver` is used, which can be run on a node as a role. This class is watching the blackboard, to see if there are services in the network that are running at two positions at the same time, while this is not desired. This is done by checking the registrations on the blackboard against a list of services that should run only once. Once two (or more) duplicate services are detected, the duplicate solver asks all the services for their fitness value. The service that is running on the fittest node will be the one that is allowed to continue its operation. The other service(s) get(s) a message that it / they should stop running. Stopping in this case also means that a service has to stop its own monitor, since otherwise it would be restarted once it stops.

Besides checking whether there is more than one copy of a service, this is also done for the duplicate solver itself. The reason for this is that it is not wise to have two services that are trying to solve duplicates. In the worst case this could mean that eventually none of the services is left.

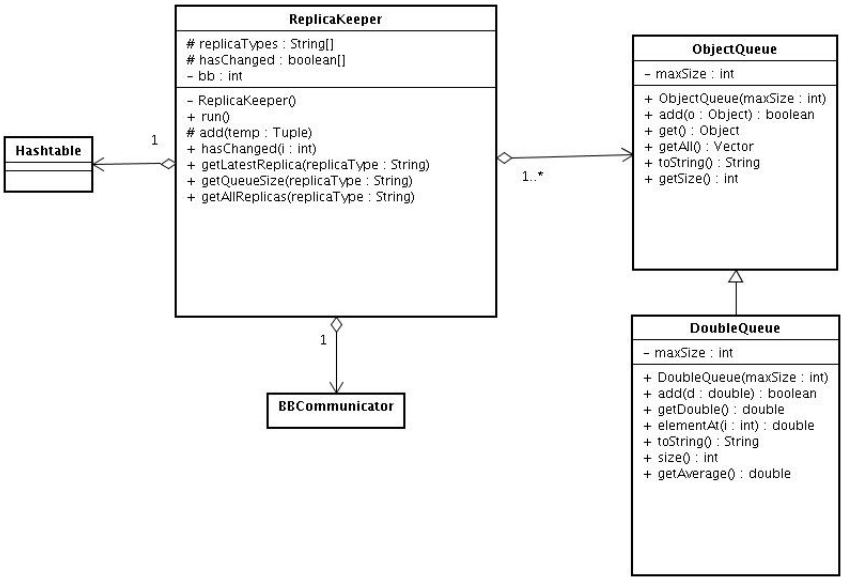


Figure 6.11: Class diagram of ReplicaKeeper

6.5.11 The fitness of a node

As indicated in section 4.6, a value for the "fitness" of a node was necessary to determine which node will be assigned to execute a task. In this section the form of the formula was also given (see equations 4.3 and 4.4). The formula that is used, is build up of a number of components that are, in turn, calculated based on the values of some performance factors that were taken into account. This section describes the way these values were measured and calculated.

Monitoring the performance of a "node" is something that turned out not to be possible by using standard Java libraries. Some experimental performance monitoring in Java turned out not to be helpful. The profiling tool that is in Java 1.3.1 is "experimental" [sun] and it only gave information *after* the program exited. To get process information while the program was running, a certain combination of keys had to be pressed (which made it not very useful during the run of the program). In the newest version for Java, Java 1.5 a.k.a. Java 5, a performance monitoring tool is available. However, the target devices, the Zaurus PDAs, could only run Java 1.3. Thus, since there were no free solutions available, some workarounds were used. As will become clear, these values are determined using platform specific commands / workarounds.

The idea was to take into account the performance of the different processes. The Zaurus devices were (luckily) running Linux, which made it possible to find some reasonable workarounds to get the information needed.
One of the first pieces of information that is needed to get some information about a process in Linux, is the id of a process. In the implemented framework, the roles (or tasks) that are executed in the system are all separate Java threads. In the Java version running on the Zaurus devices, the different Java threads have their own process id. The Java versions on regular pc's that were used during development did not have separate id's for Java threads. However, to do some performance measurements anyway, the pc's forked off new Java Virtual Machines instead of threads. This way the operating system considered it as saperate processes and thus gave it separate process id's. Of course this is inefficient but we considered this to be acceptable, since it was part of a "proof of concept". Furthermore, the used pc's were fast enough to "keep up" with the Zaurus devices.

Now the id of the processes could be found with a native call from the process under consideration. In the implemented system, to get the process id from the operating system the Java Native Interface (JNI) had to be used for the native call. The Java code thus calls a (simple) C program that returns the process id of the current running thread (see appendix A for the C code).
However, because of the different architectures, it was not possible to use the same library with the native C code for the Zaurus devices (with an arm processor) on the one hand and a laptop (with an i386 architecture) on the other hand. The C code is identical, but the C code had to be cross-compiled[5] for the arm processor. The Java code was quite simple, as can be seen in figure 6.12.

```
/**
 * This method is for returning the process id of a process. It is
 * implemented in C.
 *
 * @return the pid of a process
 */
public static native long getpid();

static
{
    if(Onto.DEVICE==Onto.MYLAPTOP)
        // the compiled library for the i386 arch.
        System.loadLibrary("j_jnicom-i386");
    if(Onto.DEVICE==Onto.ZAURUS)
        // the compiled library for the arm arch.
        System.loadLibrary("j_jnicom-arm");
}
```

Figure 6.12: The Java code for getting a Linux process id

Besides writing this Java and C code, some technical details had to be taken care of. These will not be further explained here. See [jnia] for more technical details about JNI.
With the process id that was now availabe, some "standard" Linux commands could be used as workarounds for getting the performance information.

One of the factors that is taken into account is the *CPU usage* of a process. This is measured using the ps-command in Linux. This command returns the following information (see Linux User's Manual):

> *"%CPU shows the cputime/realtime percentage. It will not add up to 100% unless you are lucky. It is time used divided by the time the process has been running."*

This value is used together with the time that a process has been running (measured within the process itself). By multiplying these values an approximate of the CPU usage of a process can be determined. For the total CPU power of the machine, a simple call to the top command sufficed on "normal" Linux machines. On the Zaurus devices, the CPU usage from the top command gave useless information, which meant that the whole output of the "top" command had to be parsed to calculate the value of the CPU usage.

[5] A cross-compiler is used, since the Zaurus devices do not have enough space for a complete version of gcc. Furthermore, the devices are to slow to compile large programs on.

For the *memory usage* of a process a similar workaround was used. Although it seems possible to get the amount of memory used by a program, by "manually" calculating the memory usage of all object and adding these values, this was expected to be too expensive for the battery limited devices such as the Zaurus. Furthermore, writing procedures to get the memory usage seemed to be much work and it was not known in advance whether the values calculated would be a reasonable approximation. Therefore an alternative solution was used in the form of parsing the output of the "top"-command.

For the *battery power*, another Linux command was necessary. This command was also dependent on the powermanagement application/kernel-module installed on the machine. Some possibilities are APM[6], ACPI [7] and Acpitool (which uses ACPI). The output of, for example, APM was:

> *"AC on-line, battery charging"*

when the battery was charging or

> *"AC off-line, battery status low: 25%"*

when the system was using the battery. This output was usable by the program, by forking the process in Java using a `java.lang.Process` and getting the output. Other information that was relatively easy to get from the (Linux) operating system, were the total usage of memory and the total usage of storage. For these values the outputs of the commands "free" and "df" respectively were used. As with the output of the top command etc. these outputs were parsed in the Java programs running on the devices.

The functions described above were all part of the class `LinuxNodeMonitor`, which implements the interface `OSNodeMonitor`. The routing information was maintained in a separate package. The routing information that is available on the system is retrieved from the routing algorithm running on the node. Since two routing algorithms were eventually used (see chapter 5), an abstract class `RouteLogParser` was made, that is extended by the parsers for the different routing algorithms (see figure 6.13). If another algorithm is to be used, only another implementation is to be provided, without changing the rest of the system. The RouteLogParser keeps a list of `RoutingEntry` objects that store information about the routes to other nodes.
The routing information was parsed every x seconds. This information is used to determine how many direct neighbours (i.e. within one hop) the node has and how many other nodes it can reach. This former information was a part of the fitness as indicated in section 4.6.

The fitness value is calculated by the class `FitnessObserver`. This class has an interface `FitnessFunction` that is used to calculate the fitness every x seconds. This could be implemented by different functions, which made it possible to compare the chosen fitness function with a linear function that is sometimes used in other approaches. The package that is responsible for calculating the fitness is shown in the class diagram in figure 6.14.
Now that we know how the components are found, it was time to determine the parameters of the fitness equation. As already explained, a multiplication was used, since it makes the fitness value more capable of reflecting the importance of certain factors in certain intervals. The shape of how the relation should look like was already indicated, and a function that exhibits this behaviour was already given in equation 4.3. The way that the variables bf, cf, mf, sf and x are determined has been described in the aforementioned. What remains is to determine the value of the parameters (or sensitivities) ss, ps, cs, ms and ns. This has been done in two steps.

[6] Advanced Power Management
[7] Advanced Configuration & Power Interface

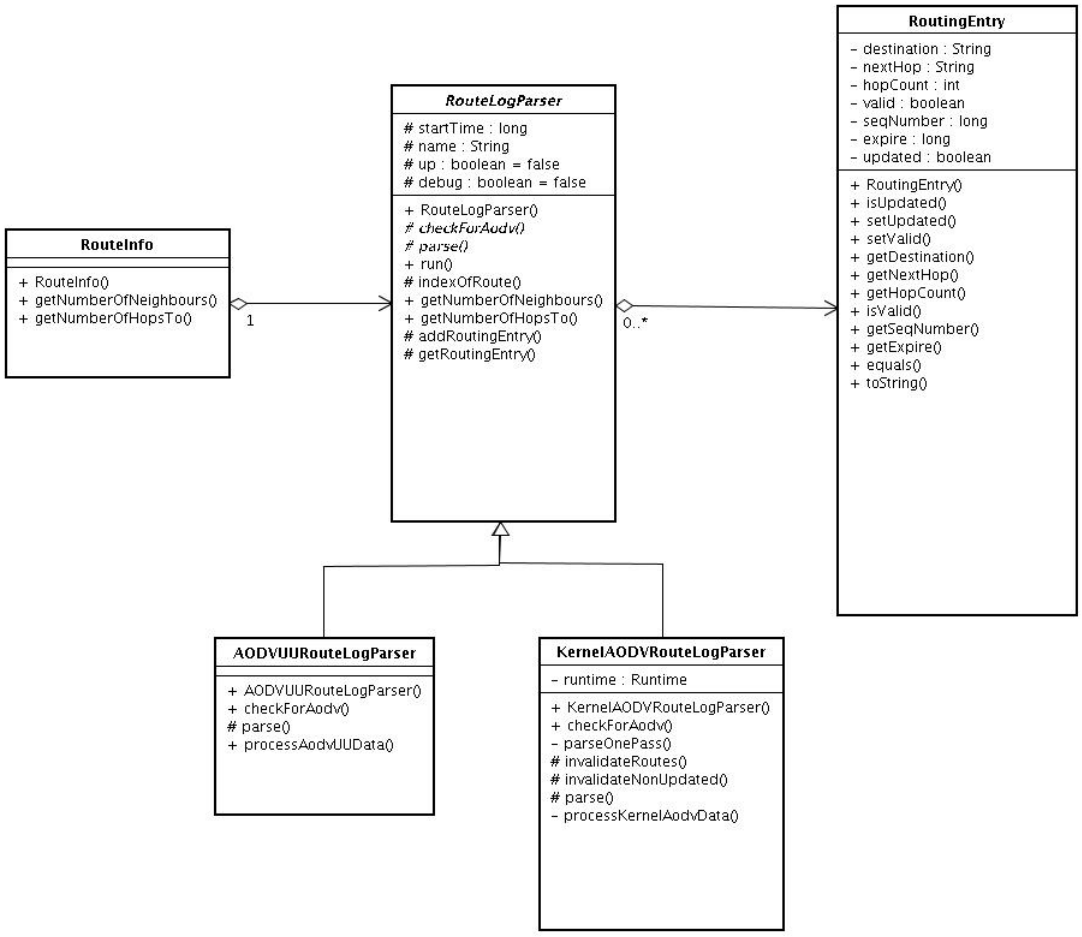


Figure 6.13: Class diagram of the package for maintaining routing information

The first step was to determine how the shape should approximatey be. This was mostly done using mathematical software[8], that could easily plot figures similar to figure 4.7. This lead to initial values for the parameters of the fitness formula. The second step was adjusting the parameters based on experiments. This mostly concerned controlled experiments (i.e. for different values of the parameters, adding tasks manually and logging the performance variables and the calculated fitness values). This way initial values were determined for the different parameters. Later, during the experiments, the performance of the whole system that is based on these calculated values was measured (see section 7.3).

6.5.12 Panic services

In order to make the system suitable for the use in crisis situations, some extra functionalities were added to the nodes, in order to prevent from data loss and show the use of the system. One of the functionalities that is already discussed in section 6.5.11 is the fact that the fitness of a node is calculated every x seconds. This fitness is not only used for the assignment of roles to a node, but is also used as a heuristic to determine whether or not a node is going down. If the fitness of a node is very low, this means that is will probably not get more roles to fulfil, but in the case that it would go down, the data that is stored on that node is lost in case this is not replicated or already sent to the (mobile) crisis center. For this reason, there is a mechanism

[8] Waterloo Maple 8

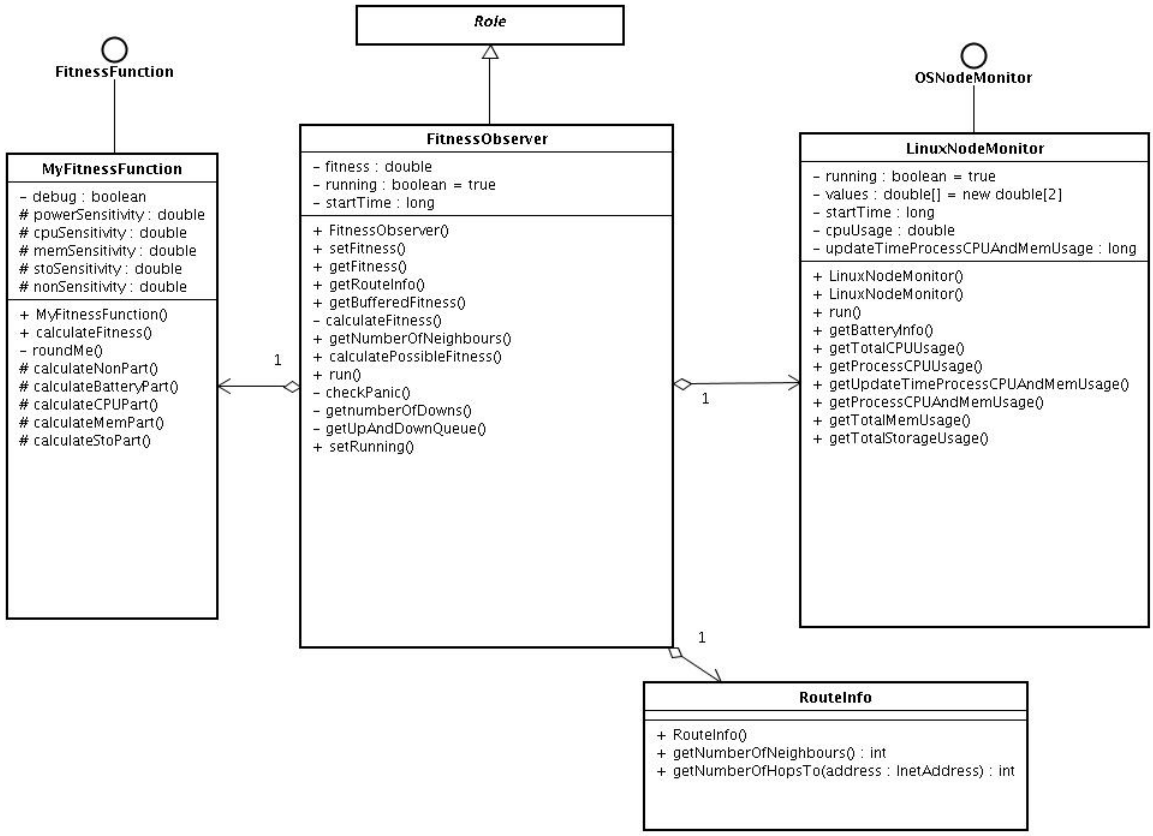


Figure 6.14: Class diagram for the classes concerning the calculation of the fitness

running on a node that decides when it is expected that the node is going down or becomes useless due to too much usage of CPU power, memory usage etc. To determine when it is time to respond by e.g. uploading data, is determined by experiments, which will be explained in section 7.3.

To make sure that all roles are notified of this, every `NamedThread` (and thus also every `Role`) in the network should implement a method `respondToPanic()`. This is the method that is called on every role running on a node in case of a fitness value that is too low. Every role can then decide what to do. In some cases this is nothing, since if a node is being monitored by a monitor that is also keeping replicas at the same time, this is not useful. However, in the case of the ISME application, for example, the report about the situation has to be sent by the user. In case that the node is going down, this information can then be sent automatically to the fusion nodes, without the intervention of the user. This way the data of the node is not lost.
At the same time, there is also a mechanism available for sending a message to one of the direct neigbours. The reason for this is that if there is no fusion node (or crisis center) available, the node might not know where to send its data. The mechanism allows the service to put a message on the blackboard, that is read by one of direct neighbours (a random one) and is put on the blackboard by this node as well.

Finally also a small service is running on all nodes that is waiting for emergency messages from the crisis center. Every node is listening to this type of messages. These messages are then spoken out loud. This is, for example, useful when an order is given that everyone has to leave a building immediately. Although this is not implemented in the system, the messages

that are to be sent might be inputted using a microphone. (Using automated speech recognition the message could then be converted to text, which can then be sent into the network, to be spoken again at the nodes.)

6.5.13 Applying some functionalities to the test programs

Besides implementing all the functionalities as discussed in this section, the functionalities were also tested. The testing was primarily done by implementing the functionalities for the services ISME and Lingua (see section 6.3.3). For the ISME and Lingua services fusion nodes were implemented for preprocessing the information from the observer nodes. The reasoning engine and the text-to-speech engine that were necessary for the applications were considered as tasks for the crisis center. These were services that were, similar to the fusion nodes, monitored by other nodes. Furthermore, some of the messages are replicated. For example, the updates that the reasoning engine of the ISME application sends into the network are replicated. If the reasoning engine would go down it is able to continue operation from that point on. The sending of messages in case of "panic", i.e. low fitness, was used for both applications to send the information that was not already sent by the user, but was already inputted to the program. Furthermore, the communication between the different parts is all done via the blackboard communication as opposed to the original programs that used socket communication for one-to-one connections.

Chapter 7

Experiments & results

7.1 Introduction

This chapter gives an overview of the experiments that were done during the project. This concerns the testing of the application during the development project and the (often time consuming) experimenting with the eventual working system. This was primarily to see whether all the parts could function together and to get some idea of the performance of the total system. This chapter starts with an overview of the general test setup. After discussing some experiments with the fitness functions and values, some test cases are discussed that were used to determine whether and how the system performed.

7.2 General test setup

For testing of the implemented system during the project the increments of the systems were tested as independently of each other as possible. As already indicated, the application was built using an incremental approach. In most cases the increments were tested in three steps. First the implemented part of the system was tested on "the development laptop". After a successful test, the next step was to test the application on the laptop together with one Zaurus. The third step was to use two or three Zaurusses. Even though this was not necessary for some tests, all the tests were executed using wireless connections between the nodes that together formed an ad-hoc network.
For the purpose of testing, one of the logging classes from the `utilities` package was used. This class provides the function to log text to the screen and to a file (or both). Furthermore, the logging output also includes the name of the role that prints the message. This was very useful, since as the project was making progress, the system consisted of more and more different threads. This made it more complex to debug the application, so the logging class (simply called

Figure 7.1: The initial test setup

`Logger`) became a vital tool to debug the application.

7.3 The fitness function

To come to the right fitness function, first some local experiments were done. The target of the experiment was to determine a function that prevented overloading of the nodes and that was useful to determine whether there was a reason to send the panic signal to the local roles.

7.3.1 Determining the fitness parameters

Some of the components of the fitness function were easy to influence for testing. The number of neighbours was easy to influence by turning extra nodes on and off. For influencing the CPU usage a "`while(true)`" loop was used or a number of heavy processes was started. Putting a large file on the current partition was enough to influence the amount of data storage that was used. By starting more roles on the same node (manually for testing) the amount of used memory would change. This way it was thus possible to see how the fitness value responded to changes in the different values and to changes in multiple values. The values of the parameters of the fitness function were determined using mathematical software[1] and by experimenting with the devices. Since the shape of the desired formula was given, some sample points were taken for the values of the components that should lead to a low value of the fitness. E.g. when the Zaurus devices reach a battery power of 5% they are, in practice, nearly out of battery power. For this point a fitness of approximately 0.15 was thought reasonable (for this component). Based on this value and another sample point at 25%, the parameter for the fitness component could be determined approximately. For the other components, a similar approach was taken, i.e. based on the performance of the Zaurus under different amounts of CPU usage and memory usage.

7.3.2 Comparison with linear combination

As indicated in section 4.6, besides the function that is explained in that section, a linear combination of the components was also used in the experiments as a comparison. The linear combination did not lead to good results. The value of the fitness function was tested in combination with the rules to determine whether there was a reason for "panic" as explained in section 6.5.12.
The linear combination did not give good results. As expected, if one of the components of the fitness value was very low (e.g. battery power very low), the overall fitness value remained relatively high. This could be fixed by adjusting the rules for "panic" or the weights of the components. However, since all components could indicate a critical situation, adjusting the weights did not help, since making one component more important, made other components less important. Another solution could, in this example, be to make it give the panic signal for higher fitness values. However, if multiple components of the fitness function were, for example, all 0.5, this would also give the signal that there was panic, despite the fact that not one of the components was critical. This indeed indicated that the linear combination of the components of the fitness function leads to overall poor results.

The second attempt was to use the function discussed in section 4.6 (equation 4.4) as a replacement for the linear combination. This function was able to give more importance to components of the fitness function in the range that the value is becoming critical, as indicated

[1] Waterloo Maple

in section 4.6. This lead to much better results. Initially was experimented to determine a value and to update some of the (heuristical) rules for determining panic. When values were found that exhibited the right behaviour, the parameters and the rules were further tested. The eventual rules came down to responding to panic in case:

- The $fitness \leq 0.1$
- Only one neighbour left
- In case that $0.1 \leq fitness \leq 0.5$ and the value was rapidly decreasing. The decreasing is determined by keeping the history of the last five values. Rapidly decreasing is defined as an overall decrease between the first and fifth value of more than 45% and the last change ≥ 0.1. Otherwise, in case that the decrease is larger that 20%, the system looks at the sign of the last five changes. If there are three or more decreases, the panic signal is given, otherwise no action is undertaken.

7.3.3 Result

Together it turned out that these simple decision rules, in combination with the fitness function from equation 4.4 gave the desired performance. In case that the number of neighbours became small when a node was leaving the rest of the network, the data on the node was saved. However, in case that a group of nodes left the rest of the network, no action was undertaken. The reason for this is of course that the amount of neighbours is only a heuristical approach when it comes to the problem of "group partition" which is actually occurring here. We will come back to this problem in section 8.2.2.
Furthermore, the discussed fitness function in combination with the way of handling the progress of the fitness value led to the desired situation in which the load was divided over different nodes and the overloading of the nodes was prevented.

7.4 Test cases

In this section some of the cases that were tested are discussed. Although the experiments are described as if it concerns individual experiments, all experiments were carried out multiple times and after performing the individual experiments, experiments were performed simultaneously.

7.4.1 Case 1: Testing communication

The testing of the communication was done throughout a large part of the development, since this was one of the main functionalities theu system was to offer. To test the communication, not only experiments were done with the total system, also some separate experiments were done with the routing algorithm. The target of these experiments was to determine the performance of the routing algorithm. This separate testing was necessary to be able to explain the behaviour of the rest of the system.

Practical experiments with AODV

The practical experiences with the AODV algorithm are based on both of the versions that were used during the development process. Many simulations have already been done of routing algorithms, but practical experiments are done in much less approaches (one example is the "Cactus" project of the Delft University of Technology). The performance of the AODV algorithm was negatively influenced by some issues that do not always get much attention in

simulations.

Test setup
Many of the experiments were carried out by using three or four devices, that were participating in the ad-hoc network. To test the routing algorithm, it was necessary to create a situation in which the algorithm was supposed to work multihop. For this reason, many experiments were carried out with one node somewhere in a room the middle of floor x, one or two nodes in the stair well on floor x, and one or two nodes a few floors up.

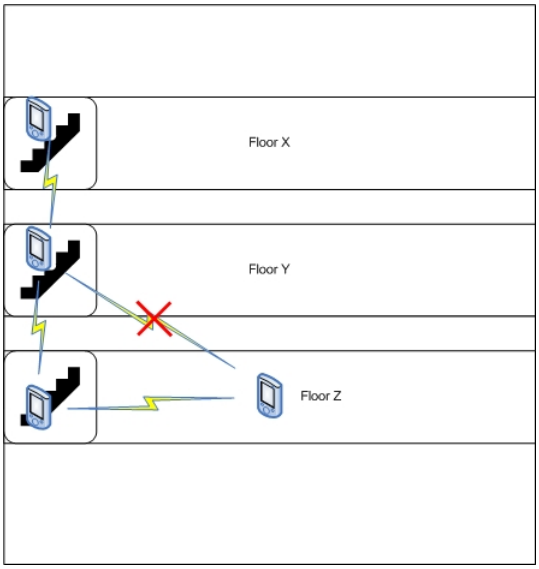


Figure 7.2: Test setup for multihop testing

In this way it was reasonably sure that the endpoints could not directly reach each other. To verify this, the tcpdump tool was used to see which packets were sent to and from the nodes. From this setup, the experiments were often started. The experiments were carried out by walking around with Zaurus devices.

Pinging vs. connections
For the implementation of the AODV algorithm from the Uppsala University, one of the findings, which is a verification of other experiments, is that the used "HELLO" messages usually have a "larger range" than actual point to point connections. Since in this implementation routes are established with help of these "HELLO" messages, it seemed as if there was a multihop connection between two nodes, but sending data was not possible. This was tested using the Java application, but also with making ssh connections to other nodes. The connection broke down in cases that the nodes were too far away. It seems likely that the implementation of the algorithm was somewhat instable due to this problem.

Indoor performance
Another experimental finding, which is frequently ignored in simulations of routing algorithms, is that indoors there are a lot of objects that reflect the wireless radiowaves. The fact that simulations of the performance of indoor MANETs are not robust, can be found in [CBH+04]. Although this is taken into account in some simulations/approaches, our experiments showed that even in the same building, under almost the same circumstances, the performance of the routing algorithm could vary a lot. This can thus be considered as a verification of the fact that reflections seriously influence multihop routing. The performance was determined when

going into rooms and there was suddenly a wall between different nodes or a door was suddenly opened or closed. All these things turned out to matter to the connections between the nodes. The second implementation of the AODV routing algorithm that was used, was more stable, but when walking around within the building, the same problems sometimes occurred. Because of this result, the assumption that the outer part of the network is highly dynamic seems not only to be justified based on the activities executed by the people carrying the nodes, but also based on the properties of the underlying technology for communication.

Delays
Another experimental finding was that the sending of messages is sometimes delayed, since routes have to be (re-)established or refreshed, since AODV is an on-demand algorithm. This was tested by logging messages and the time of sending and of arrival at the source and destination. This observation, which can be considered as a verification of earlier work, initially led to some problems with the sending and receiving of messages. In a MANET the connections are always dynamically changing. Therefore, if a message does not arrive back "soon", this could mean that the node has left the network and time-outs for the connections are necessary. Initially these were not set right and only after the experiments with AODV these were found to be too low in combination with the rest of the system. Since the use of a multihop algorithm in combination with Lime was never tried, this can be considered as our own experimental finding. The time-outs were adjusted in the system to make sure that the communication did not fail in cases that there is a route, but this route was in the process of being (re-)established.

Sending updates via fusion nodes

The sending of the updates via fusion nodes was tested in a single hop setup, i.e. a setup in which all nodes could directly reach each other, and in a multihop setup. In both cases the messages would eventually arrive. This was tested by sending a number of messages from an observer node. The number of messages was counted. At the crisis center the incoming messages from the observer nodes were logged. This way, the amount of messages that arrived could be measured. This came down to between 80 and 100% of messages that arrived, as long as the observer nodes were not suddenly turned off before they had sent their data. Furthermore, when the fusion nodes went down, this meant that messages were not delivered anymore, except when the fusion nodes were restarted by the monitors. This delayed the messages, since they had to wait for the new fusion node that was being started. Other cases of nodes leaving the network are discussed below. The amount of time that it took before a delivered messages arrived, differed depending on the number of hops between a node and the crisis center, as expected.

In case that the route to the fusion node was broken and the message could not be directly sent and when the connection was being re-established, it was delivered as required. Furthermore, due to the fact that the messages from the outer nodes to the (mobile) crisis center were sent via the fusion nodes, a lack of a direct connection (single or multihop) to the crisis center was not always a problem. The fusion nodes sometimes acted as "ferries" for the messages (see e.g. [ZAZ04]). This was not only applicable for the test situations, but this is also likely to happen when looking at the different functions people actually fulfil in crisis situations.

The overall performance of the communication was reasonably good, although the system relies on the underlying routing algorithm. For this reason it was, in case that there was no established route (either due to a broken link or no route used yet), difficult to send messages, since the route had to be (re-)established. The application had to wait for the routing algorithm. This is one if the disadvantages that is inherent to using MANETs, specifically of using an

algorithm that is communicating "on-demand".

7.4.2 Case 2: Testing role assignment

The second thing that was to be tested, as soon as the communication was working, was the role assignment procedure. This was first tested as a way to start the system and later for testing the monitoring of nodes. The target was to see whether this procedure would lead to a structure in which the data from the observer nodes was fused and delivered at the crisis center.

Startup

The use of role assignment was first used to startup services. First a crisis center was setup, by letting two nodes, a Zaurus and a laptop act as crisis center. After setting up this crisis center, two others Zaurusses were added to the network in a multihop configuration. One of these two Zaurusses started the client part of the ISME application. This did not find a reasoning engine and thus started one. This was assigned in the right part of the network, i.e. in the crisis center. However, if the crisis center was not setup first, the reasoning engine was sometimes assigned to the "wrong node" (e.g. a node that was to be used for observing). In both cases this was the behavior that was expected.
In the latter case, setting up a crisis center resulted in the activation of the roles responsible for the re-assignment of roles.

Re-assignment after change in network

If the Zaurus running the reasoning engine left the hop range that was associated with this task, the role was re-assigned by the role assignment procedure. Although this was done in the right way, the speed of the re-assignment of the role was not very high. This was primarily due to the fact that the Zaurus devices were slow in comparison to "normal" computers.

Monitoring

The monitoring of nodes was also tested. This was tested by starting three nodes in a network. On one of these nodes the reasoning engine was started, which in turn assigned a monitor and started a monitor-monitor. First the timing parameters were set to low. The monitor then re-assigned the role since it did not get a reply soon enough. If a route was being re-established, this took some time and for this reason the monitors were adjusted to check the blackboard multiple times before deciding that a node is gone and timing parameters were set to higher values (due to the slowness of the Zaurus devices).

In the next experiments, with the same setup, the monitoring worked as expected. One of the two Zaurus devices was turned off. The service was restarted by its monitor and for the reasoning engine of the ISME application a replica of the latest world model was placed back on the blackboard. However, in case that two nodes were turned off, the approach was not always successful. As can be expected, in case that the node with a service and node with the monitor of this service both leave the network, the remaining part of the network did not restart the service. An advantage of the use of the fitness function for role assignment was that the assignment of the role was not always done to the direct neighbour of a node, but to a node that, at the time of starting the monitor, was the strongest node. This reduces the probability that if a small group leaves, both a service and its monitor are in the group.
After these tests, the monitoring of nodes was enabled as a standard functionality during testing. This was to test whether the monitoring was also working when more roles were running on the nodes. This was also tested successfully.

7.4.3 Case 3: Duplicate services

In case there was a duplicate service started and a `DuplicateSolver` was running, one of the services was shutdown. During the experiments it turned out that the solving of duplicates was not working for the case that there were more than one `DuplicateSolver`, i.e. `DuplicaSolver`s were not capable of "killing" each other. From this experiment it turned out that this was to be handled separately, since it was not easy to implement this by using the `DuplicateSolver`s themselves.

The solving of duplicate services was somewhat tricky as was seen at the tests / experiments, due to the mechanism of monitoring. Although this may sound simple, at development time this was one of the things that was initially overlooked: if a service is to be stopped, the monitor that belongs to this service is to be stopped as well. If not, the duplicate is only gone for some time, after which it is re-assigned by the monitor.

7.4.4 Case 4: Panic / group partition

Even though the approach described in this thesis was not primarily focussing on the group partition problem, some small experiments were done to test how the system would respond. Furthermore, the heuristical rules used for determining panic were also included in some experiments.

As already suggested in section 7.3 the fitness function and its parameters were determined in combination with the heuristical rules. After some tweaking of the parameters and of the rules, some acceptable results were reached. The case that one node leaves the rest of the group, i.e. one person with a PDA walked away of a group of three nodes, was seen as a case of panic. In case that the number of nodes started to decrease to a value higher than 1 (i.e. a group, instead of one node was leaving the rest), the algorithm did not always indicate that the situation was to be marked as "panic". This was primarily depending on the value of the sensitivity to the number of nodes. If this was set to a high enough value, than the algorithm would respond the way it should. However, the value of this parameter determines when the fitness should decrease and a different value for this parameter has different influence on the problem of group partition. The question is what amount of neighbours is considered to be enough to form a "group" that is (usually temporarily) not connected to the rest of the network, without having to send all information to the remainder of the group. This was considered to be a question for which the algorithm was to be tested during more realistic experiments (i.e. a real crisis situation or a real exercise).

Chapter 8

Conclusions & further research

8.1 Conclusions

8.1.1 General conclusions

A general conclusion drawn from this project is that it is possible to build a wireless communication infrastructure for crisis situations based on blackboard communication. Despite some performance problems, the approach described in this thesis can be considered as a step towards a communication and coordination infrastructure that could be used in future systems for crisis situations. Already some tools are available that can assist rescue workers (see for example [Sch05]). Hopefully this work encourages the development of new and more sophisticated, task oriented services for the different rescue workers, to enable better crisis response in the future.

8.1.2 Literature survey

The literature survey that was to be executed mainly concerned the currently available techniques in the field of MANETs. The choice for this topic was due to the fact that MANETs are considered to be of great use in crisis situations which is the problem domain of the project this thesis is part of. This is also shown by the recent assignment that has been given by rescue services to a company to try and develop a wireless communication system. From this literature survey it became clear that much work was already done on the communication in MANETs on a low level, i.e. the lower levels of the OSI model. Most of the work done in this field is done using simulations of the problem, in order to focus on specific algorithms (mainly routing algorithms, but also on algorithms for data storage etc.). Less work seems to be done on the more practical aspects of MANETs, which can only be determined with practical experiments. Furthermore, on the application level, less research seems to be done, despite the fact that many people claim the usefulness of MANETs in many environments, of which crisis situations is one of the two most often used examples (the other example being military application). This thesis shows that this claim is indeed justified.

8.1.3 Model

The model made of the problem domain at hand, crisis situations, was primarily based on research by other people that was not made part of the literature survey, but was considered to be part of a sort of extra preliminary research within this thesis project. Based on the results of this research, the layered structure was brought into the communication network, which seems to fit the way that rescue workers actually execute their tasks.
This approach can be considered as more than just a solution applicable to one situation. The model of the situation and the solution that is discussed throughout this thesis, will hopefully

encourage more research in this area, since bringing a structure into the network opens new possibilities. Not only research in the areas as discussed in section 8.2.1 will lead to better handling of crisis situations, but the wireless infrastructure offers new possibilities for new applications that can be specifically built for rescue workers. To develop new services to assist rescue workers is a task that should be executed in tight cooperation with the users of the system.

8.1.4 The used design methodology

The used design methodology turned out to be an applicable, but difficult approach. The incremental approach that was used, implied that there was no fully designed/specified system in advance, only global ideas. The reuse of software and the "gluing" together of different parts of the system required much time. This was due to bugs and incompatibilities between the different parts of the overall system. Besides this, it also required applications, such as ISME, to be converted from e.g. client-server application into an application that can run distributed in a MANET, running a different version of Java etc. However, considering the complexity of the used tools and software, implementing the prototype from scratch, would have been impossible within one (thesis) project.
The incremental approach turned out to be very useful when testing and debugging of the application, which speeded up the development. Therefore, to fulfil the research assignment, it turned out to be a successful approach.

8.1.5 The implemented solution

Overall conclusions implementation

One of the parts of the research assignment was to implement, as a proof of concept, a prototype that provides some of the necessary functionalities of the solution that is designed. This was done as described in chapter 5 and 6 and with this prototype tests and experiments were executed, to see whether the system and the chosen approach work. This was also one of the parts of the research assignment.
The overall results of the tests and experiments were satisfying. The experiments could be carried out only on a small scale, since there was no real crisis environment available to test the prototype. Furthermore, the speed of the application was not high enough in certain cases, due to the limited processing capabilities of the nodes. Therefore it is not yet completely suitable for use in real time environments, i.e. an actual crisis situation. Other limitations of MANETs, in the form of limited antenna range and limited battery power, were clearly encountered during the implementation and during the experiments. Especially for the former of these two, measures were taken to reduce this problem (in the form of multihop routing and the structure in the network).

Communication

Part of the research assignment was to design a communication layer, based on a blackboard structure. Due to taking the right, open source tools, it turned out to be possible to do this as a part of the thesis project. Despite some bugs in this code and some other inconveniences, the use of some existing tools speeded up the whole development process, so that it became possible to do this in one thesis project. The overall conclusion concerning the communication is that it is reasonably reliable in case the routing algorithm is stable. Furthermore the prerequisites for a communication infrastructure are facilitated, thanks to the role assignment procedure that is used in the system, in combination with the blackboard structure for communication. The efficiency of the communication in terms of message overhead and bandwidth usage depends

on the underlying blackboard structure. In this thesis this was taken into account in the design and implementation. In case a different underlying blackboard structure would be used, this design or implementation might need to be adjusted.

Role assignment

The infrastructure that is provided by the prototype, was built for applications, such as escape routing and observing, which are to be executed in the MANET. These were to run on nodes in the network. Furthermore, other tasks for keeping the nodes in the network "up" were also to be run on the different nodes in the network.

The role assignment procedure was specifically designed to make the MANET flexible and adaptive to the changing topology. Furthermore, it brings a structure in the network that seems to be most appropriate in the situation at hand, a crisis situation. The procedure (and the maintaining of the role positions) offers, in combination with the blackboard communication structure, an optimal communication infrastructure to emerge. This facilitation of this emergent behaviour is necessary, since the optimal communication in a dynamic situation requires a solution that can handle the dynamics and that does not fail in this environment.

Besides communication, the introduction of fusion agents as a middle layer in the network for preprocessing, provides a communication flow from the lower level (sensor) observations, up to high level decisions in the local, mobile crisis center, without overloading the crisis center with too much information. The high level information in the crisis center is suitable for transmission to the main crisis center, which is usually setup in e.g. the mayors office.

Another conclusion that can be drawn from the experiments, is that the solution works, but that parts of the solution are, in this implementation, too slow for extremely dynamic environments. However,the concept works and with some better hardware, which is perhaps already available, the solution can be made suitable for real crisis situations.

Monitoring

The mechanism of monitoring is part of the design of the structure that takes into account the fact that nodes can go down. The monitors, together with the monitor-monitors can be considered as measures to prevent from data loss and to prevent that certain tasks/services stop executing while they are not finished/still necessary. This mechanism works reasonably well in case the multihop routing, and therefore the entire communication functions as it should. The procedure for the assignment of roles to the fittest node assigns the role of monitoring to the best node that is not necessarily close by. This can mean that the role is assigned to a node that will soon leave the network. On the other hand, since the role is assigned to the *fittest* node, this should be the "best" node around. Whether this mechanism works best in this way, or whether it would be better to assign it to the fittest node in the subset of nodes that is within e.g. 1 hop from a node, is something that is to be tested in more real-life experiments in (simulated) crisis situations.

Fitness

The fitness value that was introduced had multiple functions. One of the main functions was to use it as a criterion to assign a role in the network. Taking the topology into account, the usual case will be that there are multiple nodes that are capable of fulfilling a task. The idea of

introducing a general fitness value, is to support decentralized decision making. For this purpose, the used function is shown to be useful and provided a division of tasks over the network, that prohibits the overloading of certain nodes.

The functionality to respond to situations in which the node is about to go down is more tricky. The fitness value, combined with the heuristical rules provided in the implemented system, can be considered as a heuristical approach to decide whether or not to take action. However, as will also be suggested in section 8.2.1, this part of the system could be extended to come to better decisions. However, there remains a trade off between what is theoretical possible and what is practically possible in terms of speed of execution, energy efficiency etc. One of the reasons for the practical approach in this thesis work is that simulating things never takes *all* aspects into account, while a real life systems is forced to do this.

Multihop routing

One of the subtopics that was part of the graduate task, was to set up an ad-hoc network of mobile devices. Due to the problems with the multicast, multihop routing algorithms, as discussed in chapter 5, quite some experimenting was done with the two routing algorithms (both implementations of the AODV algorithm). Based on these experiments it can be concluded that this routing algorithm perhaps functions very good in simulations, in practice the performance is much worse. Especially issues as reflections, walking behind walls are usually not completely taken into account in simulations. This can have great effect on the theoretical results as shown in [CBH+04].

8.2 Further research & possible improvements

8.2.1 Further research

Using in practice

As a part of the Intelligent Systems project, this system was implemented to offer an infrastructure to applications. To evaluate the performance in practice, first the services to assist rescue workers are to be further developed. These services can also be evaluated on a smaller scale. Firemen can for example test these when a house is on fire. When better applications are available, the performance of the system can be determined in real crisis simulations as in Amsterdam (April 6th 2005). Besides the application in crisis situations, the solution (or parts of it) can perhaps also be applied in other situations. First of all, crisis situations are considered one of the standard examples of the use of MANETs. Another standard example is the use of these systems in the military field. In this case one can imagine that the model of the situation described in Chapter 3 could be applicable as well. The stable part of the network could in this case be the commanders of the troops, the "middle ring" the soldiers that are behind the front line, and the "outer ring" the front line. In this case the same ideas might be of use concerning the fitness of nodes, the dynamic assignment of roles and the means of communications via the blackboard. However, it requires some more domain specific research to exactly determine how the ideas are to be fit in a solution for these types of networks.

It is hard to state that in general, the fact what node is "stronger" than other nodes, can make certain types of networks more stable and more fault tolerant. However, as illustrated by the following example, it can be of great importance in certain situations.
One of the properties that was indicated in section 4.6.1 that it distributes the load more evenly, Furthermore, if a node "thinks" that it cannot fulfill a role, it will not offer to do it. An interest-

ing example of a network that was overloaded and would not have been if a fitness criteria was used, can be found in [Wat04]. In this book a nice description is given of the power breakdown in August 1996 on the West Coast of the United States. The cause of the breakdown was that other parts of the network "blindly" took over the task of other "nodes" (in this case nodes were power lines) in the network. In this case, the power lines that took over the power load of a line that broke, were already close to their limits. However, the lines took over tasks that they could not handle, which led to more plants going down, and this way a chain reaction was started, leading to a large power breakdown.

The use of monitoring in other situations is also not imaginary. As will be indicated in section 8.2.2, the mechanism described in this report can be further extended, which can make a network more fault tolerant. This is, of course, desirable for all distributed systems, especially in systems that consist of (more or less) equal nodes and certain tasks or applications are to be executed in a distributed manner.

Different fitness functions

In this thesis a choice was made for a fitness function that intuitively seems to give a reasonable fitness value and seemed useful in comparison to situations that roles are assigned randomly to nodes. Although chapter 7 shows that the fitness value can be useful, it is not possible to tell whether the fitness function is the best possible one. A possible way to get an overview of the performance of different algorithms, could be by making a simulation of a mobile ad-hoc network. Within this simulation different ad-hoc scenarios are to be implemented, such as (besides of course different sorts of disasters) battlefield situations, conference rooms and so on. However, as already mentioned before, simulations are usually simplifications of real life situations. In [CBH+04] is shown that "simplified" simulations models for the indoor evaluation of MANET routing algorithms are not robust. For this reason, the simulation should take into account a sophisticated model of the indoor properties. Furthermore, before the simulations could take place, mobility models are to be developed and made available of the mobility of rescue workers in crisis situations. Taking into account all this, will make it the research very broad, but as indicated, simplified simulations are not (always) suitable for drawing general conclusions.

More measures

Further research can also be done on the variables of which the used fitness function exists. More measures can be taken into account, such as the strength of the wireless connections, amount of network traffic and whether or not the network will split in two groups that cannot reach each other (group partition). However, if more measures are taken into account, more calculations have to be executed. This can lead to the situation that too much time is spent on the calculation of the fitness, while this value is only used to keep the services in the network up and at the "right" place. Of course it should not be the case that the underlying framework uses too many resources or becomes too slow as in the case of certain replication allocation algorithms (see for example [Har01] and [Har03]).

Routing

Another idea for further research could be to adjust the routing algorithms for the network and for the dynamics of parts of the network. As already suggested in [Kla04], a meta-algorithm to determine what the best routing algorithm is for the situation at hand would be applicable here. A hybrid approach in which different routing algorithms are combined would perhaps be

a way to make the routing more efficient and more appropriate with regard to the structure of the network. However, in this case, instead of a meta-algorithm, it could perhaps be handled in a simpler approach, by taking into account the structure that is brought in the topology of the network. It seems that in the stable part of the network, a proactive routing algorithm is more appropriate and that in the dynamic part of the network a reactive routing algorithm is to be used. This would, however, make the hybrid routing only applicable for the kinds of environments in which a similar topology is brought into the network. An approach in which the dynamics are automatically processed in the routing algorithm would make the approach more suitable for MANETs in general.

8.2.2 Possible improvements

General improvement

Some general aspects of the implemented system that can be improved are the speed and the portability. The speed concerns the fact that the application runs somewhat slow when multiple services are active. This is not only because of the application that is running, but also because of the fact that the (graphical) display of applications is very slow on the Zaurus devices. This can of course be solved by using devices that are faster than the Zaurus devices currently are. In the future, one can expect that PDAs will become faster in general, which could make the application faster. Perhaps another programming language than Java might be faster, although the speed of Java is getting closer to the speed of "older" languages (like C++). Furthermore, a newer Java Virtual Machine might be faster, but these are not available for the Zaurus devices.

Monitoring

A possible improvement for the monitoring of other nodes is also possible. In section 4.5.2 is indicated that is it not desirable that nodes monitor each other randomly, since this can lead to an "infinite" number of nodes that monitor each other. This was solved in the system by letting two nodes monitor each other. This idea can be further expanded to four, five and eventually to x nodes monitoring each other. This idea does have a limitation, in the sense that the monitoring generates extra network traffic. This is not really a problem in the case that two nodes monitor each other, but becomes more a problem as x increases. Besides extra message overhead, if the number of nodes that takes part in the monitoring of one service increases, the probability that one of the monitoring nodes goes down is also increasing. This will lead to a situation in which much time is spent on the monitoring and re-assigning of role in the network.

Group partition

Another improvement, that is also shortly mentioned in section 8.2.1, is to use a special approach to handle the problem of group partition. In case of the implemented system, group partition can have as a result that a group of nodes that is monitoring each other is isolated from the rest of the network. From that moment on services that are available and monitored in this group of nodes, are not available anymore to the nodes in the other part of the network. For the detection of group partition some algorithms are available. In [Har01] and [Har03] information about group partition is used for replication. Despite the fact that this algorithm takes to much time to execute (getting all information about the structure of the network and let the algorithm make decisions about where to allocate replicas), this might improve the availability of services and data in the network.

Performance measurements

As indicated in section 6.5.11, some of the performance factors were measured with workarounds that were platform specific. This might be improved when the newest Java version, Java 1.5, is used to implement this part of the application. It was, as already indicated, not possible to use Java 1.4 or 1.5, since there is no Java Runtime Environment version 1.5 available for the Zaurus. If this would become available for the Zaurus (or any PDA that is fast enough), this could make the performance measurements platform independent.

Besides the way that the fitness information is determined, some improvements can also be made to the fitness function by including more factors. As already indicated in section 8.2.1, some possibilities are the amount of network traffic, the group partition problem, but also other performance measures, such as system information about nodes, such as CPU speed etc. This could be useful when different nodes in the network are different devices (e.g. not only Zaurus devices, but also laptops or even fixed computers if used in other situations than the one assumed in this thesis work). As already indicated in section 4.6.2, the connectivity can also be taken into account. If all routes in the network are known, a value for the connectivity can be defined based on the stability and strength of the connections. This approach costs a reasonable amount of calculations and the question is whether this is practically possible. However, in theory it can improve the "heuristical" approach of taking the number of neighbours as a component of the fitness.

Bibliography

[ADA04] S. Quarteroni A. Datta and K. Aberer. Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in wireless mobile ad-hoc networks. International Conference on Semantics of a Networked World, June 2004.

[ant] http://ant.apache.org/.

[aod] http://www.cse.ohio-state.edu/~jain/cis788-99/ftp/adhoc_routing/.

[BB03] Roberto Beraldi and Roberto Baldoni. Unicast routing techniques for mobile ad hoc networks. In *The Handbook of ad hoc wireless networks*, chapter 7, pages 127–148. CRC Press, Inc., 2003.

[BCPR03] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. Jade - a white paper. *Exp*, 3(3), September 2003.

[BD00] B. Bruegge and A.H. Dutoit. *Object-oriented Software Engineering*. Alan Apt, 2000.

[Bel04] Lorenzo Bellini. Lime ii: Reengineering a mobile middleware. Master's thesis, "Politecnico di Milano" Technical University, 2004.

[Ben05] T. Benjamins. Untitled. Literature survey, 2005.

[BMB+05] B. Brown, I. MacColl, M. Bell, M. Chalmers, and C. Greenhalgh. Collaboration in the square: an infrastructure for collaborative ubicomp. 2005.

[BR05] T. Benjamins and L.J.M. Rothkrantz. Crisis management procedures. Technical report, Delft University of Technology, 2005.

[bru] http://wiki.decis.nl/crisisdomain/tiki-index.php?page=Examples+of+information+failures+in+crises.

[c20] http://www.c2000.nl.

[Cai03a] G. Caire. Jade tutorial for beginners, December 2003.

[Cai03b] G. Caire. Leap user guide, December 2003.

[CBH+04] A.L. Cavilla, G. Baron, T.E. Hart, L. Litty, and E. de Lara. Simplified simulation models for indoor manet evaluation are not robust. In *Proceedings of the First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, Santa Clara, California, October 2004.

[Cha05] J. Chau. Title not known yet. Unpublished, 2005.

[CN01] K. Chen and K. Nahrstedt. An integrated data lookup and replication scheme in mobile ad hoc networks. In *Proc. of SPIE International Symposium on the Convergence of Information Technologies and Communications (ITCom 2001)*, Denver, Colorado, August 2001.

[com] http://combined.decis.nl/.

[cou04] Cougaar architecture document, 2004.

[CSN02] K. Chen, S.H. Shah, and K. Nahrstedt. Cross-layer design for data accessibility in mobile ad hoc networks. *Wireless Personal Communications*, 21(1):49–76, 2002.

[dec] http://wiki.decis.nl/crisisdomain/tiki-index.php?page=HomePage.

[Dib03] H. Dibowski. Hierarchical routing system using ant based control. Master's thesis, Delft University of Technology and Dresden University of Technology, July 2003.

[dsr] http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt.

[ecl] http://www.eclipse.org/.

[Fit04] S. Fitrianie. An icon-based communication interface on a pda, 2004.

[GJG02] Le Gruenwald, Muhammad Javed, and Meng Gu. Energy-efficient data broadcasting in mobile ad-hoc networks. In *Proceedings of the 2002 International Symposium on Database Engineering & Applications*, pages 64–73. IEEE Computer Society, 2002.

[GSB02] M. Güneş, U. Sorges, and I. Bouazizi. Ara - the ant-colony based routing algorithm for manets. In *InternationalWorkshop on Ad Hoc Networking (IWAHN 2002)*, August 18–21 2002.

[Har01] T. Hara. Effective replica allocation in ad hoc networks for improving data accessibility. In *Proceedings of IEEE INFOCOM 2001*, pages 1568–1576, 2001.

[Har03] T. Hara. Replica allocation methods in ad hoc networks with data update. *ACM-Kluwer Journal on Mobile Networks and Applications (MONET)*, 8(4):343–354, August 2003.

[HG00] L. Hunsberger and B.J. Grosz. A combinatorial auction for collaborative planning. In *Proc Fourth International Conference on Multi-Agent Systems*, pages 151 – 158, 2000.

[HHL02] Z.J. Haas, J.Y. Halpern, and L. Li. Gossip-based ad hoc routing. *IEEE INFOCOM 2002 - The Conference on Computer Communications*, 21(1):1707 – 1716, June 2002.

[INPS03] L. Iocchi, D. Nardi, M. Piaggio, and A. Sgorbissa. Distributed coordination in heterogeneous multi-robot systems. *Autonomous Robots 15*, 2:155 – 168, 2003.

[JM96] D.B. Johnson and D.A. Maltz. Dynamic source routing in ad hoc wireless networks, 1996.

[jnia] http://java.sun.com/docs/books/tutorial/native1.1/.

[jnib] http://java.sun.com/j2se/1.4.2/docs/guide/jni/.

[jom] http://homepages.cs.ncl.ac.uk/einar.vollset/home.formal/jomp.html.

[jxt04] Jxta technology: Creating connected communities, January 2004.

[KC01] T. Kunz and E. Cheng. Multicasting in ad-hoc networks: Comparing maodv and odmrp. In *Proceedings of the Workshop on Ad hoc Communications*, pages 16–22, September 2001.

[ker] http://w3.antd.nist.gov/wctg/aodv_kernel/.

[Kla04] P. Klapwijk. Distribution of data, decisions and roles in wireless ad-hoc networks. Technical report, Delft University of Technology, 2004.

[KMP02] G. Karumanchi, S. Muralidharan, and R. Prakash. Efficient peer-to-peer data dissemination in mobile ad-hoc networks. In *Proceedings of the 2002 International Conference on Parallel Processing Workshops*, page 152. IEEE Computer Society, 2002.

[lig] http://lights.sourceforge.net/.

[lim] http://lime.sourceforge.net/.

[MBK+04] S. Mehrotra, C. Butts, D.V. Kalashnikov, N. Venkatasubramanian, K. Altintas, R. Hariharan, H. Lee, Y. Ma, A. Myers, J. Wickramasuriya, R. Eguchi, and C. Huyck. Camas: a citizen awareness system for crisis mitigation. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 955–956, New York, NY, USA, 2004. ACM Press.

[McM92] J.W. McManus. *Design and analysis techniques for concurrent blackboard systems.* PhD thesis, The College of William and Mary in Virginia, 1992.

[Mil05] Filip Miletic. Progress digest workpackage world modeling. March 2005.

[MMDM04] N. Milanovic, M. Malek, A. Davidson, and V. Milutinovic. Routing and security in mobile ad hoc networks. *IEEE Computer*, February 2004.

[NFM04] R. De Nicola, G. Ferrari, and G. Meredith, editors. *A Lightweight Coordination Middleware for Mobile Computing*, Pisa, Italy, February 2004. Springer-Verlag.

[odm] http://www.hpl.hp.com/personal/Sung-Ju_Lee/abstracts/papers/draft-ietf-manet-odmrp-02.txt.

[OPBS03] J. Odell, H. Van Dyke Parunak, S. Brueckner, and J. Sauter. Changing roles: Dynamic role assignment. *Journal of Object technology*, 2(5), September-October 2003.

[PB94] C.E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244. ACM Press, 1994.

[pla] http://www.placelab.org.

[pow] http://news.bbc.co.uk/2/hi/asia-pacific/4125837.stm.

[PR99] C.E. Perkins and E.M. Royer. Ad-hoc on demand distance vector routing. In *Proc. IEEE WMCSA '99*, pages 90 – 100, februari 1999.

[Qua99] E.L. Quarantelli. Disaster related social behavior: Summary of 50 years of research findings, 1999.

[RDFT05] L.J.M. Rothkrantz, D. Datcu, S. Fitrianie, and B. Tatomir. Personal mobile support for crisis management using ad-hoc networks. In *Proceedings of the 11th international conference on Human Computer Interaction*, July 2005.

[res] www.itr-rescue.org/.

[rmi] http://java.sun.com/products/jdk/rmi/.

[RN95] S.J. Russell and P. Norvig. *Artificial Intelligence, a modern approach*. Prentice Hall, 2nd edition, 1995.

[Rot04] L.J.M. Rothkrantz. Crisis management using mobile ad-hoc networks, 2004. Whitepaper made for students to give an overview of the activities inside this project.

[Sch95] D.G. Schwartz. *Cooperating heterogeneous systems*. Kluwer, 1995.

[Sch05] Paul Schooneman. Isme: Icon based system for managing emergencies. Master's thesis, Delft University of Technology, 2005. Unpublished.

[sen] http://brownback.senate.gov/pressapp/record.cfm?id=225805.

[SLL03] Boon-Chong Seet, Bu-Sung Lee, and Chiew-Tong Lau. Route discovery optimization techniques in ad hoc networks. In *The Handbook of ad hoc wireless networks*, chapter 17, pages 301–311. CRC Press, Inc., 2003.

[Str04] S.R. Strijdhaftig. Research assignment jade, 2004. Literature survey.

[sun] http://java.sun.com.

[Sun02] J.Z. Sun. Mobile ad hoc networking: An essential technology for pervasive computing. In *Proceedings International Conferences on Info-tech and Info-net*, pages 316 – 321, 2002.

[Tan03] A.S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.

[tsu] http://nl.wikipedia.org/wiki/Vloedgolf#2004.

[voi] http://www.voicesofsept11.org/911ic/022805.htm.

[vV05] M. van Velden. Manetloc: a location based approach to distributed world-knowledge in mobile ad-hoc networks. Master's thesis, Delft University of Technology, 2005.

[Wat04] D. J. Watts. *Six degrees*. W.W. Norton & Company Ltd, 2004.

[wir] http://www.thefeature.com/article?articleid=101651.

[Ww05] Whitepaper-werkgroep. Whitepaper: Ict delft research centre. Technical report, Delft University of Technology, 2005.

[ZAZ04] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proc. of ACM Mobihoc*, Tokyo, Japan, May 2004.

[zeea] http://archief.www.ad.nl/artikel?text=dodental%20tsunami\&SORT=date\&ED=ola\&PRD=20y\&SEC=%2A\&SO=%2A\&FDOC=1.

[zeeb] http://archief.www.ad.nl/artikel?text=zeebeving%20\&SORT=date\&ED=ola\&PRD=20y\&SEC=%2A\&SO=%2A\&FDOC=7.

Glossary

ABC routing **A**nt **B**ased **C**ontrol routing. A routing algorithm for MANETs, 10

ACPI **A**dvanced **C**onfiguration & **P**ower **I**nterface. A part of the Linux operating system kernel that controls the power management of a device, 74

AODV routing **A**d-hoc **O**n-Demand **D**istance **V**ector routing algorithm. A routing algorithm specially developed for MANETs, 10, 49

APM **A**dvanced **P**ower **M**anagement. The part of the Linux operating system kernel that controls the power management of a device, 74

Blackboard A Blackboard System can be viewed as a collection of intelligent agents who are gathered around a blackboard looking at pieces of information written on it thinking about the current state of the solution and writing their conclusions on the blackboard as they generate them.[McM92], 14

DSDV **D**estination-**S**equenced **D**istance-**V**ector routing. A routing algorithm for MANETs, 10

DSR **D**ynamic **S**ource **R**outing. A routing algorithm for MANETs, 10

Fusion Combining information from multiple sources, 22

GUI Graphical User Interface, 38

ISME **I**con based **S**ystem for **M**anaging **E**mergencies. Application for reporting about crises using icons. Used as a test application, 56

JNI **J**ava **N**ative **I**nterface. A standard programming interface for writing Java native methods (and embedding the JavaTM virtual machine into native applications). The primary goal is binary compatibility of native method libraries across all Java virtual machine implementations on a given platform ([jnib]), 73

Lime **L**inda **i**n a **m**obile **e**nvironment. A Java implementation of the Linda model / blackboard model specifically made for wireless networks, 15

Lingua The Lingua application is an application for communication with strings of icons, 57

MANET **M**obile **A**d-hoc **Net**work. In this thesis this is defined as the cooperative engagement of a collection of Mobile Hosts without the required intervention of any centralized Access Point ([PB94]), 3

MAODV **M**ulticast **A**d-hoc **O**n-Demand **D**istance **V**ector routing algorithm. A routing algorithm specially developed for MANETs which also supports multicasting, 49

Multihop routing Routing in a way that if nodes A and C cannot reach each other but both nodes can reach a third node (e.g. B) the messages are forwarded by node B. This thus means that all nodes are behaving as “routers and as “senders of messages, 21

ODMPR **O**n **D**emand **M**ulticast **R**outing **P**rotocol routing algorithm. A routing algorithm specially developed for multicasting in MANETs, 52

PDA **P**ersonal **D**ata **A**ssistant. General term often used for mobile computing devices, 28

RMI **R**emote **M**ethod **I**nvocation. It enables the programmer to create distributed Java technology-based to Java technology-based applications in which the methods of remote Java objects can be invoked from other Java virtual machines that are possibly located on different hosts ([rmi]), 64

Appendices

Appendix A: C program for JNI

The code of the C program that was made to get the process id of a certain class, looks as follows:

```
#include "j_unistd.h"
#include <unistd.h>
#include <string.h>

JNIEXPORT jlong JNICALL Java_roles_fitnessobserver_JNICom_getpid
(JNIEnv *envp, jclass clazz)
{
    return getpid();
}
```

Figure 8.1: The C code for getting a Linux process id

As can be seen in this C code, the package name should also be included in the method name. This is a necessity for the method in the library to be found. If the package name of the class changes, the C-code is to be recompiled. Furthermore, the header is to be re-generated using the javah tool. The parameter JNIEnv *envp is also a necessity for JNI.

Appendix B: Communication flow

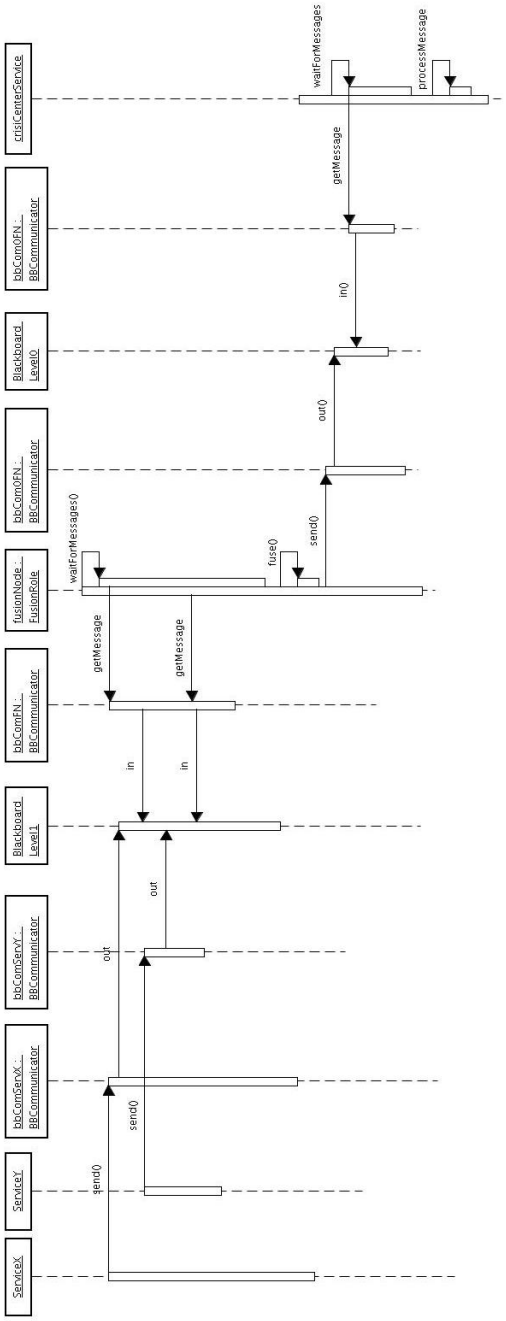


Figure 8.2: Communication flow from the outer to the inner part of the network

Appendix C: Paper