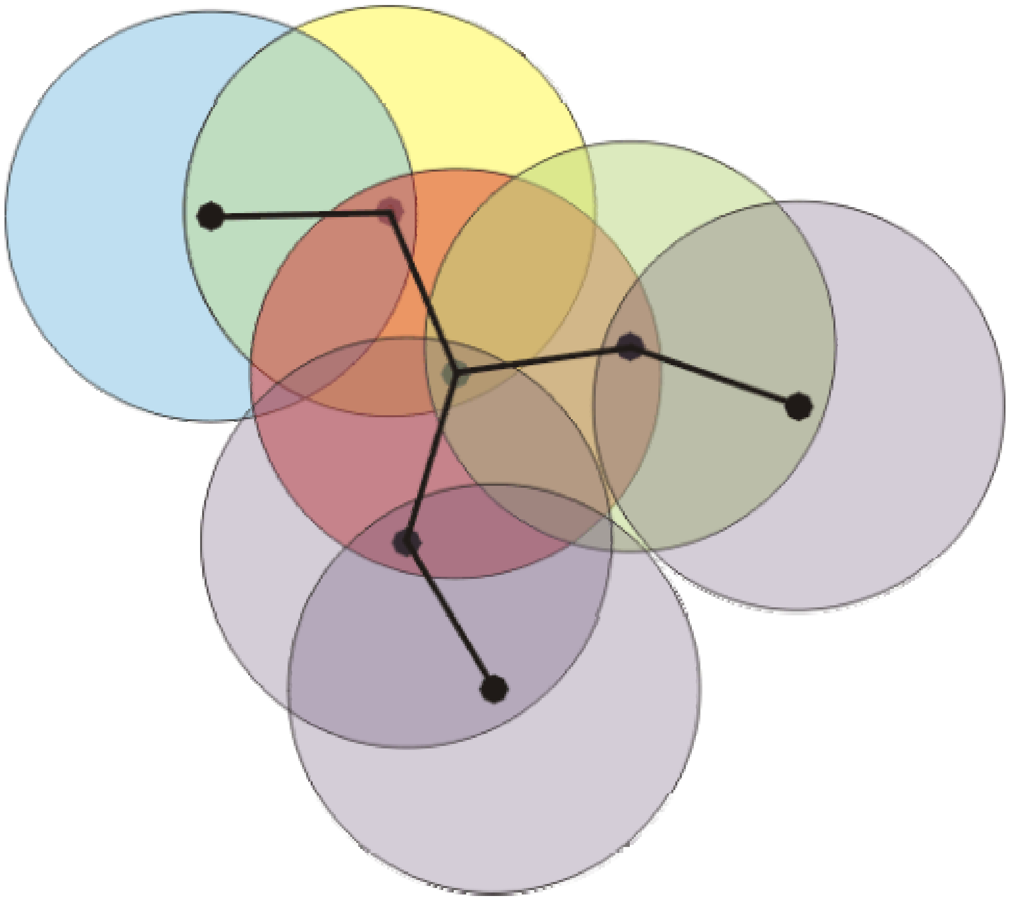
ManetLoc
A location based approach to distributed world-knowledge in mobile ad-hoc networks



Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Department Mediamatica / Man-Machine-Interaction

TUDelft
Delft University of Technology

Master Thesis of:
Marcel van Velden
Student number: 9862097
Date: April 2005

In memory of my grandfather

Graduation committee:

Drs. Dr. L.J.M. Rothkrantz
Dr. A.H.J. Oomes
Dr. K. van der Meer
Ir. F. Ververs

Abstract

In every aspect of our lives people are becoming more and more dependent on the availability of (information) systems. To have access to such systems often an infrastructure of wired and/or wireless networks is required. In case of a crisis such networks are likely to become overloaded, as more than the regular amount of service request will occur. Systems might even completely go down because of overload, incidents (such as an explosion) or sabotage. Also the environment an individual in is likely to be (partially) unknown.

In this thesis a concept of a system for making multi-agent systems in mobile ad-hoc networks aware of their environment without the need of any infrastructure is provided. The main focus is on automatically building a map of the world by using observations from individuals in such an infrastructureless network. We have designed and implemented this proof of concept, ManetLoc, in the form of a simulation. No specific agent platform was used in developing the simulation, but the future use of the JADE platform was always kept in mind. The system was built upon a preexisting ad-hoc network simulation environment, Ad-hoc Simulator [Boel04].

ManetLoc simulates a building like environment where individuals are exploring an unknown world. It is assumed that each individual in the field is equipped with a Personal Digital Assistant (PDA) and can communicate with other PDAs in the vicinity. Together the PDAs dynamically form ad-hoc networks. Users can enter their own observations to the PDA. Agents on these PDA's work together, to supply the users an as complete view on the world as possible (a topological map). An agent will also provide guidance to the user if requested.

Keywords: mobile ad-hoc network, agent, communication, topological map, infrastructureless, location, emergency, awareness, crisis

Preface

This thesis is the result of my graduation project at the Man Machine Interaction (MMI) group of the faculty Electrical Engineering, Mathematics and Computer Science at the Delft University of Technology.

The research is done as a part of the project 'Crisis management using mobile ad-hoc networks'. The crisis management project is related to the Combined Project [Comb] at DECIS LAB, a collaboration of Delft University of Technology, University of Amsterdam, THALES Nederland, and TNO. The collaboration focuses on the research of decision support systems, seeking to order information in complex and chaotic situations.

Acknowledgements

First of all I would like to express my gratitude to my supervisor Leon Rothkrantz for enabling me to do this research assignment. Next I would like to thank my fellow project members for giving me useful input and assistance. They are, in no particular order: Paul Schooneman, Paul Klapwijk, Stefan Strijdhaftig, Jan Chau, Joost Boehlé and Bogdan Tatomir. Needles to say I would also like to thank my family, friends, neighbors and especially my girlfriend Welmoed, for their support, advice and in helping me test the system.

Table of Contents

TABLE OF FIGURES

Chapter 1: Introduction

In every aspect of our lives we are becoming more and more dependent on the availability of (information) systems. These systems by themselves depend on other systems to be available, i.e. an infrastructure. In times of crisis not all of these systems might always be readily available. This could be caused by power cuts for instance or simply because there is no physical infrastructure at all.

If a telecommunications infrastructure is not available at a certain area, it should still be possible to set up an infrastructureless network or mobile ad-hoc network (MANET) under most conditions. Setting up such a network enables us to share information concerning the state of the world and coordinate actions. These ad-hoc networking technologies are making it possible to exchange information anywhere, anytime without prior network infrastructure. Using handheld devices that operate in a wireless environment, communication is still possible when major infrastructural communication links have been damaged, destroyed or overloaded. So in case of a major disaster within a city, emergency services can communicate without the need for preset-up access points or other such infrastructural requirements.

Our multi-modal interfaces (MMI) department is doing extensive research on multi-agent systems using wireless ad-hoc networks. This research is part of the project 'Crisis management using mobile ad-hoc networks', which focuses on intelligent crisis management using mobile ad-hoc networks. The crisis management project is closely associated with the Combined project (**C**haotic **O**pen world **M**ulti-agent **B**ased **I**ntelligently **NE**tworked **D**ecision support systems) of DECIS LAB our group participates in. Roughly the aim of the project is to develop an environment wherein rescue services can communicate using handheld devices dynamically forming MANETs. Users of such networks should be able to exchange observations through agent technology and intuitive GUI's located on a handheld set. Agents aid the user in finding, storing and retrieving information from the network. Our specific interest in this thesis is the distribution of world knowledge in these ad-hoc communication networking environments.

1.1 Project overview

In order to be able to find the answers we are looking for and to limit the scope of this thesis, we will look into three main topics: *multi-agent systems, mobile ad-hoc networks* and *location awareness*. We will attempt to combine the three in such a way that we are able to develop a location aware multi-agent system which is able to run in a mobile ad-hoc network.

Storing and making available knowledge in agent enabled MANETs is a challenging problem that still needs to be tackled on many fronts. This particularly is the case when there is no secondary infrastructure available at all, no special hardware on the portable devices and the network is highly dynamic. Apart from the major difficulties, the possible uses of this kind of technology seem to be countless and contain – but are not limited to - crisis situations and military use. Related to this there is a demand for mobile devices to become more aware of their environment. Again this is a challenging problem on its own as well but if there is no infrastructure available at the relevant location this is an even more complex issue. All this seems to make research in the area more than worthwhile.

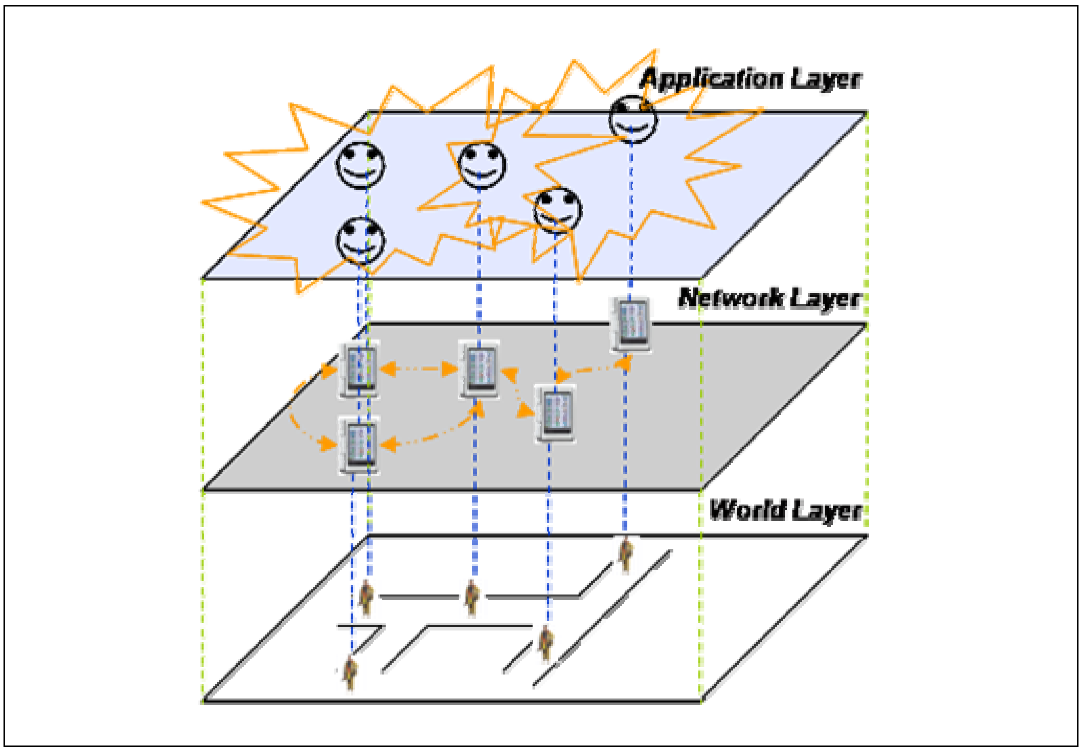


Figure 1 Project layers

A useful way of looking at our project is by splitting it up in the following three layers [Figure 1]: *World layer, Network layer and Application layer*. The world layer symbolizes the actual world a user finds itself in. As the user is carrying network enabled devices, there is a network which can be used to communicate, i.e. the network layer. In the application or agent layer knowledge about the world is shared and services are offered by agents in the network (i.e. applications). All three layers play an important role in our system concept, but we tried to focus on the application layer, where the other two serve as an essential basis for the later.

The goal of this research was to provide a concept - and a proof of this concept - of a potential solution for making multi-agent systems operating in mobile ad-hoc networks aware of their environment. From the multi agent-nature of such a system almost naturally follows the approach of distributed construction of world models by the agent software. The fact that the system is intended for use in highly dynamic ad-hoc network environments makes that no central and/or permanent services can be depended upon, thus the concept should be decentralized. Another aspect of the concept is that no knowledge of the world will be available beforehand and during the lifetime of a system component the knowledge available will incrementally grow but at most times will be incomplete and fuzzy. Therefore a non-deterministic probabilistic approach seems logical.

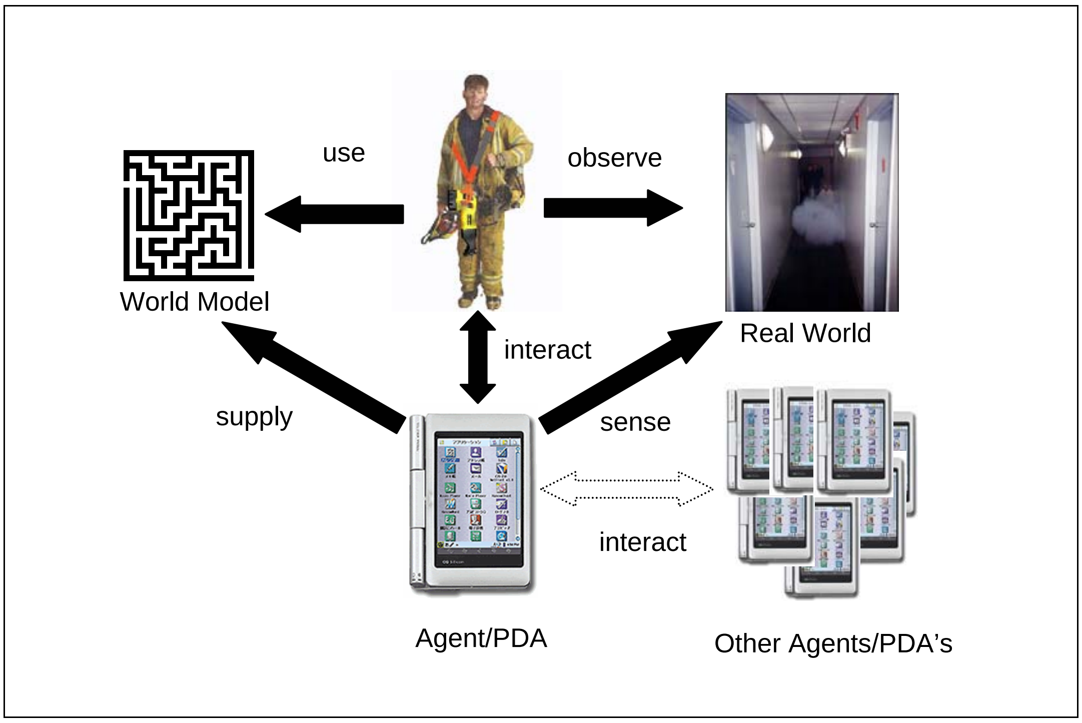


Figure 2 Project Overview

In [Figure 2] a visual overview is provided of what we have tried to achieve in this project: People observe the world and interact with their PDAs. A PDA running agent software can also sense the world and interact with other agents in a wireless ad-hoc network. From the information an agent gathers by communicating with the user of the PDA, sensors and other agents, it assembles and continually updates a world model. This world model can then be provided to the user and other (agent) software running on the PDA.

To summarize this section the three main areas of interest mentioned (*multi-agent systems, mobile ad-hoc networks* and *location awareness*) can be further specified into key facets of the system architecture concept. Thus the system will involve the combination of:

- *Emergent data structures*
- *Multi-Agent collaboration*
- *Distributed data*
- *Decentralized systems*
- *Probabilistic, fuzzy information*
- *Mobile ad-hoc networking applications*
- *Location awareness*
- *Human-agent collaboration*

1.2 Problem formulation

Our ideal system would be one that is always operational regardless of the availability of any infrastructure, that (at any location) is aware of any (relevant) information – which means a correct and up to date world model - at any time, and performs intelligent distributed planning and execution based on this information. Furthermore as the real world is a highly dynamic place it should be able to easily adapt to changes in the environment it operates in. Finally, as with almost any system, in some cases it would have to interact with human users, so the system should be able to do this in a user-friendly way.

Taking into account that all this cannot be handled in a single thesis work (or in 10 for that matter), this work focuses on location awareness in infrastructureless environments. Our problem formulation is defined as follows:

Design and implement a multi-agent-system that can operate in environments without a pre-setup infrastructure (only a mobile adhoc network) and without any pre-knowledge of the world, which is able to process and fuse location information from different users and sensors remote in space and time and distributes location information and location based services (such as guidance) to its users.

In chapter 3 we will elaborate on the architecture we have setup in answer to this problem description. First relevant and related work studied during the course of this research will be discussed.

Chapter 2: Related work

In this chapter some related work will be described which was studied before starting and also during the design of the system. We will look at the topics of multi agent systems, ad-hoc wireless LAN, agents on small devices, agents in ad-hoc networks and location awareness. As many different topics are discussed, it goes too far too go in to much detail. Therefore the information provided is mostly global, for more details on a specific topic we advice to follow the references.

2.1 Multi-agent systems

The first topic that is important for our work is that of agent systems, and more specific, multi-agent-systems (MAS) [Aaai]. The study of agent based systems evolved from the field of Distributed Artificial Intelligence in the early to mid 1980's. In contrast to classical applications in artificial intelligence, often viewed as dedicated, centralized and standalone, the specific ideas underlying agents in so called agent based systems globally are the following:

- *Autonomous to a degree*
- *Exhibit goal directed behavior*
- *Interact with and negotiate with other (possibly human) agents to achieve their goals*
- *React 'intelligently' to a dynamic and unpredictable environment*

An important 'application' of agent based systems is building them in a way so they can perform tasks that would normally require human intervention, with much of their intelligent behavior being emergent rather than preprogrammed. The study of multi-agent systems is about systems in which many intelligent agents interact with each other and possibly the user. Solving the problems associated with, and taking advantage of the opportunities offered by, distributed and unstructured environments are a major application area for intelligent and Multi-Agent systems. In the next paragraphs we will talk about FIPA [Fipa], an important organization in the standardization of agent software, two agent platforms – JADE [Jade] and Cougar [Coug] - and the concept of swarm intelligence [Swar1] [Swar2].

2.1.1 FIPA

In the world of agents the Foundation for Intelligent Physical Agents (FIPA) plays a major role. FIPA is a non-profit organization aimed at producing standards for the interoperation of heterogeneous software agents and started its activities in 1995 with the aspiration to standardize different aspects related to agent technology and multi-agent-systems. FIPA's model for agent systems is build around the concept of agent communication. Which means agents can pass semantically meaningful messages to one another in order to accomplish the tasks required by the application. The standards do not constrain the low-level implementation of agents to any great extent, nor, except for defining agent platform services, does it constrain the infrastructure a great deal. There are many agent platforms available which (partly) implement the FIPA standards, some major platforms are: Agent Development Kit, April Agent Platform, Comtec Agent Platform, FIPA-OS, Grasshopper, JACK Intelligent Agents, JADE, JAS, LEAP and ZEUS.

2.1.2 JADE

JADE (Java Agent DEvelopment Framework) is a open source Java implementation of FIPA-standards [Jfip] which is also used in some subprojects of the 'Crisis management using mobile ad-hoc networks' project. JADE is still actively maintained and supported. As it is implemented in Java a JADE agent platform can be distributed across machines which not even need to share the same OS. The jade framework supplies also a set of graphical tools that supports the debugging and deployment phases.

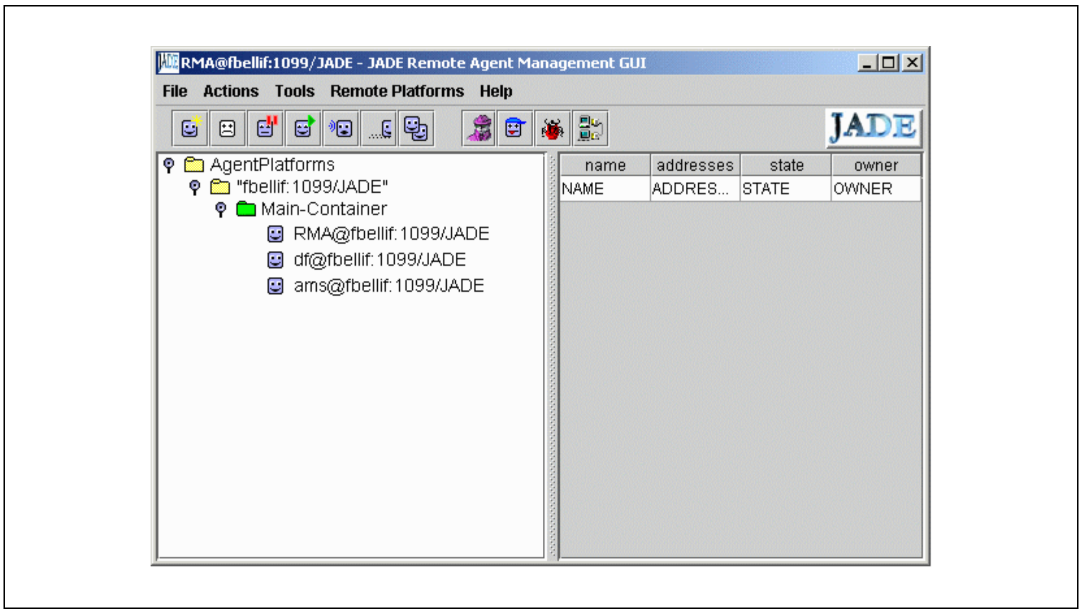


Figure 3 JADE Remote Agent Management GUI

Some important JADE specifics are [Hels04]:

- *Inter-agent communication messages are formatted in ACL or XML*
- *Pluggable transport protocols, which include RMI, IIOP, and HTTP*
- *Java utility classes simplify the construction and handling of FIPA-compliant ACL messages*
- *Security is supported through SSL and socket-based proxy agents*
- *Helpful GUIs are provided to debug agent communications and registries*
- *Each agent has its individual thread*
- *Suited to developers of relatively simple agent applications*

Although JADE attempt to be FIPA-compliant it has also implemented some features for mobile devices of its own. We will discuss the possibilities, problems and solutions for using jade in ad-hoc environments on small devices later on.

2.1.3 Cougaar

Another important agent platform within the project is Cougaar (Cognitive Agent Architecture). Cougaar also is a Java-based architecture for the construction of large-scale distributed agent-based applications. It is the product of a multi-year DARPA research project into large scale agent systems and includes not only the core architecture but also a variety of demonstration, visualization and management components to simplify the development of complex, distributed applications

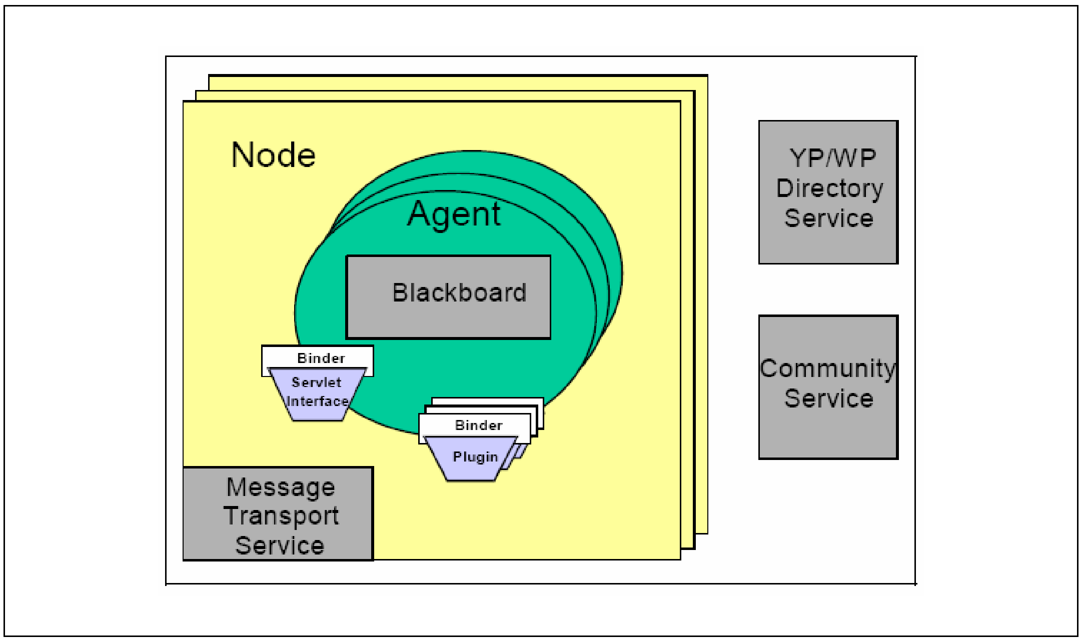


Figure 4 Cougaar components schematic

Some important Cougaar specifics are [Hels04]:

- *A focus on scalability and modularity*
- *Pluggable transport protocols, including RMI, IIOP, SMTP, and UDP*
- *Not standards compliant*
- *Messages are encoded using Java object serialization*
- *Security is pluggable and flexible, SSL, X.509, role-based authentication*
- *Uses a thread pool with resource monitoring*
- *Suited to developers that want to create complex, large-scale, robust, or highly secure agent-based applications*

2.1.4 Swarm intelligence

Agents in a mobile ad-hoc network, can only directly be aware of a small part of the environment, a concept for dealing with this issue in such environments is Swarm Intelligence *[Fern05]*:

"Swarm Intelligence (SI) is the property of a system whereby the collective behaviors of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge. SI provides a basis with which it is possible to explore collective (or distributed) problem solving without centralized control or the provision of a global model."

I.e. as agents in such systems have very limited capabilities of itself, thus has to cooperate with its neighboring agents without always being aware of the global picture. In our case there are for example limited communication possibilities. A very successful swarm intelligence technique is so called ant colony optimization [Dori91], which is a probabilistic technique for solving computational problems. In this concept problem are reduced to finding good paths through graphs and is inspired by the behavior of ants in finding paths from the colony to food. The ant colony optimization concept is extensively researched in our department and also applied in mobile ad-hoc networks [Rado03] [Tato04]. In the concept explained in chapter 3, a single agent will likely not always be able to observe the whole situation on its own, but by acting on its local environment and sharing relevant information about it to its neighbors, it should be possible to create a global 'swarm intelligence' of single mobile ad-hoc nodes in some way.

2.2 Ad-hoc wireless networks

Mobile ad-hoc computing is possible because of new technologies for short-range wireless data communication such as Wireless LAN and Bluetooth. Devices with the same type of technology make the communication and collaboration between them possible, as soon as the devices come into communication range. Mobile Ad-hoc Networks (MANETs) are wireless networks consisting entirely of mobile nodes that communicate on the move without base stations. Nodes in these networks will both generate user and application traffic and carry out network control and routing protocols. MANETs are very flexible because of the dynamic topology where nodes are free to move arbitrarily and it allows a Peer-To-Peer (P2P) communication in an asynchronous manner.

These networks have problems like rapidly changing connectivity, network partitions, higher error rates, collision interference, and bandwidth and power constraints together. These problems are particularly in the design of higher-level protocols such as routing and in implementing applications with Quality of Service requirements. As our system concept is supposed to operate in a simulated ad-hoc wireless network [Kant03], some specifics of the network protocols used in our simulation environment will be given below.

2.2.1 IEEE 802.11B

IEEE 802.11B [IEEE99a, IEEE99b] is a popular standard and implementation of wireless LAN technology. The Adhoc Simulator (AHS) developed by J. Boehlé [Boel04] simulates an IEEE 802.11B network. As our system is built on this simulation, the implementation of this protocol is also used in our system. Information is taken from [Boe04] and slightly adapted, for more details see this thesis. Below we will give a global overview some of the ad-hoc 802.11B specifics, which were implemented in AHS.

The IEEE 802.11B Medium Access Control (MAC) Layer consists of a series of agreements concerning the sending and receiving of data. Before nodes, operating in an ad-hoc wireless environment can start to transmit data to each other they must first find each other. In order for a node to discover its neighbor nodes, which are within its communication range, the node sends out a *probe*. In order to realize communication between two wireless IEEE 802.11B compatible nodes a set of delay timers and packets is needed. Nodes operating in the wireless ad-hoc mode cannot start sending data immediately. The *transmission of a data packet* to a node starts when the sending has found the receiver to be idle. The sending node broadcasts a Request To Send (RTS) packet to the intended destination then waits a certain amount of time for a Clear to Send (CTS) packet. If the CTS packet is not received within the waiting time the RTS packet is transmitted again, if the destination node is still idle. The CTS packet is send back using the broadcast method to notify other nodes of the pending transmission.

Nodes that receive either the broadcasted RTS or CTS message set an NAV timer specifying the length time in which the sending and receiving nodes are not available for transmissions. The time period for the NAV timer is taken form the duration field of the RTS or CTS packet. If no transmission is detected after the exchange of RTS or CTS a node may reset its NAV timer after a period. After receiving the CTS packet and waiting for a certain amount of time the node can transmit the data packet to the destination. Once transmitted the node waits a specified time for the Acknowledgement (ACK) packet to arrive confirming a good transmission.

To minimize the bit error rate the 802.11B specification has limited the payload to 2312 bytes. Data that exceeds this size needs to be fragmented by the sender before transmission can take place. Once the sending node finds the destination node to be idle it sends out an RTS packet. The node that overhears this RTS packet sets it NAV timer for sender and destination using the duration field of the RTS packet, the destination node sends back a CTS. The sending node can now start to transmit its first fragment, nodes that overhear the fragment update their NAV timer for sender and destination, and the sender confirms the successful reception of the data packet using the ACK packet, which also may be overheard by other nodes. This process is repeated until the last fragment specifies zero for the more fragments field of the data packet. If the ACK period expires on the sending node signaling a missed ACK packet the data transmission stops. The nodes that overheard the last fragment will have their NAV timer still running for the duration specified by the last fragment. The sending node may try to rebuild the communication with the destination node, if this does not happen the destination node will be available to all other nodes as soon as their NAV timers expire.

2.2.2 The Ant-colony-based routing algorithm for MANETs

As the Adhoc Simulator uses the Ant-colony-based routing algorithm (ARA) for routing and our system is built on this simulation an overview of the phases of this routing algorithm is provided here. Text is taken from [Boe04] and slightly adapted, for more details see this thesis. ARA is a multi-agent system in which ants form the individual agents. ARA is based on the behavior that ants exhibit when searching for food. Ant based routing algorithm uses the notion of ant based optimization, see [2.1.4]. As stated in [Boel04], the ARA routing algorithm globally consists of three distinct phases: *route discovery*, *route maintenance* and *route failure handling*

Route Discovery

In order to communicate with other nodes the ARA protocol sends out a uniquely identifiable forward ant (FANT) that tries to establish a route to node somewhere in the network. At each node (hop) that it encounters on the way to the destination node it updates the route tables. If there is no address entry for the route taken by the FANT, the FANT senders address is noted is the 'destination address' field, the hop that relayed the packet is noted in the 'Next hop' field and the pheromone value is set to default. Otherwise only the pheromone value is updated. As soon as the FANT finds/arrives at the destination node a uniquely identifiable backward ant is launched (BANT). The BA NT traces back the route to originating FANT node by making use of the routing table entries previously made by the FANT. While tracing back the route to the BANT updates the routing tables at each hop in the same fashion the FANT did. Once the FANT originator node is reached communication between the nodes can be established.

Route maintenance

Individual nodes perform route maintenance by analyzing received packets. Each packet received by the node either for processing or relaying is used to update the routing table and the pheromone value. A timer determines when to decrease the pheromone value. Once the pheromone value for a certain route has reached zero, the route can safely be discarded.

Route failure

The MAC layer implemented notifies the ARA protocol in case of route failure by returning a ROUTE_ERROR message. Once notified by the MAC layer the ARA protocol searches the routing table for an alternative route. If no alternative route is found the packet is relayed to all the neighbors of the node who will check their routing tables for a route to the destination. If the packet cannot be relayed to the destination node the packet is send back to the destination node, each receiving hop tries to find an alternative route as described above before sending the packet back.

2.3 Agents on small devices

Small devices are portable computing devices with networking capabilities, such as a mobile phone or a PDA. Apart from the great advantage of being small and lightweight and therefore portable, small devices have some issues that need to be mentioned:

- *Reduced processing power*
- *Memory limitations*
- *Limited permanent storage with no file system generally available in phones.*
- *Limited battery life*
- *Intermittent connectivity due to areas not covered, shielded environments and the need of turning the device off to save battery*
- *High network latency and low bandwidth*
- *Small screen size*
- *Restricted input mechanism such as numeric keyboard*

Figure 5 Examples of small devices

Limited battery life and connectivity are the current most relevant issues in our project. As we are not depending on infrastructure but on ad-hoc network technologies the latter is this most constraining issue. We will discuss this in the next chapter. Though the processing power and related specifics of small devices are nothing compared to that of current desktop computers it is currently possible to execute relatively complex software applications such as route planners on small devices and it is to be expected that these devices will become more and more powerful in the near future. The device we will base us on is the Sharp Zaurus SL-C760. On the SL-C760 a version of Linux is installed running Qtopia, X and Java. It has a maximum resolution of 640x480 pixels, a 400 MHz Intel XScale PXA255 processor, and 62 MB of RAM. This is sufficient for running JADE.

2.3.1 JADE on small devices: LEAP

Though it is probably possible to get the standard version of JADE running on some small devices by making much effort, the fact is that the current release of JADE is to 'heavy' for use on mobile devices. Luckily there is a special version of JADE available, called LEAP [Leap][Lawr02]. The most important goal of the LEAP project was to make JADE lightweight and executable on small devices.

LEAP, when combined with JADE, replaces some parts of the JADE kernel forming a modified runtime environment that can be deployed on a wide range of devices varying from servers to Java enabled cell phones. In order to achieve the possibility of running jade on different small devices, running different versions of Java, LEAP can be shaped in three different ways corresponding to three types of Java environments that can be found on devices, j2se, pjava and midp:

- *j2se*: to execute JADE-LEAP on PC and servers in the fixed network running JDK1.2 or later
- *pjava*: to execute JADE-LEAP on handheld devices supporting PersonalJava such as most of today PDA's
- *midp*: to execute JADE-LEAP on handheld devices supporting MIDP1.0 only such as the great majority of Java enabled cell phones

Though different internally, the three versions of JADE-LEAP provide the same set of API's to developers thus offering a homogeneous layer over a diversity of devices and types of networks. Please note: that only a few features that are available in JADE-LEAP for j2se and pjava are not supported in JADE-LEAP for midp as they are related to Java classes that are not supported in MIDP. Our intended device, the Sharp Zaurus SL-C760, runs pjava by default, so we should be able to use JADE-LEAP for an actual implementation of the concept.

2.4 Agents in mobile ad-hoc networks

In this chapter we will not concentrate on the problems with ad-hoc networking technologies mentioned in chapter 2.3, but on the possibilities of using existing agent technology and the modifications necessary to use it in combination with this kind network technology to build real-life applications.

Though developers of multi-agent systems often do not try to solve the problems in mobile ad-hoc networks, they will still have to live with them to be able to build real applications. Therefore an agent platform used cannot depend too much on network availability and has too anticipated ahead for it to change or even go down at any moment in time. After an event like this the platform should be able to recover and continue working.

In mobile ad-hoc environments each of the devices may host agents offering specific services to the surrounding which can directly be used or may be combined to more complex services. Based on the traditional Directory Facilitator (DF) definition, the search of remote services is accomplished by using the concept of DF federations: DFs, besides registering services offered by local agents, may also register other local or remote DFs. This allows them to extend the search for services to remote platforms. This mechanism is not efficient, even less for mobile ad-hoc environments, e.g. because the searcher first has to find the remote DF and afterwards has to look if the services he s searching for are registered there.

Allowing registering and discovering agent services using existing ad-hoc / P2P discovery technologies, which are specifically developed for these environments, can enable a more efficient management of service descriptions and directories, as well as an efficient search and result filtering. Furthermore, once working in mobile ad-hoc environments, ad-hoc and P2P technologies can also be used as mechanisms for agent (platform) societies in the fixed network.

The development of dynamic service discovery technologies is still an ongoing research topic. It is not yet presumable which technology will finally be widely adopted and be the leading one. All of them have specific advantages and disadvantages and do not completely fit all requirements. E.g., some are not dealing well with the spontaneity of the peer communication and fast changing service provisioning, while others are not dealing well with the scalability for a huge amount of services and users. Some existing discovery mechanisms and technologies which are worth noticing are: JINI, PDP, UPNP, BT-SDP, JXTA, Gnutella, Chord, DEAPspace, SLP and Salutation.

2.4.1 FIPA and agents in mobile ad-hoc networks

At the time of writing FIPA specifications currently do not specify any explicit support for agents in ad-hoc environments [Sanc03], but FIPA recognized the potential of ad-hoc computing and decided to adopt current dynamic service discovering technologies. In February 2002 FIPA has created an ad-hoc technical committee with the mission to develop solutions enabling agents to interoperate in mobile ad-hoc environments [Fadh1]. In [Fadh2][Finf][Berg03] possible answers are given to the question of which extensions and changes to the FIPA architecture are needed to better support agents in ad-hoc environments In these propositions the central idea for change is that each agent platform periodically broadcasts an agent platform announcement message of itself. If at least two mobile devices are in range for ad-hoc communication, both agent platforms receive the agent platform announcement messages and become aware of each other.

2.4.2 Jade in mobile ad-hoc networks

Using the current version of JADE, application developers can build mobile agents, which are able to migrate or copy themselves across multiple network hosts. In this version of JADE, only intra-platform mobility is supported, that is a JADE mobile agent can navigate across different agent containers but it is confined to a single JADE platform. Though using the functionality to replicate Main-Containers it is possible to create a fault-tolerant platform [Jadm]. This implies creating a platform consisting of multiple main-containers replicating each other and could possibly be used in an ad-hoc network, but not without problems and massive overhead.

As explained above there are still a lot of issues for getting JADE agents to behave as we like in ad-hoc networks. Most likely the best option is to go for JADE-LEAP as there seem to be some efforts to prepare it for combination of small devices and mobile ad-hoc networks. At the moment though, LEAP also isn't fully capable of being used in these kinds of networks. In [Lawr02] the following modifications to the JADE-LEAP platform and current FIPA standards are proposed:

- *Adding a Discovery Agent (DA) to handle peer-to-peer platform and agent discovery*
- *Removing the distributed container concepts currently present in JADE-LEAP to allow an ad-hoc node to be a fully contained platform*
- *Removing the Discovery Facilitator (DF) and Agent Management System (AMS) as mandatory components of a platform. But allowing their activation should a device be capable and an environment require them*
- *Leasing directory entries within the DF and AMS*
- *Providing a notification mechanism to allow the propagation of directory changes*

2.5 Location awareness

Let alone running agents in mobile ad-hoc networks is already difficult enough, the goal of this research is to find out how to make agents in these networks context aware and able coordinate actions in an operating environment like a building on fire. Physical context information can be very diverse, and include local system information such as battery level or signal-noise ratio, or environmental information such as light intensity, temperature, or ambient noise. Among all, location is possibly the most relevant context element for our application area, in that it often qualifies the values of the others. For example, a temperature reading becomes more meaningful when accompanied by the identity of the room where it was sensed. The point, however, is that the actions of an application component in a mobile environment may depend on one or more of these context information values and modeling physical context becomes a necessity.

As stated above location awareness plays a crucial role for context aware agents. From this point we will focus on the possibilities for the determination of the position of nodes in mobile ad-hoc networks. If a system like GPS is available, each node can be easily aware of its own location, but if GPS is not available (for some or all nodes) relative positions have to be determined using other methods. There are systems and system concepts available, which use network readings such as time of arrival to calculate positions. First we will give some information about GPS and why we can not always depend on its availability. Then we describe and give reference to a non-exhaustive list of possible GPS-less algorithms.

2.5.1 Satellite-based localization

Global Positioning Systems (GPS) is a technology that utilizes satellite data to provide highly accurate location, navigation, and timing information. Most systems, which have anything to do with localization, make use of GPS or a similar system, such as GLONASS

The GPS system consists of 24 satellites that circle the earth in controlled orbits, while continually broadcasting their position to locations worldwide. A GPS device receives data from these satellites and can calculate its distance from each visible satellite. When at least three satellites are visible, it can use triangulation to calculate latitude and longitude location coordinates. With fourth satellite signal, the GPS device can also calculate its altitude [Murp97]. Although highly unlikely at present time – in the future anything can happen – it would be possible that GPS isn't available at a certain time, for example because the US government decided to disallow access. A probably more important and convincing argument for not depending on GPS is that inside buildings, GPS cannot provide location information with satisfactory accuracy for most tasks [Kita03].

2.5.2 Infrastructure-free localization

There are a variety of ways in which position can be derived from the measurement of wireless network signals. The most important measurements are the *angle of arrival* (AOA), the *time of arrival* (TOA) and *time difference of arrival* (TDOA), received *signal strength indicator* (RSSI).

T(D)OA and RSSI methods use range measurements from the mobile device to several base stations and/or other mobile devices to obtain its position. This means that the accuracy of the estimated position depends on the accuracy of the range measurements. An example of such a system is the Self-Positioning Algorithm (SPA) [Srda01]. SPA is an infrastructure-free positioning algorithm, which uses range measurements between the nodes to build a network coordinate system. The Time of Arrival (TOA) method is used to obtain the range between two mobile devices. The algorithm provides relative positions of the nodes in respect to the network topology.

Another system is the Ad-hoc Localization System (AHLoS) [Ahlos]. AHLoS uses TDOA. Like TOA technology, TDOA also relies on extensive hardware that is expensive and energy consuming. Another approach is that of Angle of Arrival. Such a system is proposed in [Nicu01]. Their system allows nodes to estimate and map relative angles between neighbors Similar to TOA and TDOA, AOA estimates require additional hardware which is in most cases too expensive to be used. RSSI methods use theoretical or empirical models to translate signal strength into distance estimates. An example RSSI technology is a system called RADAR [Bahl00]. The advantage of RSSI systems is that they can more easily be used on hardware-constrained systems such as sensor nodes, but also PDA's.

In all the systems mentioned above, problems such as background interference and irregular signal propagation make range estimates inaccurate. There is some work done which tries to limit such errors take advantage of averaging, smoothing, and alternate hybrid techniques to reduce error to within some acceptable limit. [Gane02] [Giro01] The conclusion though is that it is still questionable if distance can be determined based on signal strength, propagation patterns alone. Therefore many other types of are proposed, such as centroid and DV-HOP based systems.

In the centroid scheme [Bulu00] for example, anchors beacon their position to neighbors that keep an account of all received beacons. Using this proximity information, a simple centroid model is applied to estimate the listening nodes' location. Another possible system is DV-HOP [Nicu03] which assumes a heterogeneous network consisting of sensing nodes and anchors. In DV-HOP, anchors flood their location throughout the network maintaining a hop-count at each node along the way. Nodes can calculate their position based on the received anchor locations, the hop-count from the corresponding anchor, and the average distance per hop.

We conclude this chapter by mentioning a lot of work is done on this subject of infrastructure free localization in MANETs, though at the time of writing there is no ready to use system which suits our need so we needs to go for a different approach, which will be based on user and local sensor input only.

Chapter 3: Architecture

This chapter discusses the implemented system architecture concept set out in the introduction more elaborative. Also, the requirements and constraints set for implementing our system will be discussed. The results of this chapter will form the basis for a system that will function as for a proof of concept (chapters 4, 5 and 6). It functions as an answer to the formulated problem description. The main element in the solution is the idea of maintaining a partially shared concept of relative locations as a mean of context awareness and distributing context information.

As an anticipated setting for the system, consider a building on fire, covered with smoke where people are trying to leave the building and firemen are performing tasks such as trying to rescue these people and exterminating the fire. We make the reasonable assumption that both the firemen and the people inside have no map of the building and therefore are hindered in their goals by their lack of world knowledge [Figure 6].

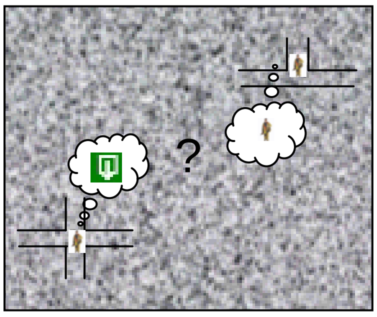


Figure 6 Lack of world knowledge hinders users

For people (including firemen) to be able to effectively reason about their environment without any pre-knowledge, as a first step a model of the world needs to be constructed in some manner. Although they have limited visibility because of the smoke, people should still be able to build a world model for themselves in their minds or on paper by only keeping a record of every hall and crossing they encounter. This includes providing information about the direction of edges and making distance estimations.

The basis for our world model is intentionally kept simple for our proof of concept and will and globally consists of the following components which will be discussed throughout this thesis:

- *Edges*
- *Intersections*
- *Exits*
- *Directions*

- *Distances*
- *Users*
- *Paths*
- *Maps*

By connecting edges and intersections together it is possible to construct a map. This does not sound as a reasonable task for people to do when they are in the situation described. They probably have something else on their minds and also even if they are carrying pen and paper, solving such a problem is not something the average person is able to do on the fly when faced with a complex world. One can imagine though that this process could also be left to a computer. All a person would have to do is indicate when he encounters a hall or a crossing. Assuming they are all carrying a portable computing device capable of wireless communication, it even is possible for them to communicate about their world models with each other [Figure 7] and make attempts to build a more or less shared world model.

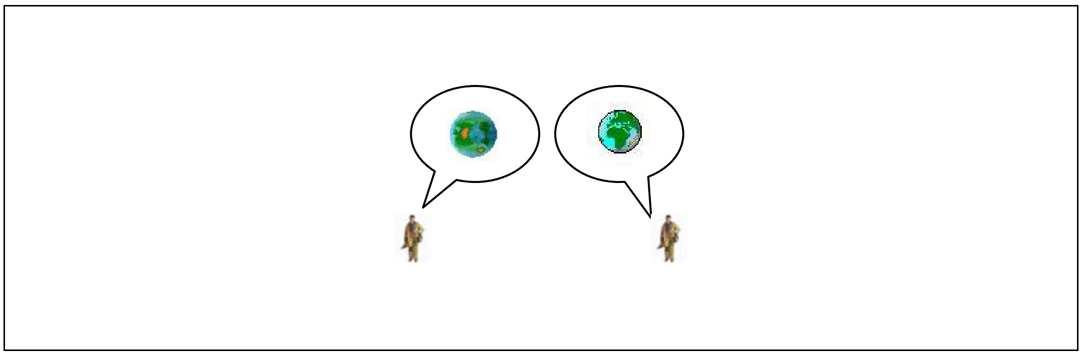


Figure 7 Different world models of the same world

If we are able to get world knowledge from the users mind into the computing device they are carrying, it is possible for the software - which we will refer to as agents from now on - on these devices to start communicating about the world with each other autonomously. We presume these agents operate in an infrastructureless environment and want to get information about their relative position to other mobile ad-hoc nodes in the world by constructing a shared world map. This so they can then again provide this information to the user of the mobile device it is running on. To expand their knowledge, agents who are within each other range will attempt to exchange context information and make efforts to match it with their current knowledge.

As mentioned before, it is very important to notice the fact that this entire concept is not dependant of any infrastructure, accept from the possibility to communicate via ad-hoc networking technology such as the widely accepted WiFi and Bluetooth standards. Please also note that this also means that the availability of satellite location systems such as GPS cannot be assumed. If such systems are available logically recommend it to be used as it can help refining the world model to greater detail. Reasons for not depending on the availability of GPS-like systems are mentioned are mentioned in [2.5.1]. Of course it is not guaranteed mobile ad-hoc communication will always work either. It could for instance happen there is too much radio frequency (RF) interference in the area where the system is deployed. Therefore our system is not suggested as a replacement to any system, but merely as an addition to other localization methods. Agents living in a mobile ad-hoc network will not likely be able to reach all other agents at all times. Therefore a particular agent will not have all and the same information available as all the other agents in the network, thus forcing them to form their individual opinion about the outside world.

The concept has three main areas of interest: *Location, Data distribution, Planning* and *Execution*, where the former is required for the latter. Only the location component and a large portion of the data distribution component will be discussed as part of our architect, the rest is future work and discussed in chapter 7.

3.1 Location

The problem of enabling agents to be able to communicate about events at certain locations without a common world map and a 'global positioning system' is far from trivial, even more in an infrastructureless environment. After all it is required that there is a well-defined mutual concept of location first, only then constructing a shared concept of the world can be attempted. We have chosen to approach this by building and sharing local partial maps first. These maps will primarily be based on user input. In practice this means the user indicates when there is an intersection etc. From this data only a map will then be built. When agents meet for the first time they will exchange their maps (or partial maps of their current area) and try to merge them, this to prevent each user has to explore the entire environment. If this succeeds the agents should have a shared concept of location. If a shared concept is in place, because the two agents met before, they can suffice by exchanging incremental updates of their maps. Not sharing already known knowledge will save precious processing power and most importantly network traffic.

There is a sufficient amount of literature available on exploring environments and building maps using sensing limited robots, which in fact is similar to our situation. In our case a person replaces the robot. Even though people have different sensing capabilities than robots, it is still possible to use many ideas of robot mapping in our system. As soon we have a collection of consistent partial maps, that have enough data in common, we can match these maps with each other. If a match is found they can be merged, resulting in more complete maps of the world. Such a matching and merging process can in essence be seen as a form of the maximum common sub graph problem [Bunk00]. Realizing the above is the first and main part of the proof of concept, which will be discussed in chapter 4. Below we will treat different approaches to this problem and the reasons for choosing an approach based upon user input only.

3.1.1 User input mapping

As noted in chapter 2 a lot of work going on in the field of determining positions in an infrastructureless manner. A reason for not choosing this approach is that we found no ready-to-use implementation, which suited all our needs, was available. In addition we prefer to focus on *application*s of agent technology in mobile ad-hoc networks, rather than indulging in electro-technical issues. Therefore we will use a positioning system, which processes user input to build maps and keep track of positions. Many systems are available using a single robot to map the world. There also has been much interest in map-making with multiple robots; see [Butl01][Dede00][Rike94].

In some of these multiple mobile robot systems, the robots must be able to cooperatively explore and map an environment to be able to perform tasks such as cleaning a building. Many different approaches have been proposed for such problems. Some systems use occupancy maps, some topological mapping, some use robots with many sensory capabilities and some with almost none. Our approach though is that the robot is 'replaced' by a human user. This means there will be different forms of environmental input than when using a robot. Many of the robot-mapping systems depend on the sensors to be very accurate. A human can never be as accurate in measuring distances, angles etc, as most combinations of sensors used in these systems.

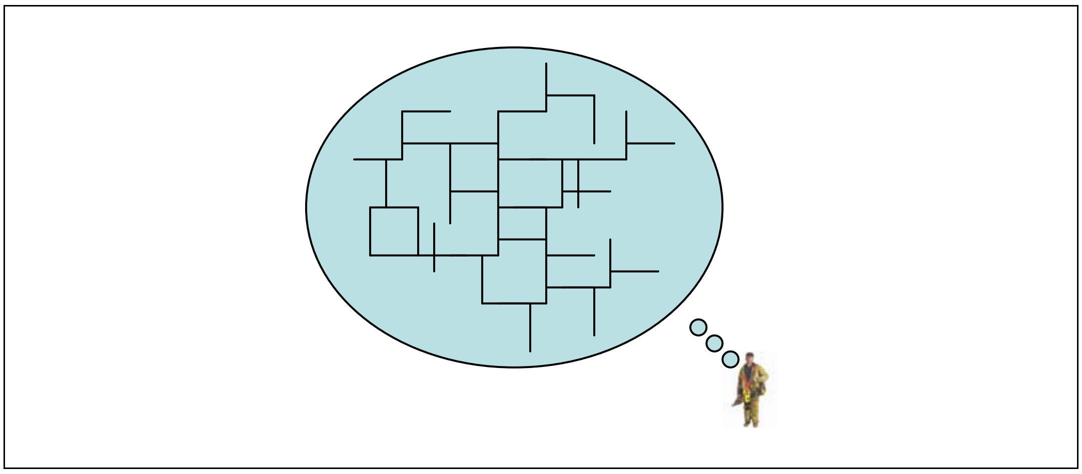


Figure 8 User world model

What a user is able to be accurate enough in (better than the average robot) is mentioning when encounters an intersections, meaning indicating there is a corridor to the left and one to the right etc. A user is also able to give a rough estimation of distances, which means he indicates one corridor is for example twice as long as the other. The idea that this input alone should be enough to build a world map [Figure 8]. More on this in chapter 4, where the system design of the proof of concept is explained.

3.1.2 Combining map information

A single human user is only able to explore so much of a particular world in a certain period. Therefore it looks promising to make attempts to combining the knowledge collected by different users [Figure 9], to be able to form more complete maps of the world [Figure 10].

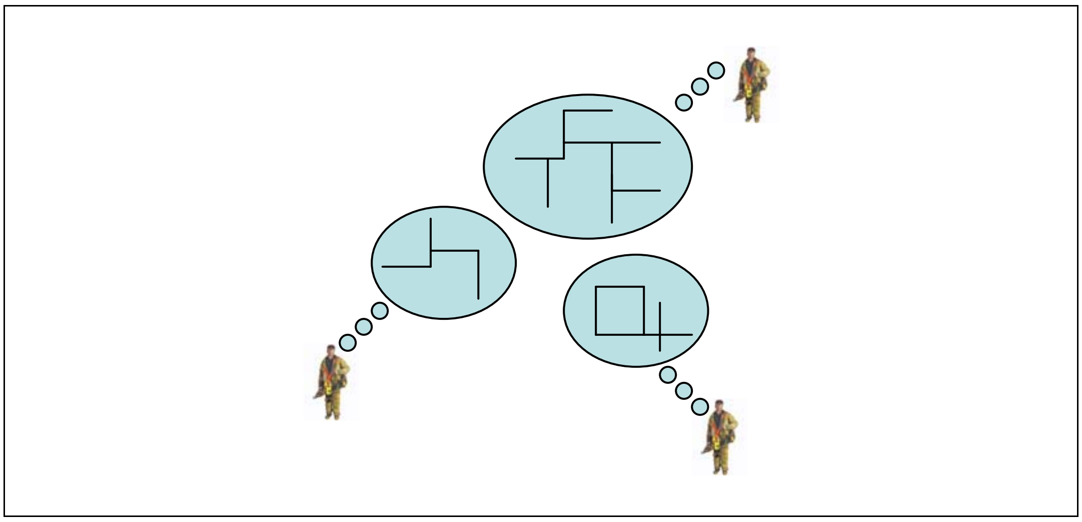


Figure 9 Users have distinct world models

Combining maps can be a difficult problem when the agents do not have a common reference frame. But finding such a common reference frame is greatly simplified when topological maps are used, as they provide a concise description of the world. By topological maps we mean graph-like representations of a world with ordinal distances between graph nodes. Another constraint - which could be released in future work - is that our world model will only use angles of exactly 90°, i.e. we will only create and merge rectilinear topological maps. In the proof of concept we will formulate an algorithm for merging two or more topological maps and prove our algorithm using simulation tests.

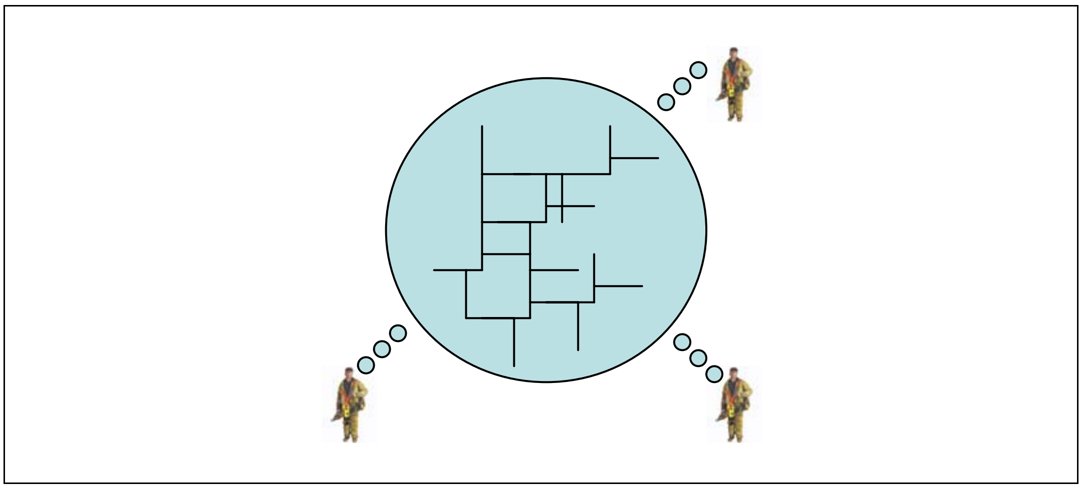


Figure 10 Combined world model

Each topological-map matching problem can be viewed as an instance of the maximal common sub graph problem and is related to the problem of sub graph isomorphism. Therefore ideas for our algorithms are partially based upon ideas from the graph matching literature [Vali01][Bunk00]. It is important to note that to be able to write an algorithm for merging topological maps the maps are assumed consistent. Meaning no two vertices may represent the same place. Not having this constraint would make merging maps virtually impossible and would require different techniques for solving the merging problem.

3.2 Requirements and constraints

When designing any system, there are requirements and constraints that need to be taken in mind. The fact that we have a concept of an *infrastructureless* system implies there are not that many requirements. There still are some worth mentioning though. Although the proof of concept will be based on a simulation, an actual system based on our concept would have to run on handheld computers. We will base us on the PDA commonly used in our project group, the Sharp Zaurus SL-C760 [Figure 11]. It has a maximum resolution of 640x480 pixels, a 400 MHz Intel XScale PXA255 processor, and 62 MB of RAM. The main constraint, following from this, is the limited processing power and memory of a node.

Figure 11 Sharp Zaurus SL-C760

Another constraint is that the system will be designed for wireless ad-hoc communication – the SL-C760 is 'wireless ready'. Because in such networks response times and data transfer speeds are much slower than a wired network, we need to make sure we keep the data traffic to an absolute minimum. At the time of writing no suitable ready-to-use 'MANET-agent-platform' was available to build the system on. Hence it was decided to build a simulation. Although not building an actual system, it is required the demonstrator takes into account the constraints which come with the future anticipated operating environment. Also to be able to focus on the problem, the virtual world will have the constraint of being a labyrinth-like environment. Summarizing, the requirement and constraints for our proof of concept are:

Requirements

- *The system should be designed to be able to run on handheld computers*
- *A node should have ways of providing local info on intersections and edges to the system; i.e. user input, GPS if available and pedometer for increased accuracy*
- *A node should be able to communicate with neighboring nodes (if they are close enough) via ad-hoc communication*

Constraints

- *From the requirement of using PDA's as nodes, there are limits on processing power requirements and memory usage*
- *Data traffic has to be kept to an absolute minimum*
- *Only a simulation will be implemented for this thesis*
- *It will be assumed everything takes place in a rectilinear labyrinth-like environment such as the floor of certain building, i.e. a rectilinear world. The main elements of this environment which are observed are:*
  - *Intersections and their directions*
  - *Edges .and relative distances*
  - *Exits*

Chapter 4: Global design

This chapter will explain how the system was built from different components, what the responsibilities of these components are, and how they are designed. First of all the system designed operates as a multi-agent system within simulation of a mobile ad-hoc network. As a basis for the simulation of a mobile ad-hoc network a system called 'Ad-hoc Simulator' (AHS) was used. Together with AHS a visualization system called Ad-hoc Visualization (AHV) was developed, which was also partially used in ManetLoc.

AHS simulates the behavior of PDAs and mobile communication in a mobile ad-hoc environment, which is visualized by AHV. The movement of mobile nodes in AHS is based upon a traffic simulation program [Kroo02], simulating cars driving in a city street-network. CityNetwork is also important for our system as the data-format of maps from the CityNetwork is used. More details on the design and implementation of these programs are provided in the next chapter. We mainly use the part of AHS that simulates wireless adhoc connectivity between nodes when they are close together in an environment, i.e. the ad-hoc WiFi component based on the ARA protocol. Also parts of AHV are used to visualize the global world state, i.e. nodes moving in the world.

The approach of building a simulation was chosen because building a real system didn't seem feasible because of time limits, availability of hardware (PDA's with ad-hoc network capabilities) and lack of a suitable ready to use platform to build the application on. The reasons for choosing AHS as a basis of our system and not other simulation software [Ns2] [Glom] [Swan] are:

- *AHS provides a simulation of an ad-hoc network, which is the environment our architecture is anticipated to operate in*
- *AHS was developed internally in our department, making the source-code freely available*
- *AHS is closely related to work currently in progress in our project group.*
- *The original system developer was still partially available for support*
- *The system was written in C#, which is very similar to Java, thus being relatively easy to port*

In the proof of concept AHS first is modified, including the visualization, in such a way that we are able to simulate user input and make node movement user controllable. A single node represents a PDA carried by a human user. Thereafter multi-agent functionality is added to the PDA nodes. Software agents will exchange partial world models over the mobile ad-hoc network. Agents receiving a world model from another agent will attempt to incorporate this new knowledge into its own and distribute it further into the network.

4.1 Overview

As a proof of concept for the main part of the architecture, described in chapter 3, and as a possible basis for further work a simulation system will was built. This involves storing and distributing location-context information based on user observation. In the simulation maps of an environment are constructed and distributed. The processes involved are primarily based on the positions different virtual people visit and any context information they might gather, such as where exits are and determining the shortest route to one.

In our program each node does will never know its exact world coordinates and initially does not have any knowledge of the world it's in at all. I.e. there is no knowledge of absolute position and no initial map of the world. The goal for each agent-node is to construct this internal map of the world by using its own observations and sharing information with other nodes. Knowledge nodes poses, is distributed to nearby nodes. In practice this will mean nodes that are within communication range. In our case the data shared is information about halls and crossings, but can be extended to suit the usage needs.

The main goal is to be able to regenerate the map - or one that is very similar - of the virtual world the simulated users are in. The system will load a predefined CityNetwork map and will allow simulated ad-hoc nodes to travel through this map. Please note that the nodes at the same time represent simulated human users and agents. As they have no preset knowledge of this virtual world and are only able to observe information like halls, crossings and exits at their current location, i.e. nodes can only observe what human users would be able to observe, or actually a lot less.

Using a system able to build a map of an environment by only having people report what they see, and without requiring any infrastructure, human users will be able to do their tasks (such as firemen) and help build a context map of an unknown environment at the same time. By doing this jointly and requiring no extra effort from a user an unknown area can be explored and mapped with less hassle, in less time and most importantly without requiring any network infrastructure. In an overview the basic modules of this system will be:

1. *Gathering data*
2. *Building topological maps from user and sensory input*
3. *Sharing and merging location context information between agents*
4. *Providing services based on location context information*

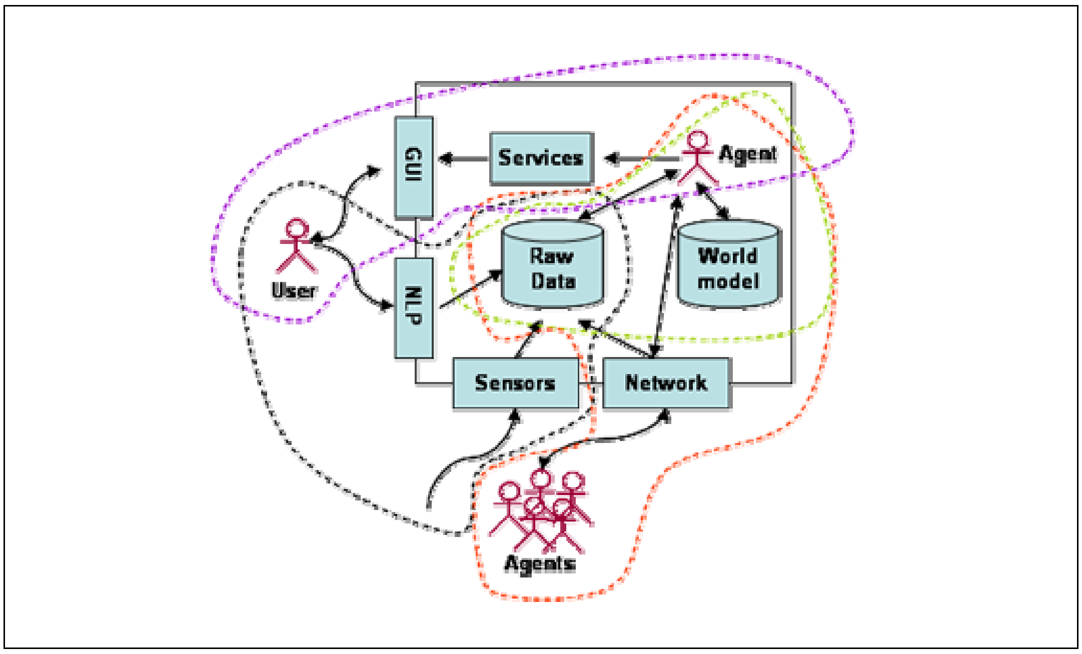


Figure 12 System modules and the PDA interfaces

[Figure 12] shows how these four overlapping modules fit into the internal and external interfaces of our PDA system (represented by the largest box). The next chapters will attempt shedding more light on these four modules.

4.2 Gathering data

Before anything else methods for gathering data on location must be in place. The collected data can then be transformed into knowledge and used for constructing and maintaining a world model. In our system data about the world is gathered in 3 ways [Figure 13]:

- *Sensory data*: the most trivial is that the system keeps track of the distances the user travels. For this it could counts the number steps the user takes and based on that (and possibly other data) make estimates of distances covered
- *User input*: preferably by voice recognition users supply information to the system, but other methods such as a point and click system could be used as an alternative or addition to the projected use of voice recognition
- *Other agents*: information provided by other agents within communication range can be merged with the agent's own world model

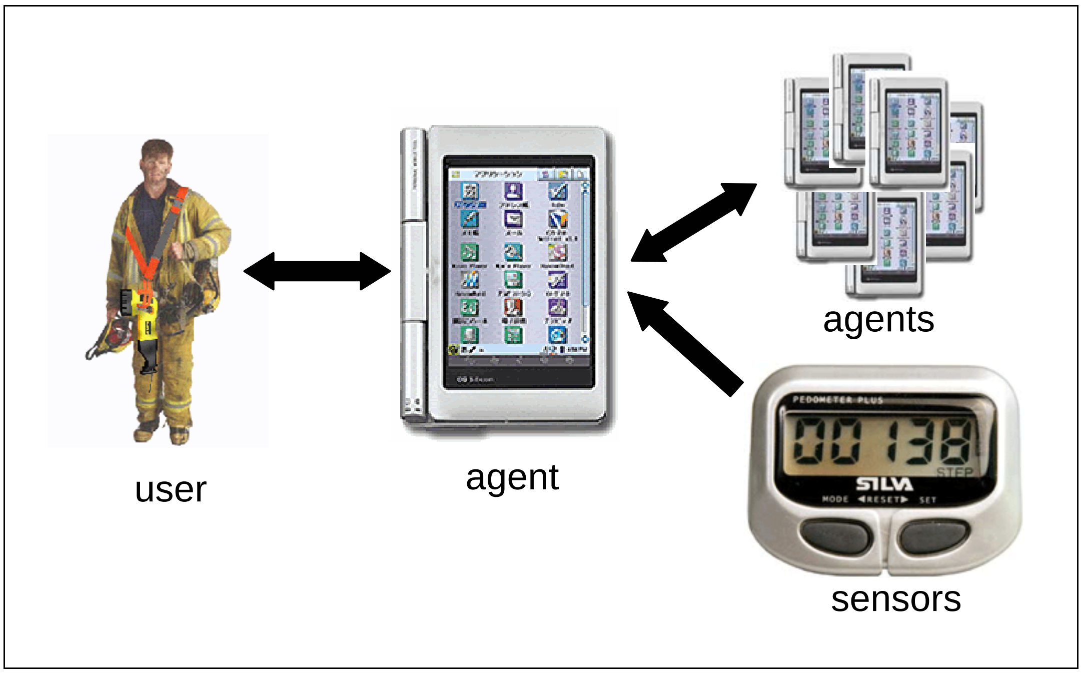


Figure 13 Potential sources of data

With the gathering of this data, certain difficulties emerge related to converting data to workable knowledge, some that are worth mentioning are:

- *As the real world is dynamic one should be able to determining if data is duplicate, new and still valid*
- *Input data will be fuzzy, this needs to be handled in some way*
- *Combining (different types of) data in one world model*
- *Combining world models from multiple sources*

Although the first two topics are also worth studying, most research for this thesis went into combining location data. This to be able to create world maps, and thus will be discussed more thoroughly in the next sections.

4.3 Building topological maps from user and sensory input

The first goal for our agent-system is to be able to construct a local word model from data the user and sensors provide about the world. For the form of the world model we chose to construct a topological map. Considering the fact that our focus is on indoor environments and road networks and as these worlds can be relatively easily translated into a graph-like representation, the choice of generating topological maps seems to be appropriate [Remo02].

The most trivial input a user can provide for building a map of a building is to indicate that at a certain time he encounters an intersection and possibly indicating the number of paths and their directions. Although it would be helpful if the user could provide more specific information such as the wind direction (i.e. 'I can go east and west here') it is not necessary. This specific example would require user knowledge of his global orientation, which cannot be depended upon. Firstly this is because we cannot assume user caries a compass. Secondly the user is not likely to be able to orientate very well because of the crisis situation he is in. Therefore, as the user is exploring a rectilinear labyrinth, just left, right, forward and backward indications will be sufficient input.

To be able to build a map from user input only it is trivial the user provides correct information. So when he has changed direction the system should always be informed of this event. It is possible to allow the user to make errors in his observations but this is beyond the scope of this research. The only provided backup for the user making mistakes is checking for inconsistencies in the stored graph(s). If one is found in the users own observations, all his observations that occurred after the inconsistency are rendered incorrect. Summarizing, the information gathering process globally consists of two components:

1. *Observations*: i.e. at time t1, when the user reports arriving at an intersection, where one can go left and right, according to his observations. In most cases it can be assumed a user can also go back, providing this isn't the first observation.
2. *Actions*: i.e. at time t2 the user report he walks into a certain direction.

From all observations and actions the user reports, a graph can be constructed. This graph will represent a topological map of the world, which should be able to be displayed on the screen of the PDA and by this visualizing it to the user [Figure 14]. To be able to generate such a map it is important to be able to estimate (relative) distances traveled. These distances can be calculated from the time between two observations, combined with the type of movement (running, walking, etc) and can be refined by using pedometer [Ches01] input. There are more possibilities but we prefer to use as little specialized hardware as possible. In our system we will make the assumption the user provides relatively accurate indications of distances traveled (+/- 10%).

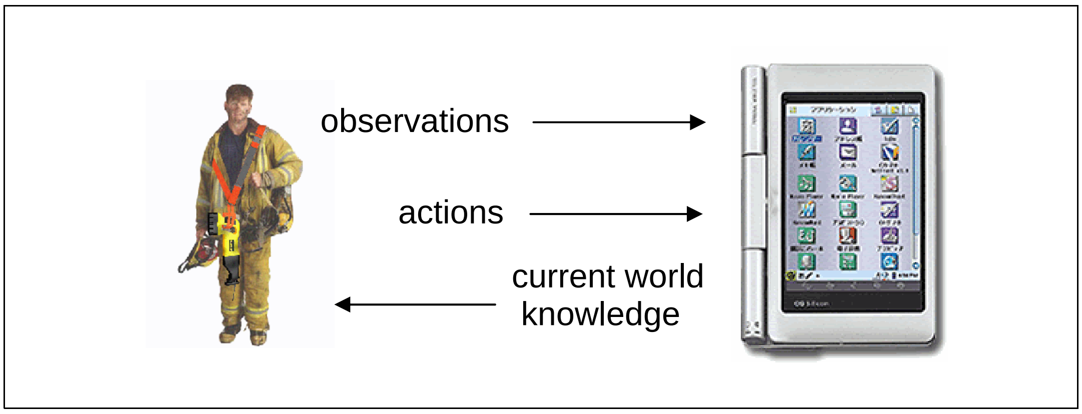


Figure 14 User input and system feedback

The main problem in a graph generated from the scarce input provided by the user detecting a loop in a path and closing it to keep the map consistent. Keeping the map consistent is critical, as we need to be able to match and merge the map with the maps of other agents. Closing loops and merging maps will be described in depth later on, but first we will provide an example.

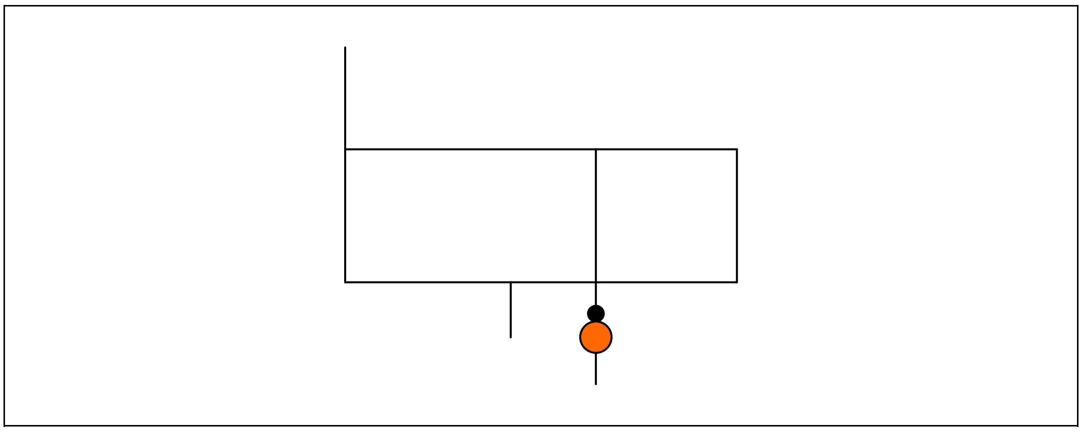


Figure 15 User in a simple world

Please imagine a user walking around in the world of [Figure 15] for this purpose. In this figure the lines indicate corridors. The user is visualized with the two dots, where the smaller indicates his orientation. The user starts at the marked location and starts exploring the world. Whenever he performs any relevant actions or does an observation the user will report it. In an actual system, providing this information should be possible via multiple ways, to allow operating under many different conditions for instance. Examples of such input methods are voice recognition, text input, pointing/clicking on a device and possibly even pointing a sensor (like a) camera at a certain object or location. Regardless of the input method used we will assume all user input is 'perfect', so the user will never forget to mention a fact like arriving at an intersection.

Time	Observation	State
1	In a hallway	Walking
5	Can go left, right, straight	Took left and walking
8	Can go, left, straight	Stopped
15		Took left, walking
17	Cant go any further	Turned 180, running
19	Can go left right	Go left, running
23	Can go right	Turned 180, walking

Figure 16 Example list of user input

The table in [Figure 16] lists information the input process could provide. The agent can then translate such information into vertex and edge knowledge. In CityNetwork-like format this would look similar to [Figure 17]:

```
intersections
{  //{number, x-pos, y-pos}
   {1, 0, 0}
   {2, -3, 0}
   {3, -3, -2}
   {4, -9, 0}
}

roads
{  //{number, from, to}
   {1, 1, -1}
   {2, 1, -2}
   {3, 1, -3}
   {4, 1, 2}
   {5, 2, 3}
   {6, 2, 4}
   {7, 4, -1}
}
```

Figure 17 Translation of user input to a CityNetwork map

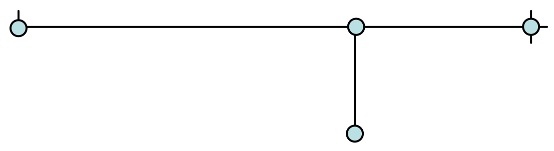


Figure 18 Example system output

Please note that a 'negative' intersection in the roads section of the map indicates an unexplored road (-1=up; -2=right; -3=down; -4=left). In the actual system the data format is a bit little different. This format is used to ease explaining the workings. Also, contrary to CityNetwork maps all roads are bi-directional in the format. Finally on a screen such a map could be visualized as [Figure 18].

4.3.1 Closing the loop

If a user would travel long enough in a building he will eventually always return to a location that was visited before from a different direction. The process described above will then result in loop in the graph, which should be detected and closed. A correctly detected closed loop is very valuable information, as it is required to make a map consistent. Consistent maps are required by our system to be able to match and merge one map with another [4.4]. The question is how to detect such a closed loop [Save04]. As an example of closing loops we take an even simpler square shaped world [Figure 19]:

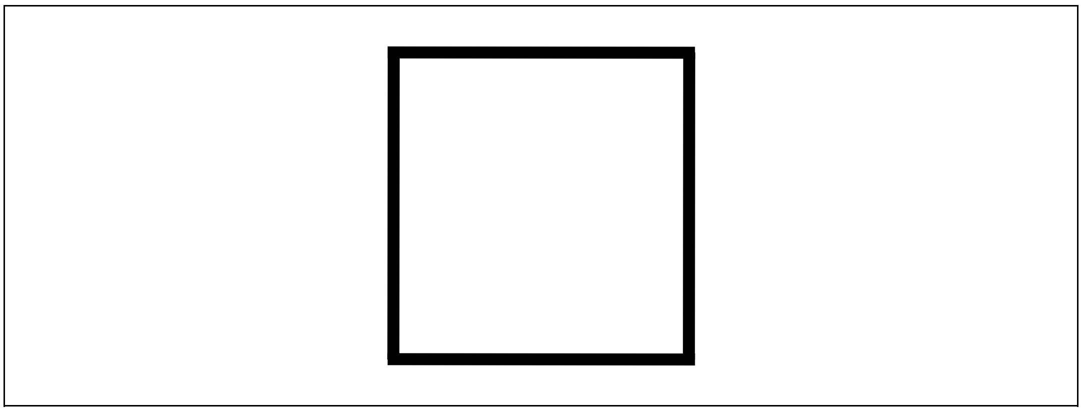


Figure 19 Example square world

Applied to the square world the process described in the previous chapter might result in the following 'map', where the two upper-left intersections, should be recognized as one and the same, but at the moment we still have 5 vertices in the graph where there are only 4 intersections in the world [Figure 20].

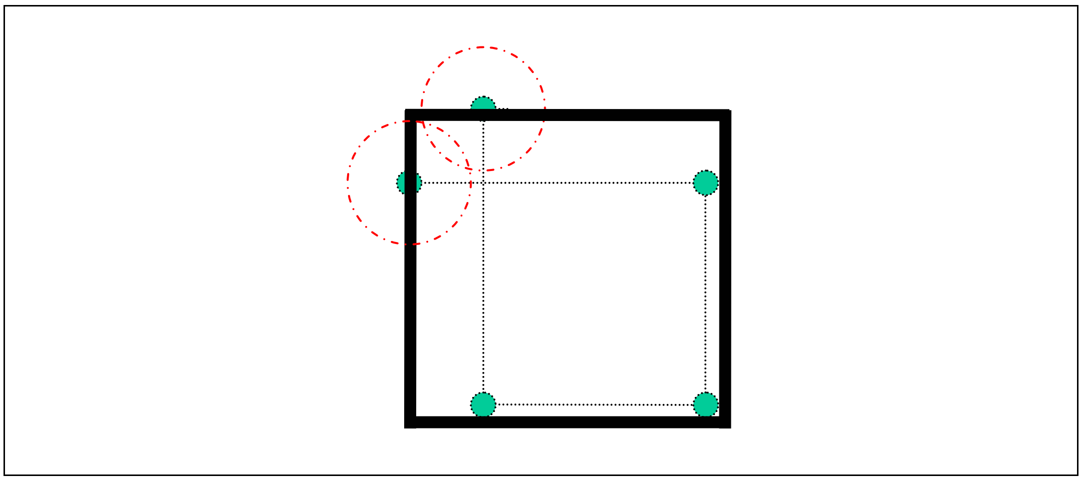


Figure 20 Open loop

Before being able to close a loop we should be able to detect and build hypotheses concerning possible loops. For this we roughly follow a procedure in which is checked for each new vertex if there is a vertex nearby that might be closing a loop. If we can find two matching vertexes a loop hypothesis is started. For our algorithm it is also important to note that a loop always has a minimum of 4 vertices and an even number of turns.

To make the graph consistent when two matching vertexes are found, small adjustments can be made to the endpoints of edges [Figure 21]. Whenever a loop hypothesis has been formed we can start testing it by comparing edge lengths of the supposed loop with new measurements. These measurements will result in accepting or rejecting the loop. It is impossible to check with 100% certainty if a hypothesis is correct, but we just use a fair amount of measurements. If inconsistencies are found later on the hypothesis and changes made based on this will be rejected.

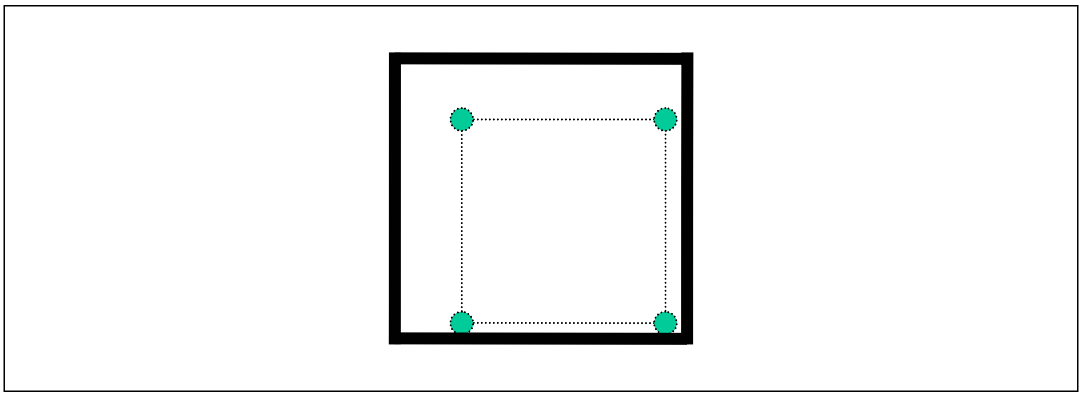


Figure 21 Possible closed-loop hypothesis

A danger is that incorrect hypotheses may be accepted as correct or vice versa. A way to deal with this is to keep multiple hypotheses and never commit to any one of them, i.e. retain the hypothesis that the loop has not been closed. Hypotheses can then be accepted or rejected later on when new observations have been added and map data was received from other agents. If other agents confirm the conclusions about a specific part of the world the more likely it is to be correct. On the other hand, agents have different opinions about this particular part of the world, decreases the likelihood of the hypothesis being correct.

It is also preferred to be able to deal with mapping errors such as incorrect hypotheses, input errors and dynamic worlds. In a dynamic world more recent information will intuitively have more meaning than older. Dealing with this is incorporated into our system by decaying the influence of individual readings over time [Thru97]. New information inconsistent with the old information could cause the old to be discarded. Though our system will not work in a dynamic world, still newer information is preferred over older. In our simulation we will only use time stamped data for repairing incorrect hypotheses and input errors, but in future work it could be adapted to be used in dynamic worlds.

4.4 Sharing and merging location context information

Our agents are operating in a mobile ad-hoc networking environment in which it is likely they encounter other agents every now and then. When an agent detects one or more (compatible) agent(s) within its communication range, they can share the knowledge they have about the world and possibly are able to merge this [Figure 22]. Each agent will attempt to merge information from other agents for themselves. Consequently it is possible for two agents to come to different conclusions. This makes it possible for PDA's to have different agent software (versions) running or to be in a different processing mode. The only requirement is that the messages they send out are compatible.

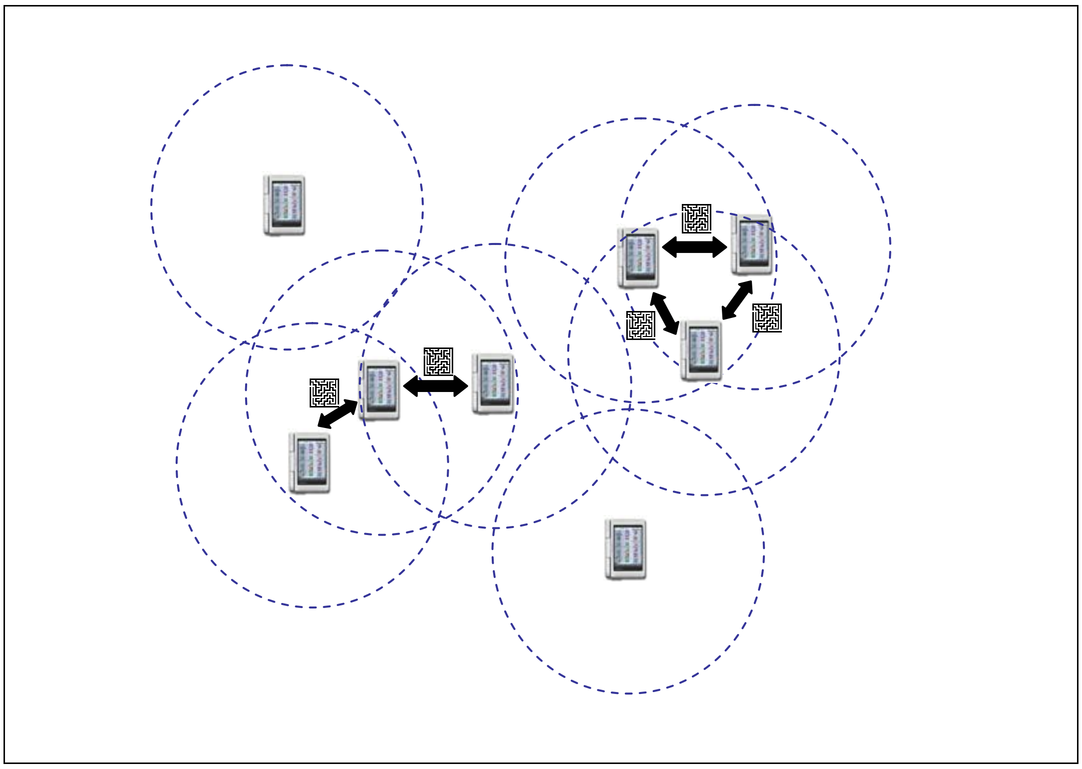


Figure 22 Agents in mobile ad-hoc networks share maps when in range

As stated above we can build a consistent topological map at any time from the user and sensory input the agent gathers in time. It is very important this graph is consistent, as we want to share and merge this knowledge with other agents.

Similar to the internally stored world models, maps are shared in the form of graphs (topological maps). They are assumed to be consistent, but not exact and do not always have a common reference frame. To prevent errors consistency is checked prior to making a merge attempt. Inconsistent maps could be repaired in some cases. Starting with consistent maps makes merging the information a lot easier and reliable.

The data format used when distributing also is similar to the format used in CityNetwork maps, but slightly adapted to fit our needs [Figure 23]. In this format, vertices correspond to intersections and edges to roads in the city network format. Exits, will be discussed later on, they denote possibilities for leaving the area. Location refers to the current location relative to a vertex of the user sending the data. If two or more users have explored overlapping regions of an environment their agents should have topological maps that have common sub graphs with identical structure. Since having a common sub graph it is possible to find a reference frame and merge the two maps into one. We will use the maps from [Figure 24] and [Figure 25] for illustrating the algorithms to accomplish this. Solving the map-merging problem is thus analogous to identifying a matching between the graphs of map A and map B. This will be discussed more in detail in the sections below.

```
vertices
{
    //{x-pos, y-pos}
    {0, 0}
    {0, 200}
    {200, 200}
}
edges
{
    //{direction, vertex number 1, vertex number 2}
    {3, 2, 1}
    {4, 3, 2}
    {3, 4, 3}
    {2, 4, 0}
}
exits
{
    //{direction, vertexnumber}
    {4, 2}
}
location
{
    //{direction, vertexnumber, edgeposition}
    {3, 4, 56}
}
```

Figure 23 Example map information shared between agents

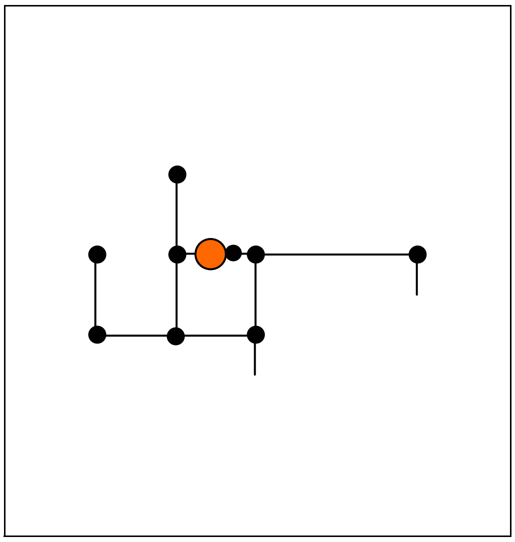


Figure 24 Map A

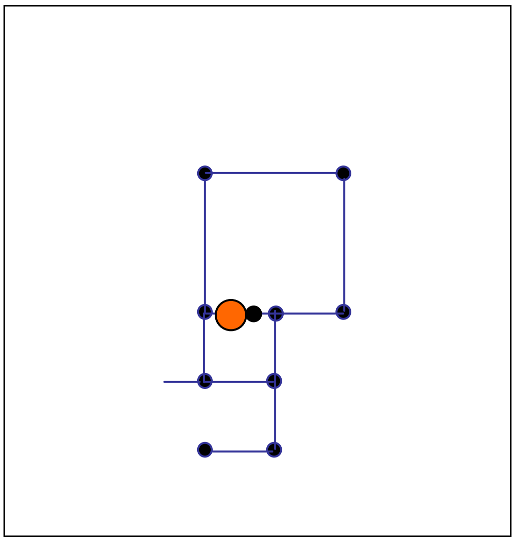


Figure 25 Map B

4.4.1 Matching

In order to be able to merge two maps, we first need to match the maps together, building hypothesis and choosing the correct one (i.e. the best match). A hypothesis is a possibly rotated sub graph that the two maps have in common. There is a chance the process described below does not supply a (large enough) hypothesis. If this is the case and not enough vertices can be matched to make a good hypothesis yet, the map received is stored internally. In that case we can try the matching process later on when a more complete world model is available. The used algorithm for matching two maps consists of three phases:

1. *Vertex matching*
2. *Growing hypotheses*
3. *Combining hypotheses*

Vertex matching
The first step taken in matching is building a list of all vertices that match each other in the two maps. Two vertices only match if they have the same edge directions. This is also the case if a vertex needs to be rotated to match, which is also stored. We expect exactly known attributes vertices, such as the type of the vertices to match perfectly. However, attributes that are subject to measurement error can be compared with a similarity test. In the case of our simulation we don't have any fuzzy variables of a vertex, but in a real environment or an extension of our simulation some could occur.

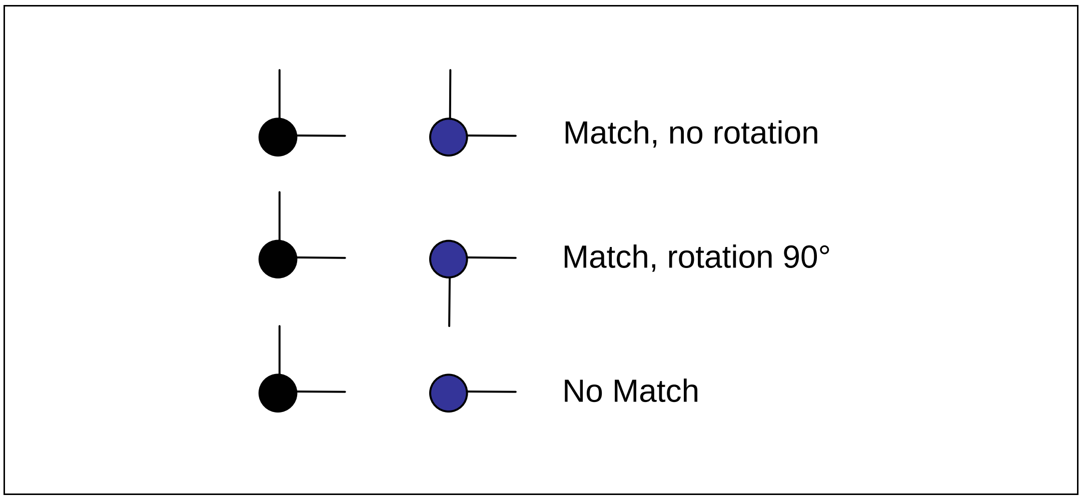


Figure 26 Matching vertexes

Growing hypotheses
After having built lists of matching vertices we grow matches by testing corresponding pairs of edges leaving the paired vertices. If the edges are compatible and the vertices at the ends are also compatible, they are added to the hypothesis. If the edges or vertices are incompatible, the entire hypothesis is rejected. The vertices are tested with the same type of criteria and similarity tests used to form the initial pair. Edges may also have both exactly and inexactly known attributes. In our system, they have their path length compared with a similarity test.

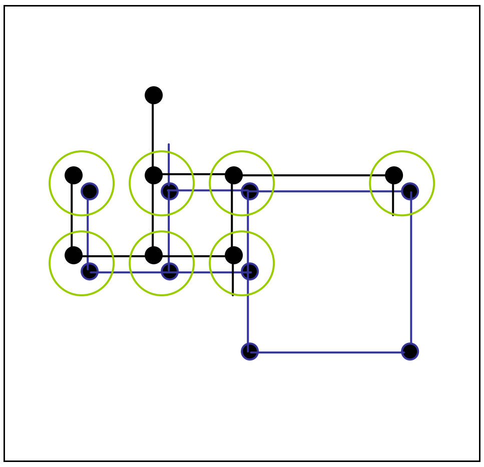


Figure 27 Successful hypothesis growth

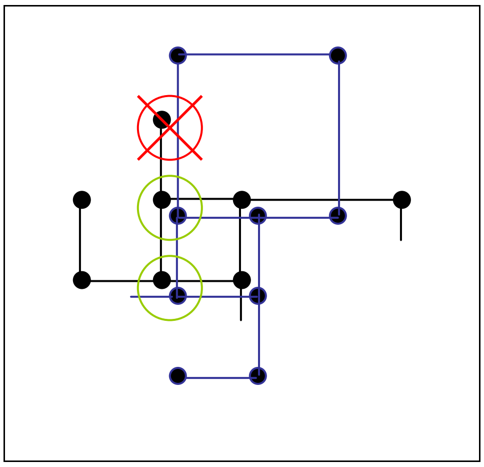


Figure 28 Unsuccessful hypothesis growth

Our hypotheses are the unique matches surviving the growing process. Duplicate hypotheses are avoided by keeping a table of vertex pairings. When vertices are paired during the growth phase, the corresponding entry is marked in the table. This entry is then ineligible as an initial pairing of vertices.

A sub graph in one map can be matched to multiple sub graphs in the other under separate hypotheses, but a pair of matched vertices with a given edge correspondence can appear in only one hypothesis. The matching and growing process is repeated until all valid vertex pairings are examined. Please note that if we would be working with imperfect user input - which is not the case in this concept - a procedure filtering noise should be added. When able to match enough common vertices and edges, and if there are a minimal amount of conflicting vertices and edges, the conflicts can be discarded and the hypothesis accepted.

Combining hypotheses
If successful the hypothesis growing process described above results in list of possibly multiple hypotheses within the same rotation. From these hypotheses one has to be selected. Before this takes place it is possible that if such a list contains more than one hypothesis, some of these entries are consistent with each other. These hypotheses are then combined with each other into one larger hypothesis cluster.

4.4.2 Merging

After the system has chosen a hypothesis cluster, the next step is to merge (or flatten) the two maps into one single map. Estimates of path lengths can be updated by combining the measurements from the two maps for corresponding edges. The edge orientations at the corresponding vertices can be similarly merged. Parts of one map not present in the other should be added. The merging process is globally performed in four steps [Figure 29]:

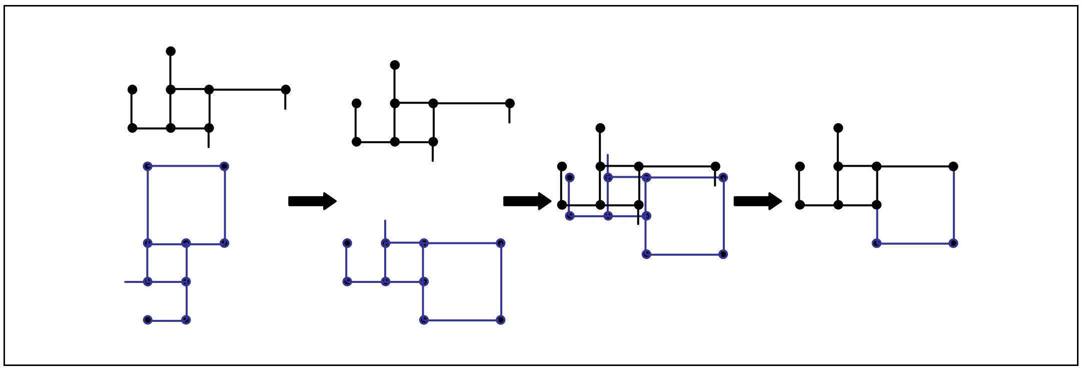


Figure 29 Map merging process

1. *Rotate the received map so its orientation matches the local node's map*
2. *Shift the rotated map so its coordinates match the local agent's map*
3. *Add any new vertexes from the rotated and shifted map to the local node's map*
4. *Connect everything together (update edge lengths, check for inconsistency's etc.)*

Please note that the choices made, may later turn out to be incorrect. For example, early in the process of exploring a self-similar environment, a user might seem to be exploring the same area when in fact they are exploring similar but distinct areas. To protect against such situations, it is remembered from whom and when the new parts of the map are received. This makes it possible for discovered inconsistencies to be removed or corrected later on without discarding the whole map. This is possible as data is marked with time stamps and the id of the agent the data was received from.

4.4.2 Incremental updating

After agents have merged their maps once two nodes may later exchange maps information again. This is also the case if they weren't able to merge their knowledge, but just exchanged their knowledge before. An incremental update can save a substantial amount of computation and bandwidth. This is possible with minimal bookkeeping effort. Each agent must maintain timestamps so that only new and modified vertices and edges since the last update are exchanged. If a successful merge took place before and if it still appears to be correct we already have a hypothesis and can computationally easy merge the new information. If the agents were not able to merge their maps before the previously received map is merely be updated with the new information, and another merge attempt can be made including the new world information. The computational and network traffic savings of incremental updating can be significant if implemented correctly. The former highly depends on the amount of information retained from the original map merging.

4.5 User guidance

When an agent running on a user's PDA gathers knowledge about the world, the knowledge can be used to provide services such as advice to the user. One of the possibilities is guidance of the user. Here one can think of the possibility of guiding the user to a certain location such as the nearest exit or an unexplored area of the map. Both options are implemented in our proof of concept.

Nearest exit
When a user indicates he wants to leave the area as soon as possible the system can instruct the user how to walk, calculated using a shortest path algorithm (i.e. Dijkstra, A*, Ant based, etc.) . The result of a request for guidance to our system is indicated, by giving the path to the nearest exit a specific color [Figure 30]. Green dots are exits; the red line is for guidance.

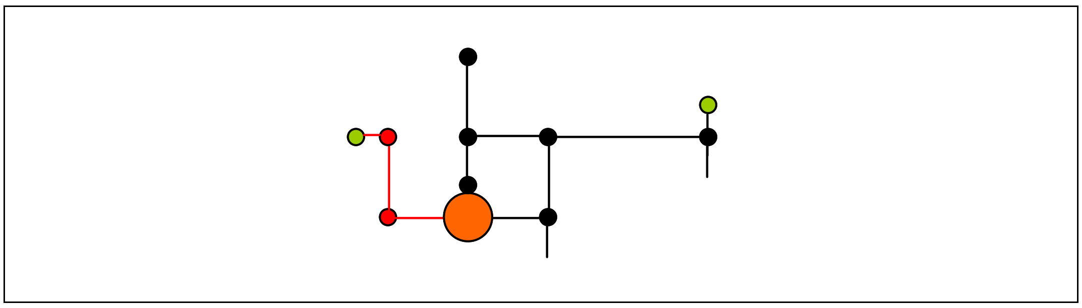


Figure 30 Guiding user

Unexplored area
When the goal of the user is to (help) explore the world the system can instruct the user where the user should go to help optimize the process. As this is very similar to instructing the user to a nearest exit the same type of shortest path algorithms can be used.

It is trivial there are many more services that can provided to the users by agents with knowledge such as in our system (although thinking of and implementing useful services is not). In the context of the crisis management project, knowledge could be gathered about crisis indicators (fire, smoke, etc) and translated into some sort of scenario (e.g. the building is on fire, south side is still clear to pass). It is not unthinkable that having this type of knowledge available in an agent network could even be used to coordinate the actions of individuals and groups.

Chapter 5: Implementation

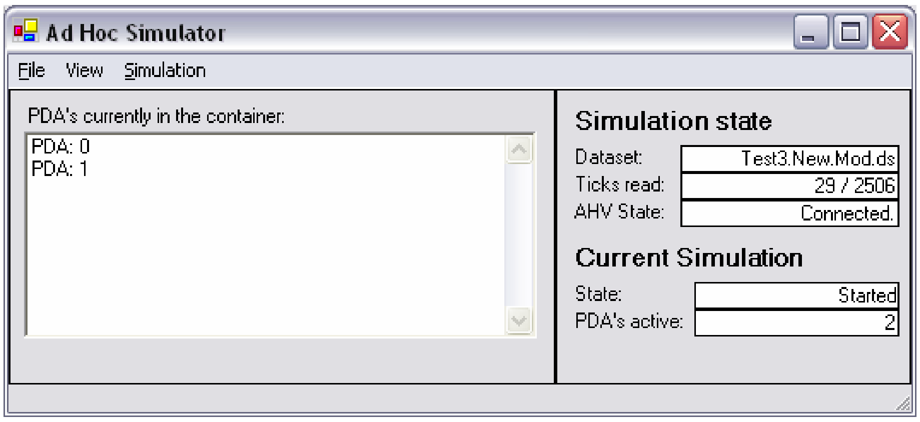
In this chapter the implementation of the system that was built – ManetLoc - will be discussed in detail. ManetLoc was built in the form of a simulation and has been implemented in the C# language. For this the Microsoft Visual Studio .NET development environment was used. At the hand of diagrams we will show the workings of the different parts of the system. The most important classes, functions and algorithms will be discussed. As ManetLoc was built on top of the AHV and AHS systems the design and modifications of these systems will be discussed first.

5.1 Existing software and data structures

The three main software components used are the Adhoc Simulation (AHS), Adhoc Visualization (AHV) and Network generator. The most important data structure, which was reused are the so-called CityNetwork files. These files bind these three applications and ManetLoc together. In the next subchapters these essential elements of ManetLoc will be discussed.

5.1.1 AHS

AHS is a multi-threaded application, simulating a mobile ad-hoc network. The program reads node information from a dataset, calculates nearest neighbors and maintains a collection of PDA's, MAC layers and ARA protocols. It provides PDA network statistics and a minimal user interface [Figure 31].



PDA's currently in the container:
PDA: 0
PDA: 1

Simulation state

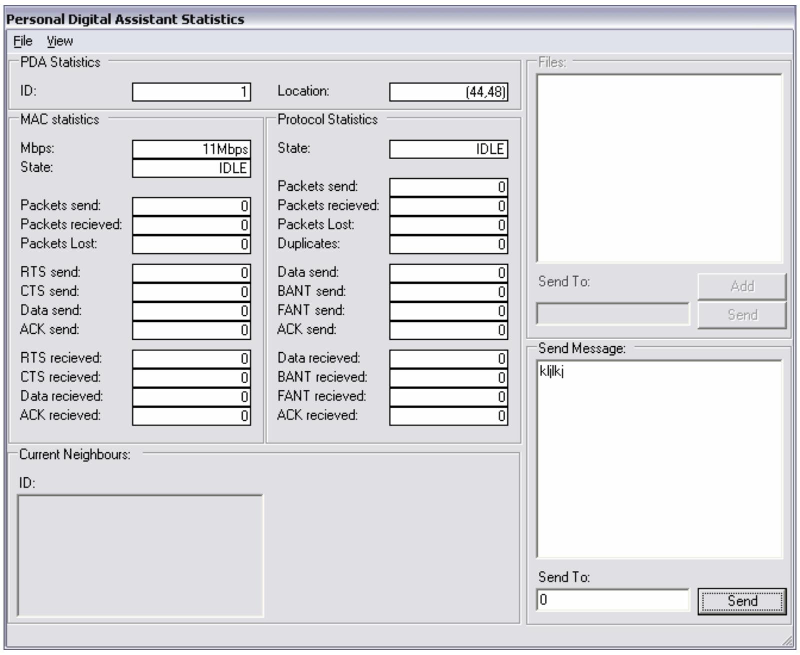
Dataset:	Test3.New.Mod.ds
Ticks read:	29 / 2506
AHV State:	Connected.

Current Simulation

State:	Started
PDA's active:	2

Figure 31 AHS main interface

On the left side of the AHS main interface one can see the list PDA's that are currently in the network. Double clicking a PDA, will show the PDA statistics [Figure 32], providing information such as the number of packets sent during the simulation by this specific PDA.



ID: 1
Location: (44,48)

MAC statistics:
Mbps: 11Mbps
State: IDLE
Packets send: 0
Packets recieved: 0
Packets Lost: 0
RTS send: 0
CTS send: 0
Data send: 0
ACK send: 0
RTS recieved: 0
CTS recieved: 0
Data recieved: 0
ACK recieved: 0

Protocol Statistics:
State: IDLE
Packets send: 0
Packets recieved: 0
Packets Lost: 0
Duplicates: 0
Data send: 0
BANT send: 0
FANT send: 0
ACK send: 0
Data recieved: 0
BANT recieved: 0
FANT recieved: 0
ACK recieved: 0

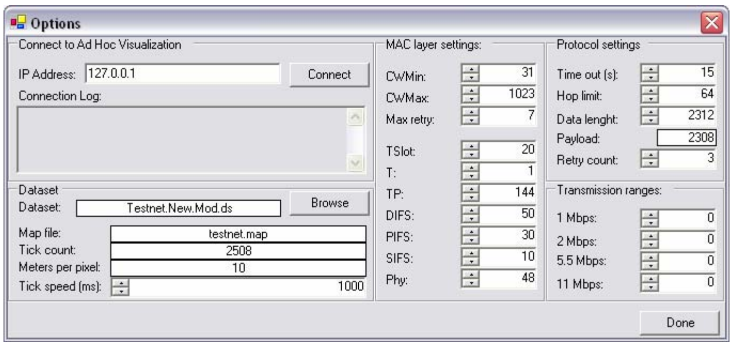
Send Message: kljlkj
Send To: 0

Figure 32 PDA Statistics

On the right side of the AHS main interface the state of the simulation is shown , i.e. :

- *The dataset currently loaded,*
- *The current tick:* a counter for the 'line number' the simulation has read from the dataset
- *If AHV is connected*
- *If the simulation is running*
- *The number of active PDA's*

To start a simulation, one has to load a dataset using the options dialog (Simulation → Options) [Figure 33]. Besides loading a dataset some simulation parameters can be modified in the options dialog, such as MAC layers settings.



IP Address: 127.0.0.1
Dataset: Testnet.New.Mod.ds
Map file: testnet.map
Tick count: 2508
Meters per pixel: 10
Tick speed (ms): 1000

MAC layer settings:
CWMin: 31
CWMax: 1023
Max retry: 7
TSlot: 20
T: 1
TP: 144
DIFS: 50
PIFS: 30
SIFS: 10
Phy: 48

Protocol settings:
Time out (s): 15
Hop limit: 64
Data lenght: 2312
Payload: 2308
Retry count: 3

Transmission ranges:
1 Mbps: 0
2 Mbps: 0
5.5 Mbps: 0
11 Mbps: 0

Figure 33 AHS options dialog

A main element in AHS is the dataset (in [Figure 33] 'Testnet.New.Mod.ds' is loaded). This system is partially replaced by a different system in ManetLoc. To understand the modified system the original will be explained first. A dataset [Figure 34] in AHS and AHV contains coordinate information output from a modified version of the traffic simulation program. A dataset comprises header, multiple ticks and node location information.

```
H|test3.map|2506|36|120|120|0|0|80
T|0|1
0|0|99.7301472860233|39.2363011519124
T|1|3
0|1|99.359357297353|39.2363011519124
1|1|58.0224533937584|63.2164449818621
0|1|93.5503141415182|39.2363011519124
T|2|0
T|3|2
1|1|58.0224533937584|63.5219245210971
0|1|93.1795241528479|39.2363011519124
T|4|3
2|0|20.3934493768668|41.0691783873225
1|1|58.0224533937584|63.8274040603322
0|-1|92.8087341641776|39.2363011519124
```

Figure 34 Dataset example

The dataset header consists out of eight parameters that are stored on the first line, prefixed by the letter 'H'. The parameters are defined as follows:

1. *Name of the traffic simulation map file on which the simulation was run*
2. *Number of ticks present in the file*
3. *Maximum number of nodes active at any given time during the entire simulation*
4. *Largest value for the x-axis as found in the traffic map file on which the simulation was run*
5. *Largest value for the y-axis as found in the traffic map file*
6. *Smallest value for the x-axis as found in the traffic map file*
7. *Smallest value for the y-axis as found in the traffic map file*
8. *Number of meters that one traffic map file unit represents*

Lines prefixed with the letter 'T' represent ticks. These ticks represent the time when the traffic simulation program started a new rendering sequence and updated the coordinates for the different nodes. A tick consists of two parameters:

1. *Current tick number, starting from zero*
2. *Number of node mutations present in this tick*

Data stored in the dataset (lines below a tick line) consists out of four parameters and is not prefixed.

1. *A node's unique number*
2. *Operation that needs to be carried out on the node*
3. *New coordinate for the x-axis*
4. *New coordinate for the y-axis*

There are three different types of node operations:

0	Indicates that the node is new and needs to be created in memory before processing coordinate information
1	Indicates that only the new coordinates need to be processed by the simulation value
-1	Indicates that the node trajectory is completed and that it can be removed from memory

Figure 35 Node operations

We conclude this chapter by supplying the global AHS class diagram. For more detailed information on AHS see [Boe04].

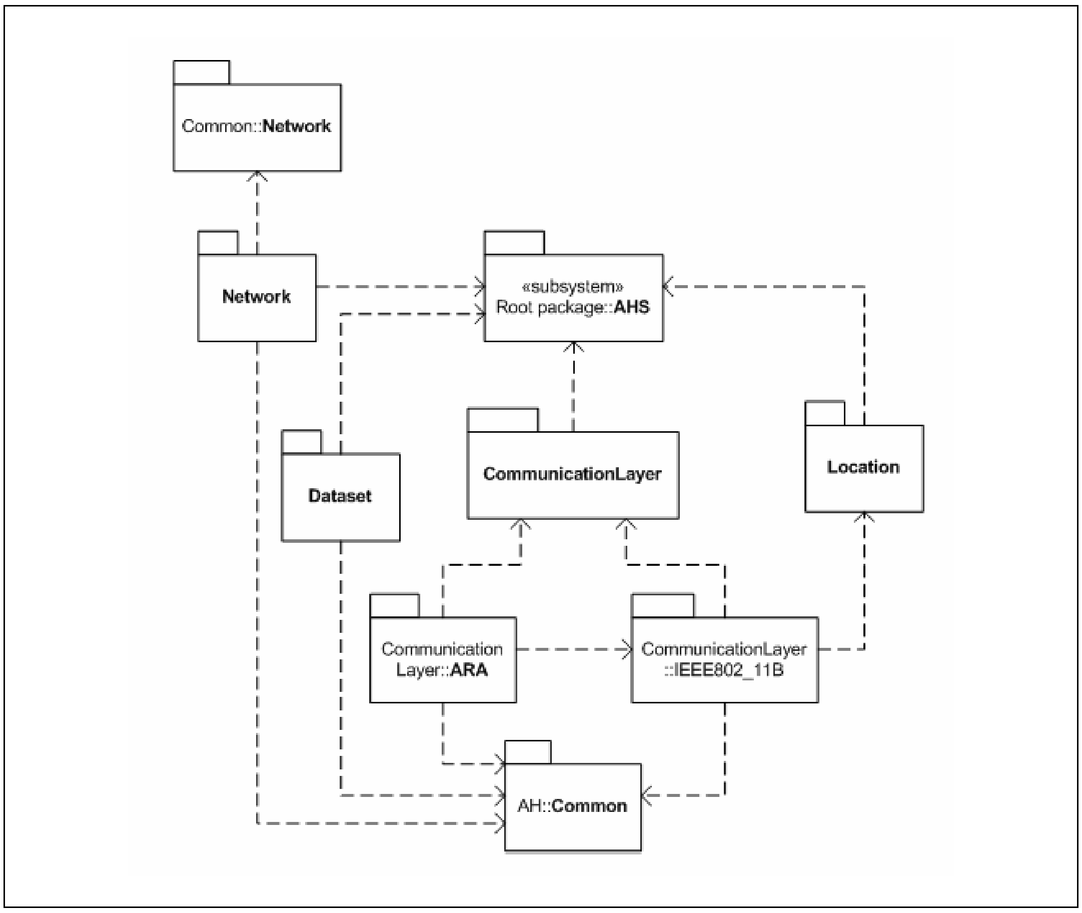


Figure 36 AHS class diagram

5.1.1.1 AHS modifications

Initially there were many problems and bugs in the AHS source-code which prevented us from building our ManetLoc system on top of it. Only the most important problems we encountered are mentioned. Please note though that AHS is currently still not free from mugs and don't recommend using it any future system.

First of all there was as a code page problem, which prevented the system to run at all on our machine, as input data would not be interpreted correctly. To fix this, changes were made to make AHS code page independent. Furthermore, there were many problems in the network simulation, which in most of the cases caused the system to hang when actually trying to transfer data between nodes. Another encountered problem relates to the multi-treading nature of the application. As these threads were not correctly implemented, often issues such as deadlocks would occur. As it was not feasible to change the entire structure of AHS only changes were made to the parts of the code critical to our requirements. Apart from the many bug fixes, much was removed from AHS. Whether it was because the functionality was not completely implemented, did not work at all (sending files for example), it was in the way or our goals (the 'tick system') or because it was replaced by something better (GUI changes).

As mentioned above the dataset format was modified, leaving only header information. As most header parameters were unnecessary, they were removed, leaving only two parameters [Figure 37]:

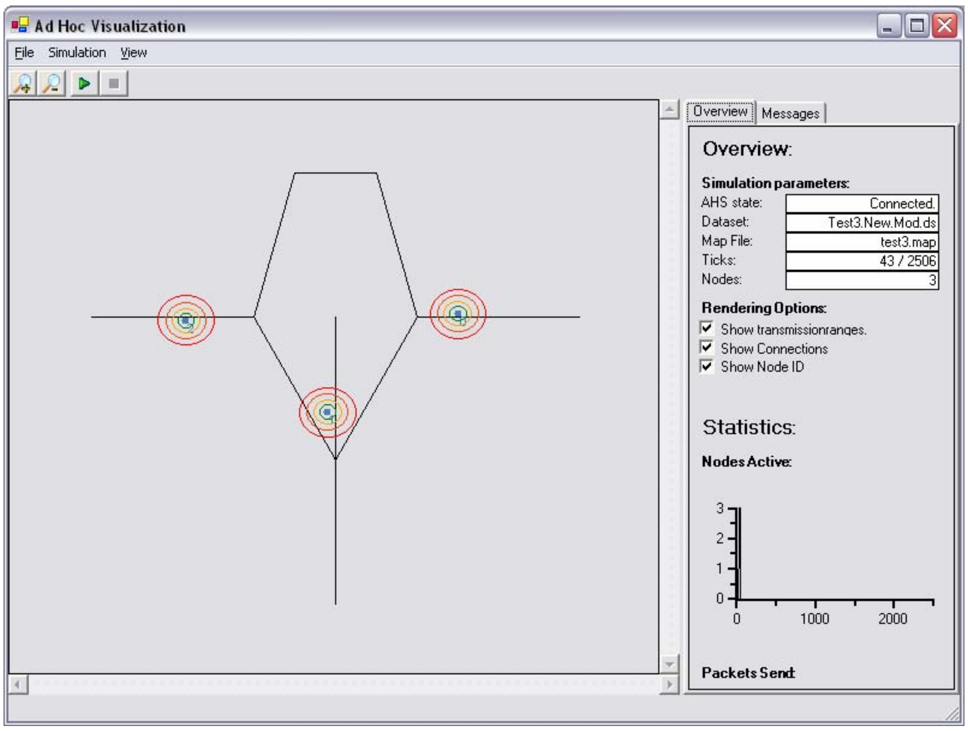
- *Name of the traffic simulation map file on which the simulation was run*
- *Number of meters that one traffic map file unit represents*

```
H|test3.map|80
```

Figure 37 Modified dataset example

5.1.2 AHV

The Ad-hoc Visualization environment [Figure 38] merely visualizes the process of AHS by actually running the simulation on its own. AHV uses a network connection to stay synchronized with AHS.



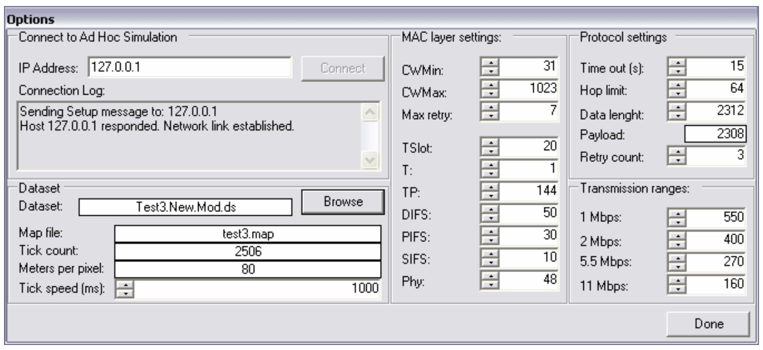
Simulation parameters:

AHS state:	Connected.
Dataset:	Test3.New.Mod.ds
Map File:	test3.map
Ticks:	43 / 2506
Nodes:	3

- ✓ Show transmissionranges.
- ✓ Show Connections
- ✓ Show Node ID

Figure 38 AHV main interface

AHV has an options dialog [Figure 39] similar to AHS allowing loading a dataset, modifying network settings and connecting to AHS.



Connect to Ad Hoc Simulation

IP Address: 127.0.0.1

Connection Log:

```
Sending Setup message to: 127.0.0.1
Host 127.0.0.1 responded. Network link established.
```

Dataset

Dataset:	Test3.New.Mod.ds
Map file:	test3.map
Tick count:	2506
Meters per pixel:	80
Tick speed (ms):	1000

MAC layer settings:

CWMin:	31
CWMax:	1023
Max retry:	7
TSlot:	20
T:	1
TP:	144
DIFS:	50
PIFS:	30
SIFS:	10
Phy:	48

Protocol settings

Time out (s):	15
Hop limit:	64
Data lenght:	2312
Payload:	2308
Retry count:	3

Transmission ranges:

1 Mbps:	550
2 Mbps:	400
5.5 Mbps:	270
11 Mbps:	160

Figure 39 AHV options dialog

The AHV class diagram is specified as follows:

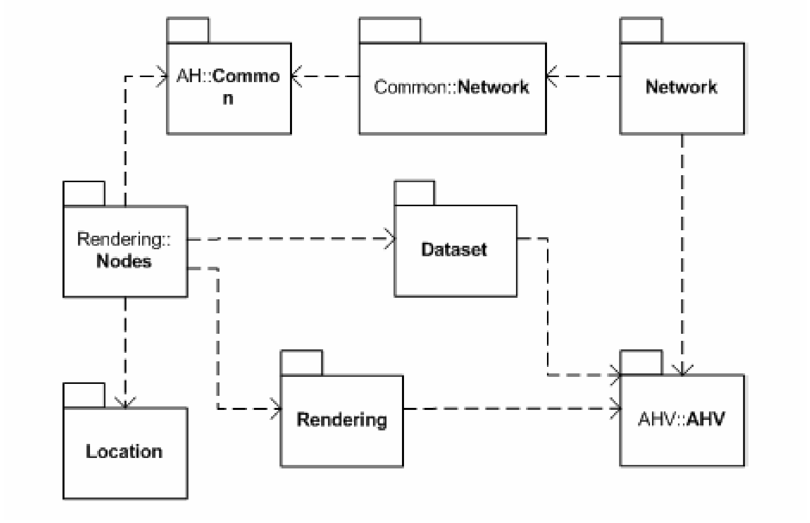


Figure 40 AHV class diagram

For detailed information on AHV we refer to [Boe04].

5.1.2.1 AHV modifications

As AHV actually is almost an exact copy of AHS, but with a visualization part added, many of the same problems that were encountered with AHS were also found and fixed in AHV. A main AHV specific issue was getting the statistic graphs to run, which requires specific libraries to be loaded (not mentioned anywhere in documentation), available (as shareware) by installing the MS Visual Basic .NET Resource Kit [Vbre]. Other notable changes in AHV are the removal of the option to start the simulation from the visualization, removal of the statistical charts and flipping the coordinate system of the drawing in such a way that the origin of the coordinate system is in the lower left corner of the screen. The main change in the AHV code base is the removal of AHS functionality, leaving only a highly modified visualization program.

5.1.3 CityNetwork map files

CityNetwork map files are the files defining the worlds used in our application. Maps in the CityNetwork format consist of roads and intersections and can be described as in [Figure 41]:

```
// First enumerate the intersections
// Then connect the intersections with roads
intersections
{       //{number, x-pos, y-pos}
            {1, 50, 0}
            {2, 100, 50}
            {3, 50, 100}
            {4, 0, 50}
}

roads
{       //{number, from, to}
            {1, 1, 2}
            {2, 2, 3}
            {3, 3, 4}
            {4, 4, 1}
            {5, 1, 4}
}
```

Figure 41 CityNetwork map, describing a square

The intersections segment provides the coordinates of any intersection in the topological map and the roads segment connects the intersections together. Please note that roads are directional. In our example [4.4.1] indicates a road from intersection 4 to intersection 1 and [5.1.4] a road from intersection 1 to intersection 4.

5.1.4 Network Generator

To be able to generate city network maps, a program called the network generator was created. The user can create the graph by placing place the nodes on desired position. We have used the network for this exact task but had to modify it slightly to suit our needs.

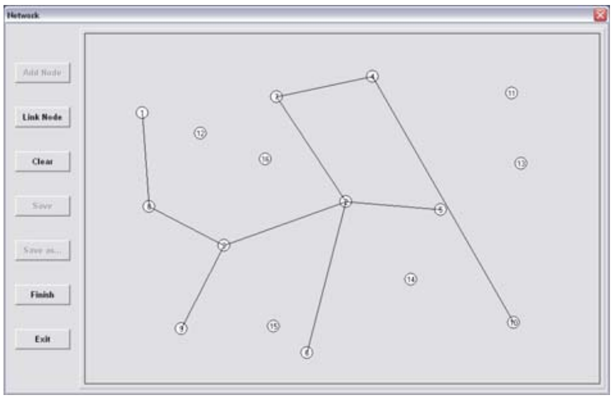


Figure 42 Network Generator

5.1.4.1 Network Generator Modifications

As our system requires rectilinear graphs, the program has been modified in such a way that it is easier to create that type of maps [Figure 44]. The most important change is the modification of the sendnode procedure, which is used when 'Link Node' is active to connect two nodes together.

The modification compares the absolute x and y differences of the two nodes. Then the smallest difference is reduced to zero by moving the x or y coordinate of node 2 to that of node 1. Another change is that it was made possible to delete nodes.

```
procedure TNetwork.sendnode(Button: TMouseButton; xx, yy: Integer);
var
 index: Integer;
begin
 if button_link.Enabled = false then
  if button = mbleft then
  begin
    for index:=1 to node_id do
    if (Sqr(xx-orig_x[index]) + Sqr(yy-orig_y[index])) <= Sqr(10) then
    begin
      nodeNumber := nodeNumber + 1;
      if nodeNumber = 1 then
      begin
       node1 := index;
      end;
      if nodeNumber = 2 then
      begin
       node2 := index;
       if Abs(orig_x[node2] - orig_x[node1]) < Abs(orig_y[node2] - orig_y[node1]) then
       begin
        orig_x[node2] := orig_x[node1];
       end else
       begin
        orig_y[node2] := orig_y[node1];
       end;
       connected[node1,node2] := 1;
       connected[node2,node1] := 1;
       paintbox1.Repaint;
       nodeNumber := 0;
       node1 := 0;
       node2 := 0;
      end;
    end;
  end;
  paintbox1.Repaint;
```

Figure 43 Modified sendnode for create rectilinear networks

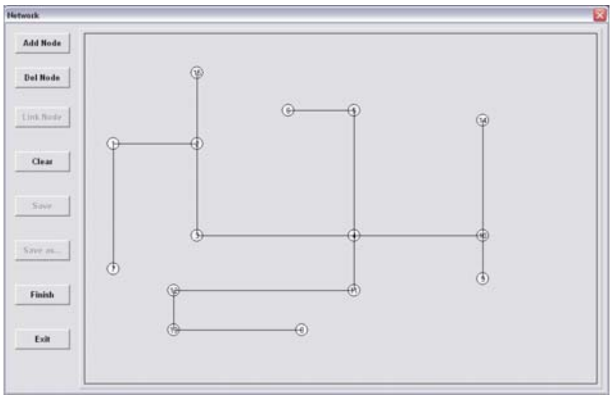


Figure 44 Modified network generator for rectilinear worlds

5.2 ManetLoc

ManetLoc serves as the proof of concept for the proposed architecture. The system is implemented as integration into the AHS and AHV code bases and fully implements the ideas stated in chapters 3 and 4.

5.2.1 Class Diagram

As ManetLoc was implemented as an integrated module into AHS and AHS much of its class structure was reused. Even though the original class structure is a 'mess' to be honest we were able to hook our program in. It seems irrelevant to explain the complete class diagram; instead a global class diagram is provided showing how ManetLoc uses parts of AHS by connecting to the communication layer. Using network socket communication ManetLoc is connected to the modified visualization program, allowing it to run the on a different machine.

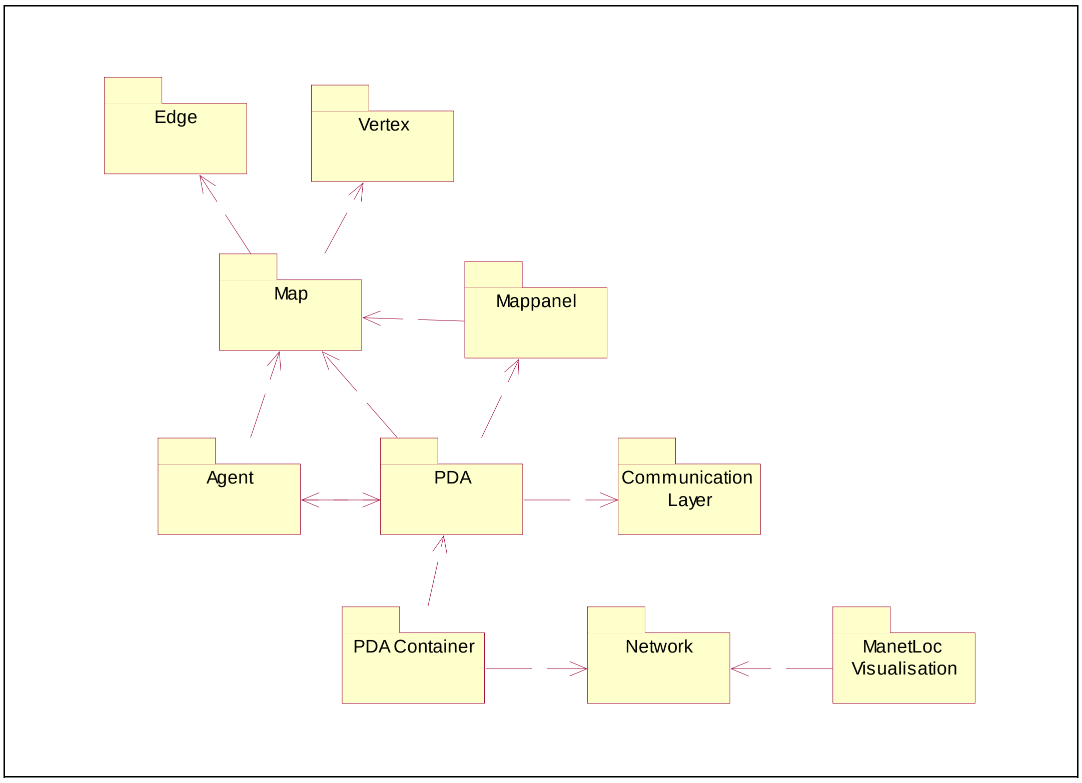


Figure 45 ManetLoc Class diagram

As one can see, the PDAContainer contains PDA's. PDA's are running agent software. And agent uses the PDA's communication layer to exchange map data (consisting of edges and vertexes) with other agents. The agent also supplies the PDA with a view of the world, which is visualized in the MapPanel.

5.2.2 Use Case Diagram

The use case diagram of [Figure 46] describes how a user can use the ManetLoc software. Users can control the simulation, which they first have to supply with a world map. After loading the map it is possible to start the simulation. When the simulation has initialized one can add, remove and control PDA's. The ManetLoc system also contains agents (running on the simulated PDA's) which possibilities are described by [Figure 47]. This use case diagram shows the options an agent has. It can decide to share data about the world with other agents in the mobile ad-hoc network. If new data is received, it can be combined with what already is knows. A user wanting advice (or the agent itself) can query the agent. If possible the agent supplies an answer based on what it currently knows about the state of the world.

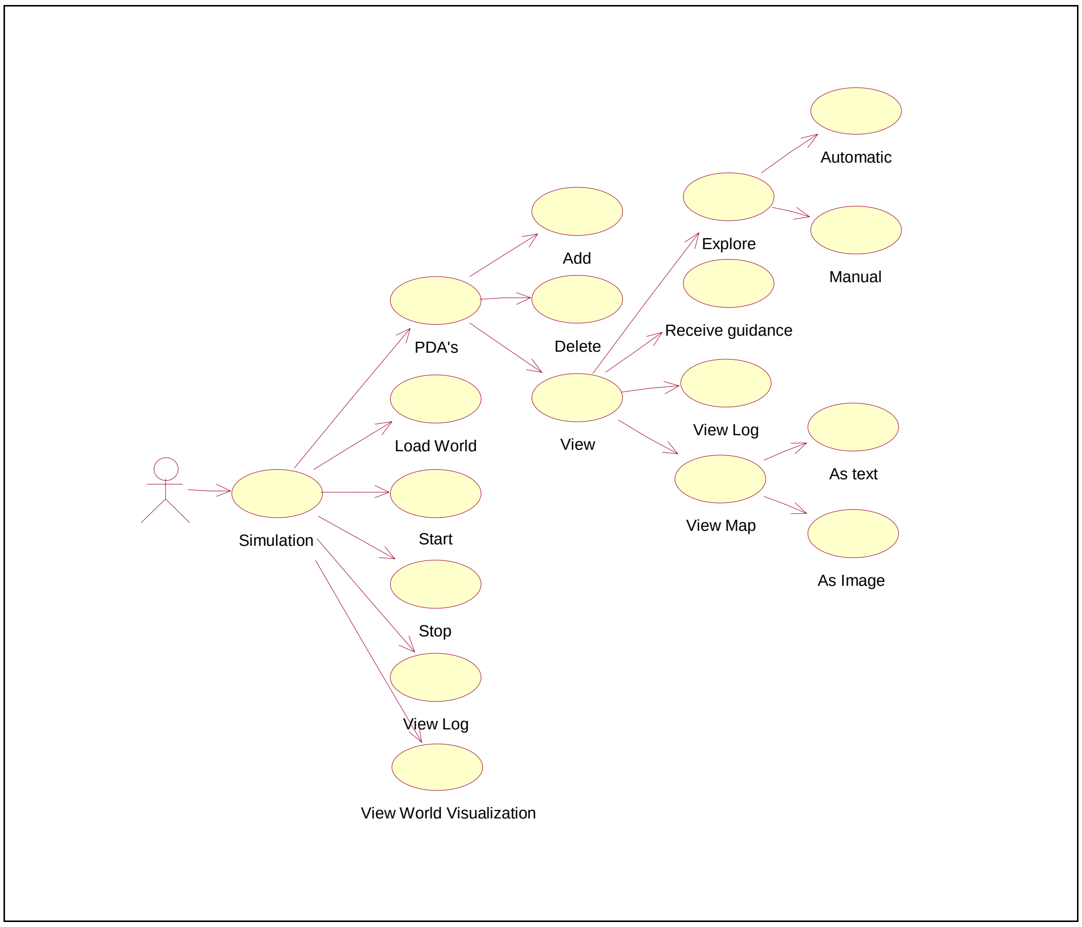


Figure 46 ManetLoc use case diagram for user

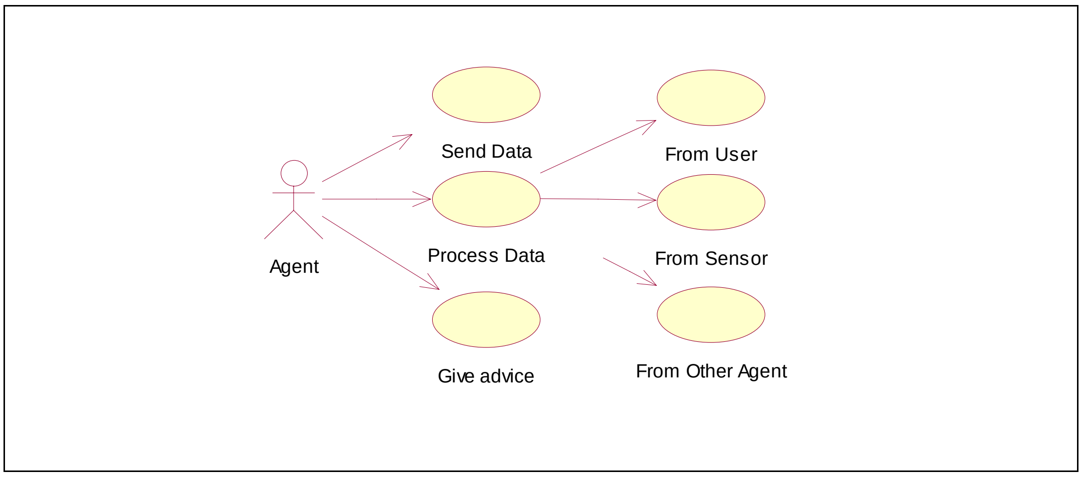


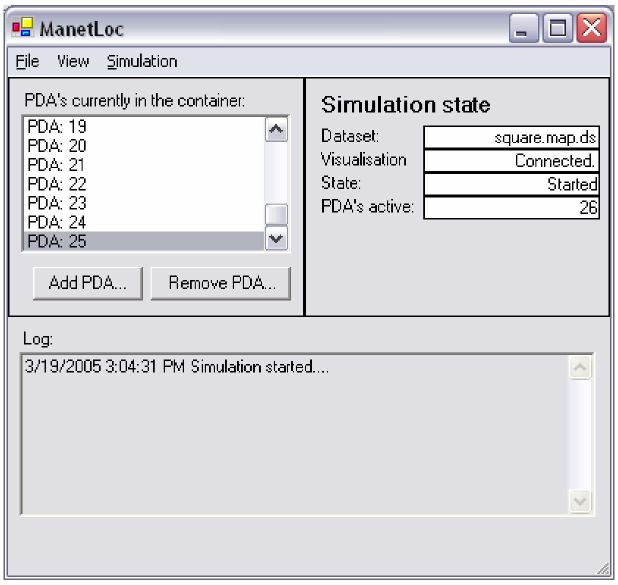
Figure 47 ManetLoc use case diagram for agent

5.2.3 Implemented GUI

The graphical user interface of ManetLoc has been intentionally been kept very basic, and close to that of the original AHS system. This makes it easier to port to different systems (possibly real PDA's) and easier to explain. The screens discussed, are: m*ain interface, options dialog, PDA world view and world visualization*.

Main interface

Like AHS the main interface [Figure 48] consists of a left side where all the PDA's currently in the simulation can be selected. In ManetLoc, only minimal dataset's are used, which don't provide simulation data like in AHS, but only the header data. These datasets mainly tells the program which map to load and the relation between map distance units and actual metric distances.



Dataset: square.map.ds
Visualisation: Connected.
State: Started
PDA's active: 26

```
3/19/2005 3:04:31 PM Simulation started....
```

Figure 48 ManetLoc main-interface

Instead of loading from pre-simulated data, PDA's can be added and removed manually from the main interface. In a real world this would be analogous to turning on a PDA. Double clicking on a PDA in the list opens up its world-view.

On the right side is the Simulation state, which shows:

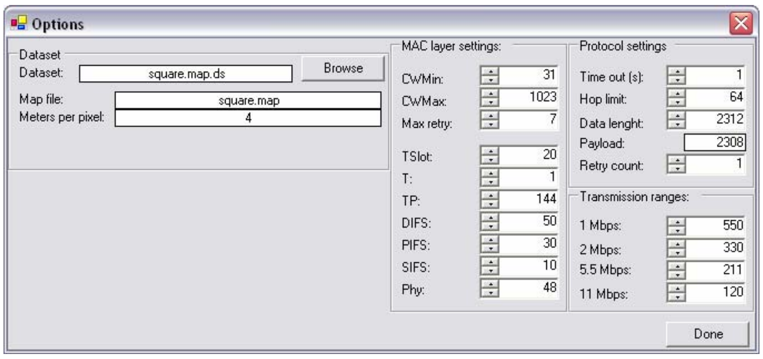
- The dataset loaded
- Whether the visualization program is connected.
- Whether the simulation is active or not
- The number of PDA's currently active.

There are three menus in this screen: *file, view* and *simulation:*

- File: allows exiting the program,
- View: allows showing or hiding the log
- Simulation: Allow setting simulation options and state (start, stop)

Options dialog

The options dialog is also very similar to that of AHS. It allows setting network parameters and supplying the dataset that has to be loaded [Figure 49].



Options

Dataset
Dataset: square.map.ds
Map file: square.map
Meters per pixel: 4

MAC layer settings:
CWMin: 31
CWMax: 1023
Max retry: 7
TSlot: 20
T: 1
TP: 144
DIFS: 50
PIFS: 30
SIFS: 10
Phy: 48

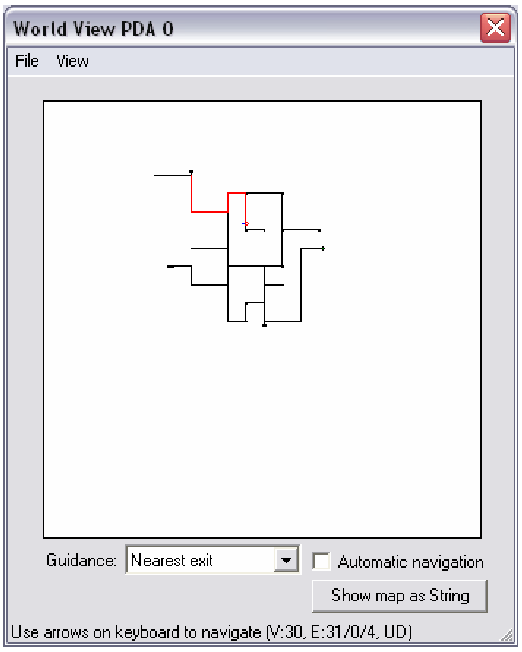
Protocol settings
Time out (s): 1
Hop limit: 64
Data lenght: 2312
Payload: 2308
Retry count: 1

Transmission ranges:
1 Mbps: 550
2 Mbps: 330
5.5 Mbps: 211
11 Mbps: 120

Figure 49 ManetLoc options dialog

PDA world-view

All PDA's are separately controllable, but also can be set to automatic exploration using the world-view screen. An instance of this screen represents a PDA. Using arrows on keyboard a user can navigate through the world. All that is known by the agent on this virtual PDA is visualized at the same time. Using the guidance combo box a user can ask advice from the agent and if set to automatic navigation, this advice is always followed if possible. Colors in the map give supply information to the user [Figure 50].



World View PDA 0

Guidance: Nearest exit

Automatic navigation [ ]

Use arrows on keyboard to navigate (V:30, E:31/0/4, UD)

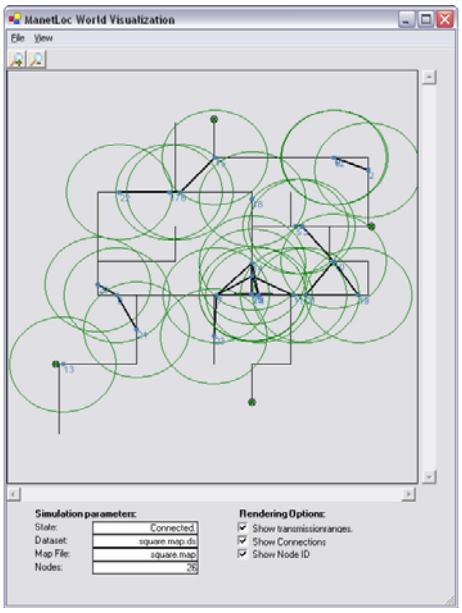
Figure 50 PDA world view dialog

- Black dots indicate intersections
- Green dots indicate exits
- Lines indicate edges.
- Red lines indicate that they are part of the path to the nearest exit
- Green lines indicate that they are part of the path to the nearest unexplored area
- Blue lines and dots indicate that data about this data was supplied by another agent
- Dotted lines indicate that this edge has not been visited before

Note: by selecting log from the view menu one can see a log of events since the birth of this PDA concerning itself.

World visualization

ManetLoc World Visualization merely visualizes the map provided by the dataset and at the same time shows the location of all of the simulated PDA's in currently active. Green circles indicate the WiFi communication radius of these nodes. And bold lines if two nodes are able to communicate.



Simulation parameters:
State: Connected.
Dataset: square.map.ds
Map File: square.map
Nodes: 26

Rendering Options:
[✓] Show transmissionranges.
[✓] Show Connections
[✓] Show Node ID

Figure 51 ManetLoc World Visualization

5.2.4 Implemented Algorithms

The basics of implemented algorithms have been explained into some extent in chapter 4, which allows us to refer to them and explain how these were actually implemented and used in the next few pages.

5.2.4.1 Exploration

The parts of ManetLoc that was implemented first, were the elements that allow a user to navigate trough a world defined by the loaded CityNetwork-based map.

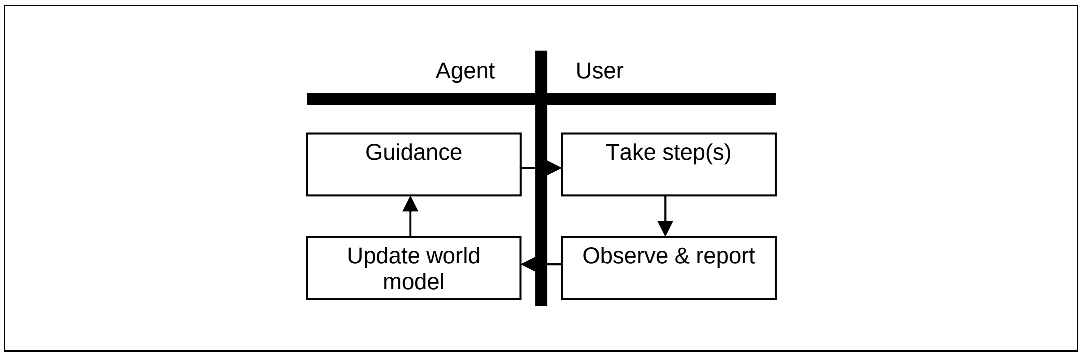


Figure 52 Overview of exploration program loop

Go direction

Go direction is a group of complex bookkeeping procedures, which keeps track of the user's location in the simulated global map and in the user's local map. If allowed it first performs an action step in the world updates the global orientation and location of the user. Then if successful the local location, orientation map of the world is updated. Which means, updating edge lengths, vertex locations, checking for inconsistencies, closed loops, etcetera. If inconsistencies are found they are restored. If closed loops are found the vertexes and edges should be connected correctly.

Nearest Exit

'Nearest Exit' is a group of iterative procedures, which perform a search for the nearest exit from the user's location and report back the path to this near exit if one exists. The search algorithm used is based on Dijkstra's shortest path algorithm.

Nearest Unexplored

Similar to 'Nearest Exit', 'Nearest Unexplored' is a group of Dijkstra-based iterative procedures, performing a search for the nearest unexplored area from the users location and report back the path to this area.

Automatic Exploration

Automatic exploration was implemented to simulate a user making exploration decisions. This is because - as we want to simulate an environment with many PDA's - it would be impossible to control the entire simulated group of PDA's manually. If automatic exploration is on – by default it is – a timer will trigger a procedure that makes the decision what movement to make. This also depends on the guidance choice that is been selected:

- *None* – meaning random movement
- *Local* – meaning that the agent make a 'smart' decision based only on what the user can see at its current location
- *Nearest exit* – the agent uses the nearest exit calculation to make its decisions
- *Nearest unexplored* – the agent uses the nearest unexplored calculation to make its decision

5.2.4.2 Map distribution, matching and merging

The second part of ManetLoc mainly concerns sharing and using map data received from other agents, were the elements which allow a user to navigate trough a world defined by the loaded CityNetwork-based map. The three main steps in this process are distribution, matching and merging of maps.

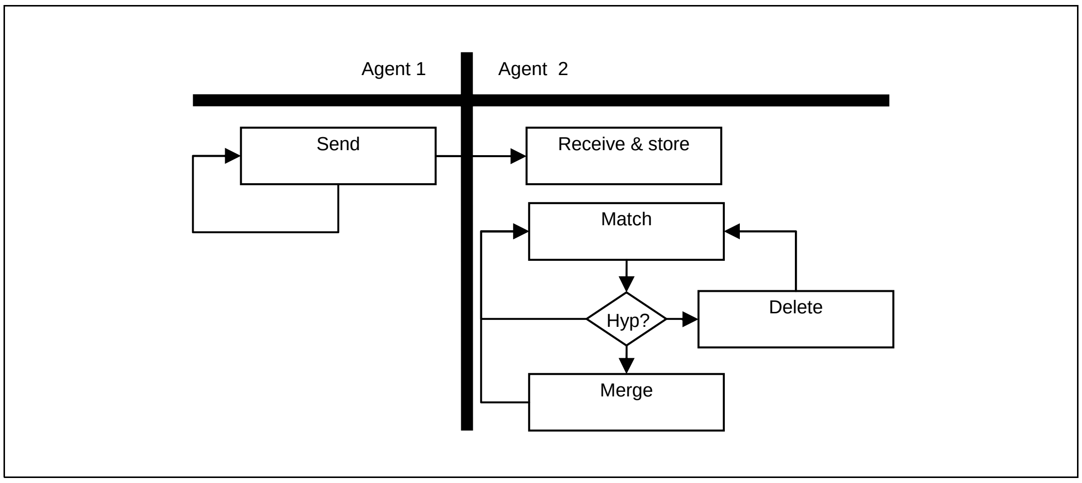


Figure 53 Overview of distribution, matching and merging program loop

Distribution

The group of procedures concerned with distribution of maps to other agents, first checks if it useful to send this data (for they may not be enough data or the map is currently inconsistent). Then it checks if there is anyone to send the data to in its range. If so, the map is converted to a string that the agent(s) on the other end can interpret. Then finally it is attempted to send the map string [Figure 49]. An agent that receives a map string from another agent converts it back to a map and stores it in a queue and processes the matching algorithm on it when requested, which is handled by a timer. If more than one map from the same agent is received, the older one will be discarded.

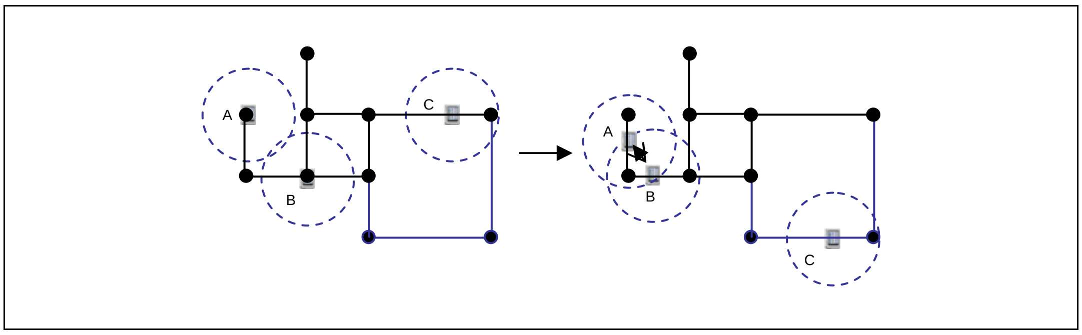


Figure 54 Map distribution

Matching

Matching maps - described in [4.4.1 Matching] - also involves a group op (partially) iterative procedures and is performed when there are one or more map from other agents are available and we don't already have the complete map. The newest map is always tried first, and is discarded after 3 attempts.

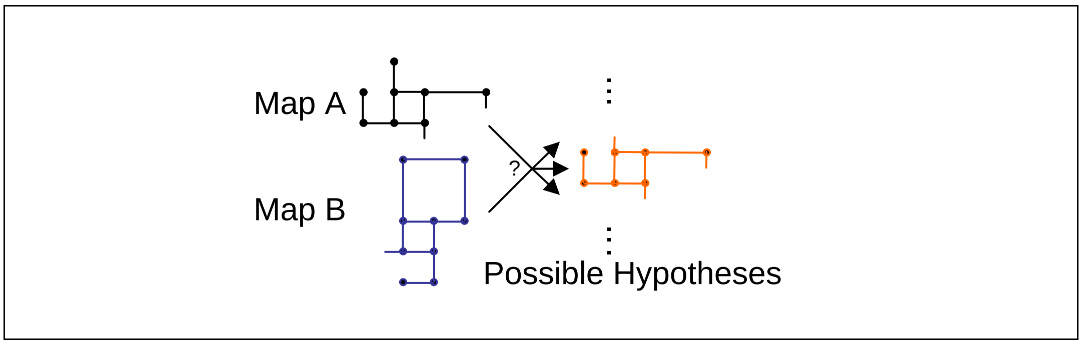


Figure 55 Matching process

Merging

Immediately after a matching process succeeded - resulting a list of hypotheses - a merge attempt [4.4.2 Merging] is executed with the best hypothesis. The process of selecting the best hypothesis globally looks at their size, but also how well the edge lengths match. After selecting the best hypothesis, the map is rotated, shifted and integrated in the original map. A record of 'changes made' is kept so it will be possible to restore from this process if the hypothesis later appears to be false.

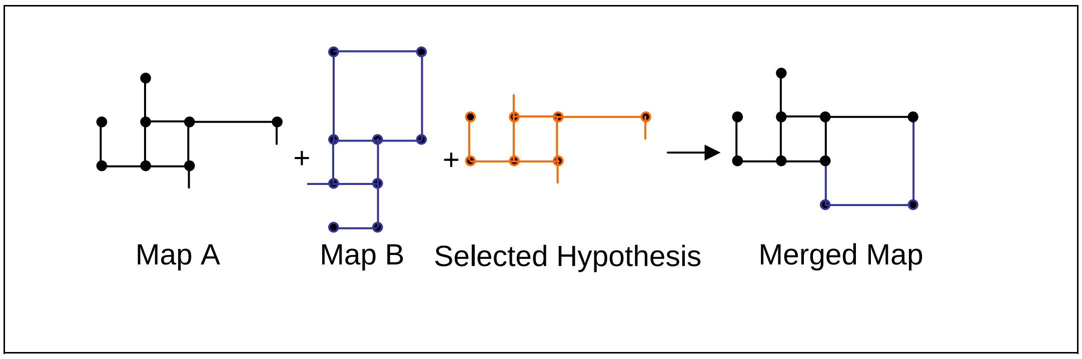


Figure 56 Merging process

5.2.4.3 Restoring from and preventing errors

As it is crucial a local map is consistent, this is checked each time a change is made. This means: when a new observation is made [Figure 57], when a loop hypothesis was accepted and closed and each time two maps have been merged. Also to prevent errors, this is checked again before sending any data to other agents.

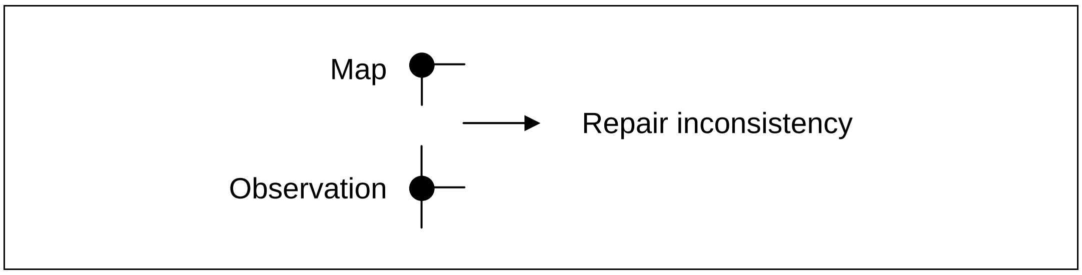


Figure 57 Inconsistency

Merging two maps can lead to inconsistencies, as the selected hypothesis might not have been correct or the map received wasn't entirely correct. Encountering such an inconstancy will in most cases only lead to merely discarding the changes made for this hypothesis which have not been verified by local observations or other agents thus far.

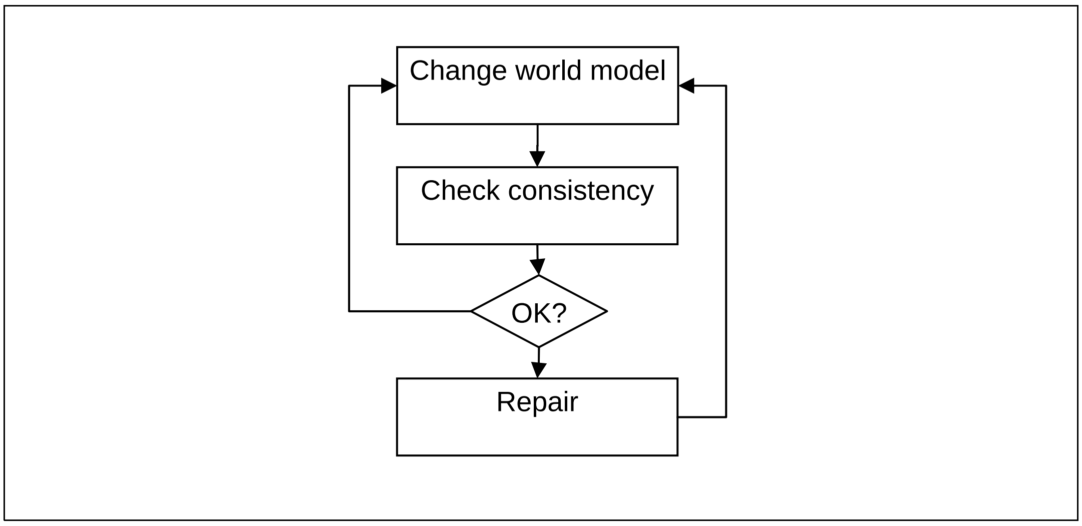


Figure 58 Overview of consistency program loop

Chapter 6: System Tests

Even though our system is a simulation and no one is likely to depend on it in a real life situation, testing a system is always important. It is very important to test if the system GUI is useable, if the simulation works correct and if it lives up to the goals we set beforehand. This chapter will discuss the tests the system has undergone, after it was marked final. Only if critical bugs are found they will be fixed, otherwise they are merely listed as future work. Globally the *usability*, *correctness*, *completeness and performance* of the system will be tested.

For performing the tests 3 world maps have been designed, one with only 10 intersections and one with 30 intersections. Also limited – due to hardware constraints - testing was performed with a 100 intersections world. This setup allows us to analyze the systems performance in small and in a larger world and see how it scales up. Needless to say is that during system implementation ManetLoc was used with many other maps of wide variety.

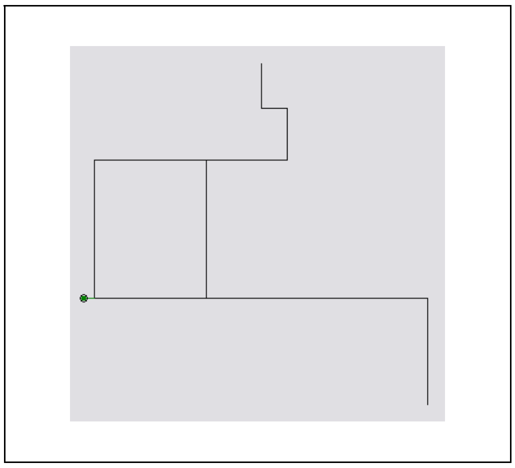


Figure 59 The 10 intersections world

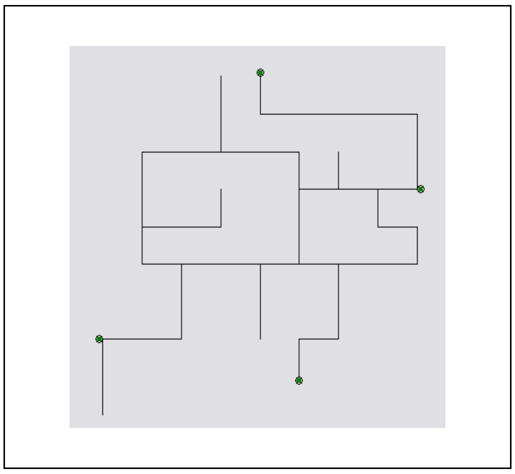


Figure 60 The 30 intersections world

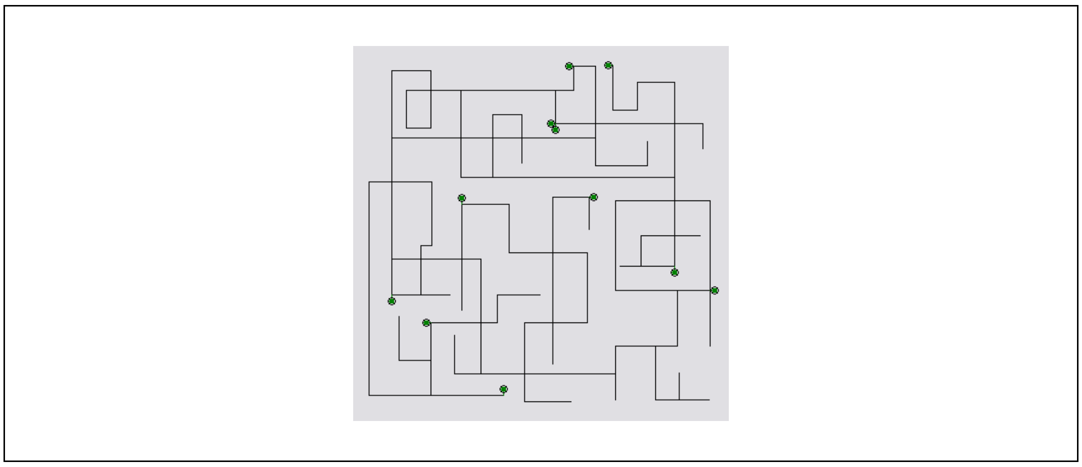


Figure 61 The 100 intersections world

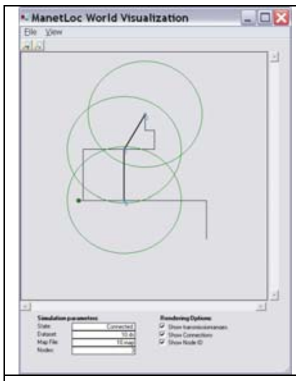
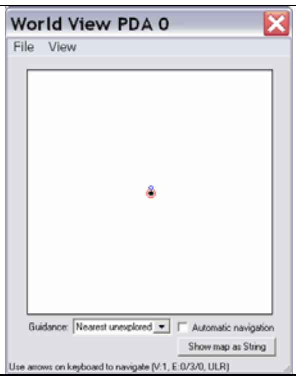
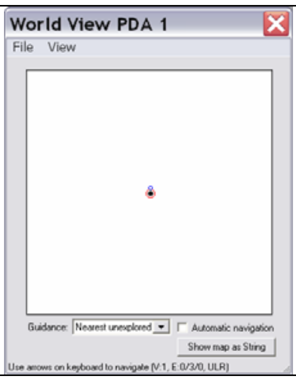
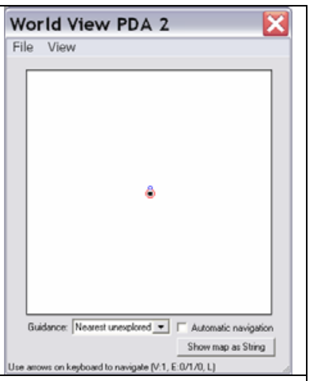
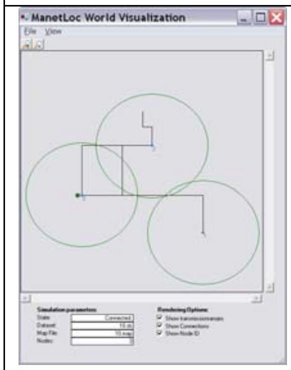
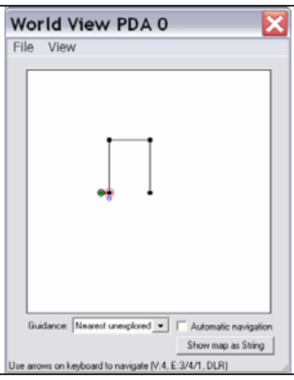
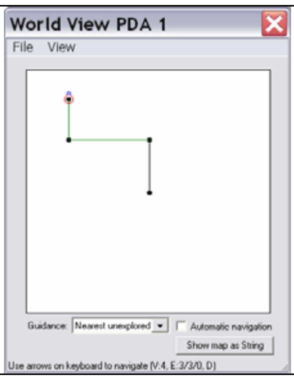
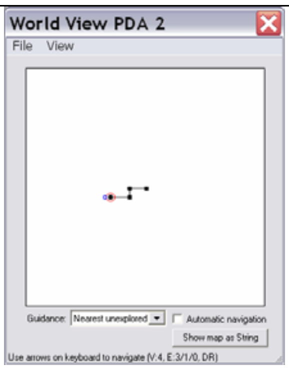
The definitions of these maps can be found in [Appendix B]. An important detail is though, the virtual sizes of the entire path length of these three maps [Figure 55]. The number of meters per unit is important for the simulated ad-hoc wireless network, as communication over such networks is bound to certain range limits.

Map	# units	# meters per unit
10.map	948	4
30.map	4800	4
100.map	9180	4

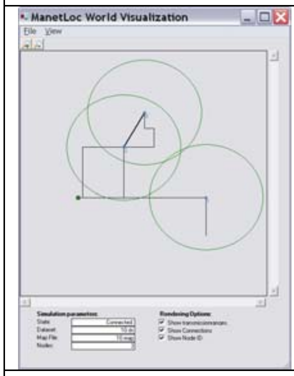
Figure 62 Map total path length sizes

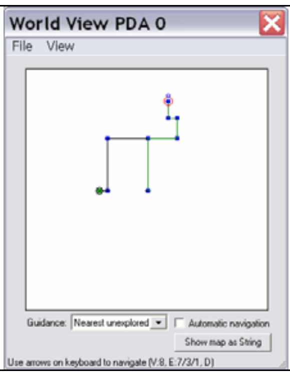
Because of time the fact that larger maps take longer to test, most test were performed on the 10 intersections map. During test with very large maps (100 or more) and many agents (5 or more) it was found that depending on the hardware configuration used there are limits to the size of the simulation one can run on a pc.

As a first test and demonstration of the system a test run on '10.map' with 3 agents is visualized below.

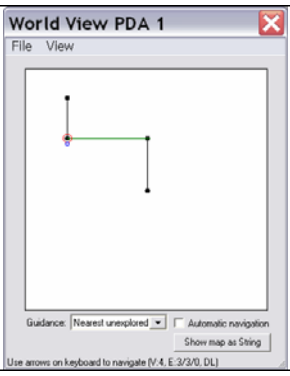
	 Guidance: Nearest unexplored; Automatic navigation: off. Use arrows on keyboard to navigate (V:1, E:0/3/0, ULR)	 Guidance: Nearest unexplored; Automatic navigation: off. Use arrows on keyboard to navigate (V:1, E:0/3/0, ULR)	 Guidance: Nearest unexplored; Automatic navigation: off. Use arrows on keyboard to navigate (V:1, E:0/1/0, L)
Initial setup showing the stating locations of the three agents. Each agent has its own local orientation visualized as up in the world view. All agents are connected in this setup and thus are able to communicate. They have nothing to talk about though as their only initial world knowledge is that of the intersection they started at.			
	 Guidance: Nearest unexplored; Automatic navigation: off. Use arrows on keyboard to navigate (V:4, E:3/4/1, DLR)	 Guidance: Nearest unexplored; Automatic navigation: off. Use arrows on keyboard to navigate (V:4, E:3/3/0, D)	 Guidance: Nearest unexplored; Automatic navigation: off. Use arrows on keyboard to navigate (V:4, E:3/1/0, DR)
After some time the users have explored parts of the world. The user of PDA 1 encountered a dead end. As his guidance combobox is set to 'Nearest unexplored' the green line indicates the path to the nearest			

unexplored area. In this setup none of the agents are able to communicate as they are not close enough together.

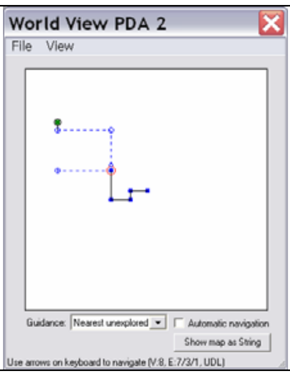




Guidance: Nearest unexplored
Use arrows on keyboard to navigate (V:8, E:7/3/1, D)

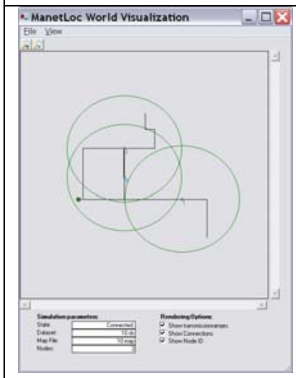


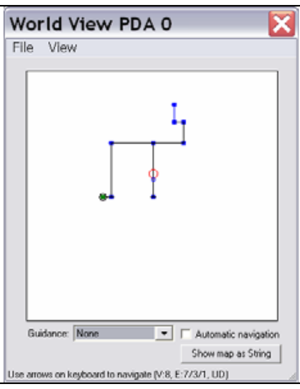
Guidance: Nearest unexplored
Use arrows on keyboard to navigate (V:4, E:3/3/0, DL)



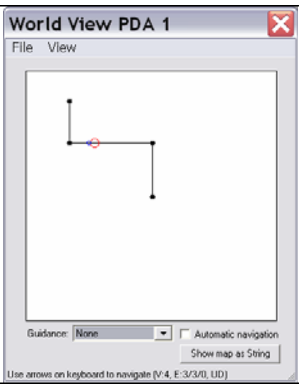
Guidance: Nearest unexplored
Use arrows on keyboard to navigate (V:8, E:7/3/1, UDL)

User 0 and 2 move into each others range (and pre-explored area), and are both able to match and merge their maps. User 0 also encountered a dead-end and now also receives guidance from the system. User is moving towards the unexplored area.

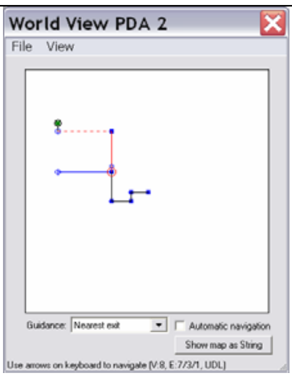




Guidance: None
Use arrows on keyboard to navigate (V:8, E:7/3/1, UD)

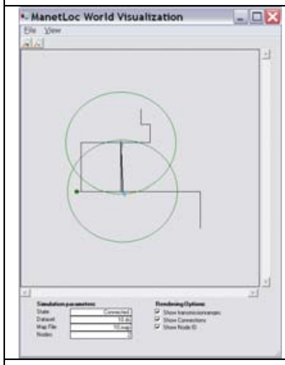


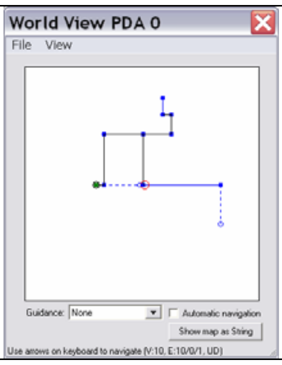
Guidance: None
Use arrows on keyboard to navigate (V:4, E:3/3/0, UD)



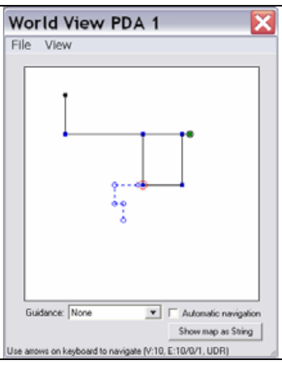
Guidance: Nearest exit
Use arrows on keyboard to navigate (V:8, E:7/3/1, UDL)

User 0 and 1 decided not to receive any guidance from now on. As user 2 is thinking about leaving the environment he set his guidance to show the path to the nearest exit, indicated by the red line.

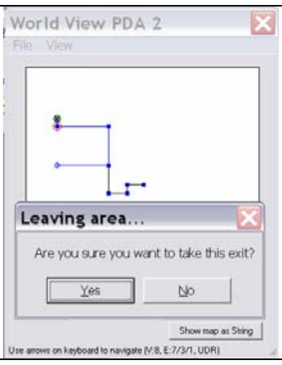




Guidance: None
Use arrows on keyboard to navigate (V:10, E:10/0/1, UD)



Guidance: None
Use arrows on keyboard to navigate (V:10, E:10/0/1, UDR)



Are you sure you want to take this exit?
Use arrows on keyboard to navigate (V:8, E:7/3/1, UDR)

User 0 and 1 explore some of the environment and together are able to find the complete map. User 2 leaves the area.

Figure 63 Test Run on 10.map

6.1 Usability

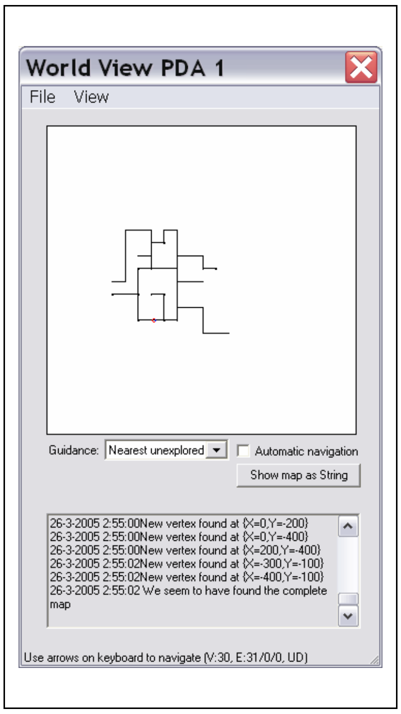
In the usability test, we look for UI bugs and performance issues. This was be done by having a number of users test the system. The test users, 5 in total, were instructed by me personally, they did not have a user manual as it wasn't finished at the time. They were first demonstrated the system usage and then asked to try for themselves, reporting anything they believed not to be correct. The problems they encountered were:

ManetLoc world visualization does not show directly when simulation has started, zoom button has to be clicked first
Some buttons are and menu enabled in certain situation when the should (Add PDA, Remove PDA, stop, start)
Simulation can become slow on very large maps
Simulation can become slow with many active agents
Simulation can become slow with many world views open
After stopping the simulation it is not possible to start it again with different options
Closing ManetLoc visualization when simulation is still running will result in errors

Figure 64 Non critical UI bugs found

6.2 Correctness

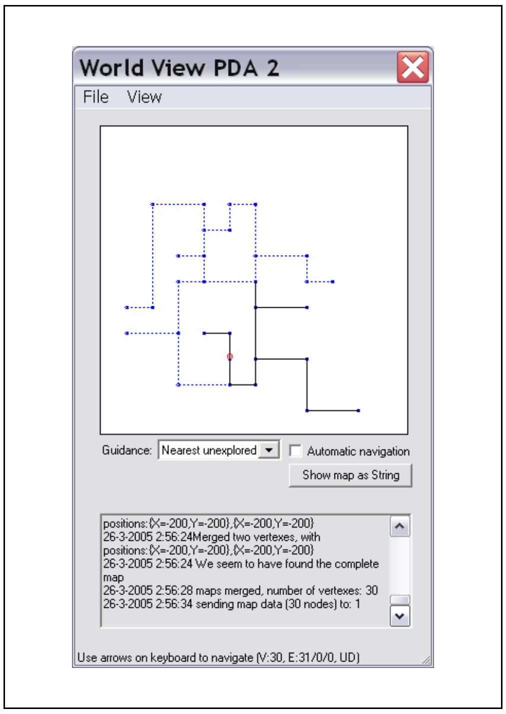
For this test we will test if the system processes data correctly and eventually comes with the correct output. Many single and multi-agent runs were performed on the 10, 30 and also the 100 intersections map, and in (most of the times) the correct complete map was found. In one or two case where the system load was too high and other less clear causes problems occurred with the AHS (threading) code base, resulting in problems such as deadlocks or the program crashing. We have extensively looked into the cause of this, and although we have found some hints to the issue, it seemed not feasible or required to completely solve these problems as it is still possible to do normal testing of the ManetLoc part, although it makes it more difficult to test the correctness of the system on large maps and setups with many agents



```
26-3-2005 2:55:00New vertex found at {X=0,Y=-200}
26-3-2005 2:55:00New vertex found at {X=0,Y=-400}
26-3-2005 2:55:00New vertex found at {X=200,Y=-400}
26-3-2005 2:55:02New vertex found at {X=-300,Y=-100}
26-3-2005 2:55:02New vertex found at {X=-400,Y=-100}
26-3-2005 2:55:02 We seem to have found the complete
map
```

Use arrows on keyboard to navigate (V:30, E:31/0/0, UD)

Figure 65 Map 30 found, single agent setup



```
positions:{X=-200,Y=-200},{X=-200,Y=-200}
26-3-2005 2:56:24Merged two vertexes, with
positions:{X=-200,Y=-200},{X=-200,Y=-200}
26-3-2005 2:56:24 We seem to have found the complete
map
26-3-2005 2:56:28 maps merged, number of vertexes: 30
26-3-2005 2:56:34 sending map data (30 nodes) to: 1
```

Use arrows on keyboard to navigate (V:30, E:31/0/0, UD)

Figure 66 Map 30 found , multi-agent setup

6.3 Completeness

In this test the realized system is compared to the goals defined by the problem description and any missing features are reported. There are five distinct main goals defined in our problem description:

Design and implement a (1) ***multi-agent-system*** *that can* (2) ***operate in environments without a pre-setup infrastructure (only a mobile adhoc network)*** *and* (3) ***without any pre-knowledge of the world****, which is able to* (4) ***process*** *and* (5) ***fuse location information from different users and sensors remote in space and time*** *and* (6) ***distributes location information and location based services (such as guidance) to its users.***

1. Multi-agent
As each simulated PDA runs its personal semi-intelligent agent, which exchanges knowledge running on other, simulated PDA's and can thus be considered a multi-agent system. No standard agent platform is used for implementing this though and can be considered as future work when implemented in an actual system instead of a simulation.

2. Operate in environments without a pre-setup infrastructure
This goal is not entirely met, as the environments in which the agents run are merely a simulation of the envisioned infrastructureless environment. Put that aside, it seems to be a realistic assumption that this setup would also work in a similar non-

simulated environment without having to change too much in the design and implementation.

3. Operate without any pre-knowledge of the world
As each agent does not have any preset knowledge of the world and is not able to cheat in any way this goal is met. The only input an agent in our system receives, which It can use to form an opinion about the world is that of simulated user input and other agents.

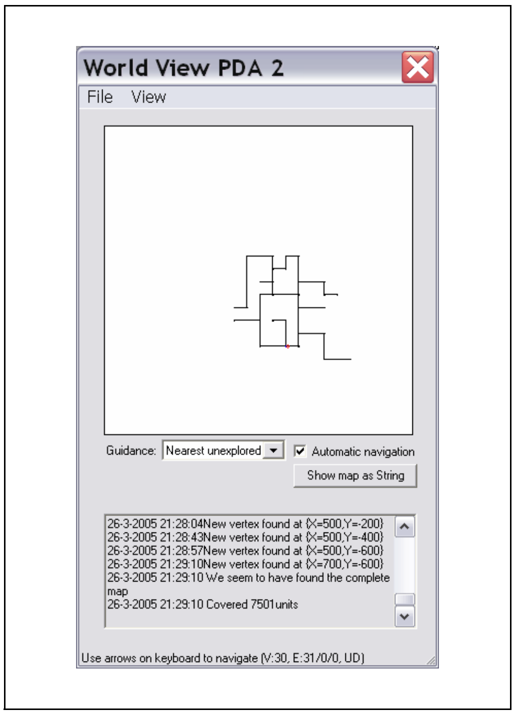
4. Process location information from user and sensory input
Each agent receives data from its (simulated) user and sensors The data received is converted in to knowledge and merged with existing knowledge. This goal is met.

5. Fuse location information from different users remote in space and time
Each agent receives data from its agents representing other users at different locations throughout time. The data received is converted into workable knowledge and merged with the agents own world model. This goal can be considered met.

6. Distribute location information and location based services
The system provides guidance in exploring the environment and in finding the nearest, so the goal can be considered met. Though many more service can be imagined which weren't implemented, including distributed planning and execution mentioned in the architecture concept. This can be considered future work.

6.4 Performance

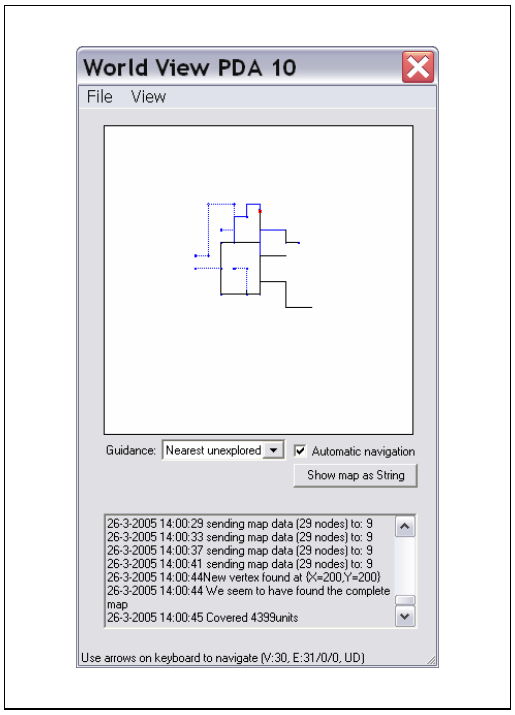
For this test the simulation was first run on the 10 and 30 nodes maps, first with just one agent exploring the world automatically using nearest unexplored area navigation, and later with more agents who were simultaneously started. The time it took an agent to find the complete map was measured and also was always checked if the output was correct (see [6.2]) The results are presented in a table [Figure 69] and a chart [Figure 70].



```
26-3-2005 21:28:04New vertex found at {X=500,Y=-200}
26-3-2005 21:28:43New vertex found at {X=500,Y=-400}
26-3-2005 21:28:57New vertex found at {X=500,Y=-600}
26-3-2005 21:29:10New vertex found at {X=700,Y=-600}
26-3-2005 21:29:10 We seem to have found the complete
map
26-3-2005 21:29:10 Covered 7501units
```

Use arrows on keyboard to navigate (V:30, E:31/0/0, UD)

Figure 67 Map 30 found with 1 agent, after 7501 units



```
26-3-2005 14:00:29 sending map data (29 nodes) to: 9
26-3-2005 14:00:33 sending map data (29 nodes) to: 9
26-3-2005 14:00:37 sending map data (29 nodes) to: 9
26-3-2005 14:00:41 sending map data (29 nodes) to: 9
26-3-2005 14:00:44New vertex found at {X=200,Y=200}
26-3-2005 14:00:44 We seem to have found the complete
map
26-3-2005 14:00:45 Covered 4399units
```

Use arrows on keyboard to navigate (V:30, E:31/0/0, UD)

Figure 68 Map 30 found with 5 agent, after 4399 units

Map	#agents	#runs	Min units	Max units	Avg. units
10.map	1	10	1248	2351	1721
10.map	2	5	943	1248	1125
10.map	3	5	847	1248	1100
10.map	4	5	953	1219	1112
10.map	5	5	868	1176	1060
30.map	1	5	9104	11205	9701
30.map	2	5	7405	10101	8569
30.map	3	5	5046	7014	6227
30.map	4	5	4399	6412	5301
30.map	5	5	4007	8701	6017

Figure 69 Test results, finding complete map

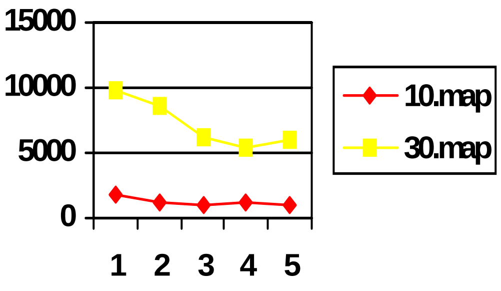
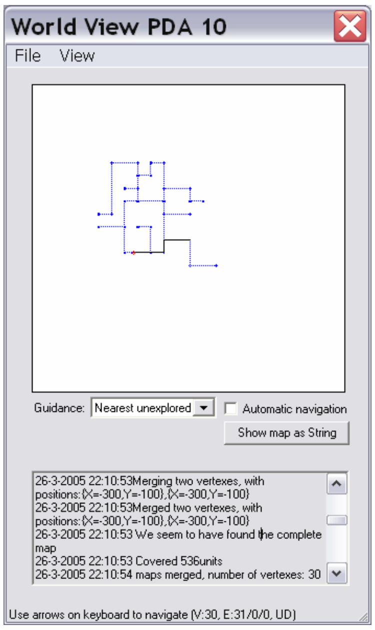


Figure 70 Test results, finding complete map

It can be clearly concluded that the process of sharing and merging maps has an effect, the larger the map the larger the effect and also the more agents the more gained. Agents starting in a map that was already explored by others logically have the most gain; after they have explored a small part of the world they can simply merge the large map parts with their own. Please note that the test result would probably be significantly lower if the AHS network code wouldn't suffer from deadlock issues.

This is illustrated in [Figure 71] with the 30 intersections map. This world was pre-explored completely by 5 agents after whom a fresh agent was added. The new agent received the complete map from multiple agents and was able to find the complete correct map within 536 distance units traveled. Considering it takes 9701 units on average for an agent to explore this world on its own this is a considerable gain (in this specific case 18 times faster).



Guidance: Nearest unexplored
Automatic navigation

```
26-3-2005 22:10:53Merging two vertexes, with
positions:{X=-300,Y=-100},{X=-300,Y=-100}
26-3-2005 22:10:53Merged two vertexes, with
positions:{X=-300,Y=-100},{X=-300,Y=-100}
26-3-2005 22:10:53 We seem to have found the complete
map
26-3-2005 22:10:53 Covered 536units
26-3-2005 22:10:54 maps merged, number of vertexes: 30
```

Use arrows on keyboard to navigate (V:30, E:31/0/0, UD)

Figure 71 Most gain when arriving in pre-explored world

Chapter 7: Conclusions and future work

In this chapter the results of the ManetLoc project are discussed. After that we will evaluate to what extend the projects goals are reached. Finally we will discuss some possibilities for further research and development of the system.

7.1 Conclusions

As anticipated, it seems very well possible to distribute, and merge world knowledge in a mobile ad-hoc multi-agent environment. Even in such an environment with limited communication possibilities our test results showed there is a significant gain found when solving a mapping problem with multiple distributed agents. As expected, the larger the map the better results on how useful distributing and merging partial maps is. Although with the fact that simulation runs on a single machine comes that there are limits because of processing power. This is an important issue still encountered in our simulation system. Though when calculations would not be performed on one machine anymore, but on one for each agent, it should not be a problem anymore. So we anticipate scalability will not be a direct problem, should the simulation be translated into a real life distributed system.

In the tests, ManetLoc was compared to our problem description from which can be concluded is the system meets the goals set in the problem description. As it was constructed with a system in mind containing the 8 facets summarizing the project goals, these will be discussed now.

Emergent data structures
In comparison to systems with a static data or world model, we consider system able to build and adapt their model on the fly to be superior in highly dynamic situations such as a crisis. It is imaginable that such a situation renders large parts of any knowledge available prior to a (series of) crisis event. Therefore ManetLoc was designed with the possibility of not having any preset knowledge. Thus the data structure is built from bottom-up as soon as observations are coming in. Each simulated user will build its own personal model of the world from scratch.

Multi-agent collaboration
The combination of many agents systems which are able to in some way, are often able to solve a problem collaboratively with less effort than when an individual agent would have to tackle the issue. As our system tests showed, this is also the case with ManetLoc. Building a world model together saves users from having to explore certain parts of the world, as knowledge of unvisited parts we shared by others in the environment and merged with the personal world model.

Distributed data
In ManetLoc partial data about the world is distributedly available to each agent in the environment. Though it is never certain if one agent has the 'complete' picture available, distributed data makes it highly unlikely for all data to get lost. Our setup makes it possible to store information only in the area(s) where it is required, thus not needing long distance communication lines to be able to operate.

Decentralized systems
Related to distributed data is that as ManetLoc is a fully decentralized it is unlikely to ever completely be unavailable. One system going down in a (partially) centralized setup could prevent it from functioning at all. In our system each agent is fully autonomous and running on its own (simulated) PDA and only makes use of the services of others only when availably.

Probabilistic, fuzzy information
Human observations about the world can be considered subjective and inexact. The input agents in ManetLoc receive is thus fuzzy, local and probabilistic. Fuzziness in our system mean although a corridor has been measured as 100 units a time t1 it can be measured 90 units at t2. Locality of information can be seen in the concept orientation of distance units each agent has. For example, where for one agent a corridor is 10 distance units for another it can be 100 different units. The probabilistic nature of our system can be clearly seen in the closing of loops. As information is inexact, the correctness of the decision the a loop has been found can at a later time seem incorrect and thus undone.

Mobile ad-hoc networking applications
Most likely caused by the fact that there is still a lot of work in progress basis of MANET's (such as routing algorithms)a lack of applications and application platforms was found. An important goal in our research looking into the possibilities and advantages of new and existing applications made possible by using MANETs as a basis. Thus ManetLoc was also designed a proof of concept of a MANET specific system.

Location awareness
The main task of agents in ManetLoc is to gather environment data and from this incrementally construct a model of the world, a map. Thus becoming aware and being able to reason about locations in the world, such as where other users, exits and potentially any object or event is located.

Human-agent collaboration
An important aspect of our concept is creating a fluent collaboration between our system (i.e. agents) and humans carrying PDAs. Simulated user input is the main information channel in our system. Though in ManetLoc individuals are simulated, the architecture is ready for actual humans in a real life when an input method such as speech synthesis would be used.

Concluding we can say that all the goals - as stated in the problem description and the project overview - are met. However, there are still many things we would like to see done in a different or more elaborate way. We will discuss ideas for possible future work in the next section.

7.2 Future Work

The first imaginable aspect of future work is converting the simulation into an real life system. As most work in our group is done using the Java programming language another wish is to convert from C# to Java. This also because of the portability to mobile devices, where Java is currently more supported. This is also the case for the Sharp Zaurus, which doesn't run C# code (at the moment) but does run Java code.

An issue was encountered in our simulation which was caused by the fact each agent has the same behavior by default. This resulted in the agent clogging together/following each other whenever they meet and exchange world knowledge. This might some situations be the behavior actual users would sometimes perform. It is not intelligent behavior though when trying to explore the world in as less time as possible. Therefore more research could go into better guidance and preventing agents following each other when this not wanted behavior.

An important part of anticipated work is improving data distribution and realizing possibilities for distributed planning and execution in mobile adhoc agent networks.

7.2.1 Data distribution, Planning and execution

Having a shared concept of location enables agents to talk about events at certain locations. This chapter will present thoughts on storing location-related knowledge in mobile ad-hoc networks. Knowledge one could think of is for example that there is a fire somewhere, or that a certain hall is no longer passable because of debris from the roof collapsing, also more abstract knowledge such as crisis scenarios

The idea of crisis scenarios also is part of the 'Crisis management using mobile ad-hoc networks' project and discussed more in-depth in [Scho05]. When they have enough information about an area, let the agents form an opinion about the scenario that might be at hand. A scenario may change the behavior of agents, i.e. the way it spreads and gathers information.

The following issues in the context of handling world knowledge in mobile ad-hoc networks will be discussed below:

- *Preventing network clogging*
- *Storing and making available data*

- *Making data available at a certain location*
- *Keeping data available*
- *Working with knowledge in different levels of detail*
- *Security*
- *Planning and execution*

It is important to notice when thinking about storing data in mobile ad-hoc nodes, is that you should no be too overenthusiastic about sharing as it has its costs. Sending too much data could clog the network, which should be avoided at all times [Chen04] [Desi04].

Also, as there is a high possibility of an ad-hoc node not being reachable by some other nodes at a certain time it would seem to be appropriate that (in most cases) information would not be transferred to specific nodes but to locations or areas. Supporting this is the fact that we are storing context information and we don't have a central data storage unit available. Most knowledge is highly related to its location and more or less useless if not available at its originating location.

As stated above the storage of information in our proposed architecture concept is primarily related to the location to where it originated and/or to where it is relevant. This is also the case for preventing of data loss or unavailability. To prevent from losing data when a node goes down or becomes unreachable, if possible it seems wise to keep multiple copies of the knowledge. We would prefer to do this in the region where it is most relevant. In most cases this would mean in the neighborhood where it originated and for high-level information at locations where decisions are made about this information (i.e. the 'command center').

As it is unlikely that many ad-hoc nodes will go down at exactly the same time the above will make sure that if a node would goes down the information is very likely still available at other nodes. Providing that the information was sufficiently important and there were other nodes in its neighborhood, it should have been distributed to others. After initial storage at a certain location keeping information at the 'correct' location(s) is also an issue. An idea for this is that if an agent would leave the area in which certain information is relevant it should make an effort to redistribute its knowledge to be sure it doesn't get lost. Due to the ad-hoc nature of MANETs information cannot always directly be transferred to all relevant locations. As direct transfer will often not be possible attempts should be made to eventually get the information at the specified location. An idea for this is basically moving the information in the right direction, i.e. closer to its target location. Combined with other tactics this should in often enable accomplish the goal.

Another advantage of this location-based approach is that as most communication will be single-hop. As mentioned earlier in this chapter network bandwidth is relatively scarce in ad-hoc networks, so it is very important to limit data being transferred. Single hop communication greatly reduces the network load, compared to multi hop communication and therefore helps preventing network overload.

An important aspect of storing context information mentioned before is that it has differing relevancy (its weight) at different location. Also though global information of a 'situation' can still be important somewhere, its details are possibly not. This would imply that it is possible and useful to store information with different levels of detail (possibly using quad tree decomposition). So for example, the more important information is, the more widely it will be distributed over the nodes in the network, but the further we get from the 'source' the less detail is sent.

As in most information systems security is also an issue here. Most likely not all information should be available to everyone, as information could be private or sensitive in some way. There are two issues here, as data will most likely travel trough air in our network (ad-hoc WiFi) anyone is able to pick up the data stream. Also some nodes we don't want to have access to certain information might be required to transfer the information to the right nodes. While not the focus of our research there are some security problems, which would probably require some or all data should be encrypted at the cost of processing power etc.

The direction of movement of an ad-hoc node could possibly matter in what information to share with other nodes. This could mean to push information that is 'better off' at other node – as information should stay near its original location - and pull info you will need about the environment you are in and/or will encounter. Direction of movement might also be used in determining the level of detail to be sent. One could for example send higher detail of the area, which lies in current direction of movement, and less detail of the area a node is coming from.

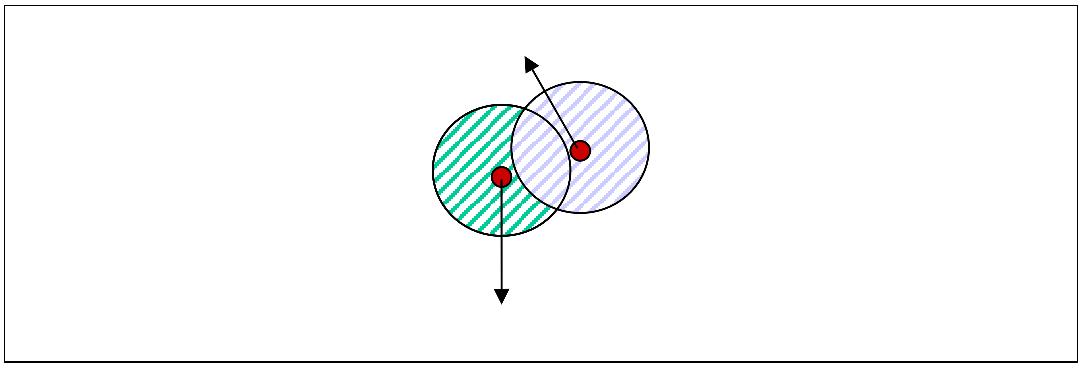


Figure 72 Direction of movement matters in distribution of data

Another idea is that knowledge gathered from own sensors or user input has higher priority of being shared than knowledge gathered from other nodes. Most often knowledge will be a combination of the two but the more it comes from the node itself the more it should be pushed. This to promote distribution of new knowledge instead of already distributed data.

When able to build a model of the current state of the world, planning and execution of tasks based on this dynamic knowledge will also become possible. If for instance someone needs help at a certain location as he has just encountered finds an emergency, such as a fire spreading.

As the network topology of mobile ad-hoc networks often highly dynamic, we must always keep in mind that requests for information at and/or about a certain location are not guaranteed to ever return. This is also an issue when coordinating actions. When communication lines between the parties that ordered a task can break up at any moment being sure if a task is being executed is a problem. Therefore it seems logical not to try not to give orders to a specific node and implicitly a specific person, but to a relative region. And leave monitoring of execution the task mostly to this region itself. In the emergency scenario mentioned, the fact that help is required can be spread across the network. When the emergency is over or when enough help has arrived this again can be distributed.

Bibliography

[Comb] *Combined Systems group*, http://combined.decis.nl

[Boel04] *A Communication Layer for Distributed Decision Making*, J.L.Boehlé, 2004, Delft University of Technology, Erasmus University Rotterdam

[Aaai] *Agents*, The American Association for Artificial Intelligence, 2000 - 2005 http://www.aaai.org/AITopics/html/agents.html

[Fipa] *The Foundation for Intelligent Physical Agents*, http://www.fipa.org

[Berg03] *Towards a FIPA Approach for Mobile Ad-hoc Environments*, M. Berger1, M. Watzke1, H. Helin, 2003, FIPA, http://www.fipa.org/docs/input/f-in-00091/f-in-00091.pdf

[Fiaa] *FIPA Abstract Architecture Specification*, 2003, FIPA, http://www.fipa.org/specs/fipa00001/SC00001L.pdf

[Sanc03] *Design of a FIPA compliant agent platform for limited devices*, G. Sancho, R. Rioja, C.Vazquez, 2003, University Carlos III de Madrid, http://www.it.uc3m.es/~celeste/papers/mata2003_uc3m_def.pdf

[Jade] *Java Agent Development Framework*, http://jade.tilab.com/

[Coug] *Cognitive Agent Architecture*, http://www.cougaar.org/

[Swar1] *Swarm Intelligence*, http://en.wikipedia.org/wiki/Swarm_intelligence

[Swar2] *Swarm Intelligence Recourses*, http://dsp.jpl.nasa.gov/members/payman/swarm/

[Dori91] *Positive Feedback as a Search Strategy*, Dorigo M., V. Maniezzo & A. Colorni, 1991, Politecnico di Milano, ftp://iridia.ulb.ac.be/pub/mdorigo/tec.reps/TR.01-ANTS-91-016.ps.gz

[Rado03] *Ant Routing for Mobile Ad-Hoc Networks*, Radovan Miloševic, 2003, Delft University of Technology

[Fern05] *Varying the Population Size of Artificial Foraging Swarms on Time Varying Landscapes*, Carlos Fernandes, Vitorino Ramos, Agostinho C. Rosa1, 2005, Technical Univ. of Lisbon, http://alfa.ist.utl.pt/~cvrm/staff/vramos/Vramos-ICANN05.pdf

[Hels04] *Cougaar: A scalable, Distributed Muti-Agent Architecture*, Aaron Helsinger, Michael Thome, Todd Wright, 2004, BBN Technologies, http://cougaar.org/docman/view.php/17/136/cougaar-bbn-0617-submitted.pdf

[IEEE99A] *LAN MAN Standards Committee. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999, IEEE

[IEEE99B] *LAN MAN Standards Committee. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band*, 1999, IEEE

[Kant03] *Modeling and Simulation of Adhoc/P2P Resource Sharing Networks,* Krishna Kant, Ravi Iyer, 2003, Intel Labs, http://kkant.ccwebhost.com/papers/simpra.pdf

[Leap] *Lightweight Extensible Agent Platform*, http://leap.crm-paris.com

[Lawr02] *LEAP into Ad-Hoc Networks, Jamie Lawrence*, 2002, Media Lab Europe, http://autonomousagents.org/ubiquitousagents/2002/papers/papers/10.pdf

[Fadh1] *FIPA Ad-hoc Technical Comitee*, http://www.fipa.org/activities/ad_hoc.html

[Fadh2] *Agents in Ad-hoc Environments, a Whitepaper, Version: 0.C*, 2002, FIPA Technical Committee “Adhoc”, http://www.fipa.org/docs/input/f-in-00068/f-in-00068A.htm

[Finf] *FIPA Inform! Volume 3 Issue 3*, http://www.fipa.org/docs/output/f-out-00128/f-out-00128.pdf

[Jadm] *JADE Administrator's guide, chapter 4*, Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa, Roland Mungenast, 2005, JADE Board, http://jade.tilab.com/doc/administratorsguide.pdf

[Jfip] *JADE- A FIPA compliant agent framework*, Fabio Bellifemine, Agostino Poggi, Giovanni Rimassa, JADE Board, http://sharon.cselt.it/projects/jade/papers/PAAM.pdf

[Murp97] *Global Positioning Systems, A Technical Assessment Paper*,Lotta Danielsson-Murphy, Thaddeus Murphy , 1997 http://www.lottaworld.com/worksample/gps.html

[Kita03] *Design of WiPS: WLAN-Based Indoor Positioning System*, Teruaki Kitasuka Tsuneo Nakanishi, and Akira Fukuda, 2003, Kyushu University, http://www.f.csce.kyushu-u.ac.jp/~kitasuka/papers/2003/kmms_kitasuka_v7n4.pdf

[Srda01] *GPS-free positioning in mobile Ad-Hoc networks*, Srdan Capkun, Maher Hamdi, Jean-Pierre Hubaux, 2001, Ecole Polytechnique Federale de Lausanne, http://lcawww.epfl.ch/Publications/Capkun/CapkunHH01a.pdf

[Ahlos] *The Ad-hoc Localization System*, http://nesl.ee.ucla.edu/projects/ahlos/

[Nicu01] *Ad-hoc Positioning System (APS) Using AOA*, D. Niculescu, B. Nath, Rutgers University, http://paul.rutgers.edu/~dnicules/research/aps/aoa-infocom.pdf

[Bahl00] *Radar: An in-building user location and tracking system*, P. Bahl, V. N. Padmanabhan, 2000, Microsoft Research, http://research.microsoft.com/~padmanab/papers/infocom2000.pdf

[Giro01] *Robust Range Estimation Using Acoustic and Multimodal Sensing*, Lewis Girod and Deborah Estrin, 2001, UCLA, http://lecs.cs.ucla.edu/~girod/papers/IROS-2001.pdf

[Gane02] *Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks*, D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin and S. Wicker, 2002, http://lecs.cs.ucla.edu/Publications/papers/Deepak-Empirical.pdf

[Bulu00] *GPS-less Low Cost Outdoor Localization for Very Small Devices*, N. Bulusu, J. Heidemann and D. Estrin, 2000, University of Southern California, http://lecs.cs.ucla.edu/~bulusu/papers/Bulusu00a.pdf

[Nicu03] *DV-based positioning in adhoc networks in Telecommunication Systems*, D. Niculescu, B. Nath, 2003, The State University of New Jersey, http://paul.rutgers.edu/~dnicules/research/aps/aps-jrn.pdf

[Butl01] *Distributed coverage of rectilinear environments*, Z.Butler, A Rizzi, R. Hollis, 2001, Carnegie Mellon University, http://www.ri.cmu.edu/pub_files/pub2/butler_zack_2000_1/butler_zack_2000_1.pdf

[Dede00] *Landmark-based matching algorithm for cooperative mapping by autonomous robots*, G. Dedeoglu,G. Sukhatme, 2000, University of Southern California, http://www-robotics.usc.edu/~dedeoglu/dars00.pdf

[Rike94] *Autonomous agent map construction in unknown enclosed environments*, K. Doty, S. Seed, 1994, University of Florida, http://citeseer.ist.psu.edu/rd/665606%2C505179%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/25819/http:zSzzSzwww.mil.ufl.eduzSzpublicationszSz..zSzpeoplezSzdotyzSzpaperszSzautonomouszSzriker.pdf/doty94autonomous.pdf

[Vali01] *Subgraph Isomorphism and Related Problems*, G. Valiente, 2001, Technical University of Catalonia, http://www.lsi.upc.es/~valiente/graph-00-01-a.pdf

[Bunk00] *Mean and maximum common subgraph of two graphs*, H. Bunke, A. Kandel, 2000, University of Bern, University of South Florida, http://www.iam.unibe.ch/~fki/publications/papersOnGraphMatching/meanAndMaximum.ps.gz

[Scho05] *Icon based System for Managing Emergencies*, P. Schooneman, 2005, Delft University of Technology

[Ches01] *Pedometer Accuracy, T. Chester, 2001,* http://sd.znet.com/~schester/grand_canyon/pedometer_accuracy.html

[Save04] *Loop-Closing and Planarity in Topological Map-Building*, F. Savelli, B. Kuipers, 2004, Universit`a di Roma “La Sapienza”, University of Texas at Austin, ftp://ftp.cs.utexas.edu/pub/qsim/papers/Savelli-iros-04.pdf

[Remo02] *Towards a General Theory of Topological Maps*, E. Remolina, B. Kuipers, 2002, University of Texas at Austin, ftp://ftp.cs.utexas.edu/pub/qsim/papers/Remolina+Kuipers-TR-02.pdf

[Vbre] Microsoft Visual Basic .NET Resource Kit, http://msdn.microsoft.com/vbasic/vbrkit/default.aspx

[Tato04] *Dynamic traffic routing using Ant Based Control* , B. Tatomir, L. Rothkrantz, 2004, Delft University of Technology

[Chen04] *TCP Performance over Mobile Ad-hoc Networks*, X. Chen, H. Zhai, J. Wang, Y. Fang, 2004, University of Florida, http://www.ecel.ufl.edu/~jwang/publications_files/tcp_cjece.pdf

[Desi04] *Sustaining Performance Under Traffic Overload*, S. DeSilva, R. Boppana, 2004, University of Texas, http://www.cs.utsa.edu/faculty/boppana/papers/Mwan04.pdf

[Thru97] *Learning metric-topological maps for indoor mobile robot navigation*, S. Thrun, 1997, Carnegie Mellon University, http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6TYF-3T0XP9D-2-2&_cdi=5617&_user=499885&_orig=search&_coverDate=02%2F28%2F1998&_qd=1&_sk=999009998&view=c&wchp=dGLbVzz-zSkzV&md5=b554820964dc89aa46a65fd9f207ad80&ie=/sdarticle.pdf

[Ns2] *The Network Simulator - ns-2*, http://www.isi.edu/nsnam/ns/

[Glom] *GloMoSim*, http://pcl.cs.ucla.edu/projects/glomosim/

[Swan] *SWAN*, http://www.comet.columbia.edu/swan/index.html

[Kroo02] *Dynamic vehicle routing using ant based control*, R. Kroon, 2002, Delft University of Technology, http://www.kbs.twi.tudelft.nl/docs/MSc/2002/Kroon_Ronald/thesis.pdf

Appendix A: User Manual

This Appendix will provide short non-technical instructions on what ManetLoc is and how to use it.

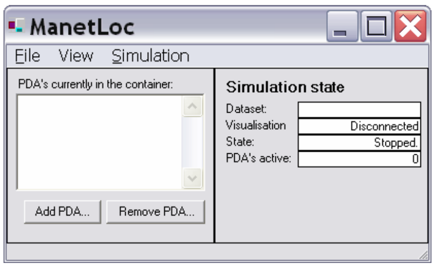
What is ManetLoc

This application is a demonstration of how users in an unknown building-like environment carrying mobile devices can explore the world, share the gathered data and merge with their current world model (in the case of ManetLoc a topological map) by using a mobile ad-hoc network as a platform.

The application is a simulation and proof of concept, and could be converted into an actual system by another developer when a agent-platform which is suitable for use in mobile ad-hoc networks is available.

Using ManetLoc

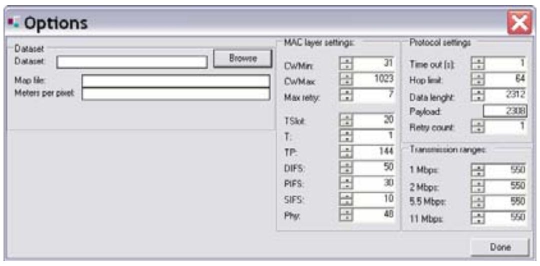
ManetLoc can be started by executing 'manetloc.exe'. This will result in the main interface to be loaded.



Simulation state

Dataset:	
Visualisation	Disconnected
State:	Stopped.
PDA's active:	0

Main interface



Dataset:	
Map file:	
Meters per pixel:	

MAC layer settings:

CWMin:	31
CWMax:	1023
Max retry:	7
TSlot:	20
T:	1
TP:	144
DIFS:	50
PIFS:	30
SIFS:	10
Phy:	48

Protocol settings

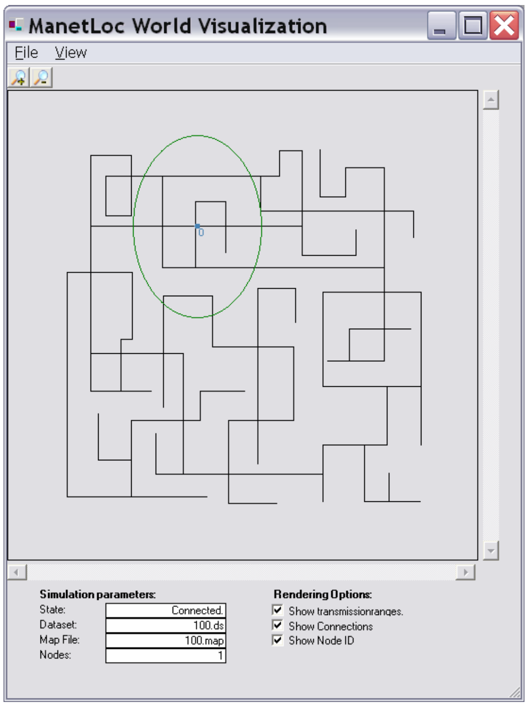
Time out (s):	1
Hop limit:	64
Data lenght:	2312
Payload:	2308
Retry count:	1

Transmission ranges:

1 Mbps:	550
2 Mbps:	550
5.5 Mbps:	550
11 Mbps:	550

Options Dialog

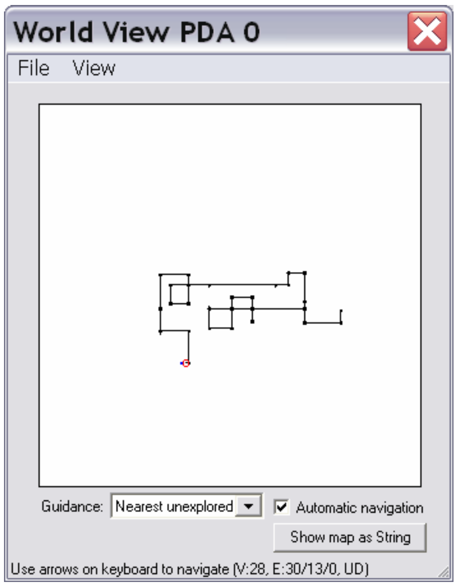
The next step is to load a dataset - which are files ending with *.ds -, defining the world for the simulation. To do this, select 'Options' from the 'Simulation' menu. The options dialog will be loaded. On the left side a dataset can be selected by clicking on 'Browse'. On the right side the network parameters can be modified if required. When ready click done and the dataset will be loaded, plus the world visualization environment will be automatically started.



Simulation parameters:
State: Connected.
Dataset: 100.ds
Map File: 100.map
Nodes: 1

Rendering Options:
☑ Show transmissionranges.
☑ Show Connections
☑ Show Node ID

World visualization



Guidance: Nearest unexplored
☑ Automatic navigation

Use arrows on keyboard to navigate (V:28, E:30/13/0, UD)

PDA world view

Now select 'Start' in the 'Simulation menu to actually start the simulation. If the map is not visible in the visualization environment, you can click on the zoom buttons in the upper left corner. If you select 'Log' in the 'View' menu the log will be shown, indicating the simulation has started. Now PDA's can be added (and removed) from the simulation. Click 'Add PDA' to add one. The new PDA will now appear in the list box on the main interface and in world visualization. Double clicking on the PDA in the listbox will open its world view interface. Using this dialog a specific PDA can be monitored and controlled. The main part consists of a drawing of the current world model the agent running on the PDA has in memory. In the 'View' menu the log for this PDA can be shown or hidden. By checking 'Automatic Navigation' the world if automatically explored, using the guidance option selected. If not checked, the movement of the virtual user can be manually controlled by using the arrow keys on the keyboard. By clicking on the 'Show map as string' the current world model is show as a string. This last option is manly available for debugging purposes.

Appendix B: Map, Dataset and Road files

<10.ds>

```
H|10.map|9999|0|500|500|0|0|4
```

<10.map>

```
// Use this file to construct the map
// First enumerate the intersections
// Then connect the intersections with roads
// Then define the exits
intersections
{	//{number, x-pos, y-pos}
	{1,0,100}
	{2,100,100}
	{3,299,100}
	{4,299,0}
	{5,0,231}
	{6,100,231}
	{7,173,231}
	{8,173,280}
	{9,150,280}
	{10,150,322}
}

roads
{	//{number, from, to}
	{1, 1, 2}
	{2, 2, 1}
	{3, 2, 3}
	{4, 3, 2}
	{5, 3, 4}
	{6, 4, 3}
	{7, 1, 5}
	{8, 5, 1}
	{9, 5, 6}
	{10, 6, 5}
	{11, 6, 7}
	{12, 7, 6}
	{13, 7, 8}
	{14, 8, 7}
	{15, 8, 9}
	{16, 9, 8}
	{17, 9, 10}
	{18, 10, 9}
	{19, 2, 6}
	{20, 6, 2}
}

exits
{	//{number, vertexnum, direction}
	{1, 1, 4}
}
```

<30.ds>

```
H|30.map|9999|0|1000|500|0|0|4
```

<30.map>

```
// Use this file to construct the map
// First enumerate the intersections
// Then connect the intersections
// with roads
// Then define the exits
intersections
{       //{number, x-pos, y-pos}
        {1,0,0}
        {2,0,200}
        {3,200,200}
        {4,200,400}
        {5,100,400}
        {6,100,500}
        {7,300,500}
        {8,100,700}
        {9,300,600}
        {10,300,700}
        {11,300,900}
        {12,400,400}
        {13,400,200}
        {14,500,400}
        {15,500,600}
        {16,500,700}
        {17,400,800}
        {18,400,900}
        {19,500,200}
        {20,500,100}
        {21,600,200}
        {22,600,400}
        {23,700,500}
        {24,700,600}
        {25,800,600}
        {26,800,800}
        {27,800,500}
        {28,800,400}
        {29,600,700}
        {30,600,600}
}
roads
{       //{number, from, to}
        {1, 1, 2}
        {2, 2, 1}
        {3, 2, 3}
        {4, 3, 2}
        {49, 24, 25}
        {50, 25, 24}
        {51, 25, 26}
```

```
        {5, 3, 4}
        {6, 4, 3}
        {7, 4, 5}
        {8, 5, 4}
        {9, 5, 6}
        {10, 6, 5}
        {11, 6, 7}
        {12, 7, 6}
        {13, 6, 8}
        {14, 8, 6}
        {15, 7, 9}
        {16, 9, 7}
        {17, 8, 10}
        {18, 10, 8}
        {19, 10, 11}
        {20, 11, 10}
        {21, 10, 16}
        {22, 16, 10}
        {23, 16, 15}
        {24, 15, 16}
        {25, 15, 14}
        {26, 14, 15}
        {27, 14, 12}
        {28, 12, 14}
        {29, 4, 12}
        {30, 12, 4}
        {31, 13, 12}
        {32, 12, 13}
        {33, 14, 22}
        {34, 22, 14}
        {35, 22, 21}
        {36, 21, 22}
        {37, 19, 21}
        {38, 21, 19}
        {39, 19, 20}
        {40, 20, 19}
        {41, 22, 28}
        {42, 28, 22}
        {43, 28, 27}
        {44, 27, 28}
        {45, 23, 27}
        {46, 27, 23}
        {47, 23, 24}
        {48, 24, 23}
```

```
		{52, 26, 25}
		 {53, 26, 17}
		{54, 17, 26}
		{55, 17, 18}
		{56, 18, 17}
		{57, 15, 30}
		{58, 30, 15}
		{59, 24, 30}
		{60, 30, 24}
		{61, 30, 29}
		{62, 29, 30}
}

exits
{		//{number, vertexnum, direction}
		{1, 2, 4}
		{2, 18, 1}
		{3, 20, 3}
		{4, 25, 2}
}
```

<100.ds>

```
H|100.map|9999|0|800|500|0|0|4
```

<100.map>

```
// Use this file to construct the map
// First enumerate the intersections
// Then connect the intersections with roads
// Then define the exits
intersections
{       //{number, x-pos, y-pos}
        {1, 159, 81}
        {2, 271, 59}
        {3, 429, 251}
        {4, 229, 241}
        {5, 429, 141}
        {6, 297, 371}
        {7, 333, 251}
        {8, 333, 328}
        {9, 568, 334}
        {10, 506, 251}
        {11, 436, 456}
        {12, 227, 433}
        {13, 162, 263}
        {14, 72, 364}
        {15, 229, 328}
        {16, 568, 103}
        {17, 706, 192}
        {18, 700, 371}
        {19, 525, 456}
        {20, 657, 103}
        {21, 706, 103}
        {22, 778, 192}
        {23, 568, 59}
        {24, 368, 59}
        {25, 368, 141}
        {26, 159, 140}
        {27, 137, 241}
        {28, 88, 81}
        {29, 72, 241}
        {30, 271, 140}
        {31, 568, 192}
        {32, 429, 340}
        {33, 700, 456}
        {34, 436, 509}
        {35, 297, 433}
        {36, 227, 509}
        {37, 72, 433}
        {38, 700, 334}
        {83, 618, 478}
        {84, 562, 478}
        {85, 562, 548}
        {86, 212, 120}
        {39, 762, 456}
        {40, 778, 334}
        {41, 525, 433}
        {42, 361, 433}
        {43, 160, 509}
        {44, 162, 364}
        {45, 137, 185}
        {46, 72, 185}
        {47, 137, 263}
        {48, 271, 241}
        {49, 306, 185}
        {50, 657, 19}
        {51, 778, 104}
        {52, 700, 521}
        {53, 618, 521}
        {54, 361, 393}
        {55, 72, 540}
        {56, 160, 540}
        {57, 506, 141}
        {58, 368, 16}
        {59, 159, 25}
        {60, 88, 150}
        {61, 306, 140}
        {62, 510, 340}
        {63, 625, 230}
        {64, 625, 279}
        {65, 700, 279}
        {66, 700, 230}
        {67, 756, 279}
        {68, 580, 230}
        {69, 227, 371}
        {70, 160, 449}
        {71, 104, 449}
        {72, 104, 509}
        {73, 361, 470}
        {74, 297, 470}
        {75, 510, 289}
        {76, 568, 143}
        {77, 711, 19}
        {78, 777, 19}
        {79, 762, 416}
        {80, 525, 389}
        {81, 640, 389}
        {82, 640, 428}
        { 31, 5, 57}
        {32, 57, 5}
        {33, 5, 98}
        {34, 98, 5}
```

```
		{87, 212, 59}
		{88, 23, 25}
		{89, 23, 364}
		{90, 568, 18}
		{91, 711, 61}
		{92, 525, 547}
		{93, 476, 547}
		{94, 476, 509}
		{95, 229, 161}
		{96, 201, 185}
		{97, 402, 185}
		{98, 429, 75}
		{99, 471, 16}
		{100, 321, 25}
}

roads
{		//{number, from, to}
		{1, 1, 26}
		{2, 26, 1}
		{3, 1, 28}
		{4, 28, 1}
		{5, 1, 59}
		{6, 59, 1}
		{7, 2, 24}
		{8, 24, 2}
		{9, 2, 30}
		{10, 30, 2}
		{11, 2, 87}
		{12, 87, 2}
		{13, 3, 5}
		{14, 5, 3}
		{15, 3, 7}
		{16, 7, 3}
		{17, 3, 10}
		{18, 10, 3}
		{19, 3, 32}
		{20, 32, 3}
		{21, 4, 15}
		{22, 15, 4}
		{23, 4, 27}
		{24, 27, 4}
		{25, 4, 48}
		{26, 48, 4}
		{27, 4, 95}
		{28, 95, 4}
		{29, 5, 25}
		{30, 25, 5}
		{31, 5, 57}
		{84, 31, 17}
		{85, 18, 33}
		{86, 33, 18}
		{87, 18, 38}
		{88, 38, 18}
		{89, 19, 33}
		{35, 6, 18}
		{36, 18, 6}
		{37, 6, 35}
		{38, 35, 6}
		{39, 6, 69}
		{40, 69, 6}
		{41, 7, 8}
		{42, 8, 7}
		{43, 8, 15}
		{44, 15, 8}
		{45, 9, 31}
		{46, 31, 9}
		{47, 9, 38}
		{48, 38, 9}
		{49, 10, 57}
		{50, 57, 10}
		{51, 11, 19}
		{52, 19, 11}
		{53, 11, 34}
		{54, 34, 11}
		{55, 12, 35}
		{56, 35, 12}
		{57, 12, 36}
		{58, 36, 12}
		{59, 12, 37}
		{60, 37, 12}
		{61, 12, 69}
		{62, 69, 12}
		{63, 13, 44}
		{64, 44, 13}
		{65, 13, 47}
		{66, 47, 13}
		{67, 14, 29}
		{68, 29, 14}
		{69, 14, 37}
		{70, 37, 14}
		{71, 14, 44}
		{72, 44, 14}
		{73, 14, 89}
		{74, 89, 14}
		{75, 16, 20}
		{76, 20, 16}
		{77, 16, 23}
		{78, 23, 16}
		{79, 17, 21}
		{80, 21, 17}
		{81, 17, 22}
		{82, 22, 17}
		{83, 17, 31}
		{136, 94, 34}
		{137, 35, 42}
		{138, 42, 35}
		{139, 35, 74}
		{140, 74, 35}
		{141, 36, 43}
```

{90, 33, 19}	{142, 43, 36}
{91, 19, 41}	{143, 37, 55}
{92, 41, 19}	{144, 55, 37}
{93, 19, 92}	{145, 38, 40}
{94, 92, 19}	{146, 40, 38}
{95, 20, 21}	{147, 38, 65}
{96, 21, 20}	{148, 65, 38}
{97, 20, 50}	{149, 39, 79}
{98, 50, 20}	{150, 79, 39}
{99, 22, 40}	{151, 41, 42}
{100, 40, 22}	{152, 42, 41}
{101, 22, 51}	{153, 41, 80}
{102, 51, 22}	{154, 80, 41}
{103, 23, 24}	{155, 42, 54}
{104, 24, 23}	{156, 54, 42}
{105, 23, 90}	{157, 42, 73}
{106, 90, 23}	{158, 73, 42}
{107, 24, 25}	{159, 43, 56}
{108, 25, 24}	{160, 56, 43}
{109, 24, 58}	{161, 43, 70}
{110, 58, 24}	{162, 70, 43}
{111, 26, 30}	{163, 43, 72}
{112, 30, 26}	{164, 72, 43}
{113, 27, 29}	{165, 45, 46}
{114, 29, 27}	{166, 46, 45}
{115, 27, 45}	{167, 45, 96}
{116, 45, 27}	{168, 96, 45}
{117, 27, 47}	{169, 49, 61}
{118, 47, 27}	{170, 61, 49}
{119, 28, 60}	{171, 49, 97}
{120, 60, 28}	{172, 97, 49}
{121, 29, 46}	{173, 50, 77}
{122, 46, 29}	{174, 77, 50}
{123, 30, 48}	{175, 52, 53}
{124, 48, 30}	{176, 53, 52}
{125, 30, 61}	{177, 53, 83}
{126, 61, 30}	{178, 83, 53}
{127, 32, 62}	{179, 55, 56}
{128, 62, 32}	{180, 56, 55}
{129, 33, 39}	{181, 58, 99}
{130, 39, 33}	{182, 99, 58}
{131, 33, 52}	{183, 59, 88}
{132, 52, 33}	{184, 88, 59}
{133, 34, 36}	{185, 59, 100}
{134, 36, 34}	{186, 100, 59}
{135, 34, 94}	{187, 62, 75}
{188, 75, 62}	{10, 93, 4}
{189, 63, 64}	{11, 11, 4}
{190, 64, 63}	}
{191, 63, 66}	
{192, 66, 63}	
{193, 63, 68}	
{194, 68, 63}	
{195, 64, 65}	
{196, 65, 64}	

```
            {197, 65, 66}
            {198, 66, 65}
            {199, 65, 67}
            {200, 67, 65}
            {201, 70, 71}
            {202, 71, 70}
            {203, 71, 72}
            {204, 72, 71}
            {205, 73, 74}
            {206, 74, 73}
            {207, 77, 78}
            {208, 78, 77}
            {209, 77, 91}
            {210, 91, 77}
            {211, 80, 81}
            {212, 81, 80}
            {213, 81, 82}
            {214, 82, 81}
            {215, 83, 84}
            {216, 84, 83}
            {217, 84, 85}
            {218, 85, 84}
            {219, 86, 87}
            {220, 87, 86}
            {221, 88, 89}
            {222, 89, 88}
            {223, 92, 93}
            {224, 93, 92}
            {225, 93, 94}
            {226, 94, 93}
}

exits
{           //{number, vertexnum, direction}
            {1, 15, 1}
            {2, 22, 2}
            {3, 26, 4}
            {4, 62, 2}
            {5, 46, 3}
            {6, 85, 4}
            {7, 100, 1}
            {8, 11, 3}
            {9, 66, 3}
```

< *.road>

```
// Use this file to assign the road properties
// the properties are:
// a number of lanes
// a length (in m)
// a speed (in km/h)
// a priority (HIGH or LOW)
road_properties
{
        //{road number, number of lanes, length, speed, priority}
}
```

Appendix C: ManetLoc paper

Marcel van Velden
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft
M.C.vanVelden@student.tudelft.nl

Abstract

ManetLoc simulates a building like environment where individuals are exploring an unknown world. It is assumed that each individual in the field is equipped with a Personal Digital Assistant (PDA) and can communicate with other PDAs in the vicinity. Together the PDAs dynamically form ad-hoc networks. Users can enter their own observations to the PDA. Agents on these PDA's work together, to supply the users an as complete view on the world as possible (a topological map). An agent will also provide guidance to the user if requested.

A concept of a system for making multi-agent systems in mobile ad-hoc networks aware of their environment without the need of any infrastructure is described. The main focus is on automatically building a map of the world by using observations from individuals in such an infrastructureless network. We have designed and implemented this proof of concept, ManetLoc, in the form of a simulation. No specific agent platform was used in developing the simulation, but the future use of JADE [Jade] was always kept in mind. The system was built upon a preexisting ad-hoc network simulation environment, Ad-hoc Simulator [Boel04].

Keywords: mobile ad-hoc network, agent, communication, topological map, infrastructureless, location, emergency, awareness, crisis

1 Introduction

In every aspect of our lives we are becoming more and more dependent on the availability of (information) systems. These systems by themselves depend on other systems to be available, i.e. an infrastructure. In times of crisis not all of these systems might always be readily available. This could be caused by power cuts for instance or simply because there is no physical infrastructure at all.

If a telecommunications infrastructure is not available at a certain area, it should still be possible to set up an infrastructureless network or mobile ad-hoc network (MANET) under most conditions. Setting up such a network enables us to share information concerning the state of the world and coordinate actions. These ad-hoc networking technologies are making it possible to exchange information anywhere, anytime without prior network infrastructure. Using handheld devices that operate in a wireless environment, communication is still possible when major infrastructural communication links have been damaged, destroyed or overloaded. So in case of a major disaster within a city, emergency services can communicate without the need for preset-up access points or other such infrastructural requirements.

Our multi-modal interfaces (MMI) department is doing extensive research on multi-agent systems using wireless ad-hoc networks. This research is part of the project 'Crisis management using mobile ad-hoc networks', which focuses on intelligent crisis management using mobile ad-hoc networks. The crisis management project is closely associated with the Combined project of DECIS LAB [Comb] our group participates in. Roughly the aim of the project is to develop an environment wherein rescue services can communicate using handheld devices dynamically forming MANETs. Users of such networks should be able to exchange observations through agent technology and intuitive GUI's located on a handheld set. Agents aid the user in finding, storing and retrieving information from the network. Our specific interest is the distribution of world knowledge in these ad-hoc communication networking environments.

Project overview

Storing and making available knowledge in agent enabled MANETs is a challenging problem that still needs to be tackled on many fronts. This particularly is the case when there is no secondary infrastructure available at all, no special hardware on the portable devices and the network is highly dynamic. Apart from the major difficulties, the possible uses of this kind of technology seem to be countless and contain – but are not limited to - crisis situations and military use. Related to this there is a demand for mobile devices to become more aware of their environment. Again this is a challenging problem on its own as well but if there is no infrastructure available at the relevant location this is an even more complex issue. All this seems to make research in the area more than worthwhile.

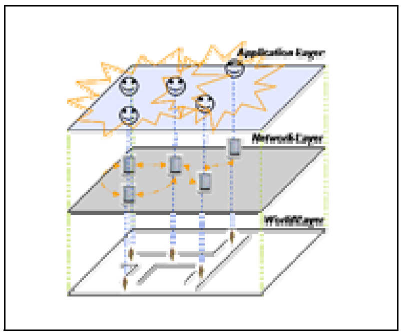


Figure 1 Project layers

A useful way of looking at our project is by splitting it up in the following three layers [Figure 1]: *World layer, Network layer and Application layer.* The world layer symbolizes the actual world a user finds itself in. As the user is carrying network enabled devices, there is a network which can be used to communicate, i.e. the network layer. In the application or agent layer knowledge about the world is shared and services are offered by agents in the network (i.e. applications). All three layers play an important role in our system concept, but we tried to focus on the application layer, where the other two serve as an essential basis for the later.

The goal of this research was to provide a concept - and a proof of this concept - of a potential solution for making multi-agent systems operating in mobile ad-hoc networks aware of their environment. From the multi agent-nature of such a system almost naturally follows the approach of distributed construction of world models by the agent software. The fact that the system is intended for use in highly dynamic ad-hoc network environments makes that no central and/or permanent services can be depended upon, thus the concept should be decentralized. Another aspect of the concept is that no knowledge of the world will be available beforehand and during the lifetime of a system component the knowledge available will incrementally grow but at most times will be incomplete and fuzzy. Therefore a non-deterministic probabilistic approach seems logical.

People observe the world and interact with their PDAs. A PDA running agent software can also sense the world and interact with other agents in a wireless ad-hoc network. From the information an agent gathers by communicating with the user of the PDA, sensors and other agents, it assembles and continually updates a world model. This world model can then be provided to the user and other (agent) software running on the PDA. To summarize this section the three main areas of interest mentioned (*multi-agent systems, mobile ad-hoc networks* and *location awareness*) can be further specified into key facets of the system architecture concept. Thus the system will involve the combination of:

- *Emergent data structures*
- *Multi-Agent collaboration*
- *Distributed data*
- *Decentralized systems*
- *Probabilistic, fuzzy information*
- *Mobile ad-hoc networking applications*
- *Location awareness*
- *Human-agent collaboration*

Problem formulation

Taking into account not all can be handled in our work, we focused on location awareness in infrastructureless environments where the problem formulation is defined as follows:

Design and implement a multi-agent-system that can operate in environments without a pre-setup infrastructure (only a mobile adhoc network) and without any pre-knowledge of the world, which is able to process and fuse location information from different users and sensors remote in space and time and distributes location information and location based services (such as guidance) to its users.

In the next sections we will elaborate on the architecture we have setup in answer to this problem description. First relevant and related work studied during the course of this research will be discussed.

2 Related work

In this section some related topics will be described. We will look at the areas of multi agent systems, ad-hoc wireless networks, agents on small devices, agents in ad-hoc networks and location awareness.

Multi-agent systems

The first topic that is important for our work is that of agent systems, and more specific, multi-agent-systems (MAS) [Aaai]. The study of agent based systems evolved from the field of Distributed Artificial Intelligence in the early to mid 1980's. In contrast to classical applications in artificial intelligence, often viewed as dedicated, centralized and standalone, the specific ideas underlying agents in so called agent based systems globally are the following:

- *Autonomous to a degree*
- *Exhibit goal directed behavior*
- *Interact with and negotiate with other (possibly human) agents to achieve their goals*
- *React 'intelligently' to a dynamic and unpredictable environment*

An important 'application' of agent based systems is building them in a way so they can perform tasks that would normally require human intervention, with much of their intelligent behavior being emergent rather than preprogrammed. The study of multi-agent systems is about systems in which many intelligent agents interact with each other and possibly the user. Solving the problems associated with, and taking advantage of the opportunities offered by, distributed and unstructured environments are a major application area for intelligent and Multi-Agent systems.

Ad-hoc wireless networks

Mobile ad-hoc computing is possible because of new technologies for short-range wireless data communication such as Wireless LAN and Bluetooth. Devices with the same type of technology make the communication and collaboration between them possible, as soon as the devices come into communication range. Mobile Ad-hoc Networks (MANETs) are wireless networks consisting entirely of mobile nodes that communicate on the move without base stations. Nodes in these networks will both generate user and application traffic and carry out network control and routing protocols. MANETs are very flexible because of the dynamic topology where nodes are free to move arbitrarily and it allows a Peer-To-Peer (P2P) communication in an asynchronous manner. These networks have problems like rapidly changing connectivity, network partitions, higher error rates, collision interference, and bandwidth and power constraints together. These problems are particularly in the design of higher-level protocols such as routing and in implementing applications with Quality of Service requirements.

Agents on small devices

Small devices are portable computing devices with networking capabilities, such as a mobile phone or a PDA. Apart from the great advantage of being small and lightweight and therefore portable, small devices have some issues that need to be mentioned:

- *Reduced processing power*
- *Memory limitations*
- *Limited permanent storage with no file system generally available in phones.*
- *Limited battery life*
- *Intermittent connectivity due to areas not covered, shielded environments and the need of turning the device off to save battery*
- *High network latency and low bandwidth*
- *Small screen size*
- *Restricted input mechanism such as numeric keyboard*

Limited battery life and connectivity are the current most relevant issues in our project. As we are not depending on infrastructure but on ad-hoc network technologies the latter is this most constraining issue. Although the processing power and related specifics of small devices are nothing compared to that of current desktop computers it is currently possible to execute relatively complex software applications such as route planners on small devices and it is to be expected that these devices will become more and more powerful in the near future. The device we will base us on is the Sharp Zaurus SL-C760. On the SL-C760 a version of Linux is installed running Qtopia, X and Java. It has a maximum resolution of 640x480 pixels, a 400 MHz Intel XScale PXA255 processor, and 62 MB of RAM. This is sufficient for running JADE.

Agents in mobile ad-hoc networks

Although developers of multi-agent systems often do not try to solve the problems in mobile ad-hoc networks, they will still have to live with them to be able to build real applications. Therefore an agent platform used cannot depend too much on network availability and has too anticipated ahead for it to change or even go down at any moment in time. After an event like this the platform should be able to recover and continue working.

In mobile ad-hoc environments each of the devices may host agents offering specific services to the surrounding which can directly be used or may be combined to more complex services. Based on the traditional Directory Facilitator (DF) definition, the search of remote services is accomplished by using the concept of DF federations: DFs, besides registering services offered by local agents, may also register other local or remote DFs. This allows them to extend the search for services to remote platforms. This mechanism is not efficient, even less for mobile ad-hoc environments, e.g. because the searcher first has to find the remote DF and afterwards has to look if the services he s searching for are registered there.

Allowing registering and discovering agent services using existing ad-hoc / P2P discovery technologies, which are specifically developed for these environments, can enable a more efficient management of service descriptions and directories, as well as an efficient search and result filtering. Furthermore, once working in mobile ad-hoc environments, ad-hoc and P2P technologies can also be used as mechanisms for agent (platform) societies in the fixed network.

The development of dynamic service discovery technologies is still an ongoing research topic. It is not yet presumable which technology will finally be widely adopted and be the leading one. All of them have specific advantages and disadvantages and do not completely fit all requirements. E.g., some are not dealing well with the spontaneity of the peer communication and fast changing service provisioning, while others are not dealing well with the scalability for a huge amount of services and users.

Location awareness

Let alone running agents in mobile ad-hoc networks is already difficult enough, the goal of this research is to find out how to make agents in these networks context aware and able coordinate actions in an operating environment like a building on fire. Physical context information can be very diverse, and include local system information such as battery level or signal-noise ratio, or environmental information such as light intensity, temperature, or ambient noise. Among all, location is possibly the most relevant context element for our application area, in that it often qualifies the values of the others. For example, a temperature reading becomes more meaningful when accompanied by the identity of the room where it was sensed. The point, however, is that the actions of an application component in a mobile environment may depend on one or more of these context information values and modeling physical context becomes a necessity.

As stated above location awareness plays a crucial role for context aware agents. From this point we will focus on the possibilities for the determination of the position of nodes in mobile ad-hoc networks. If a system like GPS [Murp97] is available, each node can be easily aware of its own location, but if GPS is not available (for some or all nodes) relative positions have to be determined using other methods. There are systems and system concepts available, which use network readings such as time of arrival to calculate positions [Kita03] [Srda01] [Nicu01].

3 Model and Implementation

This section will explain how the system was built from different components, what the responsibilities of these components are, and how they are designed. First of all the system designed operates as a multi-agent system within simulation of a mobile ad-hoc network. As a basis for the simulation of a mobile ad-hoc network a system called 'Ad-hoc Simulator' (AHS) was used. Together with AHS a visualization system called Ad-hoc Visualization (AHV) was developed, which was also partially used in ManetLoc.

AHS simulates the behavior of PDAs and mobile communication in a mobile ad-hoc environment, which is visualized by AHV. The movement of mobile nodes in AHS is based upon a traffic simulation program [Kroo02], simulating cars driving in a city street-network. CityNetwork is also important for our system as the data-format of maps from the CityNetwork is used. We mainly use the part of AHS that simulates wireless adhoc connectivity between nodes when they are close together in an environment, i.e. the ad-hoc WiFi component. Also parts of AHV are used to visualize the global world state, i.e. nodes moving in the world.

The approach of building a simulation was chosen because building a real system didn't seem feasible because of time limits, availability of hardware (PDA's with ad-hoc network capabilities) and lack of a suitable ready to use platform to build the application on.

Overview

As a proof of concept for the main part of the architecture and as a possible basis for further work a simulation system will was built. This involves storing and distributing location-context information based on user observation. In the simulation maps of an environment are constructed and distributed. The processes involved are primarily based on the positions different virtual people visit and any context information they might gather, such as where exits are and determining the shortest route to one.

In our program each node does will never know its exact world coordinates and initially does not have any knowledge of the world it's in at all. I.e. there is no knowledge of absolute position and no initial map of the world. The goal for each agent-node is to construct this internal map of the world by using its own observations and sharing information with other nodes. Knowledge nodes poses, is distributed to nearby nodes. In practice this will mean nodes that are within communication range. In our case the data shared is information about halls and crossings, but can be extended to suit the usage needs.

The main goal is to be able to regenerate the map - or one that is very similar - of the virtual world the simulated users are in. The system will load a predefined CityNetwork map and will allow simulated ad-hoc nodes to travel through this map. Please note that the nodes at the same time represent simulated human users and agents. As they have no preset knowledge of this virtual world and are only able to observe information like halls, crossings and exits at their current location, i.e. nodes can only observe what human users would be able to observe, or actually a lot less.

Using a system able to build a map of an environment by only having people report what they see, and without requiring any infrastructure, human users will be able to do their tasks (such as firemen) and help build a context map of an unknown environment at the same time. By doing this jointly and requiring no extra effort from a user an unknown area can be explored and mapped with less hassle, in less time and most importantly without requiring any network infrastructure. In an overview the basic modules of this system will be:

- *Gathering data*
- *Building topological maps from user and sensory input*
- *Sharing and merging location context information between agents*
- *Providing services based on location context information*

Gathering data

Before anything else methods for gathering data on location must be in place. The collected data can then be transformed into knowledge and used for constructing and maintaining a world model. In our system data about the world is gathered in 3 ways:

- *Sensory data*: the most trivial is that the system keeps track of the distances the user travels. For this it could counts the number steps the user takes and based on that (and possibly other data) make estimates of distances covered
- *User input*: preferably by voice recognition users supply information to the system, but other methods such as a point and click system could be used as an alternative or addition to the projected use of voice recognition
- *Other agents*: information provided by other agents within communication range can be merged with the agent's own world model

Building topological maps from user and sensory input

The first goal for our agent-system is to be able to construct a local word model from data the user and sensors provide about the world. For the form of the world model we chose to construct a topological map. Considering the fact that our focus is on indoor environments and road networks and as these worlds can be relatively easily translated into a graph-like representation, the choice of generating topological maps seems to be appropriate [Remo02].

The most trivial input a user can provide for building a map of a building is to indicate that at a certain time he encounters an intersection and possibly indicating the number of paths and their directions. Although it would be helpful if the user could provide more specific information such as the wind direction (i.e. 'I can go east and west here') it is not necessary. This specific example would require user knowledge of his global orientation, which cannot be depended upon. Firstly this is because we cannot assume user caries a compass. Secondly the user is not likely to be able to orientate very well because of the crisis situation he is in. Therefore, as the user is exploring a rectilinear labyrinth, just left, right, forward and backward indications will be sufficient input.

To be able to build a map from user input only it is trivial the user provides correct information. So when he has changed direction the system should always be informed of this event. It is possible to allow the user to make errors in his observations but this is beyond the scope of this research. The only provided backup for the user making mistakes is checking for inconsistencies in the stored graph(s). If one is found in the users own observations, all his observations that occurred after the inconsistency are rendered incorrect. Summarizing, the information gathering process globally consists of two components:

- *Observations*: i.e. at time t1, when the user reports arriving at an intersection, where one can go left and right, according to his observations. In most cases it can be assumed a user can also go back, providing this isn't the first observation.
- *Actions*: i.e. at time t2 the user report he walks into a certain direction.

From all observations and actions the user reports, a graph can be constructed. This graph will represent a topological map of the world, which should be able to be displayed on the screen of the PDA and by this visualizing it to the user. To be able to generate such a map it is important to be able to estimate (relative) distances traveled. These distances can be calculated from the time between two observations, combined with the type of movement (running, walking, etc) and can be refined by using pedometer input. There are more possibilities but we prefer to use as little specialized hardware as possible. In our system we will make the assumption the user provides relatively accurate indications of distances traveled (+/- 10%).

The main problem in a graph generated from the scarce input provided by the user detecting a loop in a path and closing it to keep the map consistent. Keeping the map consistent is critical, as we need to be able to match and merge the map with the maps of other agents. Closing loops and merging maps will be described later on, first we will provide an example.

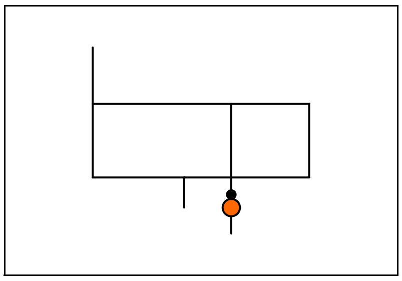


Figure 2 User in a simple world

Please imagine a user walking around in the world of [Figure 2] for this purpose. In this figure the lines indicate corridors. The user is visualized with the two dots, where the smaller indicates his orientation. The user starts at the marked location and starts exploring the world. Whenever he performs any relevant actions or does an observation the user will report it. In an actual system, providing this information should be possible via multiple ways, to allow operating under many different conditions for instance. Examples of such input methods are voice recognition, text input, pointing/clicking on a device and possibly even pointing a sensor (like a) camera at a certain object or location. Regardless of the input method used we will assume all user input is 'perfect', so the user will never forget to mention a fact like arriving at an intersection. The agent can translate information provided by the user into vertex and edge knowledge.

Closing the loop

If a user would travel long enough in a building he will eventually always return to a location that was visited before from a different direction. The process described in the example will then result in loop in the graph, which should be detected and closed. A correctly detected closed loop is very valuable information, as it is required to make a map consistent. Consistent maps are required by our system to be able to match and merge one map with another. The question is how to detect such a closed loop [Save04]. As an example of closing loops we take an even simpler square shaped world. Applied to this square world the process described above might result in the following 'map', where the two upper-left intersections, should be recognized as one and the same, but at the moment we still have 5 vertices in the graph where there are only 4 intersections in the world [Figure 3].

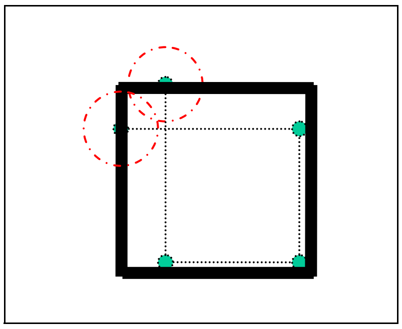


Figure 3 Open loop

Before being able to close a loop we should be able to detect and build hypotheses concerning possible loops. For this we roughly follow a procedure in which is checked for each new vertex if there is a vertex nearby that might be closing a loop. If we can find two matching vertexes a loop hypothesis is started.

To make the graph consistent when two matching vertexes are found, small adjustments can be made to the endpoints of edges. Whenever a loop hypothesis has been formed we can start testing it by comparing edge lengths of the supposed loop with new measurements. These measurements will result in accepting or rejecting the loop.

A danger is that incorrect hypotheses may be accepted as correct or vice versa. A way to deal with this is to keep multiple hypotheses and never commit to any one of them, i.e. retain the hypothesis that the loop has not been closed. Hypotheses can then be accepted or rejected later on when new observations have been added and map data was received from other agents. If other agents confirm the conclusions about a specific part of the world the more likely it is to be correct. On the other hand, agents have different opinions about this particular part of the world, decreases the likelihood of the hypothesis being correct.

It is also preferred to be able to deal with mapping errors such as incorrect hypotheses, input errors and dynamic worlds. In a dynamic world more recent information will intuitively have more meaning than older. Dealing with this is incorporated into our system by decaying the influence of individual readings over time [Thru97]. New information inconsistent with the old information could cause the old to be discarded. Though our system will not work in a dynamic world, still newer information is preferred over older. In our simulation we will only use time stamped data for repairing incorrect hypotheses and input errors, but in future work it could be adapted to be used in dynamic worlds.

Sharing and merging location context information

Our agents are operating in a mobile ad-hoc networking environment in which it is likely they encounter other agents every now and then. When an agent detects one or more (compatible) agent(s) within its communication range, they can share the knowledge they have about the world and possibly are able to merge this [Figure 4] [Butl01][Dede00] [Rike94]. Each agent will attempt to merge information from other agents for themselves. Consequently it is possible for two agents to come to different conclusions. This makes it possible for PDA's to have different agent software (versions) running or to be in a different processing mode. The only requirement is that the messages they send out are compatible.

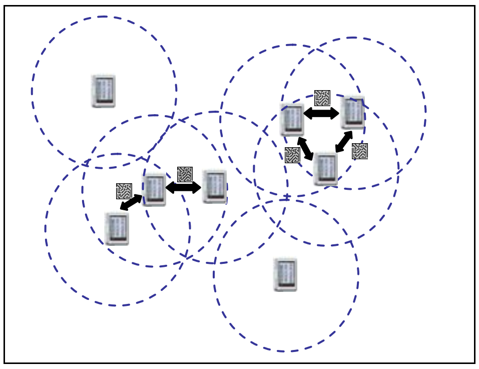


Figure 4 Agents in MANETs sharing maps

Similar to the internally stored world models, maps are shared in the form of graphs. They are assumed to be consistent, but not exact and do not always have a common reference frame. If two or more users have explored overlapping regions of an environment their agents should have topological maps that have common sub graphs [Vali01][Bunk00] with identical structure. Since having a common sub graph it is possible to find a reference frame and merge the two maps into one.

In order to be able to merge two maps, we first need to match the maps together, building hypothesis and choosing the correct one (i.e. the best match). A hypothesis is a possibly rotated sub graph that the two maps have in common. There is a chance the process described below does not supply a (large enough) hypothesis. If this is the case and not enough vertices can be matched to make a good hypothesis yet, the map received is stored internally. In that case we can try the matching process later on when a more complete world model is available. The used algorithm for matching two maps consists of three phases:

- *Vertex matching*
- *Growing hypotheses*
- *Combining hypotheses*

Vertex matching

The first step taken in matching is building a list of all vertices that match each other in the two maps. Two vertices only match if they have the same edge directions. This is also the case if a vertex needs to be rotated to match, which is also stored. We expect exactly known attributes vertices, such as the type of the vertices to match perfectly. However, attributes that are subject to measurement error can be compared with a similarity test. In the case of our simulation we don't have any fuzzy variables of a vertex, but in a real environment or an extension of our simulation some could occur.

Growing hypotheses

After having built lists of matching vertices we grow matches by testing corresponding pairs of edges leaving the paired vertices. If the edges are compatible and the vertices at the ends are also compatible, they are added to the hypothesis. If the edges or vertices are incompatible, the entire hypothesis is rejected. The vertices are tested with the same type of criteria and similarity tests used to form the initial pair. Edges may also have both exactly and inexactly known attributes. In our system, they have their path length compared with a similarity test.

Our hypotheses are the unique matches surviving the growing process. Duplicate hypotheses are avoided by keeping a table of vertex pairings. When vertices are paired during the growth phase, the corresponding entry is marked in the table. This entry is then ineligible as an initial pairing of vertices.

A sub graph in one map can be matched to multiple sub graphs in the other under separate hypotheses, but a pair of matched vertices with a given edge correspondence can appear in only one hypothesis. The matching and growing process is repeated until all valid vertex pairings are examined. Please note that if we would be working with imperfect user input - which is not the case in this concept - a procedure filtering noise should be added. When able to match enough common vertices and edges, and if there are a minimal amount of conflicting vertices and edges, the conflicts can be discarded and the hypothesis accepted.

Combining hypotheses
If successful the hypothesis growing process described above results in list of possibly multiple hypotheses within the same rotation. From these hypotheses one has to be selected. Before this takes place it is possible that if such a list contains more than one hypothesis, some of these entries are consistent with each other. These hypotheses are then combined with each other into one larger hypothesis cluster.

After the system has chosen a hypothesis cluster, the next step is to merge (or flatten) the two maps into one single map. Estimates of path lengths can be updated by combining the measurements from the two maps for corresponding edges. The edge orientations at the corresponding vertices can be similarly merged. Parts of one map not present in the other should be added. The merging process is globally performed in four steps [Figure 5]:

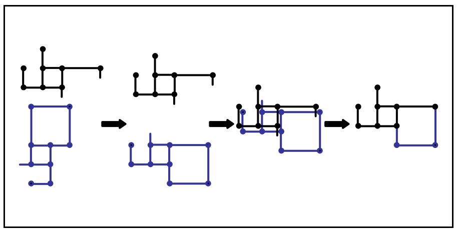


Figure 5 Map merging process

- *Rotate the received map so its orientation matches the local node's map*
- *Shift the rotated map so its coordinates match the local agent's map*
- *Add any new vertexes from the rotated and shifted map to the local node's map*
- *Connect everything together (update edge lengths, check for inconsistency's etc.)*

Please note that the choices made, may later turn out to be incorrect. For example, early in the process of exploring a self-similar environment, a user might seem to be exploring the same area when in fact they are exploring similar but distinct areas. To protect against such situations, it is remembered from whom and when the new parts of the map are received. This makes it possible for discovered inconsistencies to be removed or corrected later on without discarding the whole map.

After agents have merged their maps once two nodes may later exchange maps information again. This is also the case if they weren't able to merge their knowledge, but just exchanged their knowledge before. An incremental update can save a substantial amount of computation and bandwidth. This is possible with minimal bookkeeping effort. Each agent must maintain timestamps so that only new and modified vertices and edges since the last update are exchanged. If a successful merge took place before and if it still appears to be correct we already have a hypothesis and can computationally easy merge the new information. If the agents were not able to merge their maps before the previously received map is merely be updated with the new information, and another merge attempt can be made including the new world information. The computational and network traffic savings of incremental updating can be significant if implemented correctly. The former highly depends on the amount of information retained from the original map merging.

User guidance

When an agent running on a user's PDA gathers knowledge about the world, the knowledge can be used to provide services such as advice to the user. One of the possibilities is guidance of the user. Here one can think of the possibility of guiding the user to a certain location such as the nearest exit or an unexplored area of the map. Both options are implemented in our proof of concept.

Nearest exit
When a user indicates he wants to leave the area as soon as possible the system can instruct the user how to walk, calculated using a shortest path algorithm (i.e. Dijkstra, A*, Ant based, etc.) . The result of a request for guidance to our system is indicated, by giving the path to the nearest exit a specific color.

Unexplored area
When the goal of the user is to (help) explore the world the system can instruct the user where the user should go to help optimize the process. As this is very similar to instructing the user to a nearest exit the same type of shortest path algorithms can be used.

It is trivial there are many more services that can provided to the users by agents with knowledge such as in our system (although thinking of and implementing useful services is not). In the context of the crisis management project, knowledge could be gathered about crisis indicators (fire, smoke, etc) and translated into some sort of scenario (e.g. the building is on fire, south side is still clear to pass). It is not unthinkable that having this type of knowledge available in an agent network could even be used to coordinate the actions of individuals and groups.

4 System Tests

The simulation was first run on a 10 and 30 nodes map, first with just one agent exploring the world automatically using nearest unexplored area navigation, and later with more agents who were simultaneously started. The time it took an agent to find the complete map was measured and also was always checked if the output was correct

Map	#agents	Avg. units
10.map	1	1721
10.map	2	1125
10.map	3	1100
10.map	4	1112
10.map	5	1060
30.map	1	9701
30.map	2	8569
30.map	3	6227
30.map	4	5301
30.map	5	6017

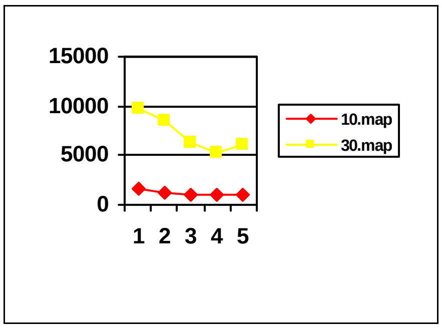


Figure 6 Test results, finding complete map

The results clearly show that the process of sharing and merging maps has an effect, the larger the map the larger the effect and also the more agents the more gained. Agents starting in a map that was already explored by others logically have the most gain; after they have explored a small part of the world they can simply merge the large map parts with their own. Please note that the test result would probably be significantly lower if the AHS network code wouldn't suffer from deadlock issues.

In another test the 30 intersections world was pre-explored completely by 5 agents after whom a fresh agent was added. The new agent received the complete map from multiple agents and was able to find the complete correct map within 536 distance units traveled. Considering it takes 9701 units on average for an agent to explore this world on its own this is a considerable gain (in this specific case 18 times faster).

5 Conclusions

Our experiments in a simulated world show that it is very well possible to distribute, and merge world knowledge in a mobile ad-hoc multi-agent environment. Even in such an environment with limited communication possibilities our test results showed there is a significant gain found when solving a mapping problem with multiple distributed agents. As expected, the larger the map the better results on how useful distributing and merging partial maps is. Although with the fact that simulation runs on a single machine comes that there are limits because of processing power. This is an important issue still encountered in our simulation system. Though when calculations would not be performed on one machine anymore, but on one for each agent, it should not be a problem anymore. So we anticipate scalability will not be a direct problem, should the simulation be translated into a real life distributed system.

Bibliography

[Comb] *Combined Systems group*, http://combined.decis.nl

[Boel04] *A Communication Layer for Distributed Decision Making*, J.L.Boehlé, 2004, Delft University of Technology, Erasmus University Rotterdam

[Aaai] *Agents*, The American Association for Artificial Intelligence, 2000 - 2005

[Jade] *Java Agent Development Framework*, http://jade.tilab.com/

[Remo02] *Towards a General Theory of Topological Maps*, E. Remolina, B. Kuipers, 2002, University of Texas at Austin

[Thru97] *Learning metric-topological maps for indoor mobile robot navigation*, S. Thrun, 1997, Carnegie Mellon University

[Murp97] *Global Positioning Systems, A TechnicalAssessment Paper*,Lotta Danielsson-Murphy, Thaddeus Murphy , 1997

[Kita03] *Design of WiPS: WLAN-Based Indoor Positioning System*, Teruaki Kitasuka Tsuneo Nakanishi, and Akira Fukuda, 2003, Kyushu University

[Srda01] *GPS-free positioning in mobile Ad-Hoc networks*, Srdan Capkun, Maher Hamdi, Jean Pierre Hubaux, 2001, Ecole Polytechnique Federale de Lausanne

[Nicu01] *Ad-hoc Positioning System (APS) Using AOA*, D. Niculescu, B. Nath, Rutgers University

[Save04] *Loop-Closing and Planarity in Topological Map-Building*, F. Savelli, B. Kuipers, 2004, Universit`a di Roma "La Sapienza", University of Texas at Austin

[Butl01] *Distributed coverage of rectilinear environments*, Z.Butler, A Rizzi, R. Hollis, 2001, Carnegie Mellon University

[Dede00] *Landmark-based matching algorithm for cooperative mapping by autonomous robots*, G. Dedeoglu,G. Sukhatme, 2000, University of Southern California

[Rike94] *Autonomous agent map construction in unknown enclosed environments*, K. Doty, S. Seed, 1994, University of Florida

[Vali01] *Subgraph Isomorphism and Related Problems*, G. Valiente, 2001, Technical University of Catalonia

[Bunk00] *Mean and maximum common subgraph of two graphs*, H. Bunke, A. Kandel, 2000, University of Bern, University of South Florida