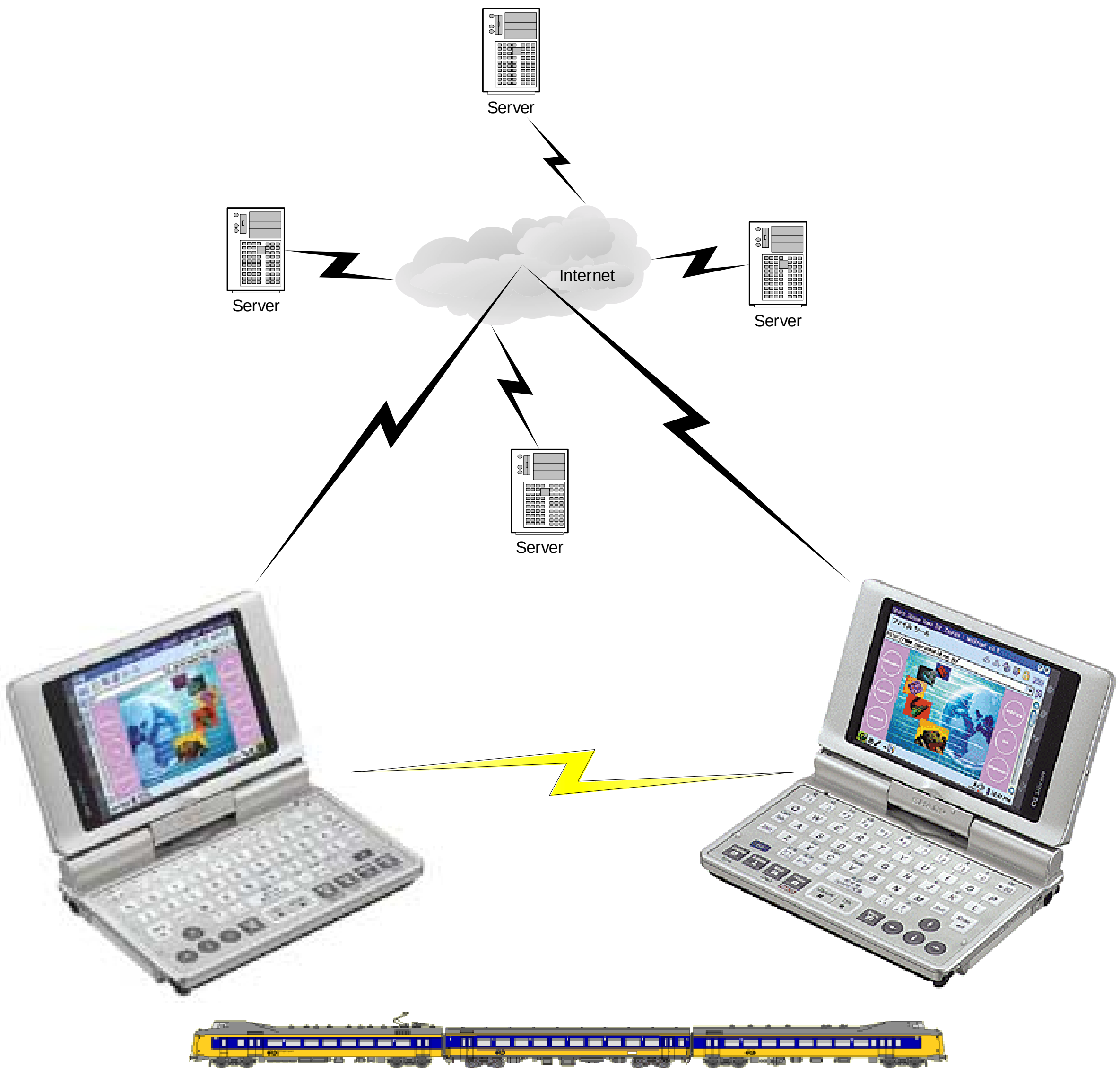
Personal Intelligent Travel Assistant

A distributed approach



Master's thesis of Martijn Beelen

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science

July 2004

TUDelft
Delft University of Technology

Beelen, Martijn (M.J.Beelen@ewi.tudelft.nl)
Master's thesis, July 2004
Personal Intelligent Travel Assistant: a distributed approach

Delft University of Technology, The Netherlands
Faculty of Electrical Engineering, Mathematics and Computer Science
Knowledge Based Systems Group

Keywords: distributed system, intelligent agent, travel assistant, public transport, dynamic route planning, multimodal interface

Preface

This master's thesis describes the research and development I have done to graduate at the Knowledge Based Systems Group, headed by Prof. dr. H. Kopperlaar at Delft University of Technology. The research was done within the Netherlands Research School for Transportation, Infrastructure and Logistics (TRAIL), as part of the Seamless Multi Modal Transportation research. The goal of the research is to bring the realisation of a next generation of public transportations, with optimal route guidance and demand (or user) driven services, one step further. This is done by designing and implementing a prototype of a system for guidance of travellers in the Dutch public transportation and a simulation environment for further research on the topic. Travellers do not need to worry about planning their route, combining information on occurring delays and rescheduling any longer, as the system does this automatically after a destination has been specified.

Project

After a visit to Leon Rothkrantz to explore different possibilities for my master's thesis, the PITA project caught my interest immediately. As a student, the delays of the NS have interested me a lot an I have been thinking about ways to prevent these from happening, however this has proven to be an endless mindgame. For peace of mind it is better to accept the fact that delays occur on the railway network and thus the best way to live with them is to know about them. If users of the public transportation are informed about delays in an appropiate way, it is easier to accept the fact that you are going to be late and you can also take actions accordingly (such as informing people who otherwise would have been waiting).

As such, the PITA project looked like an exciting challenge. The first problems were however to get relevant data on the train schedule from NS. As this information is sold commercially by NS, it was not very easy to acquire this and so the decision was made to copy the train schedule of a working day from the NS website, using a brute force approach with a HTML parser. The next problem was finding routes within this enourmous amount of data. By first implementing a straight forward search algorithm (which I later found to be an implementation of Moore's algorithm), this was done. Then the PhD thesis of Eduard Tulp caught my attention and the interesting features (including train changes and user preferences in the search) made me replace my implementation with a new implementation of the DYNET algorithm, of course adapted to the dynamic situation using actual delays within the route planning.

Combining these systems into a single distributed system has proven to be a real challenge. A distributed system, especially using autonomous agents, is not as straight forward to develop as a procedural application and definitely not easy to debug. However in the end, the prototype was realised, also running on the Zaurus handheld and providing the user with an optimal advise to get to his destination using the public transportation.

Acknowledgements

First of all I would like to thank Leon Rothkrantz for our inspiring (two) weekly sessions, which really challenged me to realise the PITA system. The amount of inspiring, creative ideas Leon gave me every time we discussed the progress of the project, always gave me a new direction to find a solution to the problems I was facing or at least made me think of new functions or possibilities for the system.

Secondly I would like to thank Wim Tiwon for his help with installing, running and compiling of software for the Zaurus handheld. Thanks to Wim I have been able to compile a speech synthesis engine for the Zaurus, which can form a starting point in the development of a true multi modal interface for the handheld.

Then I would like to thank Wouter Pasman and Jan Misker for introducing me to Jade and Jadex and supporting me to get to understand the basics of ontologies and agent communications. Also I would like to thank Lars Braubach and Alexander Pokahr of the Distributed Systems and Information Systems Group at the University of Hamburg for their help and feedback on the Jadex plan scheduler.

Lastly I would like to thank my family, my fellow resident Herbert van Willenswaard and Steve Hardy and John van der Kamp for their moral support and having to listen to my enthousiastic stories each time I realised another small part of the system.

Martijn Beelen,
Delft, July 2004.

Summary

In The Netherlands one million people use public transportation every working day. The railroad network is overloaded and minor incidents lead to a large amounts of consecutive delays. Travellers are informed of delays by signs along the railway track, but there is no system, helping travellers to find the optimal route by public transportation, which also uses the delay information. The Personal Intelligent Travel Assistant provides the user with a personalised, up-to-date advice on a handheld device, which takes into account the actual delays.

The travellers themselves provide information about the delays of trains, using positioning technology (like GPS) to estimate whether a user is experiencing a delay. Furthermore the NS website provides information on delays for 83 stations in the Netherlands. This information can be imported into the prototype. Using the actual delay information, each user gets an optimal route advice, which also takes into account his personal preferences.

To find optimal routes, considering train changes and the user preferences, the DYNET algorithm of Eduard Tulp was selected and implemented. By extending the static information on the train schedule with dynamic information on delays, a prototype of a dynamic route guidance system was designed and implemented, which runs on the Sharp Zaurus handheld and a distributed server backend.

A model was designed for distributing the delay data over a set of servers, which can also be used for simulation purposes. Furthermore interfaces were designed for manually adding delays, changing the system time and speeding up the simulation. The system was realised by agents modelled after the Beliefs, Desires and Intentions agent model, using Jade combined with Jadex, forming a FIPA compliant distributed agent platform.

Table of contents

Chapter 1: Introduction

This study explores the possibilities of an application on a handheld device, providing users of public transportation with an up-to-date advice based on the current delay situation on the railroad network in The Netherlands. In the first part of this chapter the current situation and the motivation for the study will be explored. The second part will present the context in which this research was done. The next part will give an overview of some interesting aspects of the Personal Intelligent Travel Assistant. After that, the goals for this project will be presented. The last section gives an overview of the structure of this report.

1.1 Problem setting

The Netherlands has a very extensive public transportations network, consisting of taxi's, busses, trams, metros and trains. Especially in the Randstad (the area between the 4 largest cities: Amsterdam, Utrecht, Rotterdam and The Hague) the services are very intensive. With busses stopping about every mile, trams and metros riding every 10 minutes and a 24-hour train service, a complete package of public transportation is offered to all people.

Every working day, the Dutch Railways (Nederlandse Spoowegen, NS) transports about 1 million people by arranging about 6000 train rides between 400 stations. Connections between two stations are made varying from once every hour to eight times every hour for every station. Every year the train schedule is designed to optimally fit the demand for public transportation at every station. However, since the demand is very high, the network is flooding during working days in the Randstad.

The flooding of the network is probably one of the main causes for consecutive delays. If one train gets delayed during working hours, this has a dramatic impact on the total train schedule. Because the network is flooded, the rescheduling of the delayed train directly impacts also other trains riding the same track or crossing the track of the delayed train. This gives a snowball-effect on the delays within the Randstad on working days.

On the other hand NS was the first railroad company, starting 70 years ago, to use a fixed hourly schedule, with trains riding a certain track every hour again on the same minute of the hour, in a certain direction. This has created high expectations among the people travelling by train and depending on certain fixed train changes at certain moments. This has lead to the current situation in which during working hours the network is flooded and people are experiencing delays very often and blaming NS for the delays.

So nowadays there exists a continuous tension between people counting on public transportation at a certain minute every hour and NS coping with the flooded network. A solution for a better experience for the users of public transportation could be to provide users with better information about train delays. Currently users are mainly informed of delays at the moment they arrive at the station and see the delays (which can increase every minute) on the signs along the track. As of March 25th, 2004 it is also possible to see actual departure times on the Internet site of NS and on an I-mode mobile phone for about 80 stations.

If a user experiences a delay he or she can decide to wait or to try to find another train route giving a faster transportation to their destination. This leads to a lot of users running between tracks to adapt their travelling plans and making it to another train at the last moment, while trying to solve their travelling problems. This does not lead to a comfortable travelling experience either. It would be a lot better if the user would get a notification on arrival at the station telling him which train to take at that moment, considering the actual delays and giving him a personalised advice to get to his destination.

The Personal Intelligent Travel Assistant (PITA) provides this service. Running on a handheld device, the PITA will consult the train schedule combined with available delay information and rerouting information and determine the best travelling option for the user, giving him an optimal, personalised advice for how to use the public transportsystem to get to his destination.

1.2 Seamless Multimodal Mobility

This research was conducted as part of the "Seamless Multimodal Mobility" (SMM) research program of the Netherlands Research School for TRAnsport, Infrastructure and Logistics [TRAIL]. This section will introduce the research school and the research program.

1.2.1 TRAIL

TRAIL is the Dutch academic research institution targeting world-wide developments in transport, infrastructure and logistics and combining top-level education and research. It is a joint initiative of the Delft University of Technology, the Erasmus University of Rotterdam and the University of Groningen. TRAIL is accredited by the Royal Netherlands Academy of Arts and Science. The mission statement of TRAIL is as follows:

"First, to educate high-level scientific researchers and research-leaders with a broad background and of high standing who will contribute scientific innovations to the solution of scientific, business and societal problems in the field of transport, infrastructure and logistics. Second, to build up and transfer new, ground-breaking interdisciplinary knowledge and insights at the highest international level in the fields of transport, logistics and associated infrastructures, by adopting an integrated and solution oriented approach aimed at supporting an economically viable, sustainable and accessible society. "

1.2.2 Seamless Multimodal Mobility

The research program "Seamless Multimodal Mobility" of TRAIL at Delft University of Technology is aimed at developing new public transport services for the future. The traveller will be transported from door to door without having to spend time on finding the best route to his destination or worrying about delays or calamities while using public transportation. Applying the results of the SMM program to current public transport systems will result in drastic changes in these transport services as new transport modalities and transfer nodes will be introduced and transport services will be provided on a more demand-driven and less fixed basis.

1.2.3 Research at TU Delft

At TU Delft several people have contributed to the research on the PITA subject. The PITA is the subject of the PhD research of Robert van Vark (see [Vark]) and he has written several papers on the subject. Also Gerritjan Eggenkamp (in [Egge01]) has designed a first version of the PITA system and also designed a multi modal route planner. Alexandru Dovlecel (in [Dovl03]) has described the PITA from a web service perspective and Erdal Yildiz has designed a dynamic traveller guidance system (in [Yild00]). Next to that, as part of the CACTUS (Context Aware Communication, Terminal and USer) project (see [CACTUS]) and the Combined systems project (see [Combined]), research is being done on the field of ad-hoc networking and intelligent mobile devices and user interfaces.

1.3 Personal Intelligent Travel Assistant

Within the research program SMM, research is carried out into the Personal Intelligent Travel Assistant (PITA). The PITA is an application running on a handheld device, providing communication between travellers and public transportations companies. It provides the user with an up-to-date personalised advice on what public transportation to use, based on current, available delay information. The public transportations companies can get a better insight in the traveller flows, giving them the possibility to optimise the scheduling of trains and carriages. A future application is the reservation of public transportation through the PITA system. Some interesting aspects of the system are described in the rest of this section.

1.3.1 Travel Assistant

The PITA application will continuously provide the user with the best advice about what public transportation to use to reach his destination, depending on his travelling preferences, his current location and available delay information. The advice is available before the user starts to travel and the advice will be updated continuously while the user travels, depending on delay information or other updated dynamic information becoming available.

1.3.2 Multi modal Intelligent Assistant

The PITA system will monitor the delay information and determine the significance of occuring delays for the user. If the delays affect the journey of the user, the system will generate alternative routes, estimate the travelling time and the risks on those routes and select the best possible option for the user to use. This releases the user from scheduling his journey and lets the user blindly trust the generated personal advice. Finally it is possible to interact with the PITA system in a multi modal fashion. This gives the user the possibility to speak to the system when in a hurry, or to click and point on the handheld when seated in a noisy environment. The system can also react multi modally, by displaying the advice or read it out loud to the user, using speech synthesis.

1.3.3 Personal Assistant

Not only will the application run on a personal device of the user, the application will also represent the user to the rest of the system. The system will learn the preferences of the user from the day to day use in order to optimise the generated advice (i.e. whether the user wants to run for a train or rather wait 10 minutes on the next train, whether the user can make a train change within 3 minutes on Rotterdam Central Station, etc.). In fact, the PITA handheld will be a copy of the user, functioning on the basis of his preferences and responsible for all requests to the system. So the user only needs to tell the system where to go and the PITA will take over everything else and only supply the user with actions to take (i.e. “take train number 1010 at platform 8b”, etc.).

1.3.4 User localisation for delay information

Crucial to the success of the PITA, is the availability of accurate delay information and other information affecting the journey of the user. In the best case, the information will be supplied by the public transport companies (i.e. NS and/or 9292OV) and made available for usage by the system. However, because this data is very distributed and currently only locally available to NS and because of other political arguments (NS does not want to supply everybody with details about the delays, because they get funds by the government for not having delays), the system has to be able to function without this information being made available.

Since all users of the system have their own handheld with them, the system will use positioning technology within the handheld to determine the position of the user on the basis of the position of the handheld. The position of the handheld is made available to the rest of the system, which can determine the delay the user is experiencing. This information is not only used to optimise the advice for the user himself, but also to advise other users. This way the PITA system can generate the delay information itself.

1.3.5 Ad-hoc networking

The PITA handheld has to connect to the rest of the system and several other information sources to combine the information in an optimal advice for the user. Depending on the devices (i.e. GSM, Wireless LAN device or GPS device) a user carries with him, the system will choose the optimal connections for acquiring information about the user’s location, the train schedule and possible delays. Once the user has configured the available connections, the system will optimise bandwidth usage and minimise the costs. Also, the possibility of ad-hoc networking between the handheld devices and sharing available information in an ad-hoc network (based on Wireless LAN) is used.

1.3.6 Emergent behaviour

Running autonomous assistants on the handhelds, that connect to each other over ad-hoc wireless networks and use mobile Internet connections to acquire data, gives a very interesting application in which each of the assistants has to acquire relevant data for the guidance of his user. The assistant has the choice to connect to the Internet himself or try to find other users who already have up-to-date information on a journey to his destination. Since each of the devices itself generates delay information on a specific train, the assistants are able to exchange information on delays and route plans. This leads to a distribution of data, driven by the demand of the users of the system in stead of centralised on a server.

1.3.7 Traveller flow planning

For NS it is very important to have accurate predictions about traveller flows across the different stations in The Netherlands. This is essential to schedule enough carriages on the different trains to be able to transport the travellers. Currently NS uses the ticket sales and passenger counting, to get an idea of how many people use a certain connection at a certain time of the day. This data is also used for designing the train schedule every year. Special arrangements are made for events (like football matches or concerts), where trains are scheduled in advance. With the PITA system, the NS will have another instrument to predict and view current passenger flows, because the system knows the position of its users and can give NS an overview of where PITA users are at any time of the day. This way NS will also have a better insight into the passenger flows, which could lead to better scheduled public transportation. Whenever the public transportation becomes more flexible, the PITA system can also be used to reserve seats in public transportation in advance and NS can provide more demand-oriented public transportations.

Figure 1.1 An artistic impression of the PITA system

1.4 Goals

The previous sections describe the functionality of the Personal Intelligent Travel Assistant. The main goal for this project is to design and implement a distributed application which provides the PITA functionality. The distributed approach is chosen to develop an application which is able to run on servers, handhelds and desktops and to improve scalability and stability of the application. To realise such a system, the use of Artificial Intelligence techniques and especially agent techniques go without saying. Therefore different agent platforms and agent communication systems have to be studied first. The second goal of this project is to design a model of a distributed PITA application. Thereafter a first prototype of the model has to be implemented. Next a simulation environment has to be designed and implemented and lastly some test experiments with the simulation environment have to be performed.

To summarize, the goals are listed below:

1. Literature survey on agent platforms and agent communication systems
    - Description of state of art agent platform technologies
    - Discuss the advantages and disadvantages of each platform
    - Select a suitable agent platform
2. Design a model of the distributed PITA system, using handheld devices
    - Design a model for distributing data and processing power
    - Use available network technologies for connecting the different components
    - Assume current mobile devices as operating environment of the application
    - Design an UML model for implementation of the system
3. Implement a first prototype of the PITA system
    - Acquire information on train schedules and delays
    - Implement a suitable routing algorithm
    - Implement a prototype which runs on a handheld device, preferably the Sharp SL-C860 Zaurus
4. Design and implement a PITA simulation environment
    - Design and implement a delay simulator
    - Automatically import delays from the NS website
    - Design and implement a graphical interface showing moving trains
5. Perform PITA test experiments
    - Perform experiments to determine the functioning of the system
    - Describe a user scenario showing different aspects of the system

The goals of this project therefore focus mainly on designing and implementing a PITA system from a technical point of view. Realising a simulation environment which can be used for other projects and as a demonstration application has therefore gotten a higher priority above other aspects, such as user or simulation test experiments. Also Human-Computer Interaction was not a main subject of this project and could be studied in another thesis project. Chapter 4 describes the subject of this thesis in more detail.

1.5 Report structure

First some background information about agent platforms and agent communication systems is presented in chapter 2. Available standards and implementations are discussed. In chapter 3 the actual used tools and their limitations are introduced. Chapter 4 describes the PITA problem, the assumptions and requirements and introduces the model for the design of the application. In chapter 5 the design of the system is discussed, followed by the implementation and implementation issues in chapter 6. Chapter 7 discusses a user scenario in which a jouney was simulated which shows several important aspects of the system. In chapter 8 the simulation environment which was developed as part of this thesis project is described. In chapter 9 the conclusions recommendations and suggestions for future developments are discussed.

Chapter 2: Literature

This chapter gives a survey of current agent technology and agent platforms. First, the GALAXY architecture is described, followed by the Open Agent Architecture. The third part of this chapter describes the FIPA architecture and available implementations of the FIPA standard. Then an overview of the platforms is presented and a platform is selected for this study. Finally the COUGAAR architecture is described.

2.1 GALAXY

GALAXY is an architecture developed by the Spoken Language Systems group of the MIT Laboratory for Computer Science (see [SHLPSZ98]). The main focus of the GALAXY project is to build a framework for integrating speech (recognition, understanding and synthesis) in an application. The architecture makes use of light-weight clients with specialized servers for the computationally heavy tasks. As can be seen in figure 2.1, a central role in the GALAXY system is taken by the programmable Hub, which controls the flow of data between various clients and servers and retains the state and history of the current conversation.

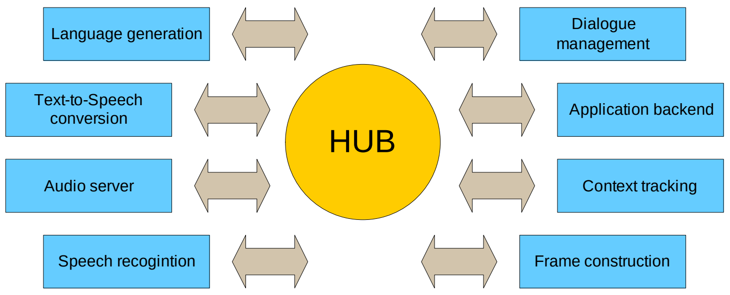


Figure 2.1 An overview of the Galaxy architecture, with a central role for the programmable hub

GALAXY has been released as an Open Source package, written in C and C++, with Java bindings. The Galaxy Communicator distribution is available in a Windows, a Sparc Solaris and a Linux version, however some features are available only on specific platforms. Especially parts of the Open Source Toolkit, containing implementations of Communicator-compliant dialogue system components, are restricted to specific platforms.

2.2 Open Agent Architecture

The Open Agent Architecture is “a framework for integrating a community of heterogeneous software agents in a distributed environment”, developed at the Artificial Intelligence centre of the Stanford Research Institute (SRI) (see [MCM99]). The main focus of the research is towards distributed communities of agents, where agents communicate with each other by means of Interagent Communication Language (ICL).

The Open Agent Architecture does not only specify the communication language between agents, it also specifies a model for how agents should communicate with each other. As shown in figure 2.2, this communication model uses a facilitator agent in a central role: “a specialized server agent that coordinates the activities of agents for the purpose of achieving higher-level, often complex problem-solving objectives”.

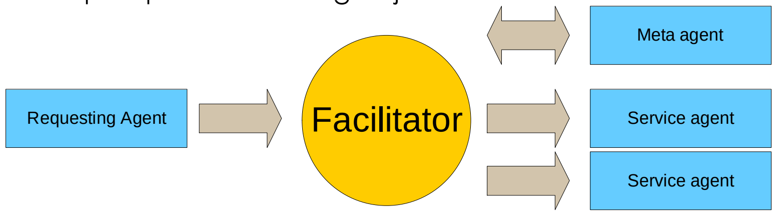


Figure 2.2 An overview of the Open Agent Architecture with a Facilitator for communication between agents

The facilitator agent combines advice or constraints from a requesting agent, specifications provided by service agents, domain-independent strategies for coordinating inter-agent cooperation and domain-relevant knowledge provided by meta-agents, in order to meet the requested objectives. Agents query the Facilitator through an unified interface (a single function-call), giving (Prolog-like) parameters for their request. The Facilitator will look at all registered agents and their capabilities and by means of some strategy forward the request to some agent(s). All requests go through the Facilitator and never directly between two agents. The Open Agent libraries are available in a variety of languages (among which Java, Prolog and C) and can be used on different platforms (e.g. Windows, Linux and Sun/Solaris).

OAA is distributed under the OAA Community License, which permits unrestricted use for government, research and other non-commercial purposes. With respect to the commercial use of OAA, SRI has thought about creating a Pro-version, which would be a more commercial-quality instantiation of the OAA framework, but currently no further information is available.

2.3 FIPA

The Foundation for Intelligent Physical Agents (FIPA) is an international organisation dedicated to developing specifications supporting interoperability among agents and agent-based applications (see [FIPA23]). By open discussions among members (companies and universities), FIPA tries to establish a common standard for agents and agent applications. FIPA does not specify how the specifications should be implemented, but only specifies the functional behaviour which the implementation should have.

The basis of the FIPA specification is formed by the Agent Management Reference Model, as shown in figure 2.3.

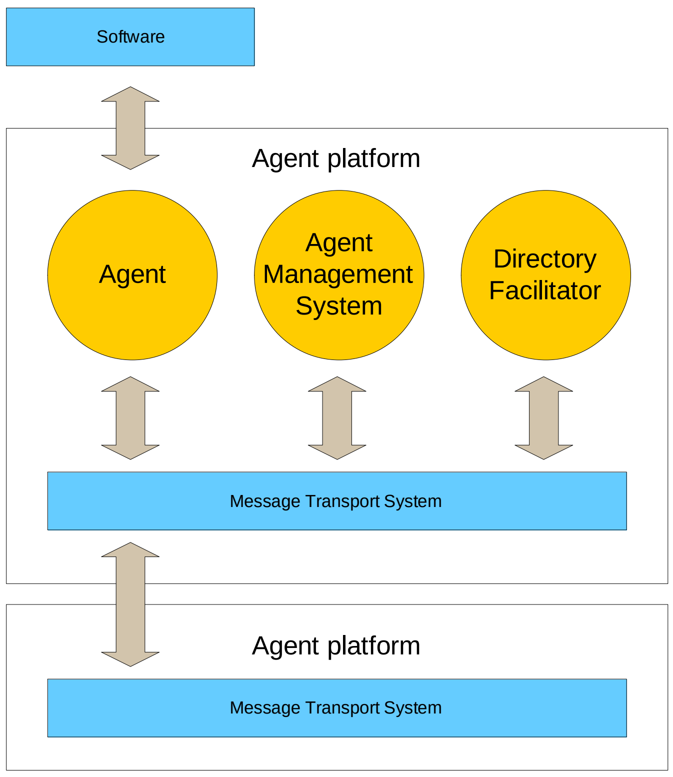


Figure 2.3 The FIPA Agent Management Reference Model

An Agent Platform consists of a Agent Management System (AMS), one or more Message Transport Systems (MTS), optionally one or more Directory Facilitators (DF) and Agents. The Agents are computational processes that implement the autonomous, communicating functionality of an application. The AMS maintains a directory of Agent identifiers and transport addresses of all registered Agents. Each Agent must register with the AMS in order to get a valid identifier. The DF is an optional component of the Agent Platform, which maintains a list of services offered by Agents. Agents may register their services with the DF and can query the DF for services offered by other Agents. The Message Transport System offers communication possibilities between different Agent Platforms.

Agents communicate with each other using Agent Communication Language (ACL), which is specified by FIPA, but which can be extended by individual programmers to fit their specific needs. FIPA has defined several ontologies, which specify the interpretation of specific ACL-messages. By designing their own ontologies, programmers can extend the capabilities of Agents and implement specific functionality.

FIPA produces open and thoroughly documented standards, but does not provide a (reference) implementation for the standard. Currently the three most important implementations of the FIPA standard are the Tryllian ADK, FIPA-OS and JADE, which are discussed in the following sections.

2.3.1 Tryllian ADK

Tryllian is a Dutch commercial organisation, which developed the Tryllian Agent Development Kit (ADK) (see [Tryllian]). Based on several open-source components, the ADK can be downloaded and used freely for non-commercial applications. The ADK is fully implemented in Java and supports the FIPA standard partly with the Tryllian FIPA Dialect. The ADK also supports SOAP for messaging and has several security and encryption features. Tryllian mainly focuses on integration with J2EE web-services for industrial applications.

2.3.2 FIPA-OS

FIPA-OS is an Open Source implementation of the FIPA standard, originally released by Nortel Networks in 1999 as the first royalty-free implementation of the standard (see [FIPA-OS]). Two different implementations are available, a Standard edition and a Micro edition (for PDA's like Compaq iPaq). FIPA-OS is also fully implemented in Java and FIPA compliant. The Micro edition can run in 2 different modes on PersonalJava compatible devices. Either the handheld device can run a complete FIPA-platform with its own AMS, DF and Agents, or the handheld device can be part of another FIPA-platform as shown in figure 2.4.

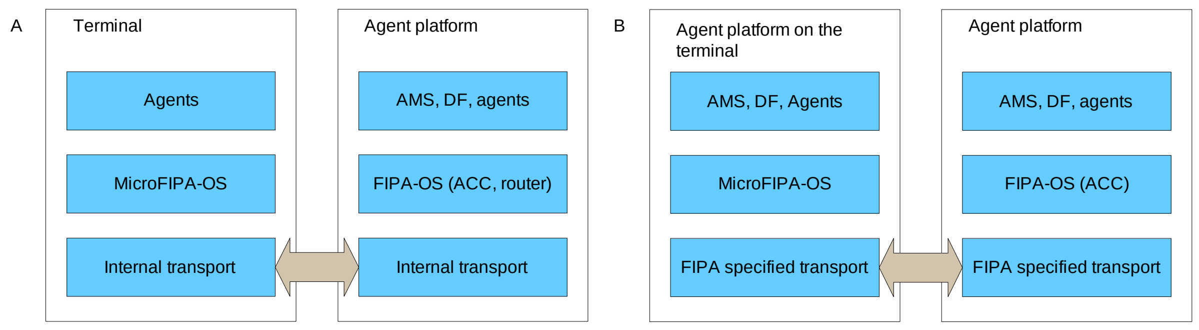


Figure 2.4 A – The terminal is part of another platform, B – The terminal runs a FIPA platform

2.3.3 JADE

The Java Agent DEvelopment framework (JADE) is another Open Source implementation of the FIPA standard, implemented in Java (see [BCPR03]). The copyright holder is Telecom Italia Lab (TILAB), but the software can be used free of charge. Specific features are the possibilities for distribution of an agent platform over multiple machines and a set of graphical (debugging) tools. Figure 2.5 shows how JADE Containers can be used to create a distributed platform:

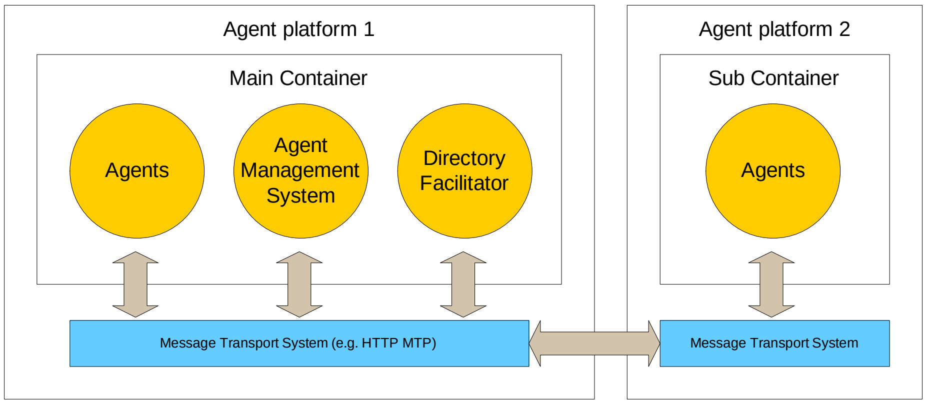


Figure 2.5 Jade Containers can be used to create a distributed agent platform

A JADE Main Container is used to host the AMS and DF agents, while other Sub-containers can be added to the platform to create a distributed platform. Since JADE 3.1 it is also possible to replicate the Main Container over multiple platforms, to ensure operation after the machine hosting the Main Container crashes.

The user base of JADE is very large, among which companies like British Telecom, France Telecom, KPN and Philips and universities like Imperial College of Londen, University of Helsinky and Parma. Several add-ons exists for the JADE platform implementation. Most notably are the LEAP add-on, which combined with JADE forms a platform for mobile devices, the Jadex add-on, which makes it possible to design agents based on the Beliefs, Desires and Intentions (BDI) model, and the HTTP MTP add-on, which allows agent communication over the HTTP protocol.

2.3.4 LEAP

The Lightweight Extensible Agent Platform (LEAP) is an extension for JADE, which enables agents to use mobile devices as agent platform (see [LEAP]). LEAP is developed by a consortium of Motorola, Broadcom, British Telecom, ADAC, Telecom Italia Lab, University of Parma and Siemens as a project for the European Commission. LEAP does not only support the Java 2 Standard Edition and PersonalJava (for PDA's), but also the Mobile Information Device Profile (MIDP) 1.0 for cell-phones supporting Java. Programmers do not need to write different code for any of the devices, since LEAP provides a homogeneous layer (API) for communicating with the mobile platform, as shown in figure 2.6. Using LEAP it is possible to create a platform which is not only distributed over different servers but can even be extended to devices which are connected by a wireless connection.

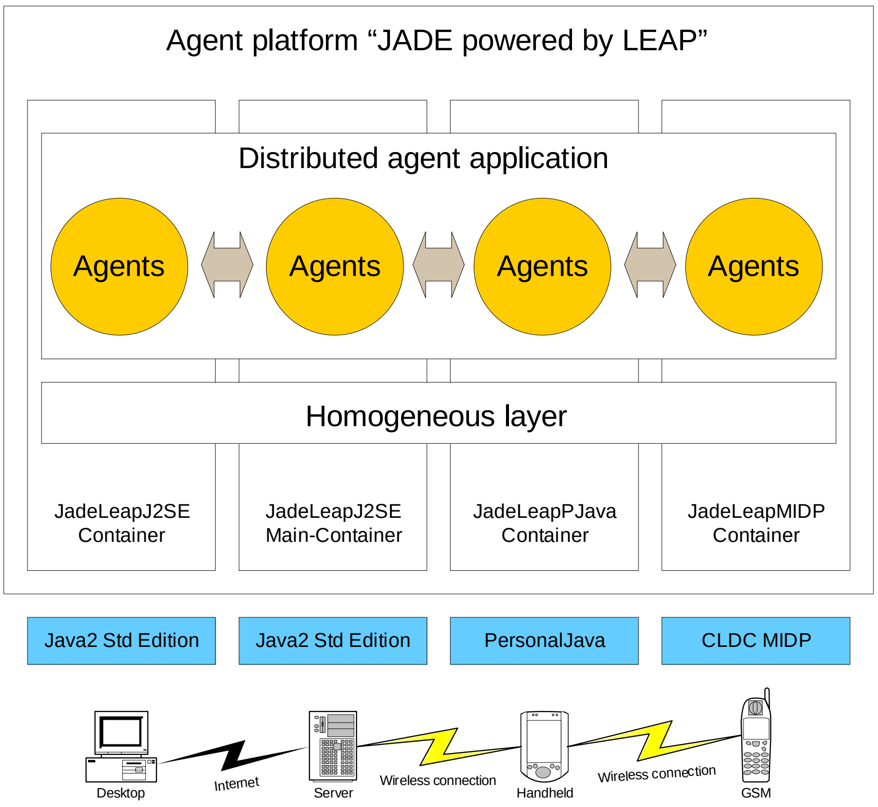


Figure 2.6 The LEAP add-on provides a homogeneous layer for agent communication on mobile devices

Figure 2.6 shows how mobile phones, handhelds, servers and desktops can be integrated, using LEAP containers that provide a homogeneous API on all platforms. This makes it possible for agents on a handheld to communicate with agents on a server and vice versa, over fixed Internet connections and wireless connections.

2.3.5 Jadex: Beliefs, Desires and Intentions agent model

Jadex is developed in Java by the Distributed Systems and Information Systems Group at the University of Hamburg under the LGPL license (see [PBL03]). It is an extension for the JADE platform which makes it possible to use the BDI (Beliefs, Desires and Intentions) model to model agents. The BDI model was first used as a philosophical model for modelling rational (human) agents and has later been introduced as a software development paradigm. The mental attitudes belief, desire and intention are used to model autonomous interacting entities that pursue their own goals and act rationally. In this case the rational behaviour of the entities means the entities try to take the best measure in order to reach their goals, seen from an ideal concept of intelligence and not with respect to human behaviour (see [RN95]).

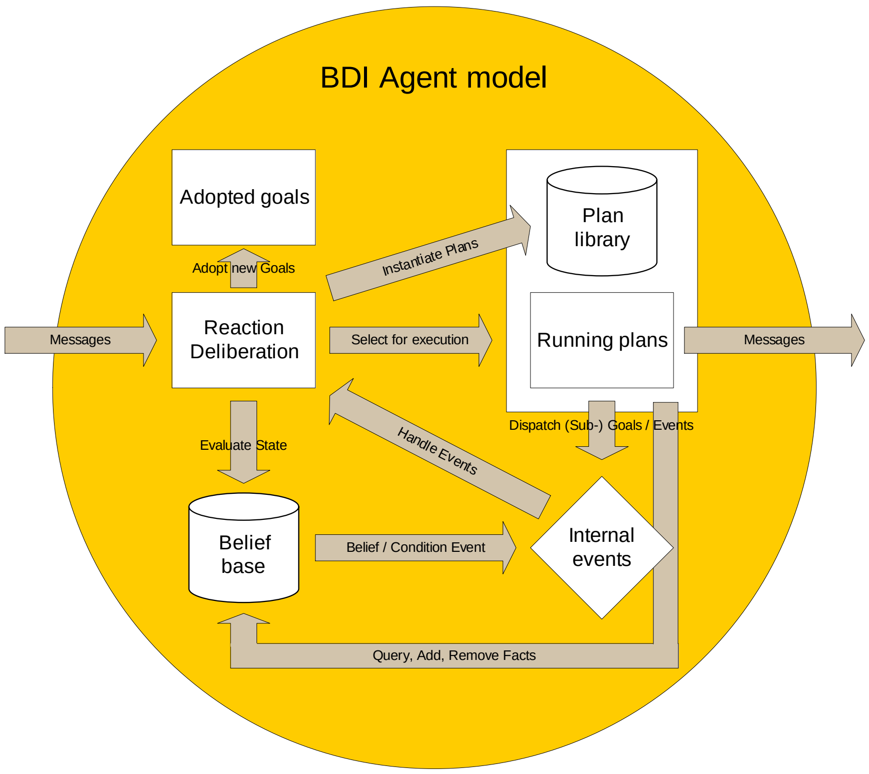


Figure 2.7 The BDI Agent model

The BDI model is represented in Jadex by beliefs, goals and plans. As shown in figure 2.7, a BDI-agent has its own belief base, containing facts about the agents perception of the world. Furthermore, each agent has a list of goals (and sub-goals) which it wants to achieve. In order to reach these goals, the agent has a library of plans at its disposal. Triggered by events the agent decides which plans to execute to achieve the adopted goals. Plans can also change the beliefs, generate events and adopt new goals, which makes it possible for the agent to function autonomously in order to reach the goals. A Jadex BDI-agent does not evaluate how successful the execution of a plan has been, which could be used to optimize the strategy to reach the goal. In fact, a Jadex BDI-agent tries out different plans until the goals have been reached.

2.4 Comparing agent platforms and implementations

In this section the features of the discussed platforms are described and a platform and an implementation of a platform are selected for this project.

2.4.1 Agent platforms

Table 2.1 shows a summary of the features of the 3 different Agent architectures. The Galaxy architecture is mainly used for multimodal user interfaces. Therefore it may be usable for designing a multimodal user interface for the PITA, but the architecture is not appropriate for the distributed personal agent application. The main disadvantages of the Open Agent Architecture are that it has been written in the C language, does not allow direct communication between agents and has an unclear license for commercial use. The main advantages of the FIPA standard are that it has different implementations with their specific benefits and licenses and does not limit agent communications but rather defines how agents should communicate with each other. The Facilitator agent of the Open Agent platform could become a bottle-neck, limiting the scalability of the application. A Java implementation is preferred because Java code is easier to understand and improve. Thus the FIPA standard was selected for this project. The next section discusses the selection of the implementation.

Tabel 2.1 Features of Galaxy, Open Agent and FIPA architecture

	Galaxy	Open Agent Architecture	FIPA
Implementation language	C / C++	C / C++	Java, different implementations available
Language interfaces	C / C++ / Java	C/ C++ / Java / Prolog	Java
Most important application area	Multimodal user interfaces	Agent based applications	Agent based applications
Support for light-weight devices	Yes	Yes	Yes
Direct Agent communication	No, through the Hub	No, through the Facilitator	Yes
License of available implementations	Free, open source	Open source, free for non-commercial usage only	Both free and commercial implementations are available
Commercial pricing	No	Not available yet	Depends on the implementation
Supported platforms	Intel Linux, Solaris, Windows	Intel Linux, Solaris, Windows	Intel Linux, Solaris, Windows

2.4.2 FIPA platform implementations

Concerning the different implementations of the FIPA standard, table 2.2 gives an overview of the 3 most advanced implementations:

Tabel 2.2 Features of different FIPA platform implementations

	Tryllian ADK	Fipa-OS	JADE / LEAP / Jadex
Lightweight client support	Separate J2ME client	Yes, PersonalJava only	Yes, PersonalJava and MIDP
License	Closed source, free for non-commercial use only	Free, open source	Free, open source
Commercial pricing	Price per runtime, based on number of users	No	No
User base	Primarily commercial users	Some commercial users, some universities	Both commercial and academic
Primary focus	Web-services, only partly FIPA-compliant	FIPA-compliant agent communications	FIPA-compliant agent communications
Distributed platform	Possible	Not possible	Different possibilities, with replication
Documentation and support	++	+	+++
Development tools	++	+	+++

The combination of JADE and LEAP makes it possible to run the agent platform on the most different devices. There are also no restrictions on the use of JADE, LEAP or Jadex because of licenses. Because JADE has the largest (both commercial and academic) user base, the documentation and development tools are the most advanced and best usable. Lastly JADE incorporates extensive possibilities for creating distributed platforms, which is an important feature for this project. This is why JADE combined with LEAP and Jadex was selected as the agent platform for this project.

2.5 The Cognitive Agent Architecture (COUGAAR)

This section desribes the COUGAAR architecture and implementation (see [KLT03]), which we describe for completeness of this chapter. At the end of this project, the COUGAAR architecture looked very interesting for consideration on other projects.

2.5.1 Background

COUGAAR is an open-source Java-based architecture for the construction of scalable distributed agent-based applications. As a DARPA (Defense Advanced Research Projects Agency) research project, the research has lead to a robust and scalable platform. It runs on Java 1.4.2 or later, but there is also a micro edition available (CougaarME) which runs on Java KVM. The COUGAAR architecture distiguishes itself from other architectures by its agent model, based on a distributed blackboard.

2.5.2 Partitioned distributed blackboard

Central in the COUGAAR agent model is the role of the partitioned distributed blackboard. Figure 2.8 shows the COUGAAR agent model.

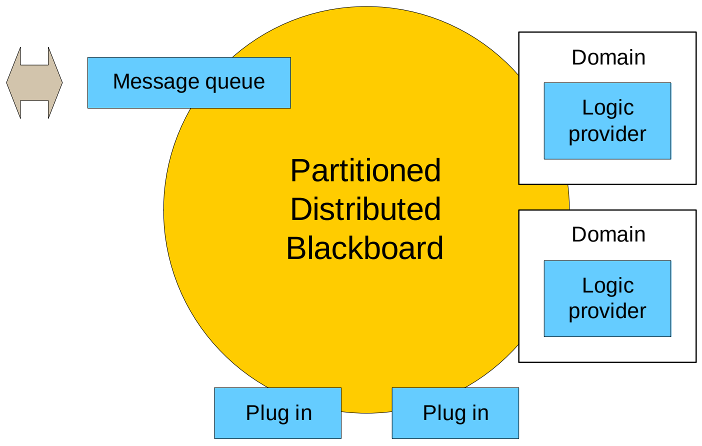


Figure 2.8 The COUGAAR agent model based on a partitioned distributed blackboard

The plug ins contain the behaviours and business logic of the agent and modify the contents of the blackboard. The contents of the blackboard are segmented into sets of logically-related objects by domains. A domain is a specification of the language used by plugins to communicate with eachother. Each domain has a set of logic providers, which act as translators between domains. The message queue is used to communicate with other agents by sending and receiving messages. Each agent owns its part of the blackboard and the contents of the blackboard are only visible to the agent itself. However the data can be shared by inter-agent tasking and querying.

2.5.3 COUGAAR compared to JADE

In comparison with JADE, COUGAAR looks very promising, but lacks the advantages of the large user base of JADE. JADE is better documented, easier to use and has more extensive mail archives with problems and solutions. Besides this, JADE complies with the FIPA standard, which enables communications with other FIPA platforms. COUGAAR does not comply with the FIPA standard and no current development is done on this subject. However, COUGAAR offers an interesting alternative architecture for the behaviour oriented or BDI agent models. One of the problems with JADE is how to share information between different agents. The blackboard of COUGAAR provides a nice solution for this problem. An interesting option is to use JADE and COUGAAR simultaneously to combine the strengths of both and reduce the disadvantages.

Chapter 3: Tools

This chapter describes the tools that were used for this project. The project is based on the JADE agent-platform. The Jadex extension for JADE was used to model and develop BDI-agents in stead of behaviour-oriented agents. Protégé-2000 was used as a modelling tool for the ontology which all agents use to communicate. Eclipse was used as graphical editor for the Java interfaces. The last section of this chapter summarises the problems which were encountered while using these tools and possible solutions.

3.1 Overview and features

Figure 3.1 shows the connection between the different tools used for this project. Protégé-2000 was used to model an ontology, which can be converted in Java source files by use of the beangenerator plug-in. Eclipse is used to develop and design more Java source files. These Java source files are used by the agent platform JADE, which uses the Jadex plug-in.

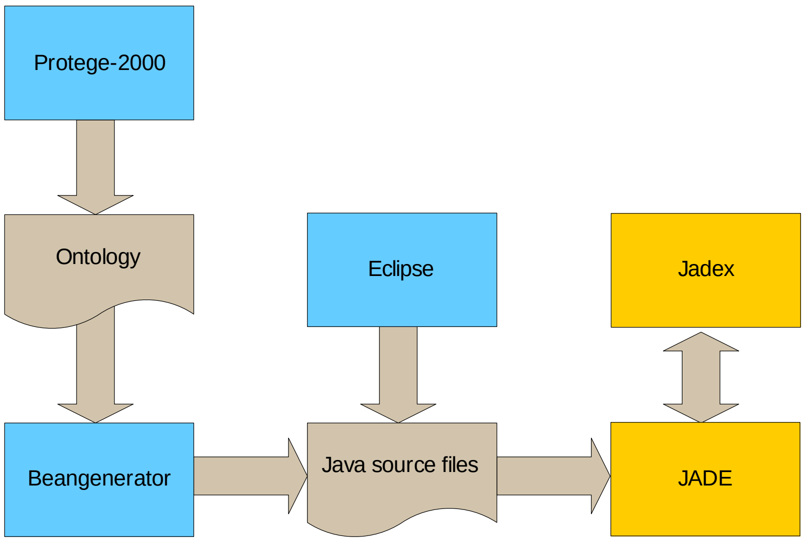


Figure 3.1 An overview of the used tools

Some common features of the used tools are:

- **All tools are open source, free to use and implemented in Java**
  Java makes it possible to run the tools on any computer platform and furthermore makes the code more understandable because it is object oriented and abstracts the low-level programming problems and Java is used extensively at Delft University. A disadvantage of Java is that it most of the time requires more processing power and memory usage as programs developed in other languages.
- **All tools have graphical user interfaces or debugging environments**
  Thanks to the graphical user interfaces it is much more user friendly to learn how to use and to use the tools, for instance for debugging an application.
- **There is a large user base using this combination of tools**
  The combination of tools is used by a lot of organisations, which ensures an active developer community and good quality documentation.

The next sections describe each of the mentioned tools.

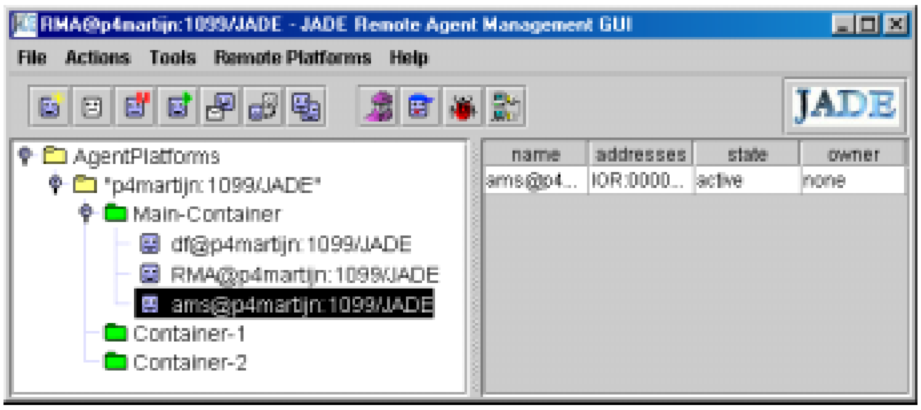
3.2 *JADE (Java Agent DEvelopment framework)*

3.2.1 Introduction

Developed and distributed by Telecom Italia Lab, the JADE software framework is a FIPA-compliant implementation of an agent platform. JADE is fully implemented in Java and comes with several tools for debugging and controlling the platform. Using the Remote Monitoring Agent (RMA), it is possible to control the platform from a Graphical User Interface (GUI). As of version 3.1 (December 2003) JADE has extensive possibilities for creating a distributed platform.

3.2.2 The agent platform

The FIPA Agent Management Reference Model, as described in section 2.3.3, is implemented in JADE as can be seen in figure 3.2.



name	addresses	state	owner
ams@p4...	IOR:0000...	active	none

<u>Legenda:</u>

🟨 Agent platform
🟩 Container
☺ Agent

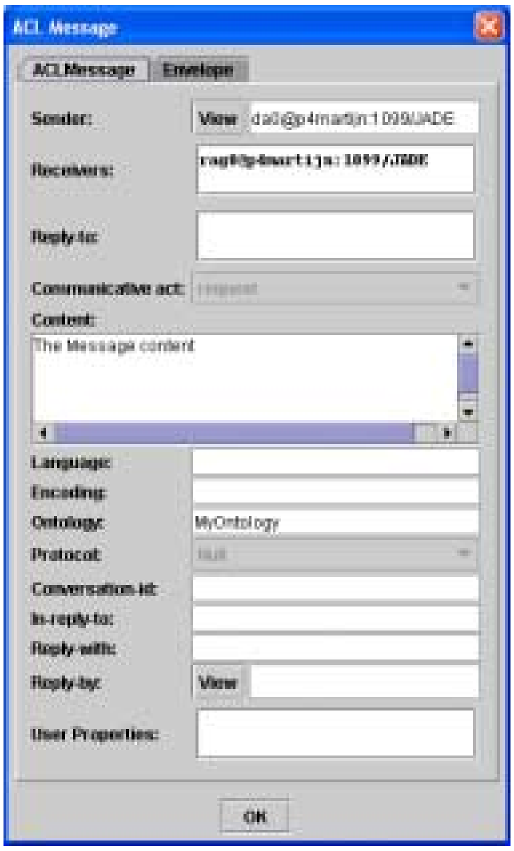
Figure 3.2 The JADE Agent Platform

The figure shows a tree starting with the agent platforms, which can contain different Containers. The Main-Container is created by default and contains the Agent Management System (AMS) and Directory Facilitator (DF). Next to these agents the Remote Monitoring Agent (RMA) is created, which enables communication with the agent platform through the GUI shown in figure 3.2.

3.2.3 Agent Communication Language

The Agent Communication Language (ACL), as described by FIPA (in [FIPA61]), forms the basis of the JADE agent platform. Figure 3.3 shows a screenshot of an ACL Message.

An ACL message can have one or more receivers. The communicative act refers to the message type of a FIPA communication protocol (i.e. request/inform/failure). The content field contains the message contents, which are encoded using the specified encoding. The ontology field is used to specify what types of objects can be stored in the content field of the message. An ontology can be seen as an object description of objects which can be exchanged by Agents that understand the same ontology.


ACL Message

ACLMessage | Envelope

Sender:	View da0@p4martijn:1099/JADE
Receivers:	rag0@p4martijn:1099/JADE
Reply-to:	
Communicative act:	request
Content:	The Message content
Language:	
Encoding:	
Ontology:	MyOntology
Protocol:	null
Conversation-id:	
In-reply-to:	
Reply-with:	
Reply-by:	View
User Properties:	

OK

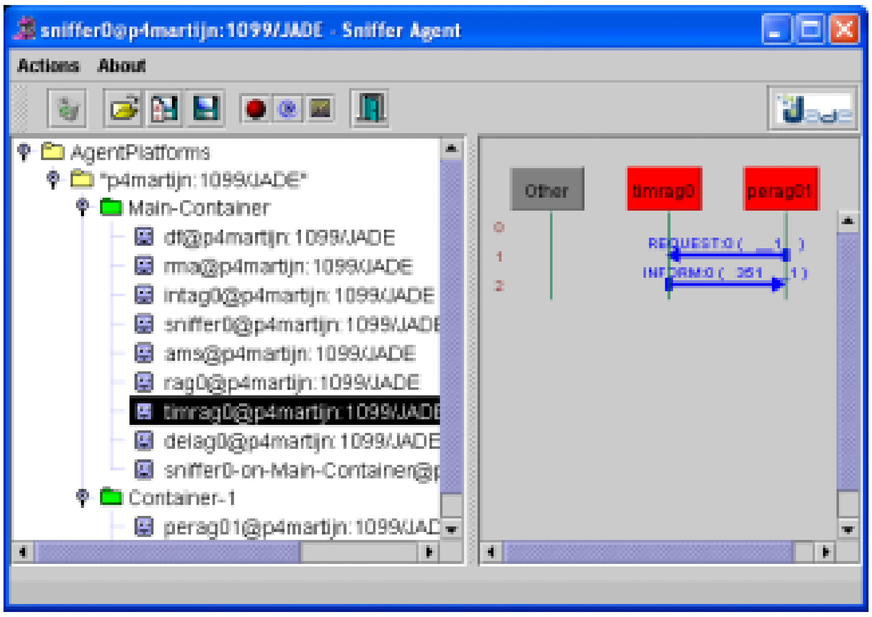
Figure 3.3 A screenshot of an ACL Message

3.2.4 Graphical debugging tools

The JADE Remote Monitoring Agent GUI comes with several graphical debugging tools, which are discussed in the following section.

3.2.4.1 The Sniffer Agent

Figure 3.4 shows a screenshot of the Sniffer Agent. The Sniffer Agent makes it possible to inspect communications between agents.


sniffer0@p4martijn:1099/JADE - Sniffer Agent

Actions About

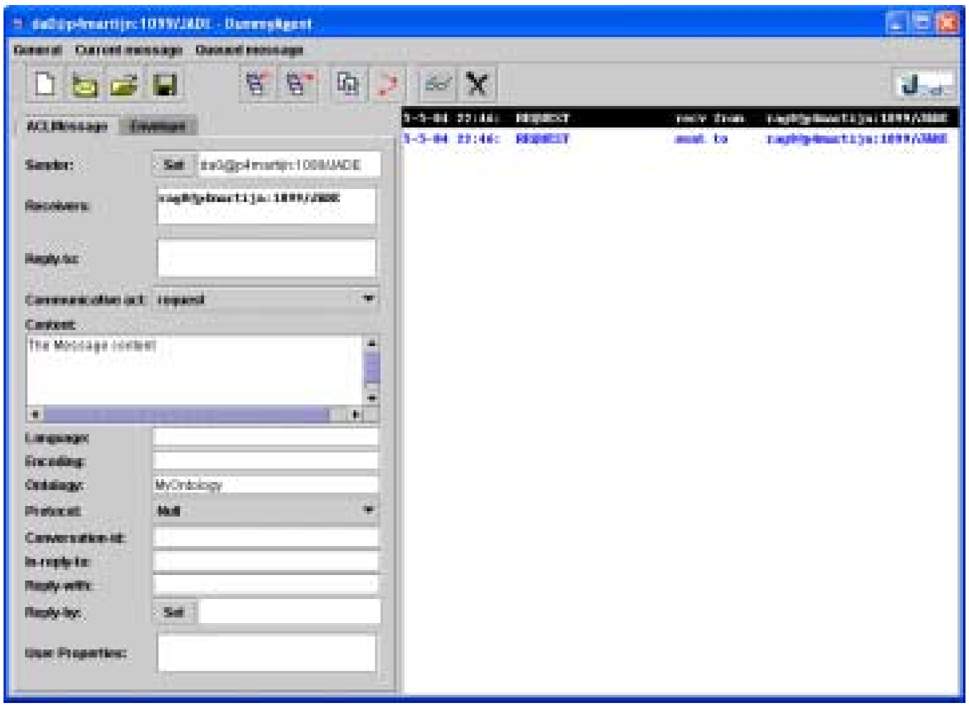
- AgentPlatforms
  - "p4martijn:1099/JADE"
    - Main-Container
      - df@p4martijn:1099/JADE
      - rma@p4martijn:1099/JADE
      - intag0@p4martijn:1099/JADE
      - sniffer0@p4martijn:1099/JADE
      - ams@p4martijn:1099/JADE
      - rag0@p4martijn:1099/JADE
      - timrag0@p4martijn:1099/JADE
      - delag0@p4martijn:1099/JADE
      - sniffer0-on-Main-Container@p
    - Container-1
      - perag01@p4martijn:1099/JAD

Figure 3.4 A screenshot of the Sniffer Agent interface

By selecting which agents to monitor, the Sniffer Agent will log all messages going to and from the selected agents and gives the possibility to inspect these messages individually.

3.2.4.2 The DummyAgent

Figure 3.5 shows a screenshot of the DummyAgent. The DummyAgent can be used to communicate with agents and inspect and log the communication.



```
5-5-04 22:46:  REQUEST      recv from   rag0@p4martijn:1099/JADE
5-5-04 22:46:  REQUEST      sent to     rag0@p4martijn:1099/JADE
```

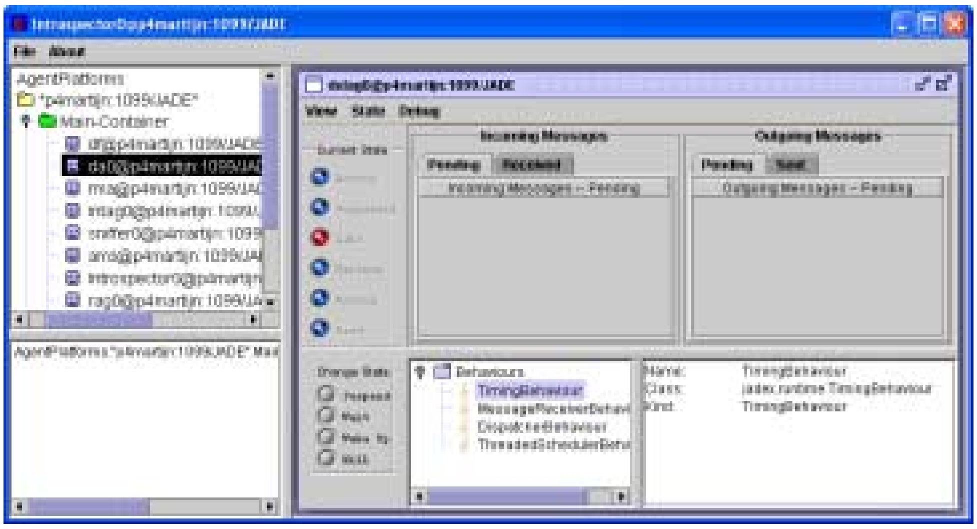
Sender:	da0@p4martijn:1099/JADE
Receivers:	rag0@p4martijn:1099/JADE
Reply-to:	
Communicative act:	request
Content:	The Message content
Language:	
Encoding:	
Ontology:	MyOntology
Protocol:	Null
Conversation-id:	
In-reply-to:	
Reply-with:	
Reply-by:	
User Properties:	

Figure 3.5 A screenshot of the DummyAgent

By manually filling out the properties of a message, messages can be sent to and received from other agents. The DummyAgent is especially useful to test the functioning of individual agents.

3.2.4.3 The Introspector Agent

The Introspector Agent, as shown in figure 3.6, makes it possible to monitor the internal state and active behaviours of individual agents as well as incoming and outgoing messages.



```
AgentPlatforms
  "p4martijn:1099/JADE"
    Main-Container
      df@p4martijn:1099/JADE
      da0@p4martijn:1099/JADE
      ma@p4martijn:1099/JADE
      intag0@p4martijn:1099/JADE
      sniffer0@p4martijn:1099/JADE
      ams@p4martijn:1099/JADE
      introspector0@p4martijn
      rag0@p4martijn:1099/JADE

Behaviours
  TimingBehaviour
  MessageReceiverBehav
  DispatcherBehaviour
  ThreadedSchedulerBeha

Name:   TimingBehaviour
Class:  jade.core.runtime.TimingBehaviour
Kind:   TimingBehaviour
```

Figure 3.6 A screenshot of the Introspector Agent

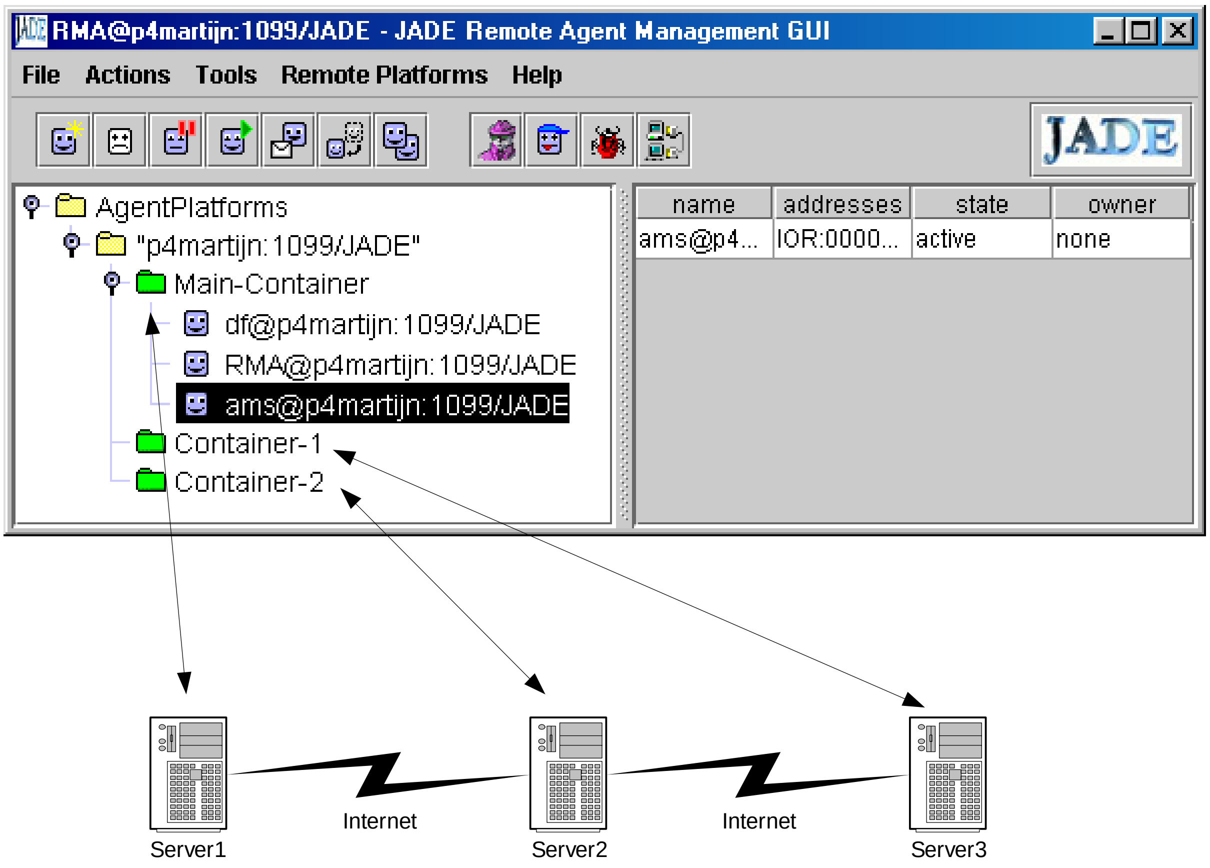
In order to debug an agent, it is possible to break execution of an agent and resume execution step by step.

3.2.5 Distributed platform

A distributed JADE platform can be created by distributing Containers of an agent platform over multiple servers. By replicating the Main-Container over different servers a failsafe system can be created. The next sections describe these features of JADE.

3.2.5.1 Distributed containers

As already shown in figure 3.2, a JADE agent platform can consist of different (sub-) containers. However, these containers do not necessarily have to be hosted on the same server. Each container can be hosted on a different server, which makes it possible to have a distributed platform consisting of containers on different servers, as shown in figure 3.7.



name	addresses	state	owner
ams@p4...	IOR:0000...	active	none

Figure 3.7 A distributed platform can be created by running agent containers on different servers

A disadvantage of distributing a platform as shown in figure 3.7 is the central point-of-failure which is the server running the Main-Container. If this server crashes, the agent platform will no longer run. There are two solutions for this problem. The first solution is to run complete agent platforms on each server (in stead of containers). However this means the application needs to manage which servers are running agent platforms and which servers have crashed. Furthermore this limits the possibilities for agent mobility, because this is not possible across different agent platforms. The second option is to use Main-Container replication, as is discussed in the next section.

3.2.5.2 Main-Container replication

In order to deal with the central point-of-failure being the server running the Main-Container of a distributed platform, JADE can be used with Main-Container replication as of version 3.1 (December 2003). By enabling this option, each server can store a copy of the Main-Container. This makes sure the Main-Container is replicated throughout the agent platform and in case the server running the Main-Container crashes, a copy on another server can take over the Main-Container role and the platform can still function. Thus a distributed fault tolerant agent platform can be created as shown in figure 3.8.

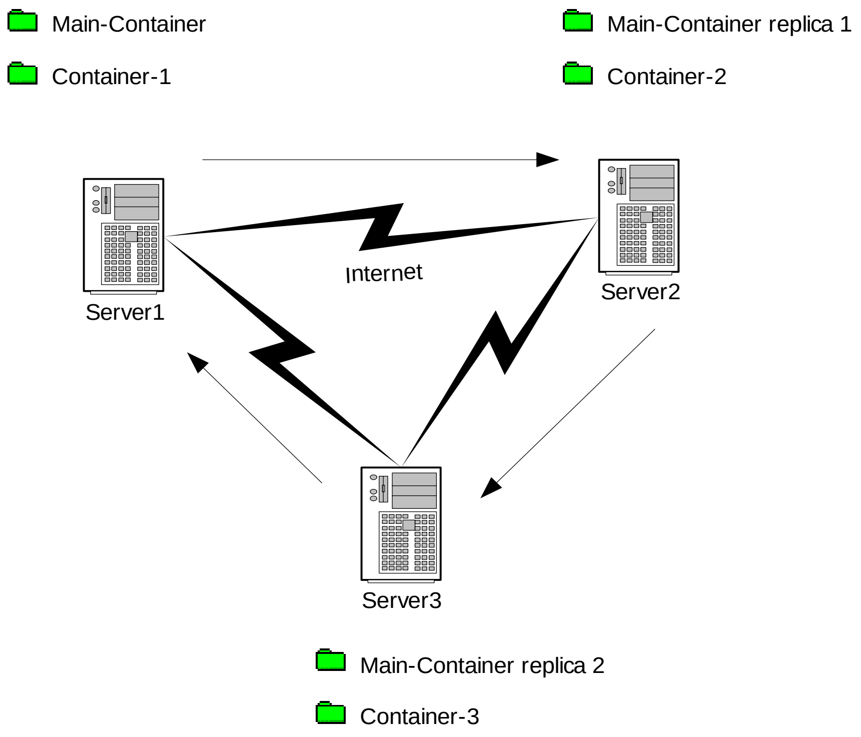


Figure 3.8 A distributed platform with Main-Container replication

By enabling Main-Container replication, a ring is formed between all replica's. This ring structure is used for replication of data, but also to ensure operation of each node of the ring. Crashed nodes are removed from the ring-structure and as soon as the Main-Container fails, the following replica will launch its Main-Container replica and ensure continued operation of the agent platform.

3.3 *Jadex – BDI Agent System*

3.3.1 Introduction

Jadex is an extension for the JADE platform which makes it possible to use the BDI (Beliefs, Desires and Intentions) model (see figure 2.7) to model agents. It is implemented in Java by the Distributed Systems and Information Systems Group at the University of Hamburg under the LGPL license. Currently the software is only available in a beta version (version 0.92).

3.3.2 Agent definition files

For each Jadex agent, an agent definition file has to be created which describes the different goals, plans, beliefs, languages and ontologies that have to be loaded when starting the agent. Agent definition files are XML-files that can contain Java-code and regular expressions. Figure 3.9 gives a overview of the contents of an example agent definition file.

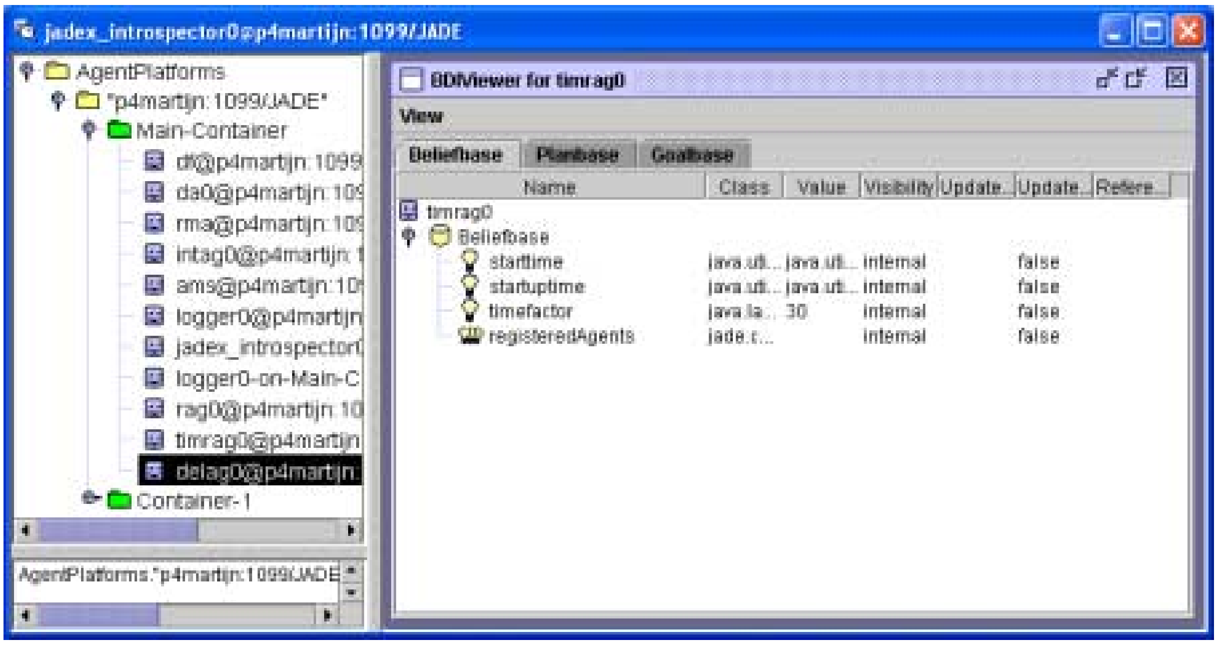
```
<agent xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://jadex.sourceforge.net/jadex.xsd" name="perag0"
class="jadex.runtime.JadeWrapperAgent">
<imports>
<import>jade.core.*</import>
…
</imports>

<plans>
<plan name="personalagentplan" instant="true">
<constructor>new PersonalAgentPlan($arg0)</constructor>
<waitqueuefilter>PersonalAgentPlan.getEventFilter()</waitqueuefilter>
</plan>
<plan name="updatepositionplan" instant="false">
<constructor>new UpdatePositionPlan()</constructor>
<condition>$beliefbase.destinationLocation != null</condition>
</plan>
<plan name="travelplan">
<constructor>new TravelPlan()</constructor>
<filter>TravelPlan.getEventFilter()</filter>
</plan>
</plans>

<beliefs>
<!-- system parameters -->
<belief name="systemTime" class="SysTime.class" category="belief">
<fact>new SysTime(60)</fact>
</belief>

<!-- travel parameters -->
<belief name="currentLocation" class="Tuple.class" category="belief">
<fact>new Tuple("Delft","dt")</fact>
</belief>
…
</beliefs>

<languages>
<language>new SLCodec()</language>
</languages>

<ontologies>
<ontology>OVRouteOntology.getInstance()</ontology>
</ontologies>

<goals>
<achievegoal name="reach"/>
</goals>

</agent>
```

Figure 3.9 An example agent definition file for a Jadex agent

Agent definition files do not limit the capabilities of an agent, since goals, plans and beliefs can be added, removed and replaced after the agent is started.

3.3.3 Graphical debugging tools

Starting JADE with the Jadex plug-in adds two new graphical debugging tools to the JADE RMA. The Logger Agent is added, which makes it possible to track debug/log messages generated by different agents. The second tool is the Jadex Introspector agent.



Name	Class	Value	Visibility	Update...	Update...	Refere...
timrag0						
Beliefbase						
starttime	java.uti...	java.uti...	internal		false	
startuptime	java.uti...	java.uti...	internal		false	
timefactor	java.la...	30	internal		false	
registeredAgents	jade.c...		internal		false	

Figure 3.10 A screenshot of the Jadex Introspector agent

The Jadex Introspector agent makes it possible to view the internal state of a BDI agent, showing the current contents of the beliefbase, planbase and goalbase. This makes it possible to accurately monitor the behaviour of different plans and goals working together.

3.4 Protégé-2000 and the beangenerator plug-in

3.4.1 Introduction

Protégé-2000 is a widely used knowledge-modelling tool (with about 15.000 registered users), developed by Stanford Medical Informatics of Stanford University (see [Protégé]). Thanks to a plug-in (called beangenerator) for Protégé-2000, developed by C.J. van Aart from the Department of Social Science Informatics (SWI) of the University of Amsterdam, it is possible to use Protégé-2000 to design ontologies for the JADE agent platform (as described in [BCPA02]).

3.4.2 Ontologies

An ontology is an abstract description of a language which agents can use to communicate. In stead of words and grammar, an ontology in this case consists of Java objects that extend the following Java classes: Predicate, Concept or AgentAction. Figure 3.11 shows a class-diagram of the Content Reference Model, containing the classes that are included in an ontology.

Objects extending either Predicate, Concept or AgentAction can be included in ACL messages directly by using built-in functions of JADE. Ontologies can be designed with the Protégé-2000 graphical user interface.

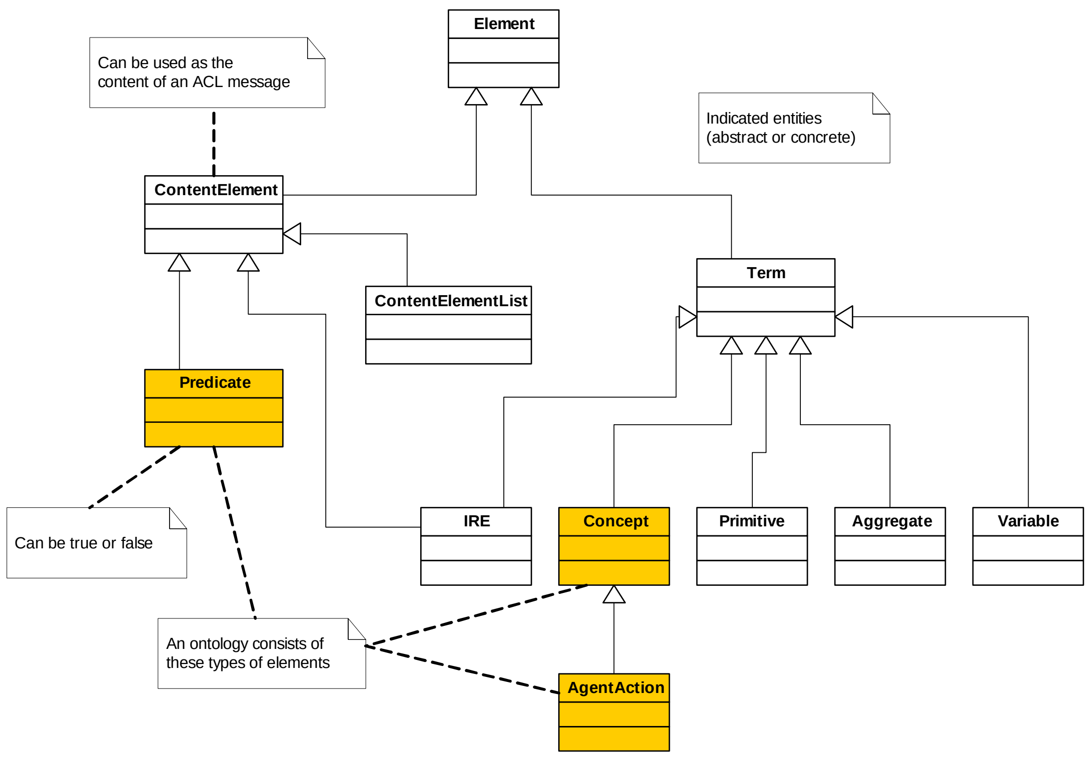
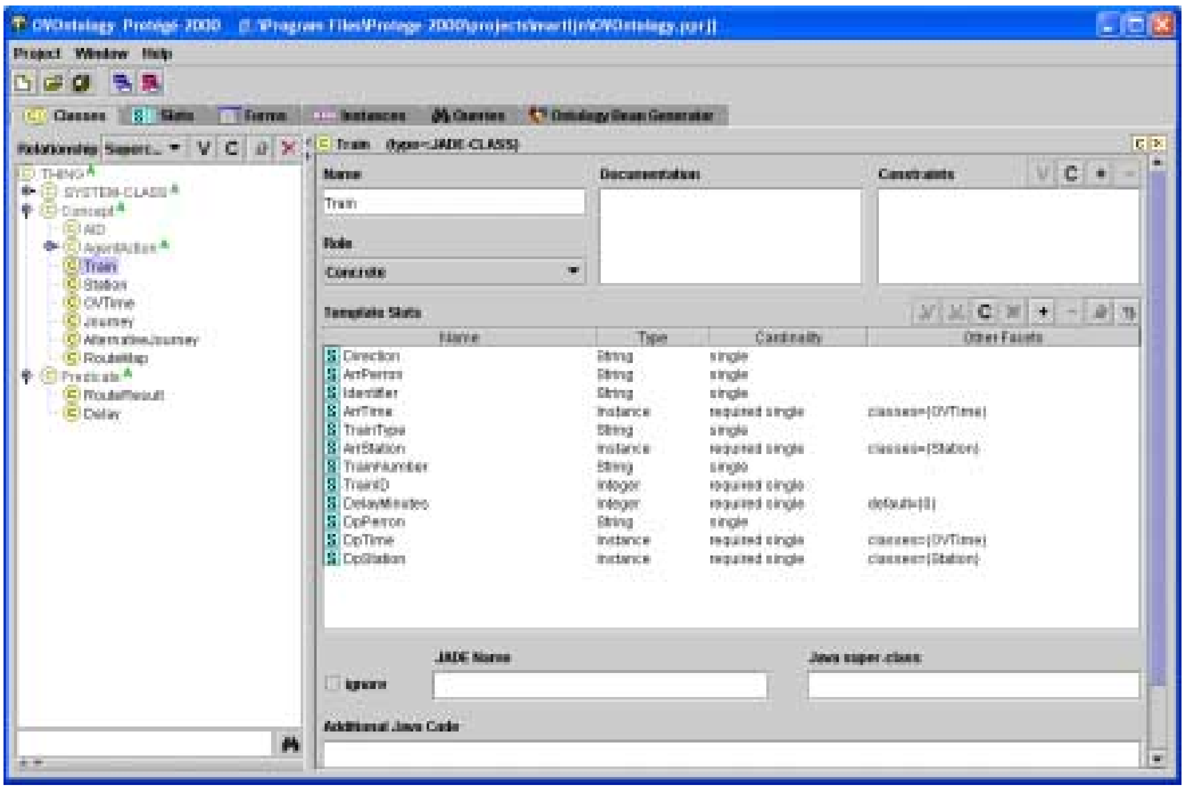

Element
Can be used as the content of an ACL message
Indicated entities (abstract or concrete)
ContentElement
ContentElementList
Term
Predicate
Can be true or false
IRE
Concept
Primitive
Aggregate
Variable
An ontology consists of these types of elements
AgentAction

Figure 3.11 The Content Reference Model for designing ontologies

3.4.3 Protégé-2000

Protégé-2000 is a graphical knowledge modelling tool. In order to function as a JADE ontology designer, the SimpleJADEAbstractOntology project has to be included, which in fact contains the model depicted in figure 3.11. After this, Protégé-2000 can be used to design Concepts, AgentActions and Predicates.



Train (type=JADE-CLASS)

Name: Train

Role: Concrete

Template Slots

Name	Type	Cardinality	Other Facets
Direction	String	single	
ArrPerson	String	single	
Identifier	String	single	
ArrTime	Instance	required single	classes={OVTime}
TrainType	String	single	
ArrStation	Instance	required single	classes={Station}
TrainNumber	String	single	
TrainID	Integer	required single	
DelayMinutes	Integer	required single	default={0}
DepPerson	String	single	
DepTime	Instance	required single	classes={OVTime}
DepStation	Instance	required single	classes={Station}

Ignore

JADE Name:

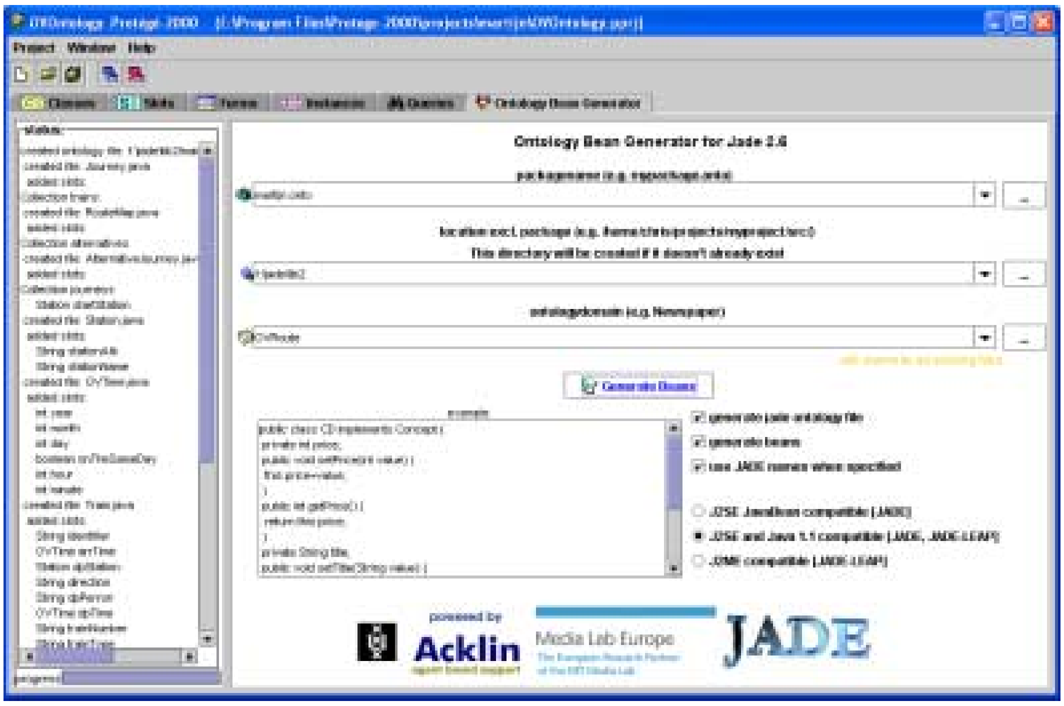
Java super class:

Figure 3.12 A screenshot of the ontology designed in Protégé-2000

Figure 3.12 shows how the design of an ontology looks in Protégé-2000. In order to create Java files from this design, it is necessary to install the beangenerator plug-in for Protégé-2000. The beangenerator makes it possible to create a set of Java source files containing the ontology, which can directly be used in JADE.

3.4.4 Beangenerator plug-in

The beangenerator is a plug-in for Protégé-2000 which generates a set of Java source files, containing the ontology which was designed in Protégé-2000.



Ontology Bean Generator for Jade 2.6

packagename (e.g. mypackage.onto): [illegible]

location excl. package (e.g. /home/chris/projects/myproject/src): [illegible]
This directory will be created if it doesn't already exist

ontologydomain (e.g. Newspaper): OVRoute

example

```
public class CD implements Concept {
 private int price;
 public void setPrice(int value) {
  this.price=value;
 }
 public int getPrice() {
  return this.price;
 }
 private String title;
 public void setTitle(String value) {
```

☑ generate jade ontology file
☑ generate beans
☑ use JADE names when specified

○ J2SE JavaBean compatible [JADE]
● J2SE and Java 1.1 compatible [JADE, JADE-LEAP]
○ J2ME compatible [JADE-LEAP]

status:
```
created ontology file: [illegible]
created file: Journey.java
 added slots:
Collection trains
created file: RouteMap.java
 added slots:
Collection alternatives
created file: AlternativeJourney.java
 added slots:
Collection journeys
  Station startStation
created file: Station.java
 added slots:
  String stationId
  String stationName
created file: OVTime.java
 added slots:
  int year
  int month
  int day
  boolean onTheSameDay
  int hour
  int minute
created file: Train.java
 added slots:
  String identifier
  OVTime arrTime
  Station arrStation
  String direction
  String depPerron
  OVTime depTime
  String trainNumber
```

Figure 3.13 The beangenerator plug-in can be used to generate Java source files

Figure 3.13 shows how by supplying a base directory, package name and ontology name, all source files can be easily generated by pressing the "Generate Beans" button. The generated beans can be either JavaBean, J2SE or J2ME compatible.

3.5 Eclipse

3.5.1 Introduction

Eclipse is an open-source IDE (Integrated Development Environment) for developing applications in all sorts of different languages, for different platforms (see [Eclipse]). IBM uses Eclipse in its WebShere Studio Device Developer. Eclipse also has a Visual Editor plug in which simplifies the development of Java graphical interfaces.

3.5.2 Graphical interface

The Eclipse IDE allows developers to both graphically design their Java graphical interfaces as well as directly modify the code, which then influences the graphical design. Also it offers code completion and online syntax checking.

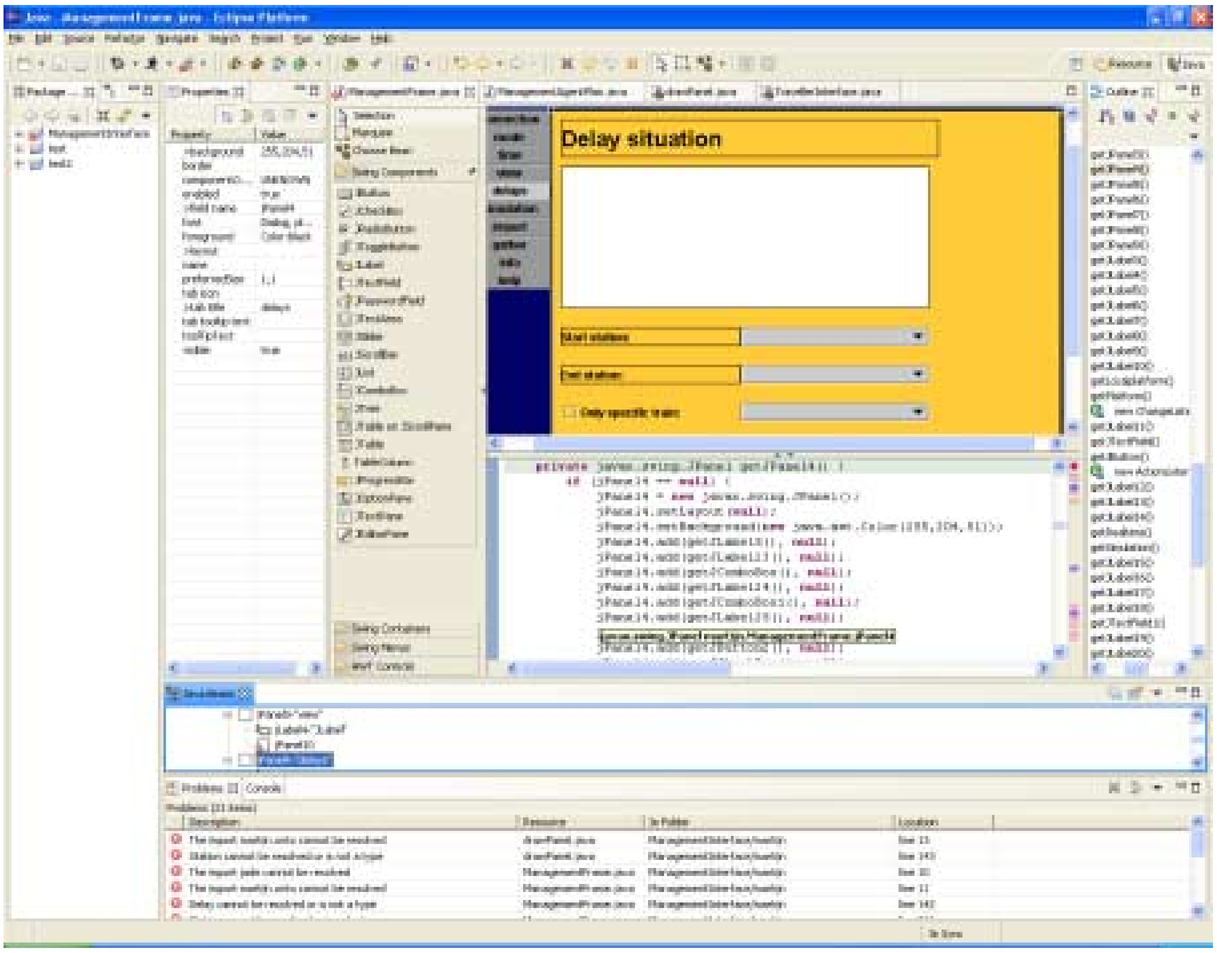


Figure 3.14 The Eclipse IDE

Figure 3.14 shows the Eclipse IDE with the Visual Editor plug in, which allows easy graphical design of user interfaces. However the usability of the IDE is not as good as some other commercially available tools.

3.6 Problems and limitations

This section describes some problems and limitations which were experienced while using the mentioned tools. Also, if available, a workaround or solution for the problem is described.

A limitation on the JADE agent platform is that agents are only mobile on the same agent platform. So, it is possible to move an agent from a Container on one server to a Container on another server, only if both Containers are part of the same distributed platform. So called intra-platform agent mobility (between different platforms) is not supported. Another limitation of JADE is that it loads and stores the class of an agent, when it is first started. Succeeding agents of the same class are instantiated from this copy in memory. If you are developing an agent, each time you alter the code and recompile it, you have to restart your agent platform to test the new version, because the platform will otherwise keep using the old version of the agent code. This slows down the development considerably.

The Jadex extension for JADE offers so called ThreadedPlans, which contain the application code. These ThreadedPlan instances are however scheduled by a single-threaded scheduler, which means that the scheduler will not schedule other plans for execution, if a plan instance blocks (for instance on a Socket InputStream). There are two solutions for this problem. The first solution is to edit the Jadex source file jadex/runtime/ThreadedSchedulerBehaviour.java and replace the following line of code:

```
monitor.wait();  // around line 105
```

with the following line, where MAX_EXECUTION_TIME is replaced by a millisecond value containing the maximum execution time, before the scheduler resumes it task:

```
monitor.wait(MAX_EXECUTION_TIME);  // replace MAX_EXECUTION_TIME
```

This will cause the scheduler to resume its task on other plans, after a plan is blocking for the specified time. A better solution is to put the longer tasks or blocking tasks in a separate Thread, which is instantiated and started from the ThreadedPlan which gives back execution to the scheduler afterwards. A limitation on both methods is that the plan which is running after the timeout and the separate Thread can no longer give back control to the scheduler and cannot send and receive messages as a consequence. To solve this, in case of a separate Thread, memory has to be shared between the ThreadedPlan and the Thread, and the Threaded plan has to check whether this memory has changed, in which case the ThreadedPlan can decide to send messages or vice versa.

A more common problem with developing agent applications is that it is very difficult to debug the application. Because different parts of code can be running simultaneously and because agents communicate autonomously, it is not possible to "step through" the whole application. Individual agents can be stopped and debugged individually, but this will not involve the communication between other agents. This makes it very difficult to correctly debug an application and the use of the mentioned tools is absolutely necessary. Another solution to cope with this problem is to build agents in an incremental way and testing the behaviour of the agent in steps. Also outputting debug information to the System.out stream can be useful to get an idea of what is going on.

Concerning the beangenerator plug in for Protégé-2000, there are two limitations on the bean generation. The first limitation is that the generated beans do not implement the (empty) Serializable interface, which enables objects to be directly saved in files or transferred over sockets. The second limitation is that the generated beans always overwrite the existing contents of the output folder, which means manual changes, made on the beans are lost between two generations. A better solution would be to merge these changes in the newly generated beans.

Finally Eclipse has some limitations, compared to other, commercially available tools. An annoyance is that it is not possible to download a complete Java IDE, but several separate packages have to be installed. Secondly it is very slow (or it requires a lot of processing power), especially on larger source files. Also the synchronisation between the graphical design and the source file can get stuck, in which case the program has to be restarted. Lastly the graphical editor misses some basic functionality in the sense that the contents of a ComboBox cannot be edited with the property editor (but have to be edited in the sourcecode) and in order to switch to a certain tab from a JTabbedPane, you have to use the object tree, in stead of clicking on the desired tab in the graphical editor.

Chapter 4: Assumptions, requirements and model

This chapter starts with a description of the PITA. After that a technical view on the problem that has to be solved as part of this thesis is described. Then the assumptions which form the basis of this thesis are described. Next, the requirements for a solution are described in the fourth part. Finally the distributed model and considerations which lead to this model are presented.

4.1 Description of the Personal Intelligent Travel Assistant

The most important aspect of the Personal Intelligent Travel Assistant is that it relieves the user of having to find the best route to his destination. By determining the position of the user, using GPS or other techniques, the system is able to optimise the route for the user and update the advice as the user travels and new information becomes available.

In order to communicate with different types of users in different kinds of situations, the PITA system provides a multi modal interface to the user. This means the communication can be based on speech, icons, text or a graphical interface depending on the preferences of the user and the situation in which the system is used. For instance an elderly user in a busy environment might prefer to communicate using a large iconic interface, while a business traveller travelling in a first class private compartment might prefer the speech interface. The speech interface uses speech recognition and speech synthesis to communicate with the user.

To optimally fit the desires and demands of the user, the system keeps track of several preferences of the user. These preferences can be changed by the user, but the system also tries to learn the user preferences from the day-to-day use. For instance, if the user always uses the speech interface in the morning and switches to the text interface in the evening, the system will learn this preference and automatically switch to the most likely interface. Also if the system detects a lot of noise on the microphone channel, it switches from the speech interface to the text or iconic interface.

Advising the user about the best route to his destination requires accurate information about delays of trains and a routing algorithm to find the optimal connection between his current location and the destination, given the train schedule and expected delays. The information about delays is partly available from the NS website, but the PITA system can also generate information about train delays itself. Since the position of each user is known at each moment, the system can compare the actual position with an expected position and estimate whether the user is experiencing a delay. This information can then not only be used to update the route of this user, but also to warn other users that are planning to take this train and possibly change their route advice.

The optimal route from the current location to the destination can depend on several user preferences. Depending on the type of user, the system can give the fastest but more risky solution based on more train changes or it can give a less risky solution which arrives a little later but is very likely to arrive at that time. The first solution is optimal for young, demanding users while the second solution will better fit the requirements of business or elderly users. The routing algorithm should use the user preferences in order to optimally fit the expectations of the user.

As handhelds become more and more powerful every day, it will be possible to run a larger part of the application on the handheld device. Ultimately the application will completely run on the handheld device and depending on information which is available optimise the advice for the user. This information can come from nearby users which are also using the PITA system, for instance by using ad-hoc networking techniques or by using dial-up or UMTS or GPRS based mobile Internet connections to gather the required information about delays. The PITA system consists of individual handheld applications, gathering and exchanging information with each other and trying to individually optimise the advice for the users. Emerging from this behaviour is a distributed application of autonomously functioning handheld applications which distribute and exchange information over the ad-hoc connections between the devices and Internet connections where available. If there is no other information available, the individual applications will still try to give the best advice to the user, based on static information stored on the handheld device.

The PITA system should be integrated with other applications on the handheld device, which together form a personal assistant for the user. For instance an application which is able to automatically schedule appointments between groups of people at a location which involves the least travelling by the group of people, could use the PITA system to guide each of the individual users to the correct location. The users then do not need to worry about planning meetings and scheduling travelling times, since the two applications, working together for all users take care of this, by communicating on behalf of each of the individual users.

Besides offering the user a comfortable solution for planning a train route to his destination, the system also offers NS several new possibilities. Since the position of each user is known by the system, it is possible to visualise traveller flows. These traveller flows can be used to design a model of the demand for capacity along every railway connection. Using this model, NS will be able to better predict traveller flows, which gives NS the possibility to optimally schedule the trains and carriages. Another possibility is to let passengers reserve a seat in advance, which gives them a guaranteed seating place and gives NS the possibility to schedule carriages even better fitted to the demand. In the case of a PITA user, the reservation of seats will naturally be done automatically by the application without intervention of the user.

The conductors can be equipped by NS with a PITA device, which gives them the possibility to advise travellers (without a PITA device) even better about the route they should take, which also includes actual delays. Besides equipping conductors with PITA devices, NS could also give travellers who buy an annual railpass a PITA device for reduced charges, as part of a better customer service. By equipping conductors and heavy users with a PITA device, the NS can take care of creating a start-up user base which is large enough to be able to gather enough information about train delays from the system itself, which is very important for acceptance of the PITA system.

The last advantage of the PITA system for NS is that trains and stations can become aware of which users are at which platform or in which train. This way it is no longer necessary to interview people in trains about which type of ticket they use, but this data can be gathered electronically. Also electronic billing becomes possible, which could simplify the work of the conductor considerably.

4.2 Description of the problem to solve

The possibilities of the PITA system, as described in the previous section, cover to much aspects to study as part of a single MSc thesis. Therefore we have divided the system in several sub-problems which have to be solved and selected a set of these problems to study as part of this MSc thesis. First a technical view on the PITA problem is presented in order to get a view on the problem domain. After that, the division in sub-problems is discussed.

4.2.1 A technical view on the problem

In order to create a Personal Intelligent Travel Assistant, it is useful to take a look at how the problem can be solved from a technical point of view. The goal is to have an application running on a handheld device, which displays a personal travelling advice for the user, as shown in figure 4.1. Using GPS or another positioning technology, the position of the user is determined. Multiple users can be near to each other and share information through an ad-hoc network, based on for instance WiFi connections. Some handheld devices have access to a GSM phone, which enables them to connect to the Internet and possibly some server or servers running another part of the application or gathering and storing data.

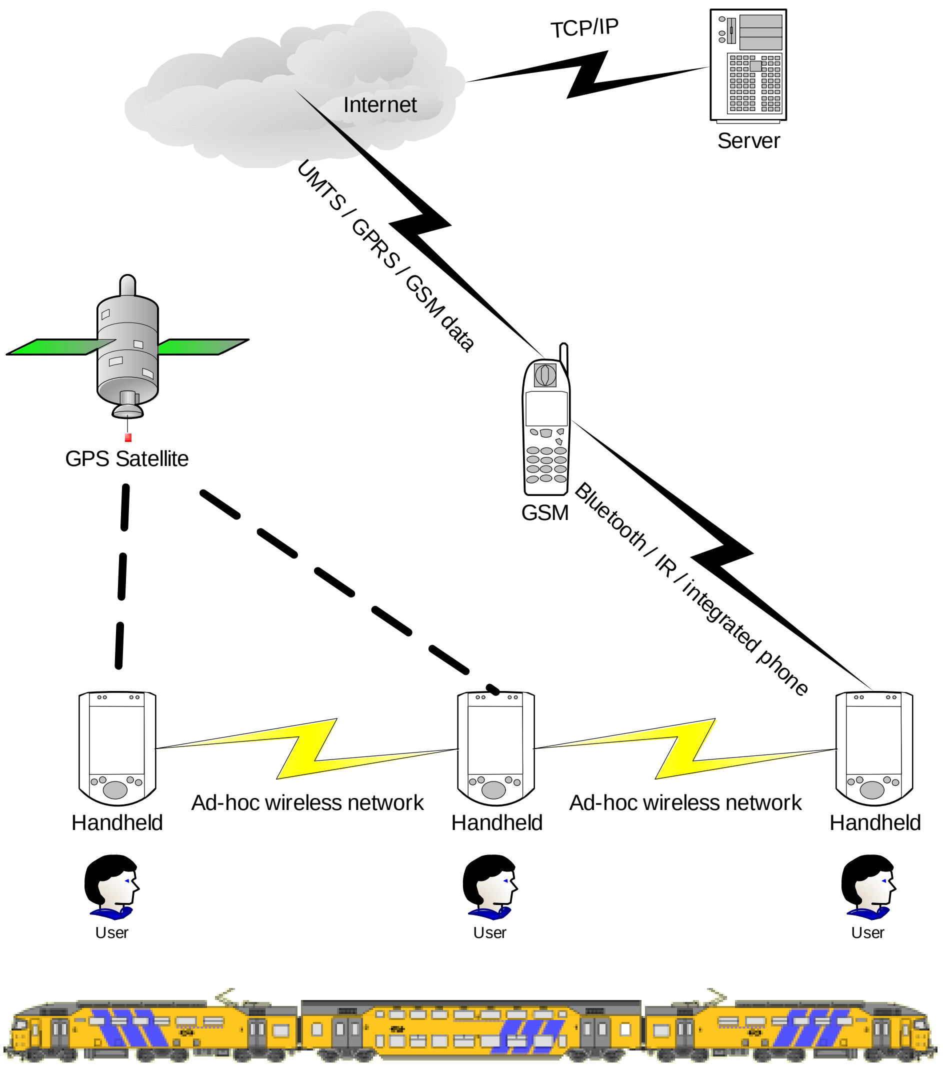


Figure 4.1 A technical view on the PITA problem

Data comes from several different sources, such as position and delay data from different users. Crucial for the PITA application is how this data can be combined into a personal advice for the user. For large scale acceptance, it is essential to minimise the costs which are involved in the wireless connections and thus the data going over the GSM data / GPRS or UMTS connections.

4.2.2 Division in sub-problems

This section describes how the PITA system was divided into sub-problems and which problems were studied as part of this MSc thesis.

The first sub problem is **acquiring the data** which is necessary to built a PITA application. This project will study the possibilities for acquiring information on the static train schedule, delay information of trains and how to determine the position of the user. Also a model to distribute this data and the processing power over handhelds and/or servers will be studied.

The second problem is the **route planner**, which will be studied as part of this project. The route planner will have to use the acquired data and the preferences of the user in order to find optimal routes.

The third sub problem is the **communication with the user**. As described, this should preferably be done multi modally (by graphical, iconic and speech interfaces) and the user should have the possibility to switch between different interfaces. This project will not focus on designing and developing a multi modal interface, but provide a graphical user interface as part of the prototype which will be developed. Also a simulation environment will be designed and implemented as part of this project. Testing the usability of the interfaces will not be part of this project.

The last problem are the **capabilities of different handheld devices**. This thesis will not study the different handheld devices which are currently available and implement different code for compatibility with different types of handhelds. In stead, a prototype will be developed which is able to run on a single handheld platform, which is as platform independent as possible. Also ad-hoc networking functionality will not be developed as part of this project, since currently research is being done in this area at TU Delft and we assume this technology to be available. The developed prototype will however be prepared to function in an ad-hoc environment.

4.3 Assumptions

Two assumptions are made for the design presented in this thesis. The first assumption is that is will be possible to generate or receive dynamic information about delays of trains. The second assumption is that it will be possible to determine the position of the user in an acceptable way.

4.3.1 Dynamic delay information

Concerning dynamic information about delays, NS currently provides information on the actual departure time of trains for 83 (of about 400) stations within The Netherlands. This information is actualised and includes delays of 5 minutes or larger. For the rest of the railway tracks, this information is not available. We assume either NS will make this information available or the information can be generated by the system itself, using handheld devices capable of determining the position of the traveller accurately (for instance using GPS).

4.3.2 Determination of the user position

The second assumption is that it will be possible to determine the position of the user in an acceptable way. One possibility is to let the user specify his position manually, another possibility is to determine the position automatically, for instance by using GPS or GSM positioning. In order to provide all described functionality, some developments have to take place on the current handheld devices, which are described in the appendix. By implementing a first prototype which is able to run on the Sharp SL-C860 Zaurus handheld, which lets the user manually specify his position, we assume to prove that a PITA handheld application can be implemented in an acceptable way for its users.

Figure 4.2 The Sharp SL-C860 Zaurus handheld

4.4 Requirements

This section describes the requirements for the PITA system. The requirements are split in system requirements and user requirements which are described in the following sections.

4.4.1 System requirements

Considering the large amount of potential users of the system (1 million each day), it is necessary for the system to be able to handle these numbers of users simultaneously. Besides this, the system should be scalable, which means it is possible to extend the running system with more processing power as more users are using the system. Besides scalability, reliability is also very important. The system should be able to handle system crashes or lost connections and minimise the possibility for users to get 'stuck', because the system is not functioning any more.

As travellers are mobile users, the system should work on a handheld device with wireless connections. This ensures the traveller to always have the optimal advice on the device in his pocket, in stead of having to remember trainnumbers or platformnumbers. The last requirement is that the system should give optimal solutions for the traveller, concerning available delay information, static route information and the user preferences. The user should be able to completely trust the solutions from the system and not have doubts about the optimality of the presented solutions.

4.4.2 User requirements

The most important requirement for the users of the system is probably the ease of use of the system. The application should not ask unnecessary questions. It should be usable by non experienced users very easily, but experienced users should have the option to change some advanced preferences in order to optimally fit their demands.

Beyond the scope of this thesis, but very important in order to realise a commercially usable system, is the design of interaction dialogues, a multi modal user interface which can be adapted to the preferences of the user, automatic learning of the user preferences and testing the usability of the system.

4.5 The PITA system model

This section describes the considerations that lead to the model which forms the basis of the global design. First the considerations between a centralised or a decentralised method are described. Next the model for the design is presented.

4.5.1 Centralised versus decentralised model

Two general approaches for designing the system were considered: the centralised approach and the decentralised approach. The centralised system would have a single server running the software and in the decentralised model several separate servers and even handheld computers are used which are linked to each other. In the decentralised model, processing power as well as data can be distributed over different computing devices.

4.5.1.1 Centralised system

A centralised system is a lot easier to design and implement, because all programs and data are available on the same server. Also because all delay data is available on this server, it is possible to find the optimal routing solution, considering all delays. A disadvantage is that the single server creates a central point-of-failure, which can cause the total system to go down because of a small problem with the hardware, software or network connection. Considering the large amount of users, it is necessary to have a very expensive server to handle all simultaneous requests.

4.5.1.2 Decentralised system

The main disadvantage of the decentralised system is the complexity involved in designing and implementing the system. In case the delay data is distributed over several computers, sub-optimal solutions can be found in stead of optimal solutions, because the delay data has to be synchronised over several computing devices. The decentralised system can however cope with the problem of failures to parts of the system and keep running on the working servers. This makes the system more fault-tolerant. Next to this, because the application is distributed over several computers, each individual computer can be cheaper, because it needs to handle only a part of the requests. Finally the decentralised model corresponds better to the availability of delay data, which is explained in the next section.

4.5.1.3 Historic background

During the process of privatization of the Dutch Railways (Nederlandse Spoorwegen), the Dutch Government decided to split the ownership of the railway tracks from the railway companies using the tracks. The ownership of the railway tracks remained with the Dutch Government and as of the first of July 2002 all stocks of Railinfratrust BV were acquired by the Dutch Government. January 1st, 2003 the organisation was renamed ProRail and consists of ProRail Railinfrabeheer (maintenance), ProRail Railned (capacity management) and ProRail Railverkeersleiding (traffic-control).

ProRail Railverkeersleiding is responsible for controlling the traffic on the Dutch railways, coordination in case of calamities and providing information about delays. To do this, The Netherlands are divided into 4 regions: North-West, North-East, South-West and South-East, which are controlled from 4 different control centres, located in Amsterdam, Zwolle, Rotterdam and Eindhoven.

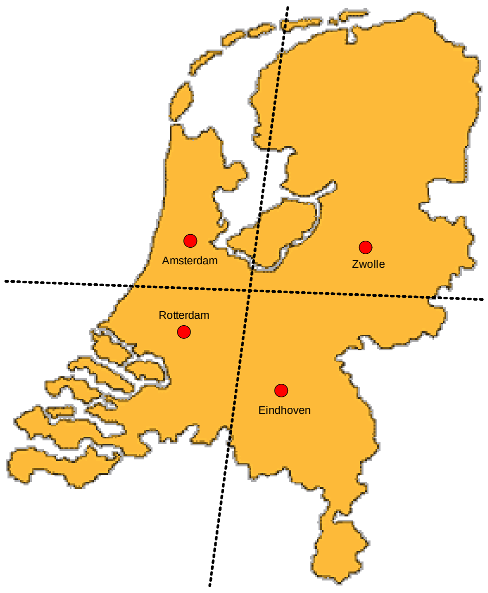


Figure 4.3 ProRail Railverkeersleiding has divided The Netherlands in 4 regions with 4 control centres

Because the control of ProRail Railverkeersleiding is distributed over 4 different locations as shown in figure 4.3, the decentralised model corresponds best to the availability of delay data.

The following table summarizes the advantages and disadvantages of the two models:

	Centralised model:	**Decentralised model:**
Advantages:	Easier to design and implement Optimal solutions can be found	No central point-of-failure Standard hardware can be used Model fits the availability of delay data
Disadvantages:	Central point-of-failure Expensive server costs	Difficult to design and implement Sub-optimal solutions are used

Considering the advantages and disadvantages, the more difficult design and implementation of the decentralized model can be seen as a challenge. This makes the decentralized model the preferred model for the system.

4.5.2 The distributed PITA model

Figure 4.4 shows the model of the distributed PITA system. The system is based on several servers which are connected by Internet, as shown by the blue inner circle. The yellow synchronisation layer realises a synchronised system clock. Also each server stores a copy of the static train schedule, which is extended with regional delay information. Interregional delay information is synchronised across the servers. Finally each server runs a JADE agent platform, shown in the brown outer layer, which all together form a single fault tolerant agent platform as described in section 3.2.5. The agent platform is used by the agents to communicate with the rest of the system.

The system connects to the external NS website in order to import information about train delays. This data is parsed from the HTML pages which are collected using a HTTP connection to the external server. This solution is sub-optimal, as it will generate a lot of data flow between the NS webserver and the rest of the system. Also the website does not give an overview of all actual delays, but only offers the possibility to query departing trains (with delay information) per station, which means 83 queries are necessary to have a complete picture of all delays.

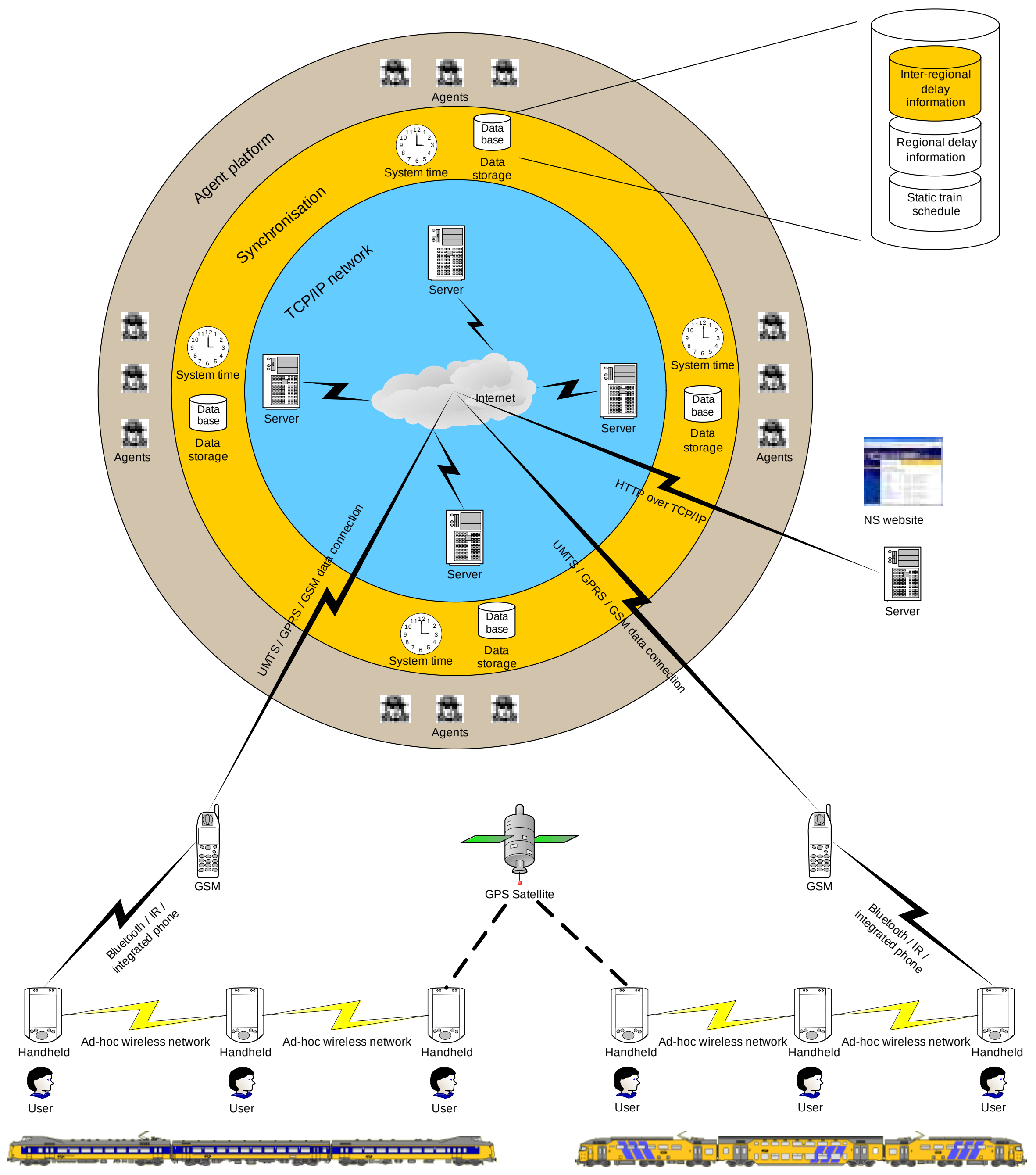


Figure 4.4 The model for the design of the distributed PITA application

Handheld devices connect to each other by wireless connections (for instance Wireless LAN) to form an ad-hoc network. Each of the ad-hoc networks connects to the Internet by (built-in) mobile phones using GSM Data, GPRS or UMTS connections to contact the disitributed server backend. These connections are currently available, but the costs of using these connections are high. The difference between a GSM Data connection and the GPRS and UMTS connections is that the GSM data connection is charged per minute of usage, while the GPRS and UMTS connections are "always on" and only the bandwidth usage is charged. Therefore the GPRS and UMTS connections are prefered for the PITA application.

The position of the user can be determined for instance by GPS or by tracking the GSM phone. The next chapter will describe the different aspects of the distributed PITA model in detail.

Chapter 5: Design

This chapter presents the design or the PITA system and application. First a design of the use cases is presented, followed by the system design. Then the data model is described containing a UML class diagram and sequence diagrams of the system. Finally the user interfaces are described.

5.1 Use case design

Three different types of users are distinguished for the use case design. First we will discuss the traveller using the system to guide him to his destination. Next, we distinguish an operator user who is responsible for running a real-time system, for instance for NS and a simulator user who simulates travellers using the PITA system, for instance for research purposes. The next sections describe the different use cases for each of the roles.

5.1.1 The traveller

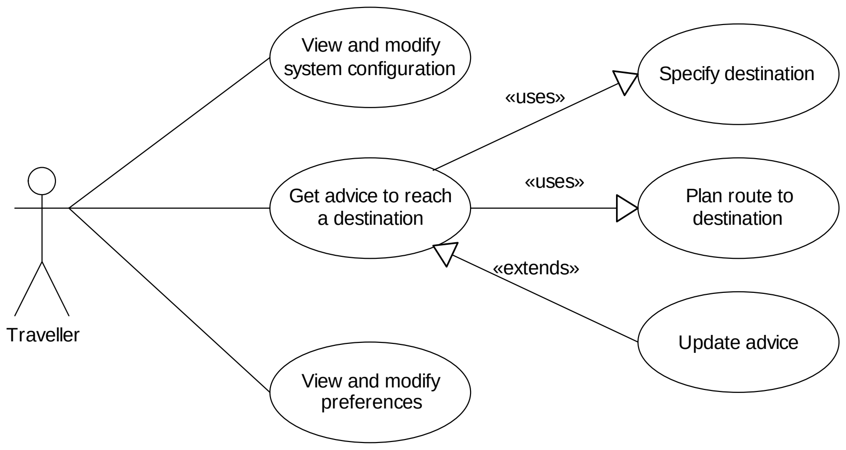

View and modify system configuration
Specify destination
«uses»
Get advice to reach a destination
«uses»
Plan route to destination
«extends»
Traveller
Update advice
View and modify preferences

Figure 5.1 The use-case diagram of the traveller

The traveller is able to view and modify the configuration of the system and his preferences and to get an advice on how to reach his destination. In order to give a personalised advice, the user needs to specify a destination and the system will plan an optimal route given the user preferences. While the user travels, the system continually updates the advice for the user, as new information about delays or cancelled trains becomes available.

5.1.2 The operator and the simulator

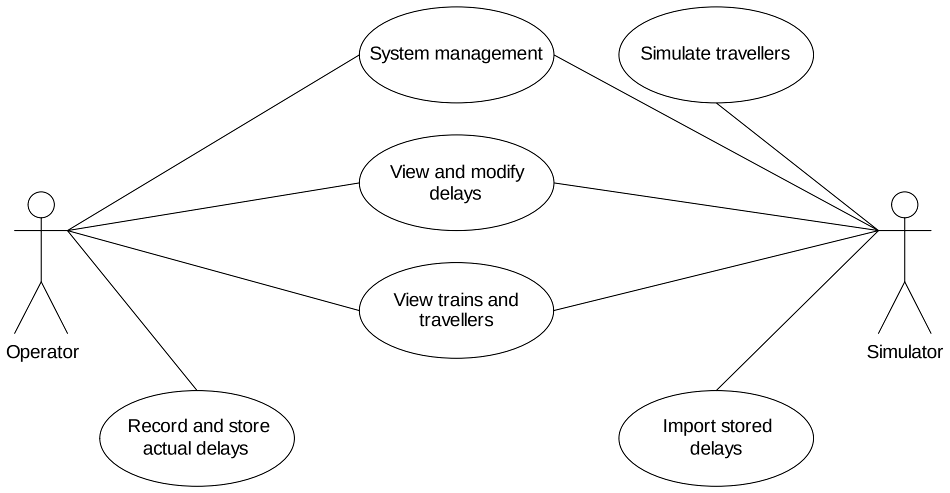


Figure 5.2 The use case diagram of the operator and the simulator

The operator is responsible for running a real-time system, which guides travellers using live data about trains and delays. In order to do so, the operator uses the system management to view the status of the different components of the system and to start and stop individual components. He can also view the actual delays and add or update the delay data. Next to that he can get a view on the current situation of moving trains and actual travellers using the system. Lastly he can record delay data, which can then later be used by the simulator to simulate delays in a simulation. Next to the other functions which a real-time operator has, the simulator is also able to simulate travellers using the PITA system.

5.2 The system design

This section describes the design of the system. First the global design showing the different subsystems is described, next each of the individual subsystems is discussed.

5.2.1 Global design

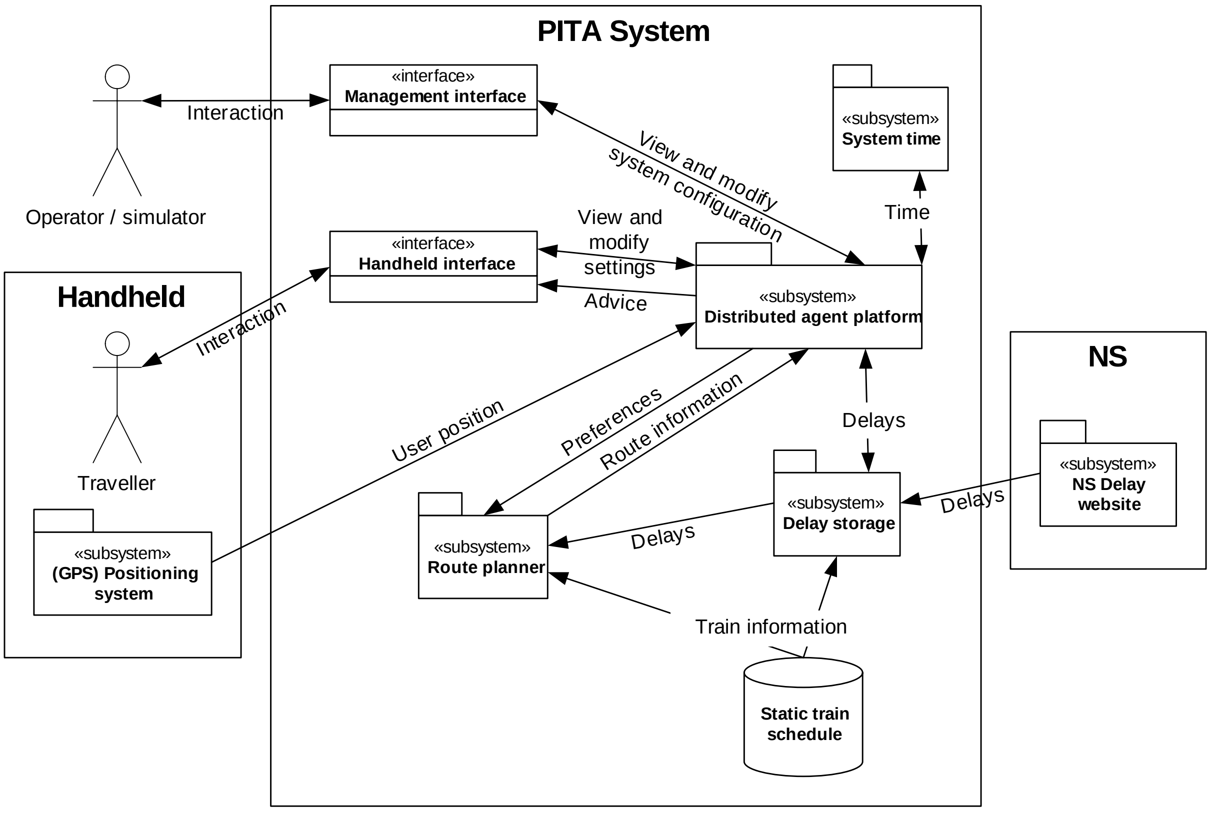

PITA System
Operator / simulator
Interaction
«interface» Management interface
View and modify system configuration
«subsystem» System time
Time
«interface» Handheld interface
View and modify settings
Advice
«subsystem» Distributed agent platform
Handheld
Interaction
Traveller
«subsystem» (GPS) Positioning system
User position
Preferences
Route information
Delays
«subsystem» Route planner
Delays
«subsystem» Delay storage
Delays
NS
«subsystem» NS Delay website
Train information
Static train schedule

Figure 5.3 The global system design of the PITA system

The PITA system has two user interfaces: a management interface and a handheld interface. The management interface is used for viewing and modifying the system configuration. The handheld interface is used by travellers using a handheld device to communicate with the system. A central role is played by the distributed agent platform, which runs personal agents for each individual traveller. In order to give an optimal advice to the traveller, the personal agents receive information about the best route to the destination, given the current delay situation and occurring delays during the trip. Using the user position it is possible to determine whether the user is experiencing a delay. This information about delays is also communicated back to the system itself, to actualise the stored delay information. Next to that the NS Delay website is used to fill the delay system. The route planner uses the static train schedule combined with the actual delay information to search an optimal route for the traveller, based on the preferences of the individual traveller.

5.2.2 Delay subsystem

The delay subsystem is responsible for gathering and storing information about train delays. Two important sources for this information are the users of the system themselves and a website of NS which gives information on actual departure times of individual trains.

5.2.2.1 Determining the position of the user

Probably the most important information for the PITA system is the location of the user. This is necessary to be able to give a travelling advice, estimate delays and to get a view on traveller flows. A possibility is to ask the user of the system to specify his position, but there are also automated ways to determine the position. One possibility is to use the Global Positioning System (GPS). Another possibility is to make use of the GSM system.

5.2.2.1.1 Global Positioning System (GPS)

The Global Positioning System consists of 24 satellites around the world which transmit position data. Using a GPS receiver (costing around 100 euros) it is possible to receive these signals and determine the position with an accuracy of about 25 meters, using the angle at which the signal is being received from different satellites. Currently every Dutch train is being equipped with a GPS receiver, but this is only for usage by the railway companies. Using a normal GPS receiver within a train is not ideal, because the train forms a Faraday cage, which blocks the signal from the satellites. Using an antenna, better results can be achieved by placing it against the window or outside the train. For the PITA system it would be ideal to make use of the GPS receivers currently being installed on every train. Another possibility is using the GSM infrastructure to determine the position of the user.

5.2.2.1.2 GSM cell-id location

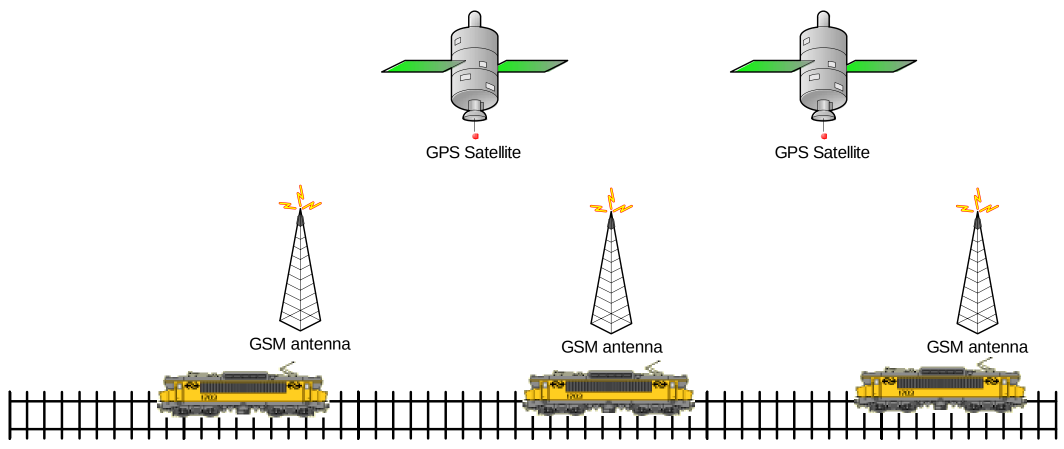
The GSM system consists of a very extensive network of antennas. The antennas are distributed according to the demand for mobile communications, which means the density of the antennas increases around cities (to about one antenna every square kilometre). Each antenna has a unique identification code (cell-id). While the user travels, the mobile phone connects to the nearest GSM antenna, having the strongest signal. Using the cell-ids, it is possible to determine the position of the user coarsely. A disadvantage of the cell-ids is that every mobile operator has its own network of antennas with its own set of identification codes, which means there are 5 networks with different cell-ids in The Netherlands.

5.2.2.2 Delay information

Several types of delays can be distinguished. In the first place there are scheduled delays, which are caused by for instance maintenance of the railroad. Information about scheduled delays is available from the website of NS well before the delays occur. Next to this, there are incidental delays, which occur because of all sorts of unexpected problems occurring while a train is riding a certain track. Information about incidental delays becomes available only several minutes up to some hours before the delay occurs. Finally there are structural incidental delays, which are delays that occur continually because of external factors, like to tight planning of the travel time on a certain track, or to tight planning of train changes at a station. Only experienced travellers know about structural delays. The structural delays require a probabilistic approach in order to predict these occurring. The other delays can be determined directly from the NS website or from travelling users using the PITA system. As a first approach, a deterministic model was implemented, which combines the scheduled and incidental delays and leaves out the structural delays.

Incidental delay information will be generated by travellers using the PITA system. After receiving information about the position of the user, this information can be used to determine whether the user is travelling according to the schedule or is experiencing a delay, by interpreting the position data. Next to this, NS provides some information about incidental and scheduled delays through their website as is discussed in the following sections.

5.2.2.2.1 *Interpreting position data*



Time:	t=00:30.00	t=00:35.00	t=00:40.00
GPS			
lattitude	N 25 dg 3,337'	N 25 dg 58,221'	N 25 dg 62,342'
longitude	E 121 dg 31,552'	E 121 dg 8,221'	E 121 dg 1,321'
GSM			
cell-id KPN	ABCD-0123	ABCD-0124	ABCD-0126
cell-id Vodaphone	6655-D1FF	6656-E2FF	6656-E300
cell-id T-Mobile	0001-0001	0001-0004	0002-0001
cell-id Orange	AABB-CCDD	ABAB-CDCD	ABAB-DDDD
cell-id Telfort	5432-1234	5433-3210	5432-1235

Figure 5.4 An example of GPS or GSM data for determining the position of the user

After receiving GPS longitude and latitude information or GSM cell-id information, it is necessary to interpret this data to be able to predict whether the user is experiencing a delay. This can be done by keeping track of a profile for every track in The Netherlands, containing timestamps and position data for a train having no delay. By looking up the position data in the table for a normal train and comparing this data with the actually received data, it is possible to estimate the delay of the user. The conversion from position data to delay data can be done with a mathematic function using the distance between the reference positions and the actual positions. Another solution would be to use a neural network or expert system to interpret this data, which is certainly necessary for the GSM system, because there is no direct mapping between the cell-ids and the location.

5.2.2.2.2 *Data provided by NS*

Next to interpreting the user position for incidental delay information, it is currently (as of March 25th, 2004) also possible to get information about the actual delays of individual trains. NS provides up-to-date delay information for 83 stations in the Netherlands on their website, giving an accurate view on the departing trains at these stations. This information includes delays of 5 or more minutes, reschedule information and cancelled trains. This information can be used as an excellent basis to extend with delay data gathered from the user positions. Next to this, scheduled delay information is also available from the NS website, which makes it possible to incorporate these also in the system.

5.2.2.3 Distributed delay information

The PITA system will be distributed over several servers, which makes it possible to distribute the delay information in several different ways over the individual servers. The first possibility is to synchronise all delay information across all servers. As there are 6000 trains in the static train schedule, this means 6000 objects have to be synchronised throughout the network of distributed servers. In order to minimise the necessary synchronisation across the servers, but to maintain the optimal routing as much as possible, another solution is chosen for the distributed PITA application.

5.2.2.3.1 Regional and inter-regional delay information

In the distributed PITA system, each server will be responsible for a certain region of The Netherlands. To avoid unnecessary synchronisation of the delay information across the servers, the delay information is modelled as shown in figure 5.5.

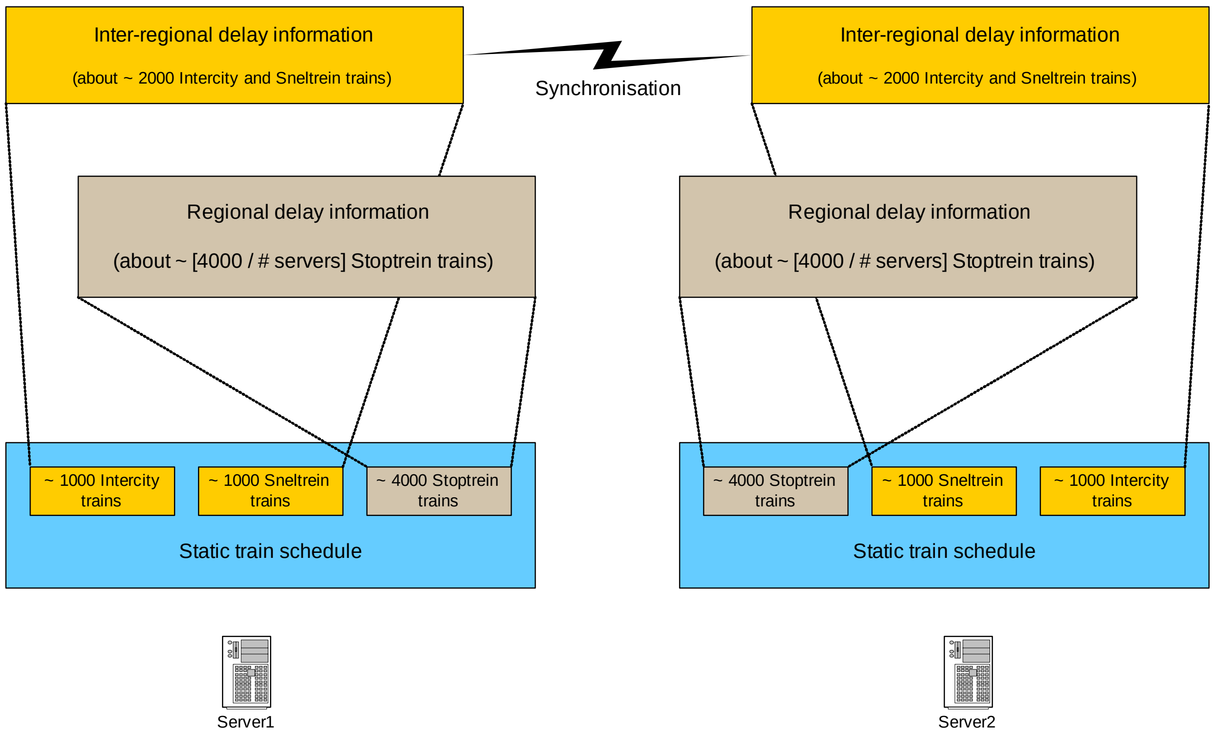

Inter-regional delay information
(about ~ 2000 Intercity and Sneltrein trains)
Synchronisation
Inter-regional delay information
(about ~ 2000 Intercity and Sneltrein trains)
Regional delay information
(about ~ [4000 / # servers] Stoptrein trains)
Regional delay information
(about ~ [4000 / # servers] Stoptrein trains)
~ 1000 Intercity trains
~ 1000 Sneltrein trains
~ 4000 Stoptrein trains
Static train schedule
~ 4000 Stoptrein trains
~ 1000 Sneltrein trains
~ 1000 Intercity trains
Static train schedule
Server1
Server2

Figure 5.5 The delay model for synchronising Inter-regional delay information

Each server will have a copy of the static train schedule, extended with regional delay information, about Stoptrein trains within the region. Regional delay information is information about trains that do not cross region-borders. On top of that, delays for trains which cross multiple regions are synchronised across all servers. This way information concerning local trains is available only on the server responsible for the specified region while information about interregional trains is available throughout the network. Another possibility would be to build an expert system with knowledge about alternative routes. In case a delay occurs on a certain track, the expert system can then determine which users should be noticed about this delay because a better route to their destination is available. This was already done by Gerritjan Eggenkamp (see [Egge01]) for automobiles on motorways, and will involve interviewing a lot of "experienced travellers" about route alternatives and modelling this information in a rule base.

5.2.2.3.2 Sub-optimal solutions

A disadvantage of the chosen model for distributing the delay information is that in some cases sub-optimal solutions can be found, since not all delays are replicated over all servers. Figure 5.6 shows an example situation of how a user at train station A with destination station C will get a sub-optimal advice due to the synchronisation scheme which is used.

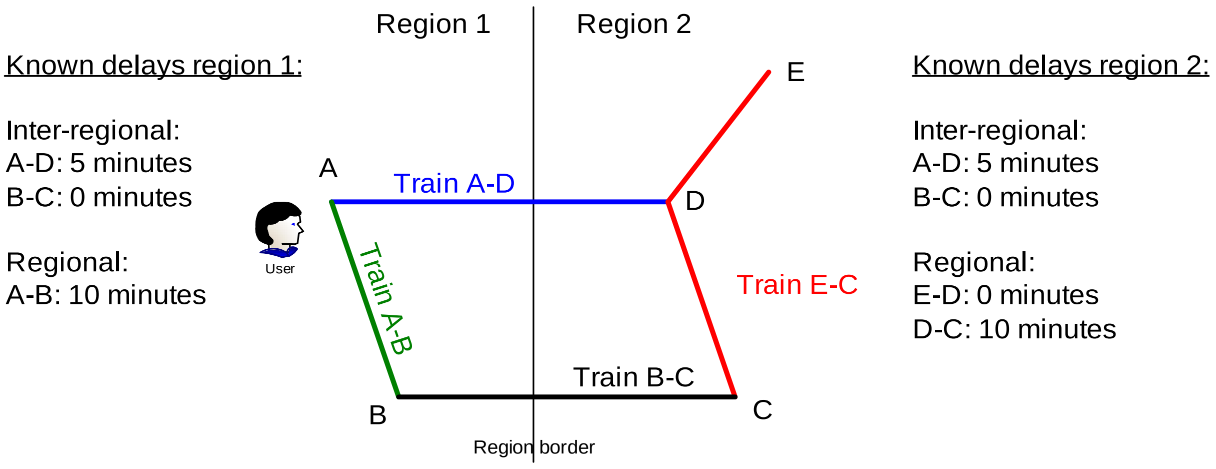


Figure 5.6 An example situation where the user will get a sub-optimal advice

Only information about inter-regional trains is synchronised over all servers, so server 1 for region 1 will have no knowledge about the delay of the regional train E-C. The optimal route in the perception of the agent on server 1 will be to go from station A to station D and then take train E-C to destination station C, with 5 minutes delay, because route A-B-C will have 10 minutes delay. The nominal travelling times of route A-D-C is assumed to be equal to route A-B-C. The example shows how delays of regional trains can have consequences for inter-regional travellers. However the (minor) disadvantages of the sub-optimal routes are compensated by less synchronisation information which has to be exchanged between the servers.

5.2.3 Route planner

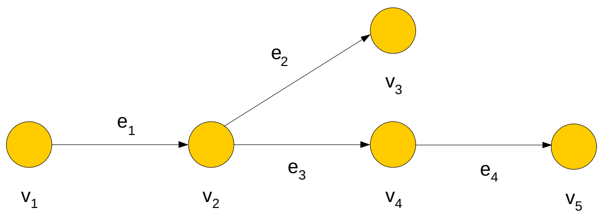
An important part of the PITA system is the route planner, which should be able to find the best route to the destination in the train schedule combined with available data about train delays and using the user preferences. Eduard Tulp (in [Tulp91]) has developed the DYNET algorithm for searching time-table networks. The next sections describe the algorithm and how it is used in the PITA system.

5.2.3.1 The best route

To find the fastest route between two stations is a well defined problem. However, the best route between two stations can depend on several factors which make it a non trivial problem to find this route. The best route can depend on situations, personal preferences and general preferences. Generally people find the route arriving as early as possible at the destination, having the shortest overall travelling time and having the least possible train changes the best route to a destination. However some people prefer a train going directly from the start to the destination which arrives later in stead of a faster connection having train changes, because they can remain seated in the same train. Another reason can be that train changes have the risk that the next train may be delayed. So by having a direct train, the arrival time at the destination is more certain than the arrival time of a route with train changes.

5.2.3.2 A discrete dynamic network

In order to represent a time-table network, it is necessary to use a discrete dynamic network in stead of a normal (directed) graph in which vertices represent stations and edges represent train connections. Trains do not depart continuously, so the travelling time between two vertices varies, depending on the arrival and departure time of trains. This is why a discrete network is used, which associates with each edge a start and end value, representing the departure and arrival times of an individual train. Another problem are train changes. Depending on which train is used to arrive at a station, it is sometimes necessary to walk to another platform to take the next train, which can take several minutes. To represent time spent on train changes, a lookup table is introduced per node which stores the amount of time necessary to go from an arriving edge to the next departing edge. In fact this introduces visiting costs per node, which will be named CON. Also each edge ***e*** gets an attribute ***id(e)*** which identifies the train riding the edge. Figure 5.9 shows an example discrete dynamic network representing 5 stations and 4 trains.



start(e_1) = 9:00, end(e_1) = 9:15	id(e_1) = train 123
start(e_2) = 9:18, end(e_2) = 9:30	id(e_2) = train 321
start(e_3) = 9:19, end(e_3) = 9:30	id(e_3) = train 111
start(e_4) = 9:31, end(e_4) = 9:35	id(e_4) = train 111

CON(e_1,e_2) = 4 minutes
CON(e_1,e_3) = 3 minutes
CON(e_3,e_4) = 0 minutes (same train)

Figure 5.7 An example discrete dynamic network

In order to go from the train represented by edge e_1, to the train represented by edge e_2, 4 minutes are necessary to go from the arrival platform to the next departure platform. Since the arrival time at the station represented by vertex v_2 is 9:15 and the departure time of e_2 is 9:18, this connection cannot be made. However, the connection from e_1 to e_3 is possible.

5.2.3.3 The DYNET algorithm

The DYNET algorithm (see [Tulp91]) uses a discrete dynamic network to search an optimal train route between two vertices. The algorithm consists of two passes, a forward pass and a backward pass. The forward pass finds a route reaching the destination vertex at the earliest possible arrival time. The backward pass optimises the route found in the forward pass by finding the route having the last possible departure time which reaches the destination vertex at the same time. Eduard Tulp describes the DYNET algorithm using macro operators, which also considers the number of train changes in the solution, as follows:

Consider a discrete dynamic network consisting of the graph *G = (V,E)* and the connection function *CON*. The maximum value of *CON* at a vertex *u* is *maxiCON(u)*. The number of train changes in a path *P* is denoted by *CHANGES(P)*. The *change_value* is the time that we are prepared to travel to avoid one train change. The corrected *end* value of a path *P*, $c_{end}(P)$, and the corrected *start* value of a path *P*, $c_{start}(P)$, are defined as follows:

$c_{end}(P) = end(P) + CHANGES(P) \times change\_value$

$c_{start}(P) = start(P) - CHANGES(P) \times change\_value$

continued on the next page…

The label of a vertex v is $\lambda(v)$ in the forward pass and $k(v)$ in the backward pass. The smallest end value that was tried to label u from v is denoted by $\omega(u,v)$. The number of train changes of the (corrected) best path arriving at a vertex v is denoted by $\varphi(v)$. A train riding a sequence of edges e_0, e_1, e_2 is identified by an extra edge attribute $id(e_0) = id(e_1) = id(e_2)$. The two special vertices s and t of the network are the starting and terminating vertices. We want to find a legal path from s to t in our discrete dynamic network, where the corrected *end* value of the path is minimum and given this corrected *end* value, the corrected *start* value of the path is maximum and its value is at least T_{start}.

Then the DYNET algorithm, consisting of two passes, is as follows:

PASS 1:

```
(1) $\lambda(s) \leftarrow Tstart$ and for all $v \in V$, $v \neq s$, $\lambda(v) \leftarrow \infty$.
    Create a partial path $P_0$ consisting of $s$ only, $c_{end}(P_0) \leftarrow T_{start}$.
    For all $v \in V$, $\omega(u,v) = \infty$ for each neighbour $u$ of $v$.
(2) $F \leftarrow \{ P_0 \}$.
(3) Let $P_m$ be a partial path $s, e_0, u_1, .. , u_{j-1}, e_{j-1}, u_j$ in $F$ for which
    $c_{end}(P_m)$ is minimum; if $F$ is empty then stop, no complete path could
    be found.
(4) if $u_j = t$, stop, $P_m$ is a complete path with an optimal corrected end
    value.
(5) if $CHANGES(P_m) = \varphi(u_j)$ and $c_{end}(P_m) < \lambda(u_j) + maxiCON(u_j)$, then for
    every relevant edge $e_j : u_j \rightarrow u_{j+1}$ :
      create a partial path $P_n = s, e_0, u_1, .. , u_{j-1}, e_{j-1}, u_j, e_j, u_{j+1}$.
      $c_{end}(P_n) \leftarrow end(P_n) + CHANGES(P_n) \times change\_value$
      if $\lambda(u_{j+1}) > c_{end}(P_n)$
      then $\lambda(u_{j+1}) \leftarrow c_{end}(P_n)$ and $\varphi(u_{j+1}) \leftarrow CHANGES(P_n)$.
      $F \leftarrow F + \{ P_n \}$.
      if $end(e_j) < \omega(u_{j+1}, u_j)$
      then $\omega(u_{j+1}, u_j) \leftarrow end(e_j)$.
(6) if $CHANGES(P_m) = \varphi(u_j)$ and $\lambda(u_j) + maxiCON(u_j) \leq c_{end}(P_m)$.
    then for the edge $e_j : u_j \rightarrow u_{j+1}$ for which $id(e_j) = id(e_{j-1})$:
      create a partial path $P_n = s, e_0, u_1, .. , u_{j-1}, e_{j-1}, u_j, e_j, u_{j+1}$.
      $c_{end}(P_n) \leftarrow end(P_n) + CHANGES(P_n) \times change\_value$
      if $\lambda(u_{j+1}) > c_{end}(P_n)$
      then $\lambda(u_{j+1}) \leftarrow c_{end}(P_n)$ and $\varphi(u_{j+1}) \leftarrow CHANGES(P_n)$.
      $F \leftarrow F + \{ P_n \}$.
(7) $F \leftarrow F - \{ P_m \}$ and go to step (3).
```

A relevant edge is defined as follows: given a partial path $u_0, .. , u_{j-1}, e_{j-1}, u_j$ and a vertex u_{j+1}, then the relevant edges from u_j to u_{j+1} are the edges $e_j : u_j \rightarrow u_{j+1}$ for which the following two *ordered* conditions hold:
(1) $start(e_j) \geq end(e_{j-1}) + CON(u_j, e_{j-1}, e_j)$, and
(2) $end(e_j) < end(e_{min}) + maxiCON(u_{j+1})$,
where $end(e_{min})$ is the minimum *end* value of any edge satisfying (1).

continued on the next page…

```
PASS 2:
(1) $k(t) \leftarrow \lambda(t)$ and for all $v \in V$, $v \neq t$, $k(v) \leftarrow -\infty$.
    Create a partial path $P_0$ consisting of $t$ only, $c_{start}(P_0) \leftarrow \lambda(t)$.
(2) $F \leftarrow \{ P_0 \}$.
(3) Let $P_m$ be a partial path $u_j, .., u_{k-1}, e_{k-1}, t$ in $F$ for which
    $c_{start}(P_m)$ is maximum.
(4) if $u_j = s$, stop, $P_m$ is an optimal complete path.
(5) if $\varphi(u_j) = CHANGES(P_m)$ and $c_{start}(P_m) > k(u_j) - maxiCON(u_j)$, then for
    every relevant edge $e_{j-1} : u_{j-1} \rightarrow u_j$, with $\omega(u_j, u_{j-1}) \leq start(P_m)$:
      create a partial path $P_n = u_{j-1}, e_{j-1}, u_j, .., u_{k-1}, e_{k-1}, t$.
      $c_{start}(P_n) \leftarrow start(P_n) - CHANGES(P_n) \times change\_value$
      if $k(u_{j-1}) < c_{start}(P_n)$
      then $k(u_{j-1}) \leftarrow c_{start}(P_n)$ and $\varphi(u_{j-1}) \leftarrow CHANGES(P_n)$.
      $F \leftarrow F + \{ P_n \}$.
(6) if $\varphi(u_j) = CHANGES(P_m)$ and $k(u_j) - maxiCON(u_j) > c_{start}(P_m)$.
    then for the edge $e_{j-1} : u_{j-1} \rightarrow u_j$, with $id(e_{j-1}) = id(e_j)$,
    and with $\omega(u_j, u_{j-1}) \leq start(P_m)$,
      create a partial path $P_n = u_{j-1}, e_{j-1}, u_j, .., u_{k-1}, e_{k-1}, t$.
      $c_{start}(P_n) \leftarrow start(P_n) - CHANGES(P_n) \times change\_value$
      if $k(u_{j-1}) < c_{start}(P_n)$
      then $k(u_{j-1}) \leftarrow c_{start}(P_n)$ and $\varphi(u_{j-1}) \leftarrow CHANGES(P_n)$.
      $F \leftarrow F + \{ P_n \}$.
(7) $F \leftarrow F - \{ P_m \}$ and go to step (3).


A relevant edge is defined as follows: given a partial path $u_j, e_j, u_{j+1}$,
.. , $u_k$ and a vertex $u_{j-1}$, then the relevant edges from $u_{j-1}$ to $u_j$ are the
edges $e_{j-1} : u_{j-1} \rightarrow u_j$ for which the following two ordered conditions hold:
(1) $end(e_{j-1}) \leq start(e_j) - CON(u_j, e_{j-1}, e_j)$, and
(2) $start(e_{j-1}) > start(e_{max}) - maxiCON(u_{j-1})$,
    where $start(e_{max})$ is the maximum start value of any edge satisfying
    (1).
```

To use the available information about train delays, the DYNET algorithm, as used in the PITA system, will add the delay of a train on a certain track e to the *end(e)* attribute of the edge *e* and to the *begin(e_j)* and *end(e_j)* attributes of all consequent edges e_j having *id(e_j) = id(e)* and for which *begin(e_j) ≥ end(e)*. The CON function was implemented as a function, taking as parameters a minimum change time and a change time per platform that has to be crossed as follows (see also section 6.3.2 for a detailed description):

CON(e_{j-1}, e_j) = 0 if *id(e_{j-1}) = id(e_j)*, or
= train_change_time + (extra_change_time_per_platform x
| getarrivalplatformnumber(e_{j-1}) - getdepartureplatformnumber(e_j) |)

5.2.3.4 Using DYNET in the PITA system

As described in the previous section, the first pass of the DYNET algorithm finds the route reaching the destination as early as possible and departing from the start as early as possible. For users who are currently already at the start station, this (first pass only) solution can be the best solution, since they can choose to wait for the fastest route or they can try to get near to the destination as soon as possible. By travelling towards the destination the risks of incidental delays are minimised. This is why the route planner of the PITA system can use either the full DYNET algorithm or only the first pass of the algorithm as a user preference. Another user preference is the *change_value* parameter, which defines the maximum extra time per train change a user would accept for a solution having one train change less compared to another route. Using these two parameters, the user is capable of optimising the route algorithm to find the best solution considering his situation as is summarised in table 5.1.

Table 5.1 The choice of routing algorithm depending on the location and preferences of the traveller

Current location:	Travel preference:	Routing algorithm:
At the start station	High risk, performance user	First pass, low change_value (e.g. 0)
At the start station	Low risk, comfortable user	First pass, large change_value (e.g. 15)
Not at the station	High risk, performance user	Two pass, low change_value (e.g. 0)
Not at the station	Low risk, comfortable user	Two pass, large change_value (e.g. 15)

5.2.4 Distributed agent platform

This section describes the distributed agent platform. First the structure of each of the distributed servers is explained and next the functioning of the personal agents running on the distributed agent platform is described.

5.2.4.1 The distributed servers

The distributed agent platform consists of several servers running JADE agent containers as described in section 3.2.5. By using Main-Container replication, the platform will remain running if a server crashes. However, the individual agents which were running on the server that crashed are lost, since individual agents are not replicated across multiple servers. Figure 5.8 shows the structure of the distributed servers as wel as what information is synchronised across the servers.

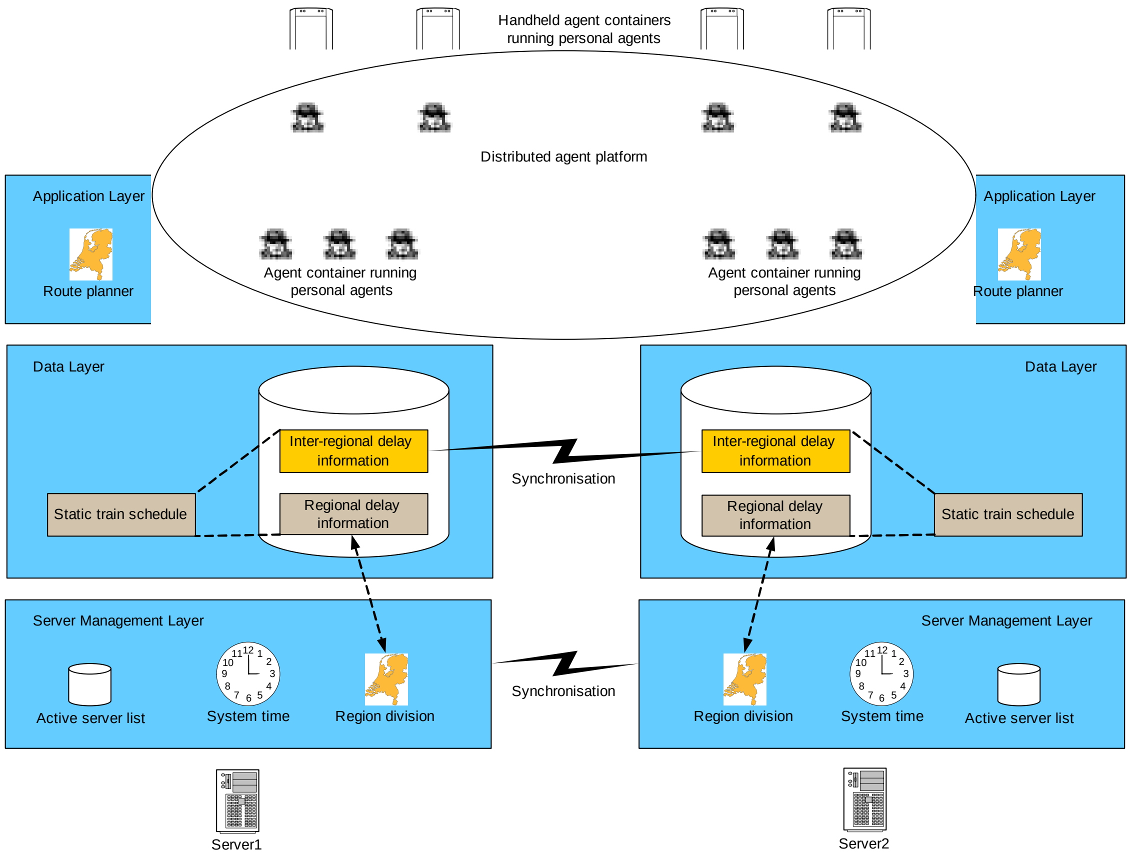


Figure 5.8 The structure of the distributed servers

The Server Management Layer is the first layer. The responsibility of this layer is to synchronise the list of active (and correctly running) servers and the system time over all servers. Also this layer divides the regions which are the responsibility for each server. This forms the basis for the rest of the application. The data layer contains a copy of the static train schedule, which is extended with regional delay information and inter-regional delay information, which is synchronised across each server. The application layer runs the route planner and the agent containers forming the distributed agent platform together with the agents on the handheld devices. The next section describes the functioning of the personal agents running in the agent containers.

5.2.4.2 The personal agent

Personal agents for each individual traveller run on top of the described distributed platform. These personal agents communicate with the rest of the system to generate an optimal advice for the user as shown in the following figure:

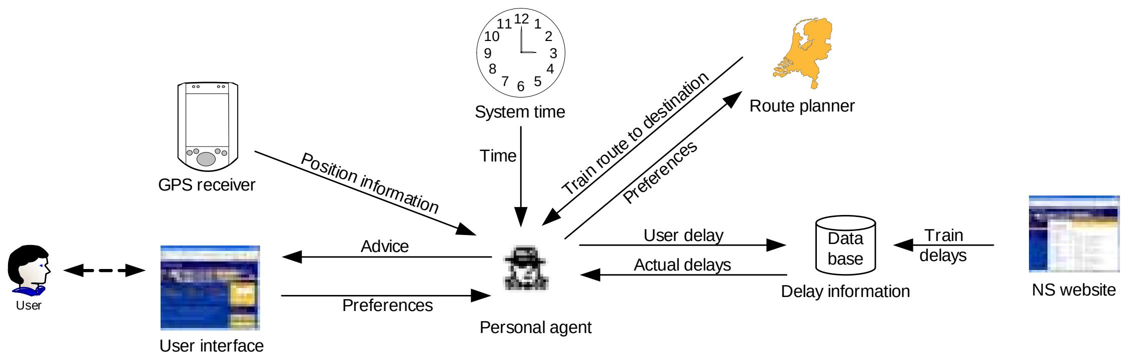


Figure 5.9 The personal agent

The personal agent uses the route planner to find the best route to the destination. Next to that the personal agent stores the preferences of the user and keeps track of the position of the user to detect whether the user is experiencing a delay. As soon as a delay is detected, this is updated in the delay database and a new route to the destination is determined. After updating the delay information, other personal agents which have planned to take this train are notified of the delay and can also determine a new route to their destination.

5.2.5 Handheld application

As described in the previous section, the personal agent and the route planner can run on the distributed servers. However as handhelds become more powerful, it will be possible to run a larger part of the application on the handheld devices as well. This section describes the functions of the handheld devices in three scenario's: either the handheld device is used as an interface only, or the handheld device runs part of the application or the full application runs on the handheld device.

5.2.5.1 The handheld interface

In the handheld interface scenario, the personal agent runs on the server and exchanges only interface commands with the handheld. The handheld itself does not do any application processing, as is shown in figure 5.10.

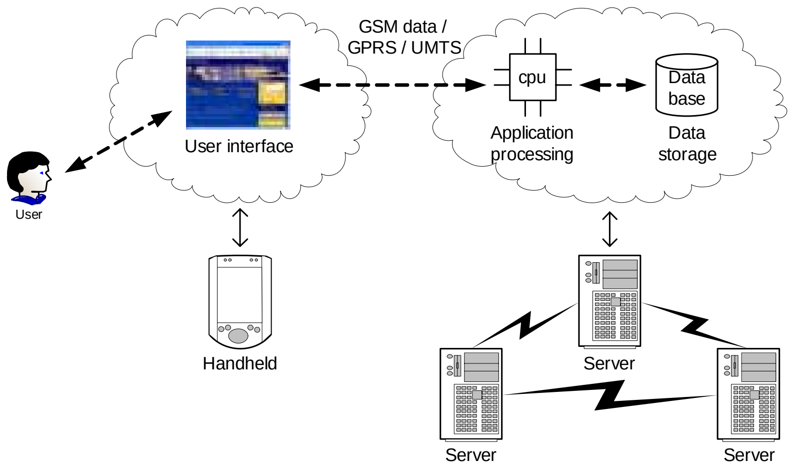


Figure 5.10 The handheld interface model

As the application runs on the server, only the interface has to be developed suitable to run on the handheld device. This makes it possible to rapidly support a large range of handheld devices. On the other hand, if the connection between the handheld and the distributed servers is lost, the application does not function any more and the traveller will not get an updated advice.

5.2.5.2 The handheld application with distributed backend

This section describes how the personal agent can run on the handheld device. Next to that the possibilities for using ad-hoc networks to efficiently share data between handhelds are described.

5.2.5.2.1 The handheld personal agent

By moving the personal agent from the distributed servers to the handheld device, the application becomes more robust and is able to advice the user based on the static information (optionally) stored on the handheld even if the connection to the rest of the system is lost. After moving the personal agent to the handheld device, the route planner can also be moved to the handheld device. This however means all information about delays has to be stored on the handheld device as well, causing a lot of synchronisation between the handheld device and the distributed server backend. Since the current costs of mobile Internet connections are very high, the PITA system will run the personal agent on the handheld device and leave the route planner and delay storage on the distributed servers, as is shown in figure 5.11.

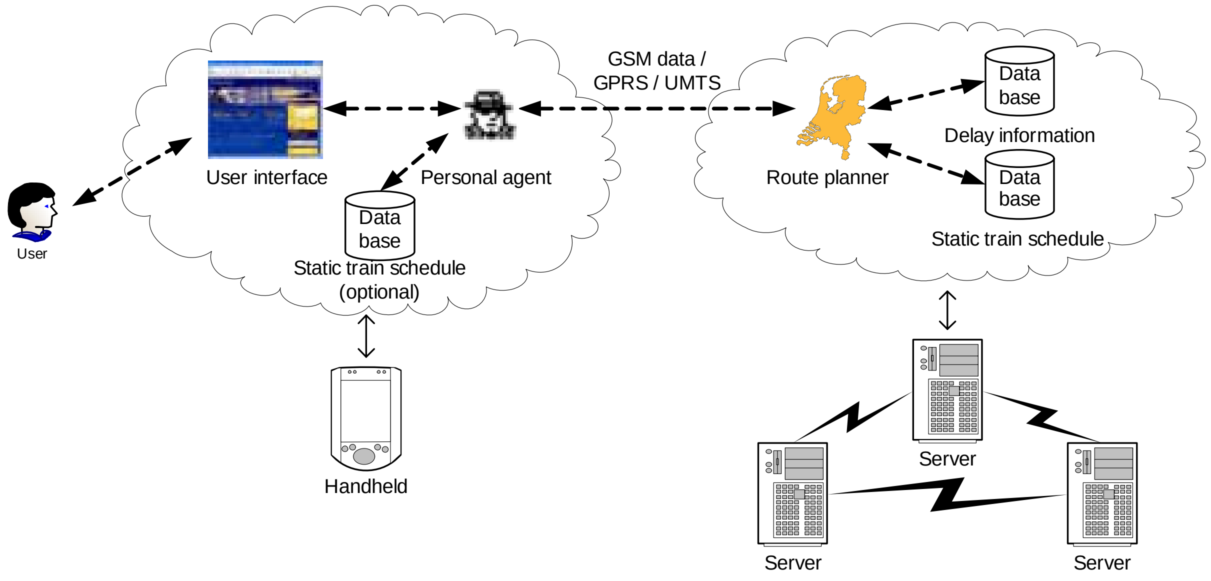


Figure 5.11 The PITA handheld application model

In this scenario, the personal agent stores the user preferences and the current advice and keeps track of the user position. The distributed servers keep track of which route information was given to which personal agents, in order to notify personal agents of delays which could possibly affect their current travelling advice. All delay information remains on the servers and the handheld can optionally store a copy of the static train schedule for situations when the mobile Internet connection is lost.

5.2.5.2.2 *Ad-hoc networking*

As the personal agent runs on the handheld device, it also becomes possible to exchange information over an ad-hoc wireless network between individual handheld devices, without having to use a mobile internet connection. This means the personal agents on different handheld devices can exchange route information and information about delays in order to optimally get to each individual destination. Figure 5.12 shows how a handheld connected only by an ad-hoc wireless network can be connected to the system.

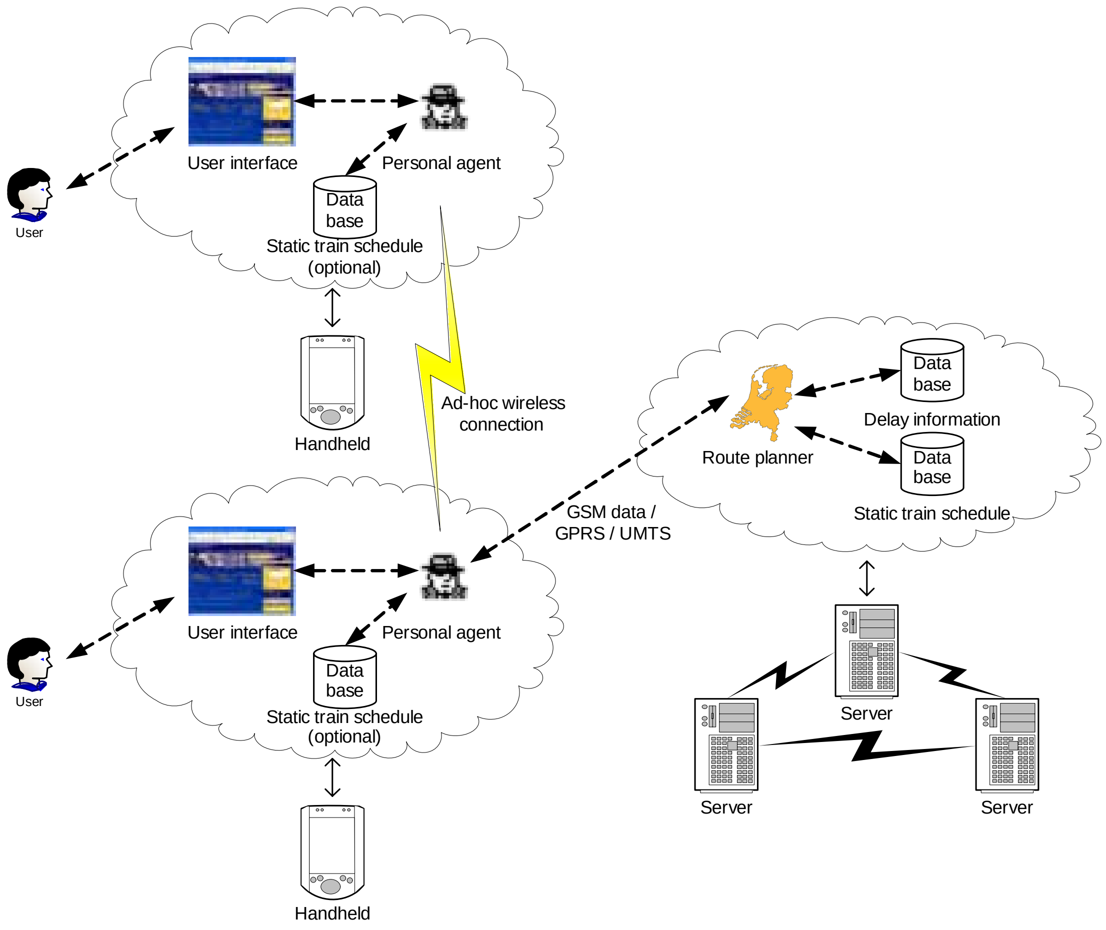


Figure 5.12 A handheld can be connected to the rest of the system by only an ad-hoc wireless connection

By sharing information over an ad-hoc network, it becomes possible to use the system without having a mobile Internet connection per device. The following information can be shared over the ad-hoc network:

- **Position information**
  If several travellers are on the same train, then only a single user needs to have a position device, like for instance a GPS receiver. By sharing this information across the wireless network, all other users are updated of their locations as well.
- **Route information**
  If several travellers (partly) have the same route to their destination, other users can use this route as well and do not need to set up their own mobile Internet connection with the distributed servers.
- **Delay information**
  By sharing information about train delays, this information needs to be communicated only once over the mobile Internet connection and can then cost effectively be shared across the wireless ad-hoc network between different handhelds.

In order for the ad-hoc networking to function in an optimal way, it is necessary to have enough travellers which are equipped with capable devices, having built-in GPS receivers and mobile Internet connection (preferably GPRS or UMTS). This is necessary to have a user base which forms a minimal coverage of the PITA system across the stations and trains in The Netherlands. To realise such a minimal coverage, NS could equip all conductors with a capable PITA device. Another solution would be to give travellers who buy an annual railpass the option to buy such a device for reduced charges.

Another problem is that nobody wants to use his mobile Internet connection all the time to help other users, but also wants to profit from the system from time to time. A solution for this would be to register credits on each handheld device. As the mobile Internet connection of the device is used, the handheld gets credits. If there are more devices with a mobile Internet connection available in an ad-hoc network, the mobile Internet connection of the device with the least credits is used to provide the network with information. This way the handhelds will negotiate what connection is used and which information is shared, leading to ad-hoc created networks and personal agents negotiating and sharing information as they travel along the Dutch railways.

5.2.6 The distributed handheld application

This section describes a possible design for running the full application on the handheld device. This scenario was not used for the implementation of the PITA system, but shows several interesting aspects which can be of interest for further research. Figure 5.13 shows the route planner is also running on the mobile device in this scenario. Besides this, the handheld device also stores delay information locally. This means a central server synchronising the delay information is no longer needed.

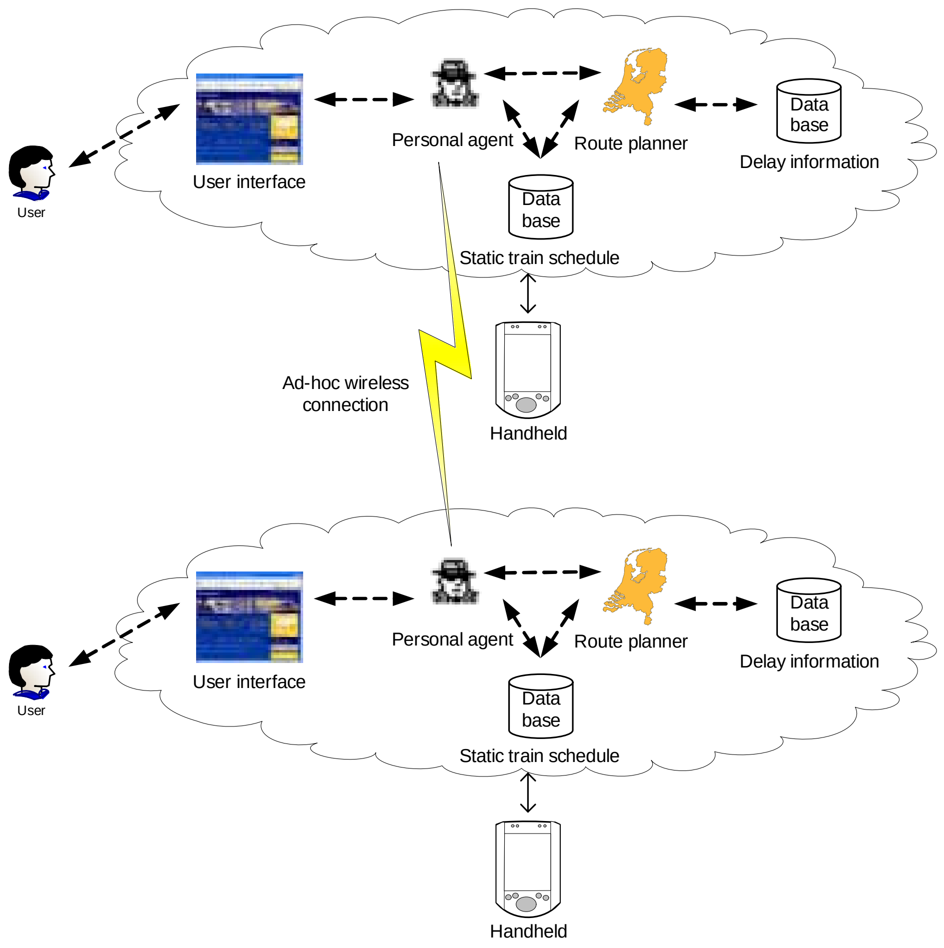


Figure 5.13 The distributed handheld application model

In this scenario the delay and route data is shared fully by personal agents in ad-hoc networks. Each personal agent is responsible for finding relevant data from nearby other agents. This leads to autonomous functioning personal agents, exchanging and storing information in a non-deterministic way, based on available information in the nearby ad-hoc networks. The delay data will be distributed in a way that suits the demand for information from the users of the system. Emerging from the autonomous functioning agents is a distributed application with a very interesting distribution model for the data.

The different handheld devices do not need to have the same capabilities in this scenario. One handheld can have positioning funtionality built-in while another has a mobile Internet connection. In this heterogeneous network, each of the users can get an optimal advice by sharing and exchanging information between the different handhelds.

5.3 The data model

This section describes the data model for the PITA system. First the system object model is described. Then some sequence diagrams present the communication between the objects. Finally the model of the ontology which is used for communication between the agents is described.

5.3.1 System object model

Figure 5.14 shows the object model of the system. It consists of the layered server model as shown in figure 5.8 and a handheld layer containing the personal agent. However it is also possible to run the handheld layer on top of the distributed servers. The most important object in the Server Management Layer is the System manager, which is the central component connecting the different layers. Each distributed server has a single system manager and the different system managers together synchronise the system time, a list of which servers are currently active and how the different regions are divided. Each server therefore has a unique number and a location, which is used to divide the regions. Through the System management interface, the Operator or Simulator can contact the System manager and change several settings and get an overview over the system.

The Data layer contains the Train schedule and a Delay manager, which is responsible for distributing the delays throughout the system. Using the NS website parser, actual data on train delays can be acquired and stored in the system. The Route planner is able to find the best route between two stations. It uses the train schedule in order to do so. Per Personal agent, the Route planner maintains what was the latest route given to that agent. This list is used to check which Personal agents have to be notified of a specific delay, once a new delay is added to the system.

The central object on the Handheld layer is the Personal agent. The Personal agent has its own System time object, which is synchronised to the System time of the rest of the system. Optionally also an instance of the Train schedule is stored on the handheld. The Personal agent furthermore keeps track of its current location through the Location object and the current advise through the Route object. It stores the user preferences and the desired destination. Lastly it keeps track of a list of other Personal agents that are nearby (more specific: in the same ad-hoc network). The Traveller can change his destination and preferences through the Personal agent interface and is also being updated on the current advice and progress through this interface.

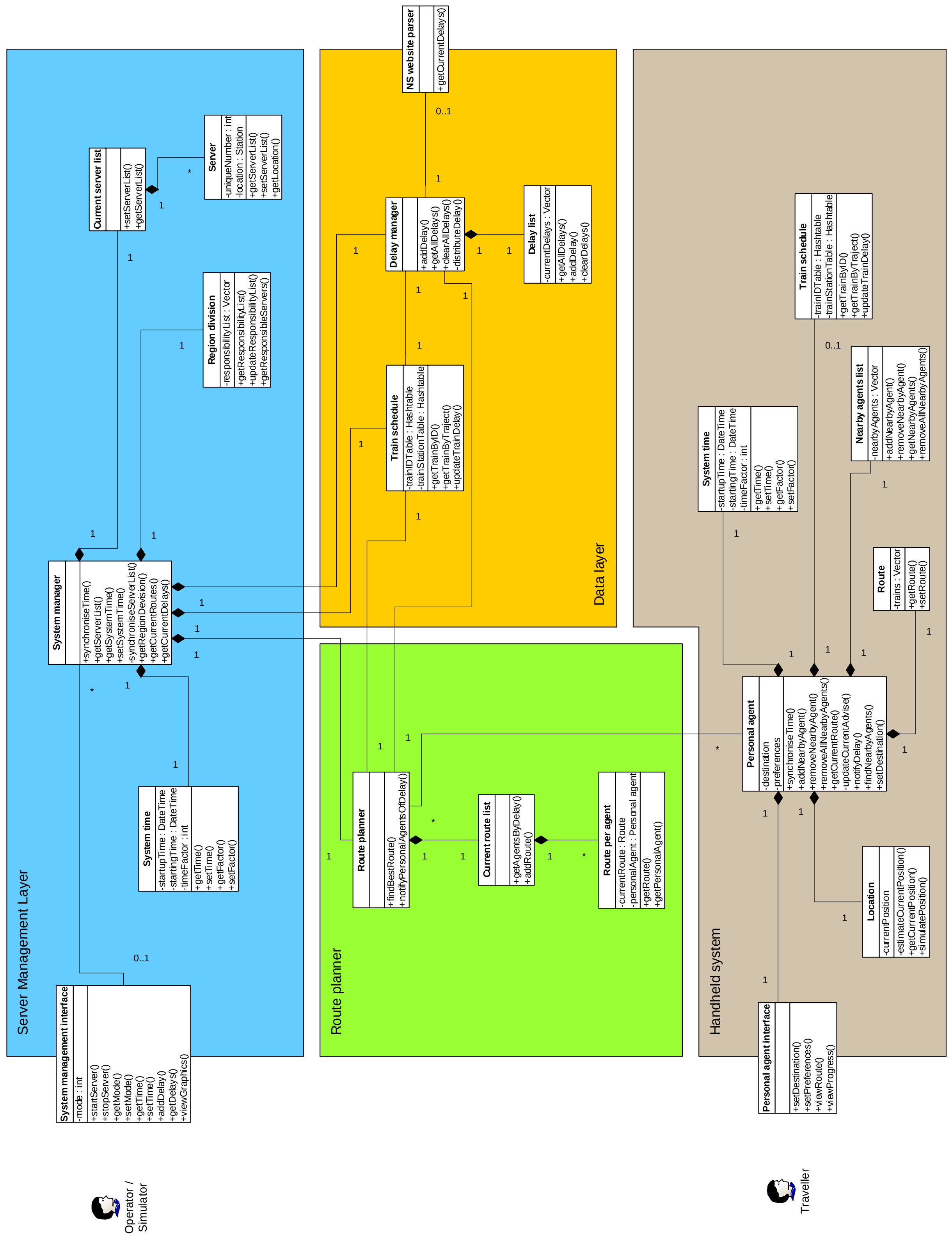


Figure 5.14 The system object model

]

5.3.2 Sequence diagrams

In order to clarify the functioning of the different objects together, the next sections describe three different functions of the system, using sequence diagrams to visualise the functions and messages which are exchanged between objects.

5.3.2.1 A new delay is added

FIgure 5.15 shows the sequence diagram for the addition of a new delay to the system.

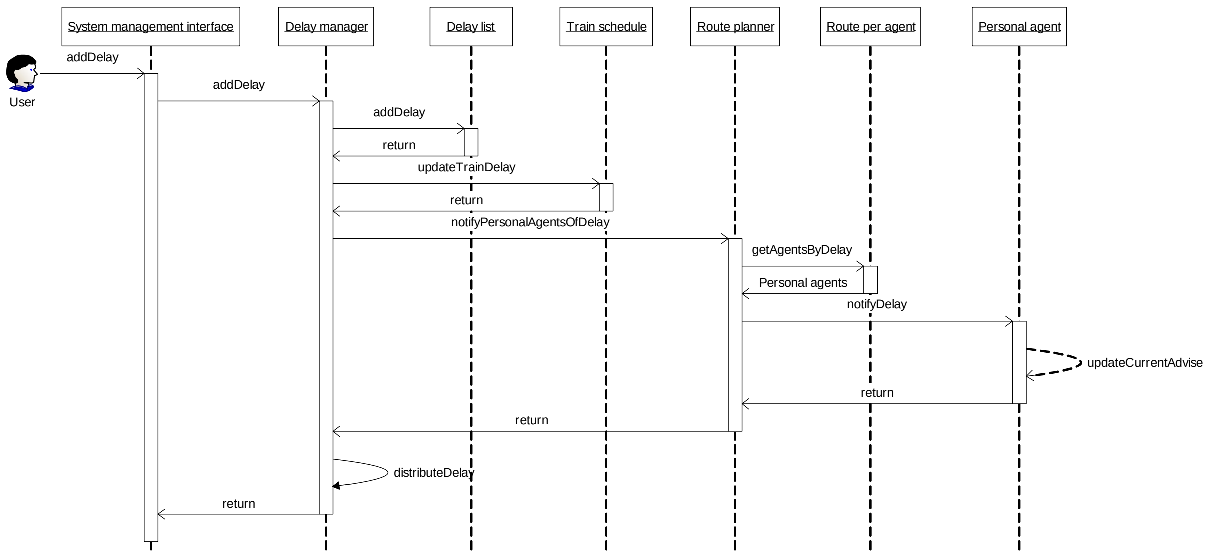


Figure 5.15 Sequence diagram of the addition of a new delay

As can be seen, the delay is stored in a list maintaining a copy of each new delay and then used to update the train schedule. After that, the routeplanner is contacted, which maintains per personal agent, a list of the last route which was advised. Using this list, the personal agents that might be interested in the new delay information are notified of the delay and they will update their advice. Finally the delay is distributed over the different servers, which is described in the next section.

5.3.2.2 The distribution of a delay

Figure 5.16 shows the sequence diagram of the distribution of delay information. Each server runs its own delay manager, which is why two different delay manager objects are used in the figure.

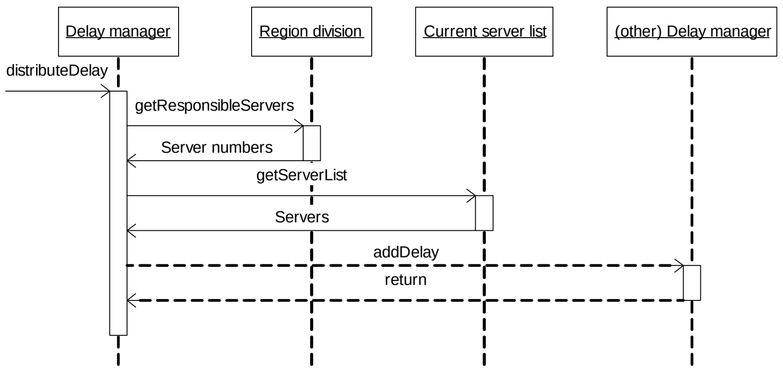


Figure 5.16 Sequence diagram of the distribution of a delay

First it is necessary to find out which servers have to be notified of the delay, which depends on the region division between the servers and the delay type (regional or inter-regional). After that, the delay managers on the different servers are notified of the delay and they will add this delay to the system as described in section 5.3.2.1.

5.3.2.3 Updating the traveller advise

Once a personal agent is notified of a delay, it will estimate whether the current route is still realizable with the new delay or if it is necessary to find a new route. If a new route is necessary, the personal agent will try to find such a route, as is shown in figure 5.17.

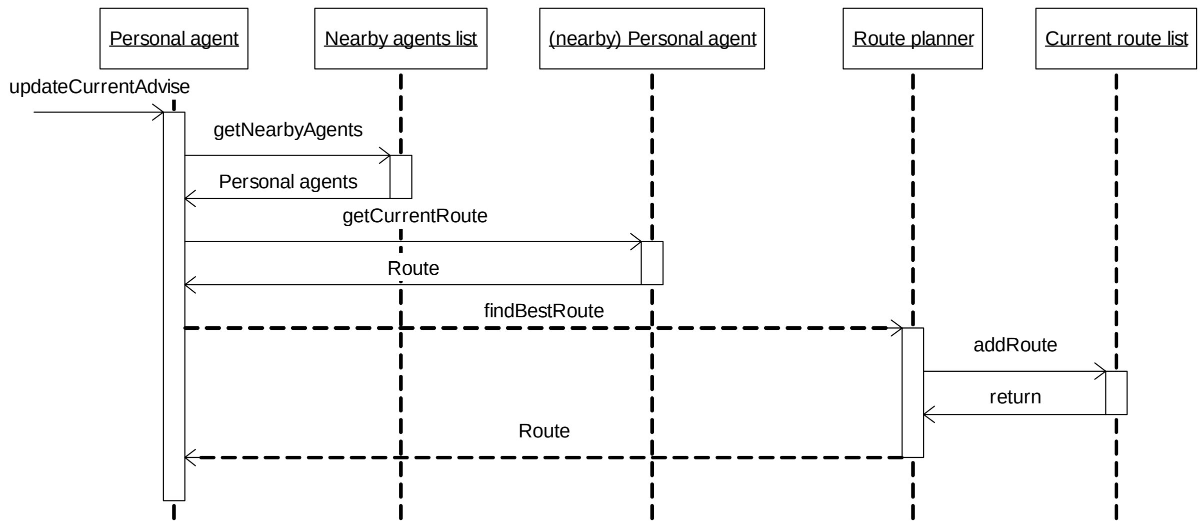


Figure 5.17 Sequence diagram showing how the advice is updated

The nearby agents list contains a list of other agents which can be contacted over the ad-hoc wireless network. These agents are first contacted to find out which route they are currently using. If a route is found, which is also usable for this traveller, this route is used by the personal agent. Otherwise, the personal agent contacts the route planner for a new route advise. The route planner updates its list with the routes which were advised to personal agents.

5.3.3 Ontology diagram

An ontology is an object description of the language which is used by the individual agents to communicate with each other. Figure 5.18 shows the ontology design for the PITA application.

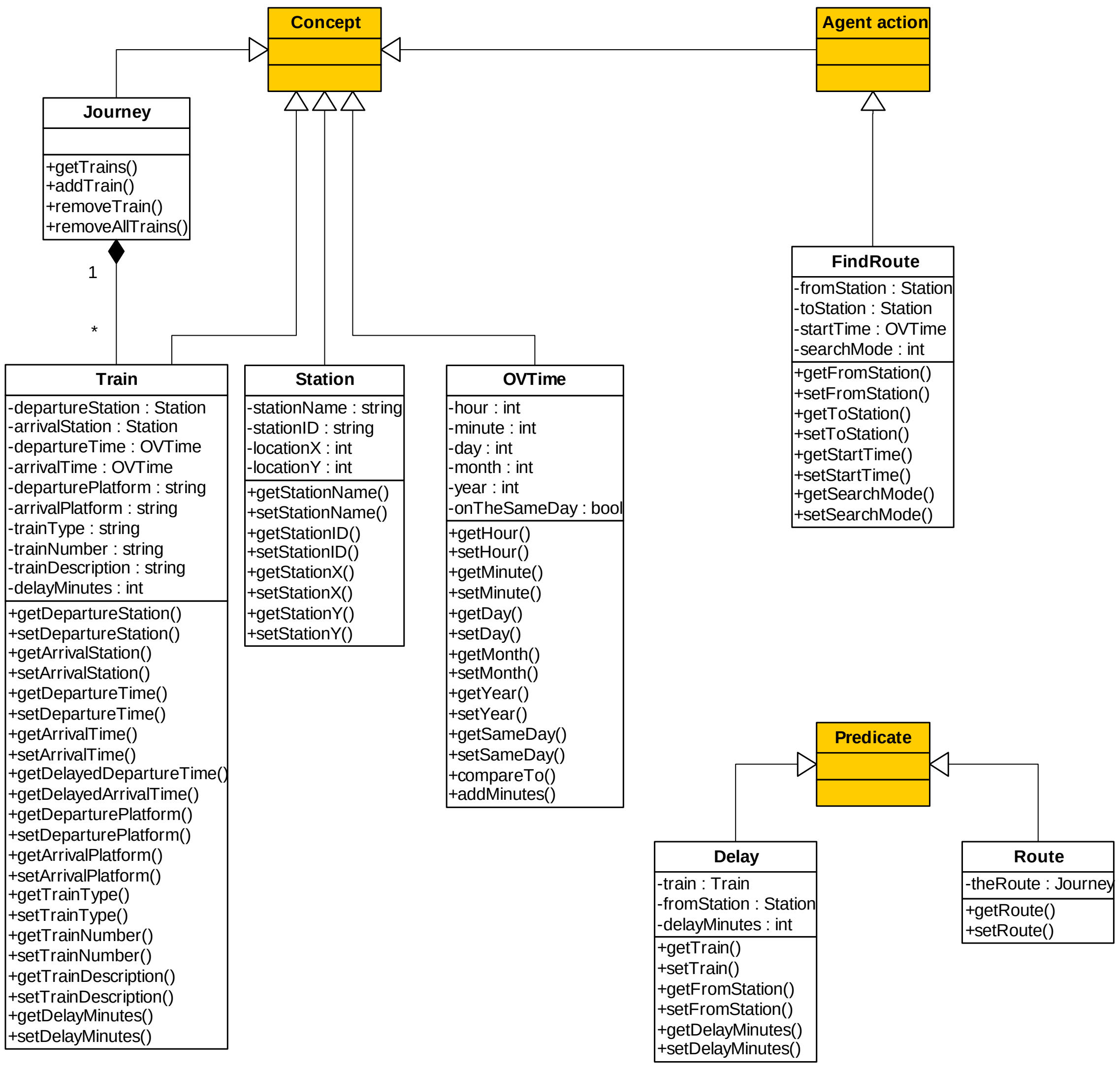


Figure 5.18 The ontology object model

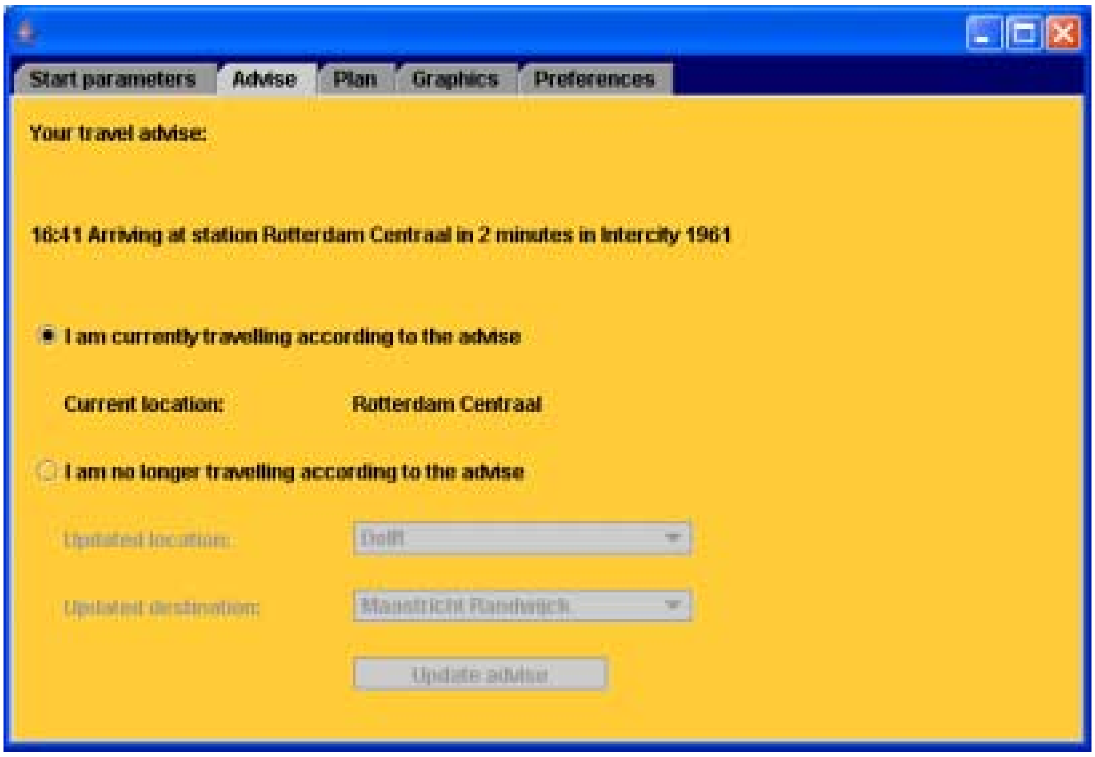
The ontology consists of the concepts Journey, Train, Station and OVTime, which form the basis to store data. The agent action FindRoute describes how route searches can be initiated. The result of a search is the predicate Route. Furthermore Delay predicates can be exchanged between agents.

5.4 The user interfaces

The two most important interfaces of the PITA system are the Traveller interface and the System management interface which are described in the following sections.

5.4.1 The Traveller interface

The Traveller interface forms the interface between the Traveller and the Personal agent. It is able to run on a handheld device. Figure 5.19 shows a screenshot of the Traveller interface. The Traveller interface is used by the user to specify his travel parameters (location and destination), his preferences and to view the advice from the PITA system.



Your travel advise:

16:41 Arriving at station Rotterdam Centraal in 2 minutes in Intercity 1961

[✓ I am currently travelling according to the advise]

Current location: Rotterdam Centraal

I am no longer travelling according to the advise

Updated location: Delft

Updated destination: Maastricht Randwyck

Update advise

Figure 5.19 The traveller interface

5.4.2 The System management interface

The System management interface is used by the Simulator and Operator users. It is used to manage and inspect the System manager objects of the distributed servers. Figure 5.19 shows a screenshot of how the System management interface can be used to inspect the current division of the regions between the different servers. In this case 4 servers are working from stations Amsterdam Centraal (pink), Utrecht Centraal (yellow), Maastricht (red) and Groningen (blue).

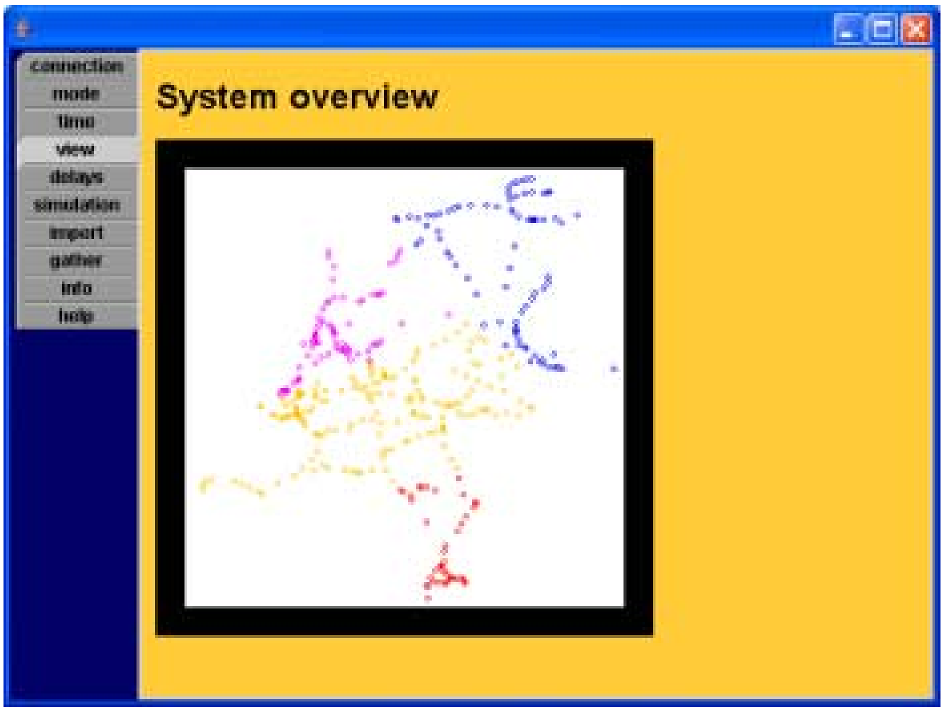
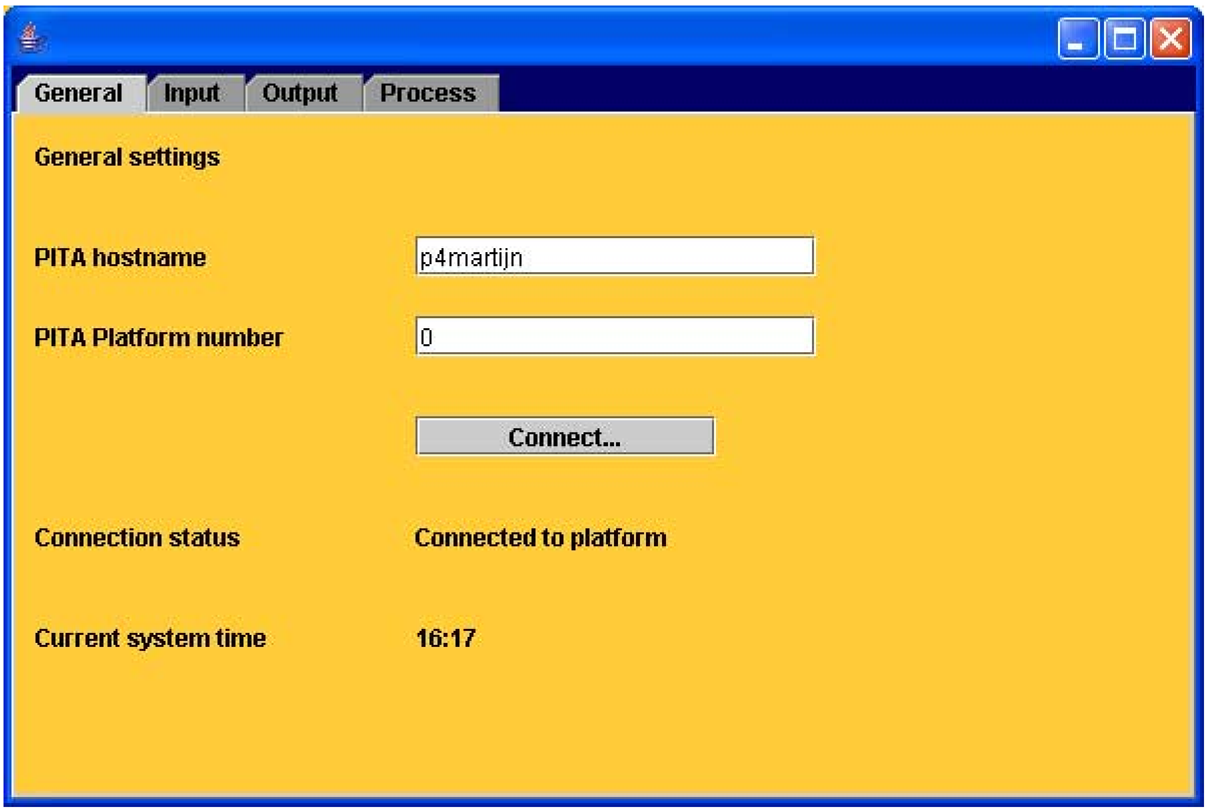


Figure 5.20 The System management interface

5.4.3 The NS delay interface

In order to simulate delays, an extra interface was designed for the delay generator. Using the delay generator, delays can be aqcuired from the NS website and send to the running PITA servers for a real time system or delays can be read from a file or outputted to a file for simulations. Figure 5.21 shows how this interface lets the user specify to which server backend it has to connect.



General settings

PITA hostname	p4martijn
PITA Platform number	0
Connection status	Connected to platform
Current system time	16:17

Figure 5.21 The NS delay interface

5.4.4 User test

To optimise the usability of the user interfaces, it is necessary to do tests with real users of the system. This is necessary to make sure the users interpret the offered functionality as it was designed by the developer: the mental model of the user should match the model of the designer. Metaphors can also be used to improve the usability. Such a user test has not been carried out as part of this project, because there was no time left to organise such a test. The provided interfaces are part of the prototype which was developed as part of this project. Other functions, such as a speech recognition interface and iconic interface are not developed. Before user tests are done, these other interfaces first have to be integrated into the current system to have a complete dialogue system for the user, in order to optimally test the system.

Chapter 6: Implementation

This chapter describes the implementation of the prototype, based on the design of chapter 5. First is described how the relevant information was acquired, such as the static train schedule and delay information. Then the configuration of the distributed agent platform is described, followed by the implementation of the route algorithm. Finally ad-hoc networking is discussed and the interface implementation is described.

6.1 Acquiring the information

Before being able to build a PITA system, it is necessary to have information on the static train schedule and delays at your disposal. Because NS does not distribute this information freely, it was necessary to parse this information from the website for the prototype. First we describe how the train schedule for a single working day was copied, followed by how GPS and GSM data can be used for determining the position of the traveller. Then we describe how the delay data can be parsed from the NS website.

6.1.1 The static train schedule

The information on departure and arrival times of trains is property of NS. NS offers several services and products which contain the information on the train schedule. In the first place, NS sells a paper guide containing a printed version of the train tables. Next to this, NS sells a program on CD called “Planner Plus”, which can be used to plan trips on a personal computer. This software does however not offer an open interface for accessing the information on the train schedule from another program. Finally there are two websites which contain planners based on the train schedule. The first website is www.9292ov.nl which is a door-to-door planner for public transportation and which also includes regional busses in the generated advise. The second website is of NS itself, www.ns.nl, it contains a planner with only information on trains.

In order to have representative data in the prototype of the PITA system, the train schedule of a normal weekday was copied from the NS site. To built a real PITA system, it is necessary to make arrangements with NS to get access to the train schedule, because there are several exceptional rules (for instance on holidays and weekends). In order to automate the process of copying the train schedule, it was necessary to make a model containing all connections between stations which are made by trains. This means all direct connections occuring in the train schedule were added to a XML file. After this, a program was developed which used this data, to query the NS website on all trains making this direct connection on a single day. This can be done by rebuilding the “search earlier and later connections” from the website in the program. Table 6.1 shows the fields which have to be supplied to query the NS webserver.

Table 6.1 The form data used to query the train schedule

Field:	Description:
from	Contains the ID of the departure station
to	Contains the ID of the arrival station
yeartraveldate	Contains the year of the date on which to search
monthtraveldate	Contains the month of the date on which to search
daytraveldate	Contains the day of the date on which to search
hourtraveltime	Contains the hour of the time to search
minutetraveltime	Contains the minutes of the time to search
state	Contains an internal state used by the webserver, which should be updated after each search
actie	Contains ‘ok’ on the first search and ‘advicequery’ on following searches
datumbutton	Contains ‘vandaag’, ‘morgen’ or ‘anders’
new_select	Contains an integer value which is the selected index

Submitting this data by a HTTP POST action to http://www.ns.nl/cgi-ns/nsbaliecgi, a search in the train schedule can be done. Consecutive searches have to increment the value in the new_select field, in order to find later connections. Also the contents of the state field have to be parsed from the returned page and then used in the new search. This way a copy of the train schedule of a single normal working day was made for the prototype and stored in a database. Later an ordered query on the database was saved to a XML file which is used as input by the prototype.

6.1.2 The user position

There are several possibilities to automatically determine the position of the user. Using a GPS receiver or a mobile phone are the two best possibilities. As mobile devices converge into a single device having all different functions built-in, it is very likely handhelds will have mobile phone and/or GPS functionality built-in in the near future. However at this moment, there are smart phones having a GSM built-in, or external GPS receivers available. To communicate with these devices, serial communication is used.

6.1.2.1 Serial communication

Serial communication is used in computers to connect external devices. Starting with mouses using the physical COM-port connector, to the modern Universal Serial Bus (USB), serial communications dominate the connection with external devices. The introduction of Bluetooth, also made it possible to use wireless serial communication to external devices, such as mobile phones and GPS receivers. On top of a serial communication link, different protocols can be used in order to exchange information with the external device. In the next two sections we will describe these protocols for communication with a (Bluetooth) GPS receiver or a (Bluetooth) mobile phone.

6.1.2.2 GPS receiver

In order to communicate with a GPS receiver, a Bluetooth GPS receiver (Fortuna GPSmart BT) and a laptop with Bluetooth adapter were used. After configuring the Bluetooth connection, this creates a virtual COM-port in Windows, which can be used to communicate with the device. The configuration is shown in figure 6.1.

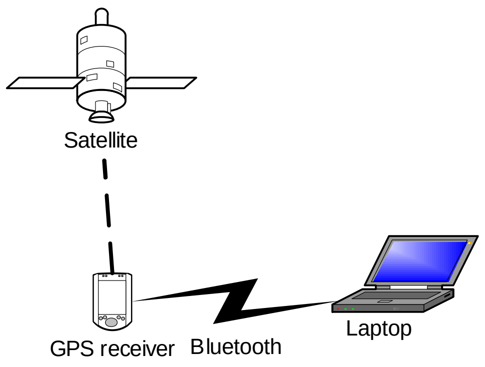
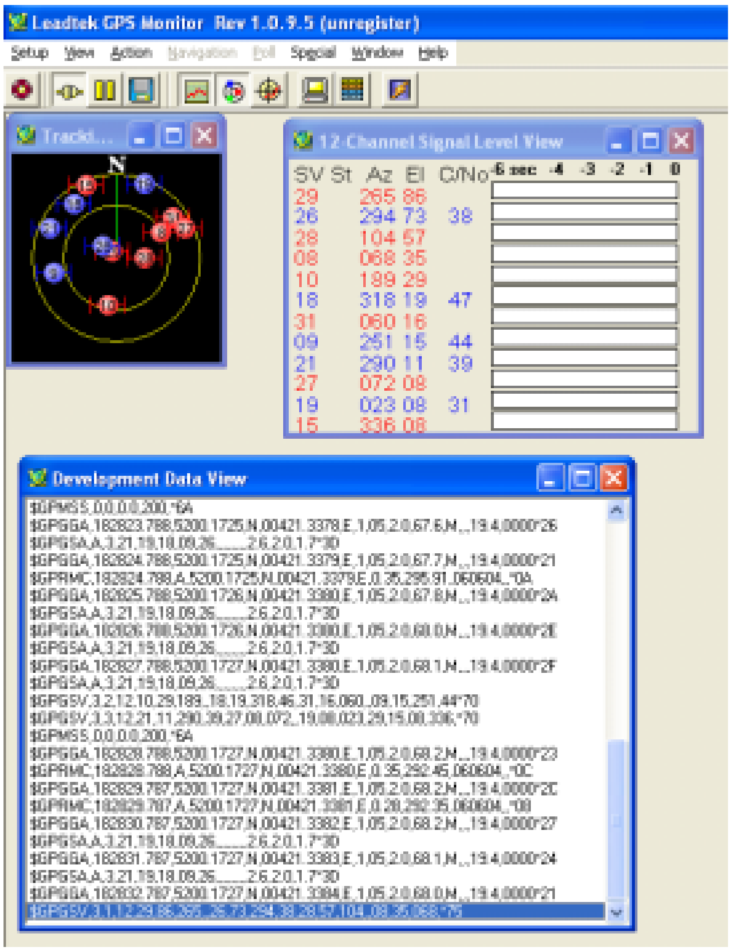


Figure 6.1 The configuration used to communicate with a GPS receiver

Currently there are two communications standards for communicating with GPS receivers: The National Marine Electronics Association standard NMEA 0183 and the SiRF standard. The used GPS receiver used the NMEA protocol, which transmits several different message types of positional data. Messages containing the current position are the $GPGGA and $GPRMC message. Figure 6.2 shows a screenshot of a GPS monitor program, which also includes a dump of the data transmitted by the GPS receiver.


Leadtek GPS Monitor Rev 1.0.9.5 (unregister)
Setup View Action Navigation Poll Special Window Help
Tracki...
N
12-Channel Signal Level View
-6 sec -4 -3 -2 -1 0
Development Data View

SV	St	Az	El	C/No
29		265	86	
26		294	73	38
28		104	57	
08		068	35	
10		199	29	
18		318	19	47
31		060	16	
09		251	15	44
21		290	11	39
27		072	08	
19		023	08	31
15		336	08	

```
$GPMSS,0,0,0.0,200,*6A
$GPGGA,182823.788,5200.1725,N,00421.3378,E,1,05,2.0,67.6,M,,,19.4,0000*26
$GPGSA,A,3,21,19,18,09,26,,,,,,,,2.6,2.0,1.7*30
$GPGGA,182824.788,5200.1725,N,00421.3379,E,1,05,2.0,67.7,M,,,19.4,0000*21
$GPRMC,182824.788,A,5200.1725,N,00421.3379,E,0.35,295.91,060604,,*0A
$GPGGA,182825.788,5200.1726,N,00421.3380,E,1,05,2.0,67.8,M,,,19.4,0000*2A
$GPGSA,A,3,21,19,18,09,26,,,,,,,,2.6,2.0,1.7*30
$GPGGA,182826.788,5200.1726,N,00421.3380,E,1,05,2.0,68.0,M,,,19.4,0000*2E
$GPGSA,A,3,21,19,18,09,26,,,,,,,,2.6,2.0,1.7*30
$GPGGA,182827.788,5200.1727,N,00421.3380,E,1,05,2.0,68.1,M,,,19.4,0000*2F
$GPGSA,A,3,21,19,18,09,26,,,,,,,,2.6,2.0,1.7*30
$GPGSV,3,2,12,10,29,199,,18,19,318,46,31,16,060,,09,15,251,44*70
$GPGSV,3,3,12,21,11,290,39,27,08,072,,19,08,023,29,15,08,336,*70
$GPMSS,0,0,0.0,200,*6A
$GPGGA,182828.788,5200.1727,N,00421.3380,E,1,05,2.0,68.2,M,,,19.4,0000*23
$GPRMC,182828.788,A,5200.1727,N,00421.3380,E,0.35,292.45,060604,,*0C
$GPGGA,182829.787,5200.1727,N,00421.3381,E,1,05,2.0,68.2,M,,,19.4,0000*2C
$GPRMC,182829.787,A,5200.1727,N,00421.3381,E,0.28,292.35,060604,,*08
$GPGGA,182830.787,5200.1727,N,00421.3382,E,1,05,2.0,68.2,M,,,19.4,0000*27
$GPGSA,A,3,21,19,18,09,26,,,,,,,,2.6,2.0,1.7*30
$GPGGA,182831.787,5200.1727,N,00421.3383,E,1,05,2.0,68.1,M,,,19.4,0000*24
$GPGSA,A,3,21,19,18,09,26,,,,,,,,2.6,2.0,1.7*30
$GPGGA,182832.787,5200.1727,N,00421.3384,E,1,05,2.0,68.0,M,,,19.4,0000*21
$GPGSV,3,1,12,29,86,265,,26,73,294,38,28,57,104,,08,35,068,*7[illegible]
```

Figure 6.2 A screenshot of a GPS monitor containing a dump of the communicated data

The content of the messages is divided in fields, separated by a comma. Table 6.2 describes the contents of the $GPRMC and $GPGGA message fields:

Table 6.2 The contents of the NMEA messages from the GPS receiver

$GPGGA message (Global positioning system fixed data)	
Field:	**Content:**
1	UTC of Position
2	**Latitude**
3	**N or S**
4	**Longitude**
5	**E or W**
6	GPS quality indicator (0=no fix)
7	Number of satellites in use
8	HDOP Horizontal dilution of position
9	Antenna altitude (w. resp. to sea level)
10	M [eters] (Antenna height unit)
11	Geoidal separation
12	M [eters] (Units of geoidal separation)
13	Age in seconds since last update from diff. reference station
14	Diff. reference station ID#, 0000-1023

$GPRMC message (Recommended minimum specific GNSS data)	
Field:	**Content:**
1	UTC of position fix
2	Data status (V= warning, A=valid)
3	**Latitude of fix**
4	**N or S**
5	**Longitude of fix**
6	**E or W**
7	Speed over ground in knots
8	Course over ground in degrees true north
9	UT date (DDMMYY)
10	Magnetic variation degrees
11	Magnetic variation, sense E or W
12	Checksum (Mandatory)

6.1.2.3 GSM mobile phone

Using the same configuration, with a laptop with Bluetooth adapter, a virtual serial port was connected to a Siemens S55 mobile phone. Current mobile phones implement extended versions of the HAYES AT-command set, once developed for modems. The AT command AT+CREG gives information on the current cell of the GSM network which is used. The device can be set to enable location information messages, by sending the following command: AT+CREG=2. After this, the mobile phone will give information on the current GSM cell and update this information as the user moves across the network, by sending response messages. These messages consist of comma seperated fields, which are described in table 6.3.

Table 6.3 The contents of the network registration message

AT+CREG response messages (Network registration)	
Field:	**Content:**
Starts with	+CREG:
1	Current mode (needs to be set to 2, to receive location information, is optional)
2	Current status: 0 = Not checked in, not seeking 1 = Checked in 2 = Not checked in, seeking a network 3 = Check-in denied by network 5 = Registered, roaming
3	Hexadecimal 2-byte string type of location area code
4	Hexadecimal 2-byte string type of cell ID

By sending the command AT+CREG? it is possible to query the current status of network registration and GSM cell. Some example messages received in Delft look like this:

```
+CREG: 2,1,"0030","2DA2"
+CREG: 1,"FFFE","006C"
+CREG: 1,"0002","0175"
+CREG: 1,"000C","0CD1"
+CREG: 1,"000C","0201"
```

As can be seen, there is no direct mathematic relation between the position and the received location information.

6.1.2.4 Determining the position in the prototype

The implementation of the first prototype of the PITA handheld application does not include automatic determination of the position of the user. In stead the user has to specify his position manually. At the moment a lot of effort would be necessary to implement automatic determination of the user position, as platform dependent libraries would have to be implemented. Since simultaneously development is being done on the Java Bluetooth API and the Java Location API and the limited amount of time available for this project, we have not started the implementation of such libraries. Figure 6.3 gives an overview of the different possibilities to automatically determine the user position.

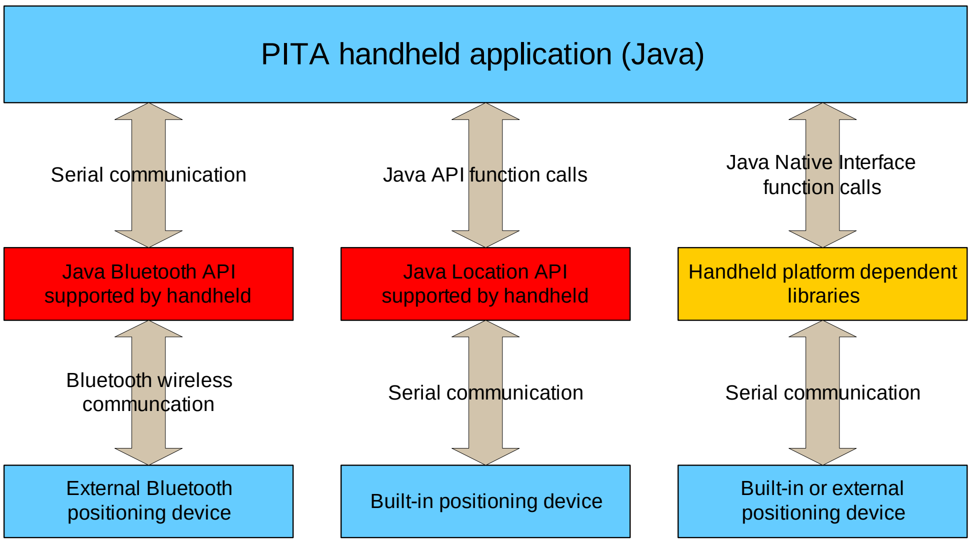

PITA handheld application (Java)
Serial communication
Java API function calls
Java Native Interface function calls
Java Bluetooth API supported by handheld
Java Location API supported by handheld
Handheld platform dependent libraries
Bluetooth wireless communcation
Serial communication
Serial communication
External Bluetooth positioning device
Built-in positioning device
Built-in or external positioning device

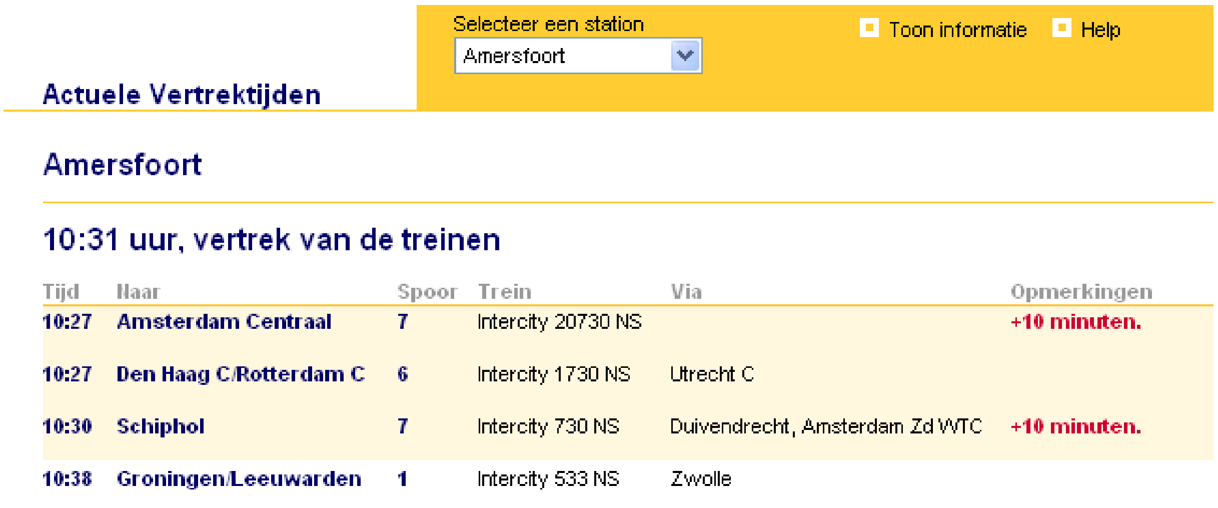
Figure 6.3 **The possibilities for automatically determining the position of the user from the PITA application**

Because currently it is not yet possible to use the Java Bluetooth API to communicate with external Bluetooth devices from Java and the Java Location API is not supported directly by any handheld device having built-in GPS, it would be necessary to implement platform-dependent libraries (for instance in C for PalmOS) which connect to the device. As it can be expected that the Bluetooth and Location API's will be supported in the near future, no effort was made to develop such hardware dependent libraries.

Another problem with GSM position determination is the lack of a direct link between the position and the received data. As each mobile operator has its own network, with different cell ID's, it is necessary to collect a lot of information on the position of the antenna's of each operator and after that it is necessary to implement a non-lineair function to determine the position. This would also require a lot of time to implement an acceptable solution.

6.1.3 Delays from the NS website

As of March 25th, 2004 it is possible to get information about the actual delays of individual trains from the NS website. The site contains up-to-date delay information for 83 stations, giving an accurate view of the departing trains at these stations, including delays. By submitting a station identifier, either the first 20 departing trains are displayed or all departing trains leaving the station in the next hour are displayed.



Amersfoort

10:31 uur, vertrek van de treinen

Tijd	Naar	Spoor	Trein	Via	Opmerkingen
10:27	Amsterdam Centraal	7	Intercity 20730 NS		+10 minuten.
10:27	Den Haag C/Rotterdam C	6	Intercity 1730 NS	Utrecht C	
10:30	Schiphol	7	Intercity 730 NS	Duivendrecht, Amsterdam Zd WTC	+10 minuten.
10:38	Groningen/Leeuwarden	1	Intercity 533 NS	Zwolle	

Figure 6.4 **A screenshot of the NS website containing delay information**

The site which is shown in figure 6.4 has been used to gather delay information. By submitting the stop field containing the station id of the departure station, as described in table 6.4, using a HTTP POST action to http://212.108.13.35/dtw/dtwClient.jsp, the list of departing trains can be requested. It is not always clear which connection a train is going to make, since the stops are only globally described in the "Via" column. That is why we used the train identifier combined with the station ID of the departure station in order to combine the delay information with the static train schedule.

Table 6.4 The form data used to query the delay website

Field:	Description:
stop	Contains the ID of the departure station

As part of this project a parser was developped which can read delay information either live from the NS website or from an input file and output the data either to the PITA system or to an output file. This parser can be used to generate delay data for simulations and to run simulations with previously saved delay data.

6.2 The distributed agent platform

The distributed agent platform forms the central backend of the PITA system. This section describes the structure of the platform, how the distribution and synchronisation on the backend takes place and which agents were implemented.

6.2.1 The agent platform architecture

The agent platform is divided over a backend consisting of different servers connected by Internet and several individual handhelds as is shown in figure 6.5.

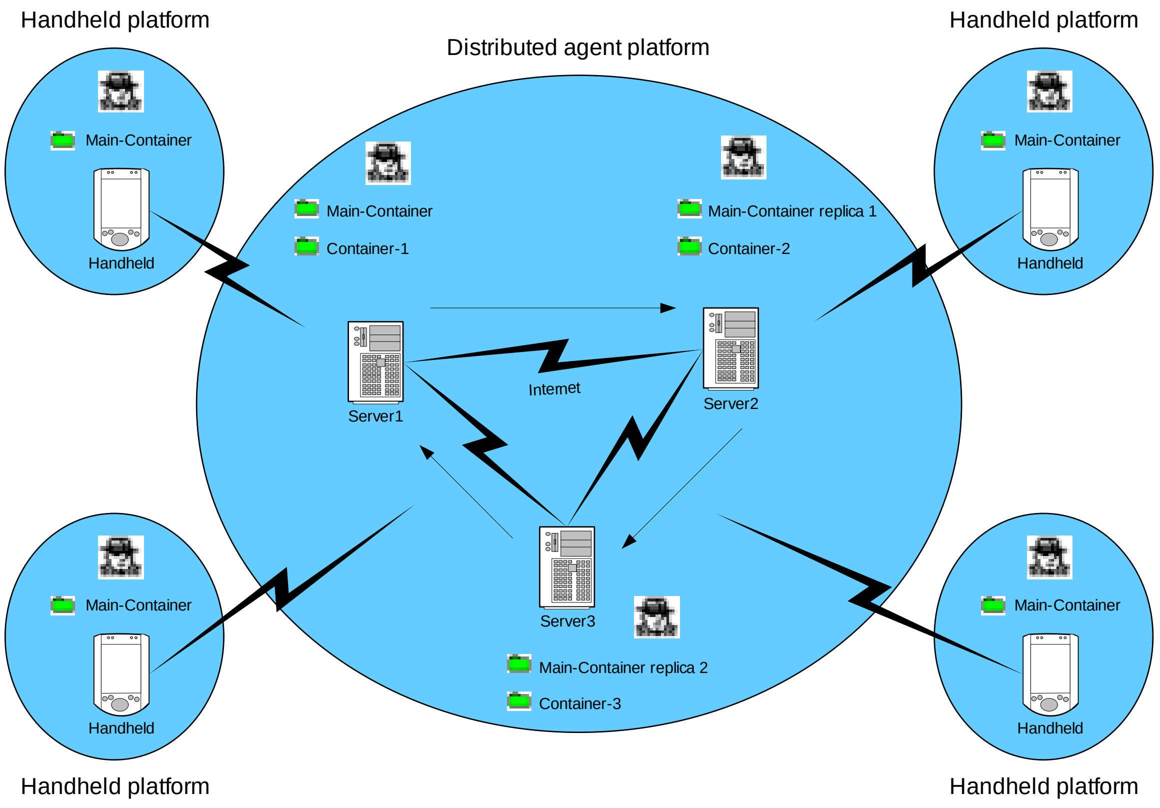


Figure 6.5 The distributed agent platform architecture

The distributed backend consists of several servers, forming a single fault-tolerant platform as described in section 3.2.5. Each of the handhelds runs a separate platform without replication or distribution, which connects to the distributed backend by a mobile Internet connection.

6.2.2 Distribution and synchronisation on the backend

The backend consists of several servers, which have to distribute the data and workload of the application across each of the servers. In order to do so, each server has to be started with two parameters: a unique (increasing) number and a train station location. The unique number is used for identification of each of the platforms and the train station location is used for the division of different regions.

As a new backend server is started, it will try to contact a server with a lower number in order to get information on the currently running configuration. Based on the train station location of each of the servers a division of the regions is made and the new server will inform all of the current running servers of this new region division. Each of the servers check whether its region changed and distribute their data according to the new division over the servers. This leads to a new configuration in which the newly added server has the responsibility for a region of which it gets the corresponding data from the other servers. Another scenario would be the failure of a server. Each of the backend servers has a random time-out, which is used in order to check the functioning of the system. As soon as the time-out of a server has expired, the server will start a check of the functioning of the current configuration by contacting each of the servers. If a server is detected which is not functioning, this server is removed from the configuration and the operation is repeated by sending each of the servers a new configuration without the failing server.

6.2.3 Autonomous agents

One of the important features of agent applications is that it is possible to model several autonomous agents, each having its own responsibility, separately in stead of having to develop the full application as a single program. Each agent in an agent application is responsible for reaching its own goals and can autonomously take decisions, execute plans and communicate with other agents in order to do so. Up front it is unclear how the application will execute, since this depends on the agent state of each of the different agents. The application will persue the given goals, but how they are reached depends fully on the way in which the agents are working and communicating together. The way the agents work together is usually referred to as the emergent behaviour of the agent application. Besides the interesting aspects of the emergent behaviour, the agent application also offers a way to divide a large problem in smaller parts which can be solved by different agents. By developing plans to solve certain problems, an ontology to communicate and appropriate goals it is possible to model the problem as an agent application. For the PITA system, five different agent types were developed, corresponding to the different systems in the system object model in figure 5.13: a server manager agent, a route planner agent, a delay agent, a NS delay agent and a personal agent. Table 6.5 gives an overview of the responsibilities of each of the developed agents.

Table 6.5 The responsibilities of the different agent types

Agent type:	Responsibilities:
Server manager	Synchronisation of the system time Maintain a working configuration of backend servers Maintain a division of different regions over the backend servers
Route planner	Plan the best route between two stations, based on preferences and delay info Maintain a list of the last route which was given to each personal agent Inform personal agents of relevant delay information
Delay agent	Maintain a list of actual delays Inform other agents of new delays
NS delay agent	Aqcuire information on delays from the NS website Import and export delays from and to files Inform other agents of delays
Personal agent	Manage personal preferences, current position and destination Maintain a list of other nearby personal agents Advise the traveller Communicate with the route planner and nearby agents

The ontology as described in section 5.3.3 was used for the communication between the agents. The goal of each of the agents is to make sure the responsibilities are fulfilled. The goal of a personal agent is to reach the destination. Each of the agents was implemented as a Jadex BDI agent, as described in section 2.3.5.

6.3 The route planner

For this project two different route planning algorithms were implemented: a Moore algorithm and the DYNET algorithm of Eduard Tulp. This section discusses the implementation of both algorithms.

6.3.1 Used algorithms

The difference between both algorithms is that the Moore algorithm is a label correcting algorithm while the DYNET algorithm is a label setting algorithm (based on Dijkstra's algorithm). This means the Moore algorithm will continually update the label of different nodes as shorter paths to the node are found, while the DYNET algorithm will build a shortest path tree containing the definitive shortest path to a node. Furthermore the implementation of the Moore algorithm in the system does not take into account time spent on train changes or alternative routes having less train changes. The DYNET algorithm can be configured with different parameters to find the best solution for a specific user. Because the DYNET algorithm can be adapted to the user preferences and takes into account time spent on train changes, this algorithm has to be preferred above the Moore implementation.

6.3.2 Search parameters

Besides the obvious parameters, such as the starting location, destination and start time, the DYNET algorithm implementation can take several parameters which are shown in table 6.6.

Table 6.6 The parameters for configuring the DYNET algorithm

Parameter:	**Description:**
change_value	The time per train change, a user is prepared to arrive later at his destination for a solution having less train changes
train_change_time	The time a user needs, to leave a train and board the following train at the same platform
extra_change_time_per_platform	The time which is added to the train_change_time per platform which a user has to cross in order to reach the next train
passes	Whether a single (first) pass algorithm is used finding the earliest leaving, earliest arriving solution or a two pass algorithm, finding the latest leaving, earliest arriving solution

Using these parameters, the best route can be found for each type of user. As we do not want to make things to complex to understand for a user, in stead of changing these values we have defined four user profiles, among which a user can select his preferred profile which includes these parameters. The next section describes the four user profiles.

6.3.3 User profiles

As described in section 5.2.3.4, four different profiles are distinguished. First we make a distinction between a user who is currently travelling and wants to get to his destination as soon as possible. This is used for the profiles with location "at the station, travelling". Users who are at home and want to take a train within an hour or so are modelled in the "not at the station, planning" profile. Next we distinguish the comfortable user, who does not want to run to make a train change and rather sits in the same train for 10 extra minutes as having to make a trainchange, and a performance user who really wants to get to his destination as soon as possible. Of course these parameters can be adjusted and should be adjusted to fit the user preferences optimally, however this is not done in the current implementation and the parameters are used statically, as is described in table 6.7.

Table 6.7 The mapping between the user profiles and the search parameters

Current location:	User type:	Parameters:

At the station, travelling	comfortable user	change_value = 10 minutes
		train_change_time = 2 minutes
		extra_change_time_per_platform = ½ minute
		passes = first pass only

At the station, travelling	performance user	change_value = 0 minutes
		train_change_time = 1 minutes
		extra_change_time_per_platform = ½ minute
		passes = first pass only

Not at the station, planning	comfortable user	change_value = 10 minutes
		train_change_time = 2 minutes
		extra_change_time_per_platform = ½ minute
		passes = two psses

Not at the station, planning	performance user	change_value = 0 minutes
		train_change_time = 1 minutes
		extra_change_time_per_platform = ½ minute
		passes = two passes

6.3.4 Performance examples

In this section we will discuss different examples of solutions which were found using the first pass DYNET algorithm and compare these with the result of the NS website. For this we selected a traveller wanting to go from Uitgeest to Haarlem at 7:40. Using the NS website planner the following solution was found:

Table 6.8 The route found by the NS website

Time:	Station:	Platform:	Direction:	Train ID:
07:42	**Uitgeest**	3	Amsterdam Centraal	Stoptrein 7327
07:47	Krommenie-Assendelft			
07:50	Wormerveer			
07:53	Koog-Zaandijk			
07:55	Koog Bloemwijk			
07:58	Zaandam			
08:04	Amsterdam Sloterdijk	6		
08:16	Amsterdam Sloterdijk	7	Haarlem	Sneltrein 2233
08:26	**Haarlem**	3		

Using the comfortable, first pass algorithm, the DYNET algorithm finds the following route in 60 milliseconds searchtime:

Table 6.9 The route found by the comfortable, first pass algorithm in 60 milliseconds

Time:	Station:	Platform:	Direction:	Train ID:
08:04	**Uitgeest**	4a	Haarlem	Sprinter 4816
08:08	Heemskerk			
08:11	Beverwijk			
08:15	Driehuis			
08:17	Santpoort Noord			
08:20	Santpoort Zuid			
08:23	Bloemendaal			
08:27	**Haarlem**	6a		

This can be explained by the change_value of 10 minutes, which gives a 10 minute penalty to the routes having a train change. Next we will use the performance, first pass profile to see which route is found in this case:

Table 6.10 The route found by the performace, first pass algorithm in 50 milliseconds

Time:	Station:	Platform:	Direction:	Train ID:
07:51	**Uitgeest**	1b	Alkmaar	Stoptrein 7314
07:55	Castricum	1		
07:57	Castricum	2	Haarlem	Sneltrein 3431
08:06	Beverwijk			
08:17	**Haarlem**	5		

The solution was found in 50 milliseconds. In the performance profile, the connection at Castricum can just be made, which gives absolutely the fastest connection but a high risk. If we use the comfortable profile in which we change the change_value parameter to 0, the same route is found as the NS website gives standard in 60 milliseconds.

The implementation of the DYNET algorithm does however not use a linear amount of time for a more complex problem. For a route from Maastricht to Groningen starting at 10:00 the following route is found using the performance profile in 591 milliseconds:

Table 6.11 The route from Maastricht to Groningen found by the performance, first pass algorithm in 591 milliseconds

Time:	Station:	Platform:	Direction:	Train ID:
10:17	**Maastricht**	4a	Roermond	Stoptrein 6838
10:23	Bunde			
10:29	Beek-Elsloo			
10:33	Geleen-Lutterade			
10:39	Sittard			
10:44	Susteren			
10:48	Echt			
10:57	Roermond	3a		
11:02	Roermond	2	Eindhoven	Intercity 838
11:16	Weert			
11:38	Eindhoven			
11:58	's-Hertogenbosch			
12:29	Utrecht Centraal	7a/b		
12:32	Utrecht Centraal	4a	Amersfoort	Stoptrein 5641
12:47	Amersfoort			
12:51	Amersfoort Schothorst			
12:57	Nijkerk			
13:03	Putten			
13:07	Ermelo			
13:11	Harderwijk			
13:19	Nunspeet			
13:25	't Harde			
13:31	Wezep			
13:40	Zwolle	5a		
13:49	Zwolle	3b	Groningen	Intercity 743
14:05	Meppel			
14:17	Hoogeveen			
14:26	Beilen			
14:36	Assen			
14:48	Haren			
14:55	**Groningen**	2b		

After this we also performed a search with the comfortable, first pass profile. In 1432 milliseconds the following route shown in table 6.12 was found, which matches the advice given by the NS website.

Table 6.12 The route from Maastricht to Groningen found by the comfortable, first pass algorithm in 1432 milliseconds

Time:	Station:	Platform:	Direction:	Train ID:
10:17	**Maastricht**	4a	Roermond	Intercity 838
10:46	Sittard			
11:02	Roermond			
11:16	Weert			
11:38	Eindhoven			
11:58	's-Hertogenbosch			
12:29	Utrecht Centraal	7a/b		
12:53	Utrecht Centraal	11a/b	Amersfoort	Intercity 1743
13:07	Amersfoort	1		
13:10	Amersfoort	2	Zwolle	Intercity 743
13:49	Zwolle			
14:05	Meppel			
14:17	Hoogeveen			
14:26	Beilen			
14:36	Assen			
14:48	Haren			
14:55	**Groningen**	2b		

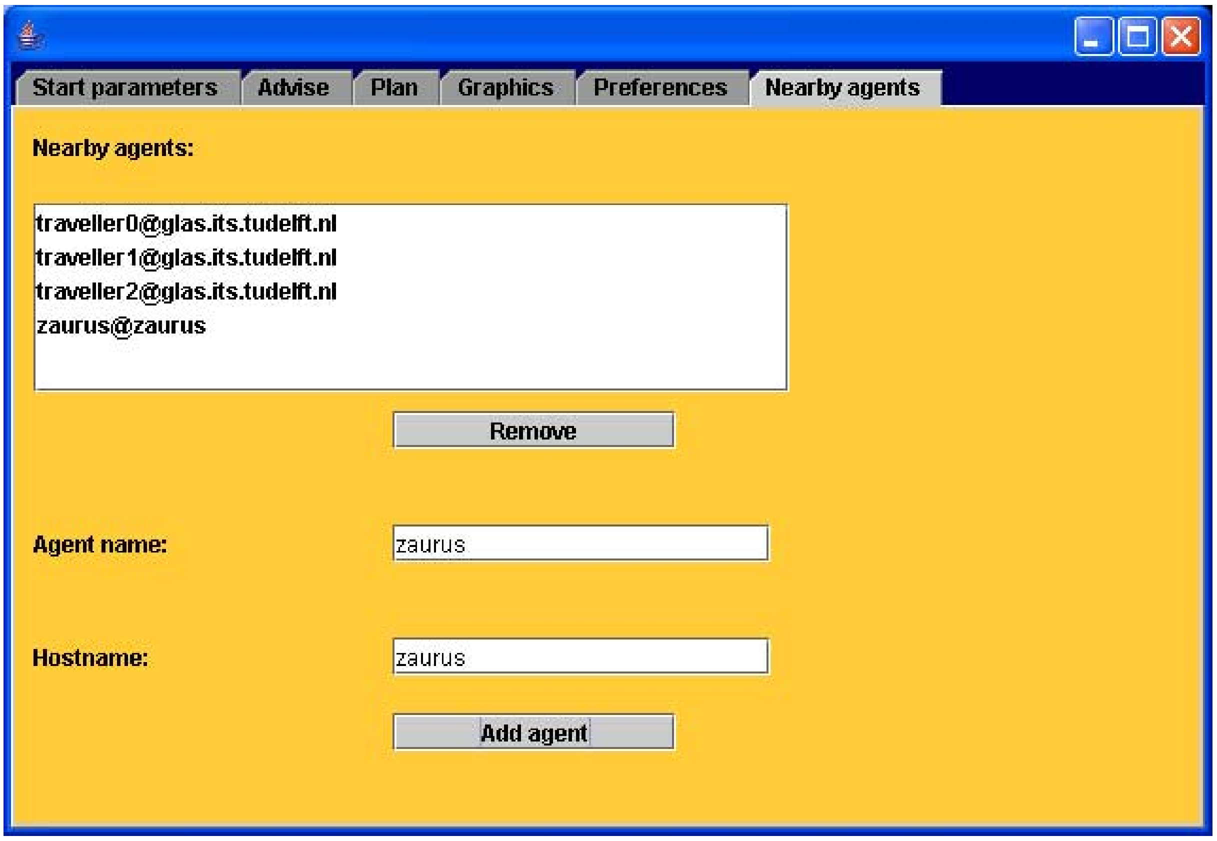
The search times which were mentioned in this section were found using a first, non-optimised implementation of the DYNET algorithm in Java. By optimising the implementation the absolute search times can be decreased. From the shown times it becomes clear the algorithm needs more than o(n) searchtime as the problem becomes more complex. So the searchtime is non linear with the complexity of the problem instance.

6.4 Ad-hoc networking

The implementation of ad-hoc networking was not a main subject of this thesis project. However, the personal agent implementation is prepared to function in ad-hoc environments. If the personal agent is provided with a list of nearby other personal agents and their (IP) addresses, the personal agent can first try to query these agents in order to find out whether these other agents are travelling (partly) in the same direction or have new information on delays.

If this information is found from nearby agents, the agent can subscribe to the other agent to receive information on delays occurring on the current route plan. This way the agent can keep up to date as long as the other agent is nearby over the wireless ad-hoc network. However as soon as the other device is to far away, the personal agent will have to search a new nearby agent to follow.

Figure 6.6 shows a screenshot of the nearby agents tab of the personal agent interface. Currently nearby agents are added and removed manually. By integrating ad-hoc network technology, the list of nearby agents can be generated from the over the wireless network reachable handhelds. This way the personal agent can first try to find the information locally over the ad-hoc network and only use the (expensive) mobile internet connection, when necessary because no information is available locally.



Nearby agents:

traveller0@glas.its.tudelft.nl
traveller1@glas.its.tudelft.nl
traveller2@glas.its.tudelft.nl
zaurus@zaurus

Agent name: zaurus

Hostname: zaurus

Figure 6.6 Screenshot of the nearby agents configuration

6.5 The interfaces

As described in section 5.4, several graphical interfaces were developed as part of the project. These interfaces were designed using Eclipse Visual Editor. A central component of each interface is the tabbed pane which is used in order to divide the different settings and views over different panes. Because the resolution of handheld devices is limited, tabbed panes offer a clean solution to arrange the different components.

Furthermore a speech synthesis engine based on Festival Lite with an American male voice was compiled with a special wrapper developed in C for the ARM processor of the Sharp SL-C860 Zaurus. The C wrapper takes care of setting up a serversocket, accepting TCP/IP connections and outputting the text which is send through the socket connection to the sounddevice of the handheld. This makes it possible to use the speech synthesis engine from Java by opening a socket connection on the device and outputting text to the socket, which is read out loud over the sound device of the handheld device. The C source code of the wrapper is included in the appendix.

However, as stated in section 4.2, the implementation of a multi modal interface was not the subject of this thesis and therefore the implementation of the speech synthesis engine has to be seen as a nice side product of the project in stead of a subject which got considerable attention. As such, no tests with users were conducted to determine the usability of the interfaces. The provided graphical interfaces are part of the prototype which was developed and are able to run on devices supporting Java 1.3.1.

Chapter 7: User scenario

This chapter describes a simulated journey with the PITA system starting in Delft at 10:00 going to station Groningen Noord. First the used backend configuration is described. Next the travel parameters are configured and the advise is started. Then a interregional and a regional delay are introduced which affect the planned route. Finally the destination is reached.

7.1 Backend configuration

For this scenario, a server configuration is used with two servers: one on location Groningen and one in Maastricht. This gives a region division as can be seen in figure 7.1, which shows a screenshot of the system management interface.

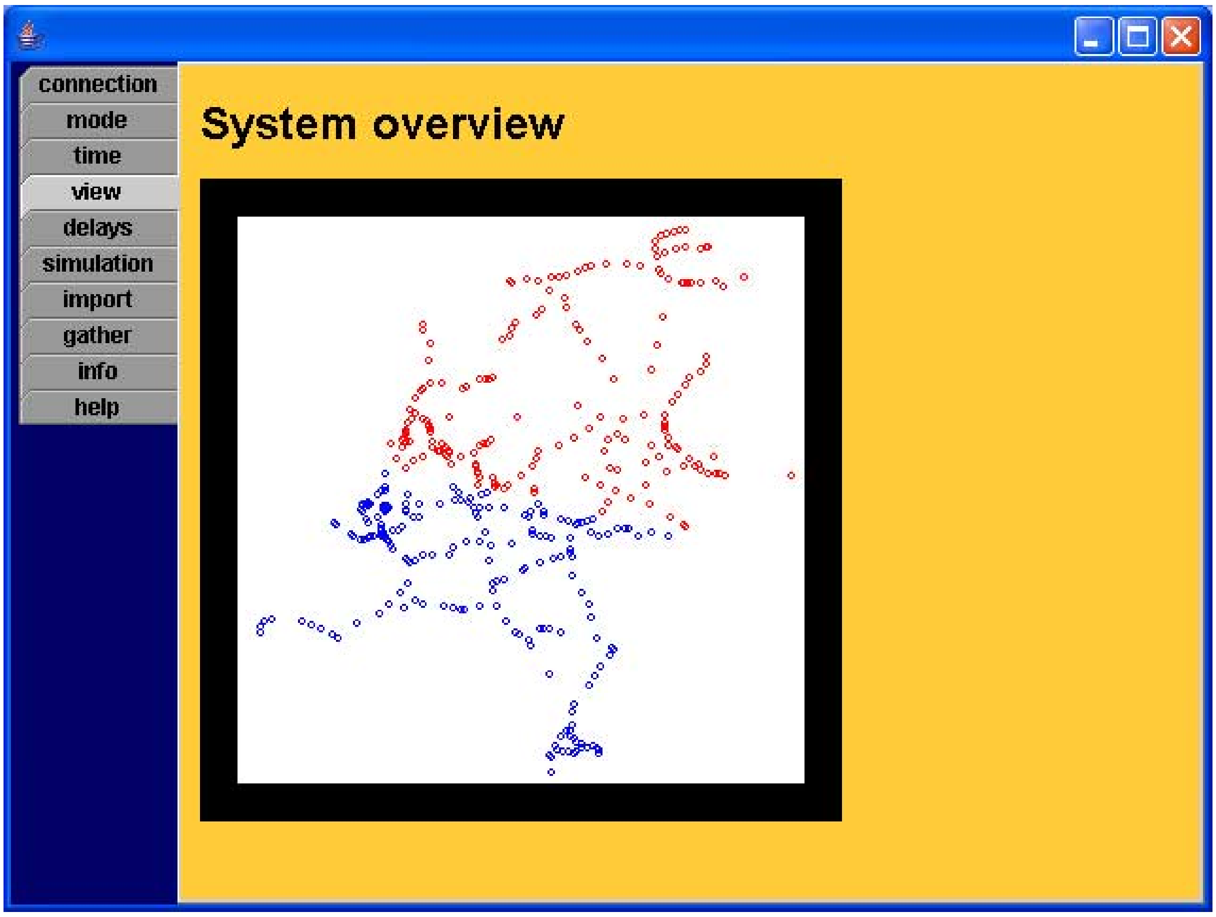
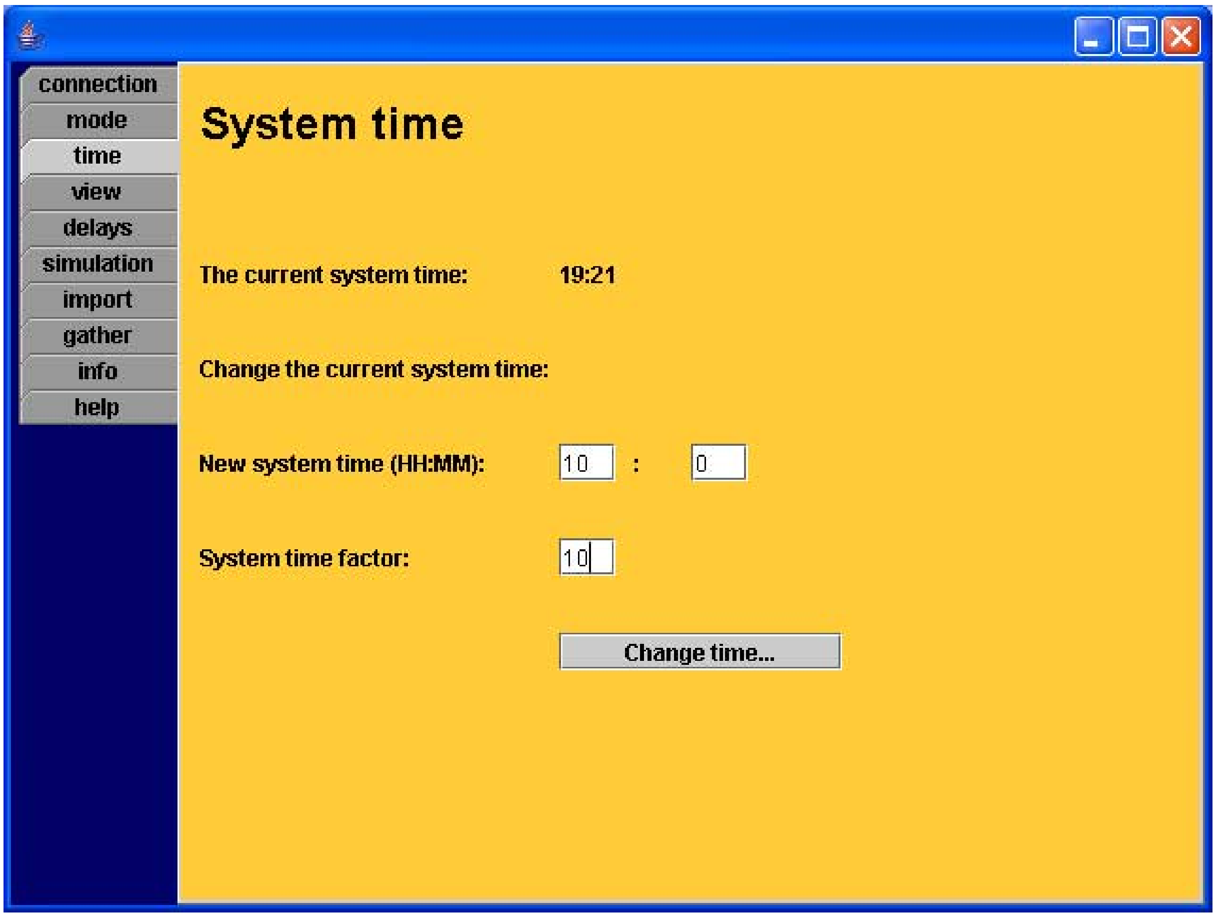


Figure 7.1 The region division between a server in Maastricht and a server in Groningen

Each of the pixels represents a trainstation. The color of a pixel determines to which region the station belongs. The blue pixels are the responsibility of the Maastricht server and the red pixels of the Groningen server. The region division is used for the synchronisation of the delay data, as described in section 5.2.2.3, and to decide which server is contacted by the personal agent to plan the route to the destination.

Because the journey starts at 10:00, we have to configure the backend servers to change their system time to 10:00. Using the same interface it is possible to do this, as is shown in figure 7.2. Besides the system time it is also possible to configure the time factor, which is the speed up factor used for the simulation. A time factor of 10 means every real second, the system advances the system time with 10 seconds.



System time

The current system time: 19:21

Change the current system time:

New system time (HH:MM): 10 : 0

System time factor: 10

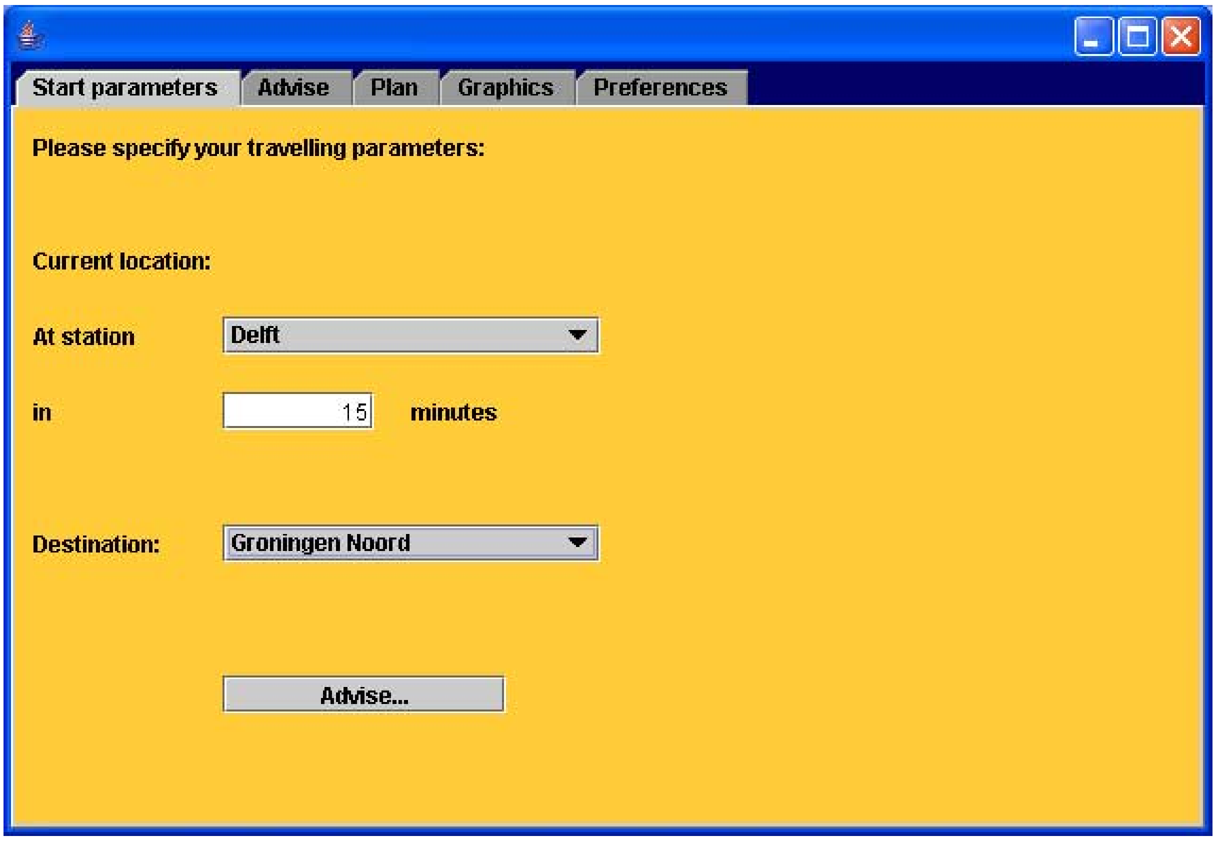
Figure 7.2 Configuration of the system time

After configuring the system time, the backend is ready for starting the simulation. The rest of this chapter will continue with the personal agent interface which provides the route guidance to the traveller.

7.2 Specification of the travel parameters

Before starting a journey, the PITA system needs to know at which station the traveller starts his journey and which destination the traveller wants to reach. This can be done using the personal agent interface, which is shown in figure 7.3. Using the combo boxes, containing all stations, the start and destination station can be selected. By pressing the first letter of the station, the selection will jump to the first station starting with this letter, which speeds up the selection process. The text field is used to specify in how much time the starting station is reached by the traveller. This makes it possible to plan a journey while the traveller is travelling to the starting station, for instance in a taxi or bus.

After these parameters are configured, the user presses the Advise button, which will start the route guidance. The personal agent interface will jump to the next tabpane, which is shown in figure 7.4. At the top of this pane, the personal travel advice for the user is shown. Below the advice, the user has two radiobutton selection possibilities to specify wheter he is still following the advise or has lost track of the advice. If the user is no longer following the advice, two new comboboxes are enabled which allow him to respecify his current location and destination. After pressing the Update advise button, the application will use his new position data to generate a new route and restart the route guidance.



Please specify your travelling parameters:

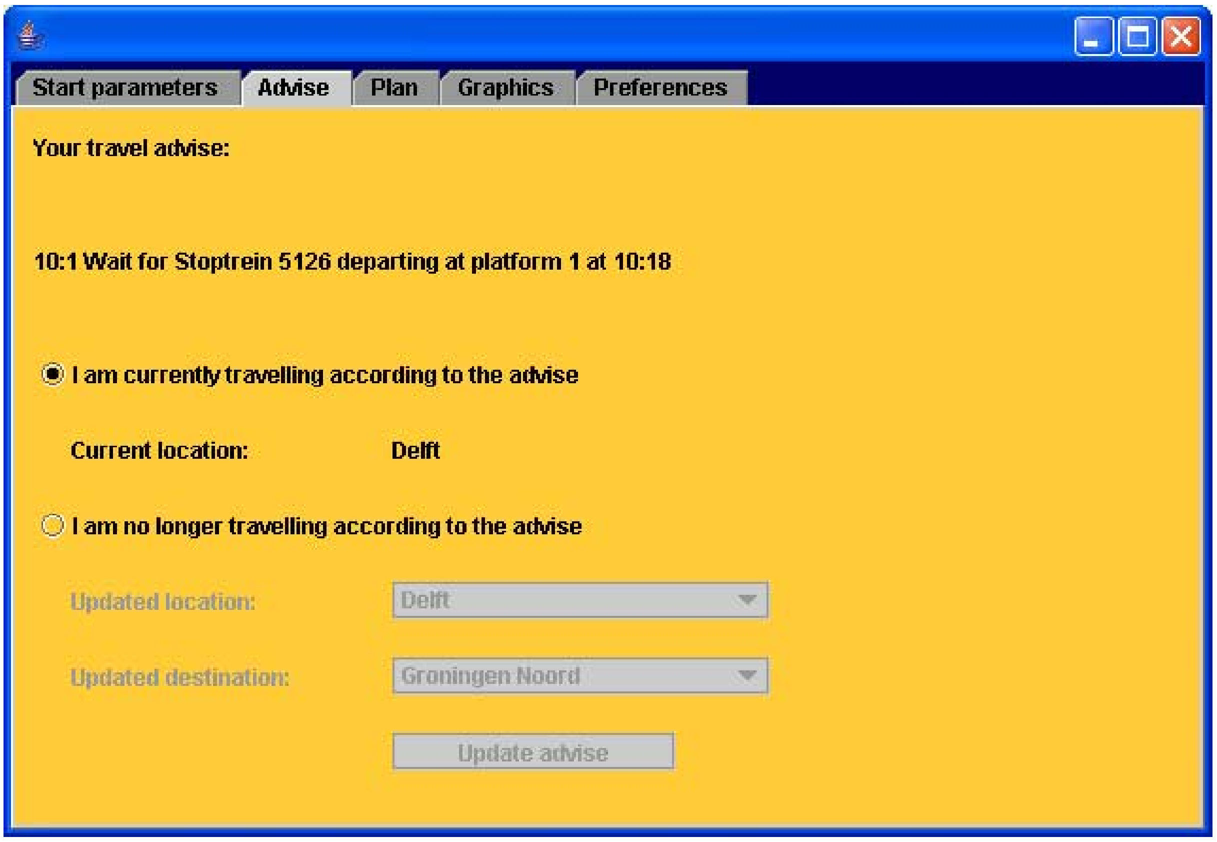
Current location:

At station Delft

in 15 minutes

Destination: Groningen Noord

Figure 7.3 Screenshot of the configuration of the travel parameters



Your travel advise:

10:1 Wait for Stoptrein 5126 departing at platform 1 at 10:18

● I am currently travelling according to the advise

Current location: Delft

○ I am no longer travelling according to the advise

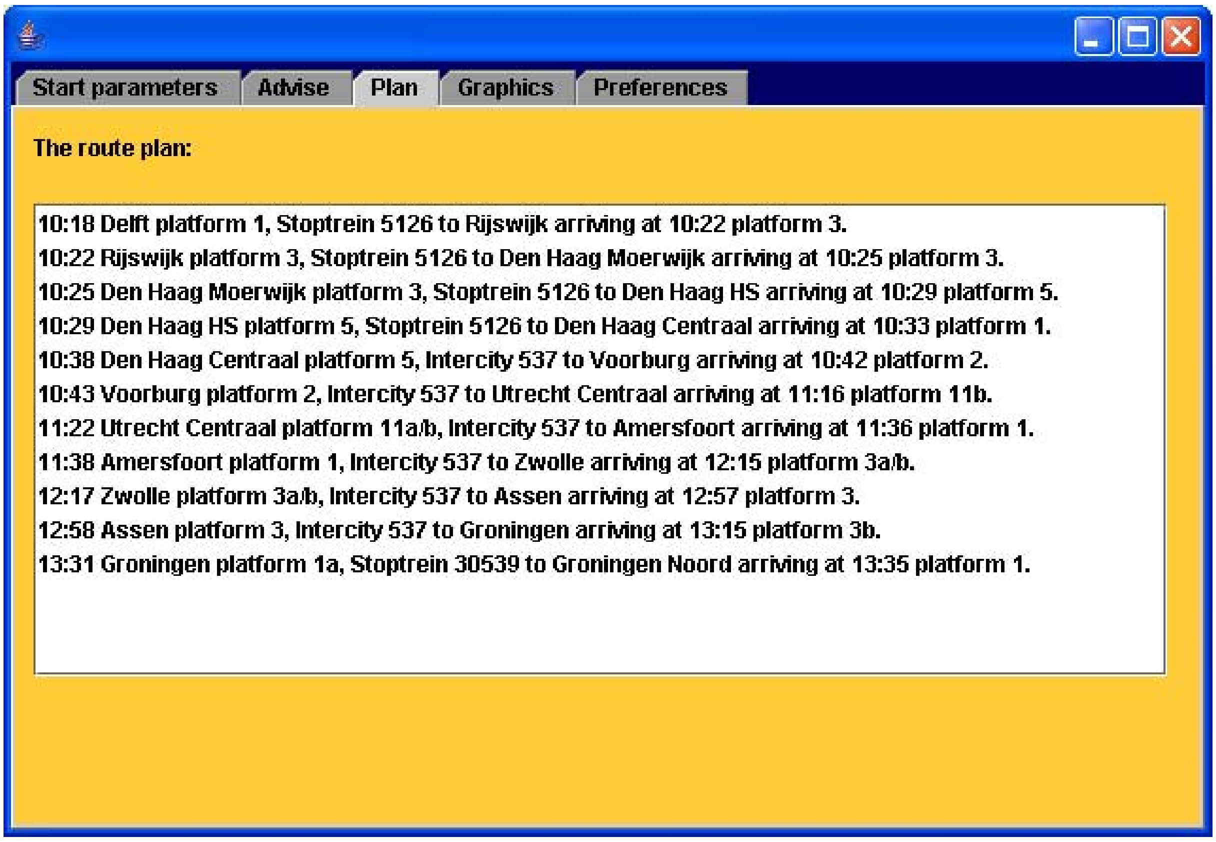
Updated location: Delft

Updated destination: Groningen Noord

Figure 7.4 The personal travel advise tabpane

7.3 The route plan

During his travel, the user can switch between the Advise and the Plan tabpanes. The Advise tabpane shows the current advice, while the Plan tabpane shows the route plan, which forms the basis for the user advice and shows which connections the system has planned to take.



```
10:18 Delft platform 1, Stoptrein 5126 to Rijswijk arriving at 10:22 platform 3.
10:22 Rijswijk platform 3, Stoptrein 5126 to Den Haag Moerwijk arriving at 10:25 platform 3.
10:25 Den Haag Moerwijk platform 3, Stoptrein 5126 to Den Haag HS arriving at 10:29 platform 5.
10:29 Den Haag HS platform 5, Stoptrein 5126 to Den Haag Centraal arriving at 10:33 platform 1.
10:38 Den Haag Centraal platform 5, Intercity 537 to Voorburg arriving at 10:42 platform 2.
10:43 Voorburg platform 2, Intercity 537 to Utrecht Centraal arriving at 11:16 platform 11b.
11:22 Utrecht Centraal platform 11a/b, Intercity 537 to Amersfoort arriving at 11:36 platform 1.
11:38 Amersfoort platform 1, Intercity 537 to Zwolle arriving at 12:15 platform 3a/b.
12:17 Zwolle platform 3a/b, Intercity 537 to Assen arriving at 12:57 platform 3.
12:58 Assen platform 3, Intercity 537 to Groningen arriving at 13:15 platform 3b.
13:31 Groningen platform 1a, Stoptrein 30539 to Groningen Noord arriving at 13:35 platform 1.
```

Figure 7.5 The route plan for reaching Groningen from Delft

The route plan shows each individual connection separately at the moment and does not combine connections of the same train into a single connection. This gives a very extensive view on which trains have to be used. The current plan corresponds to a static route, which can also be found on the NS website or in a trainschedule. In order to make the situation more interesting, we will now introduce delays which affect the route plan to see how the system uses delay information.

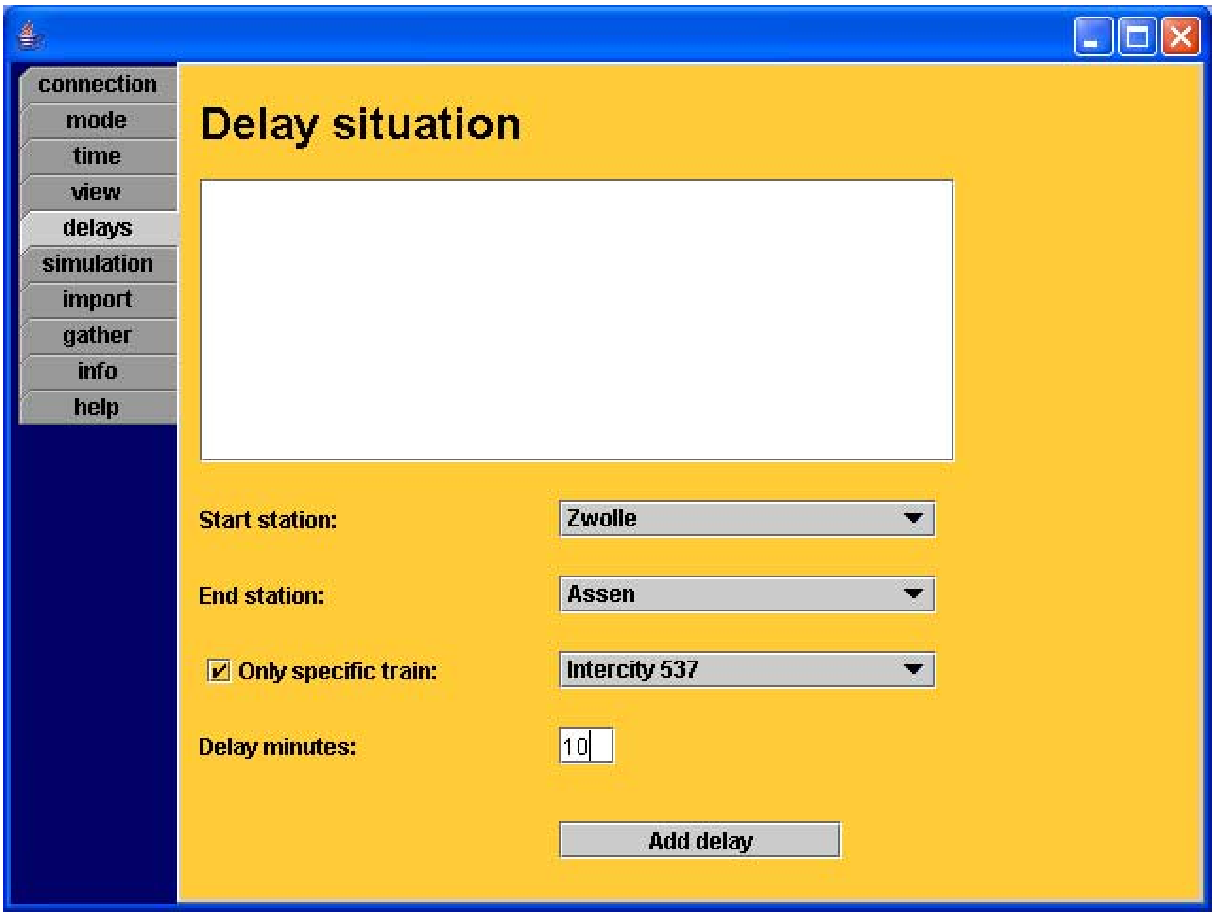
7.4 The introduction of delays

In this section we will introduce several different delays. First a interregional delay will be introduced which does not make a reschedule necessary. By increasing the delay, the system will be forced to reschedule the journey. Also a regional delay will be introduced, which is not communicated to the personal agent until the traveller reaches the other region.

7.4.1 Interregional delay

To start we will introduce a 10 minute delay for Intercity 537 from Zwolle. This can be done using the system management interface. On the Delays pane, first the starting station has to be selected. This will update the contents of the end station selection box, as this box will only contain stations to which connections are possible from the start station. After selecting the end station, the third combobox will contain all trains which ride along this track. Using the checkbox, a single train can be selected or in the other case all trains in the third combobox will be delayed.

As can be seen in figure 7.6, we introduce a 10 minute delay for Intercity 537 from station Zwolle. This means the train will depart from station Zwolle at the normal (static) departure time, but arrive later at Assen. Subsequently the train will depart 10 minutes later from station Assen and arrive and depart from all following stations with a delay of 10 minutes.



Start station: Zwolle

End station: Assen

[✓] Only specific train: Intercity 537

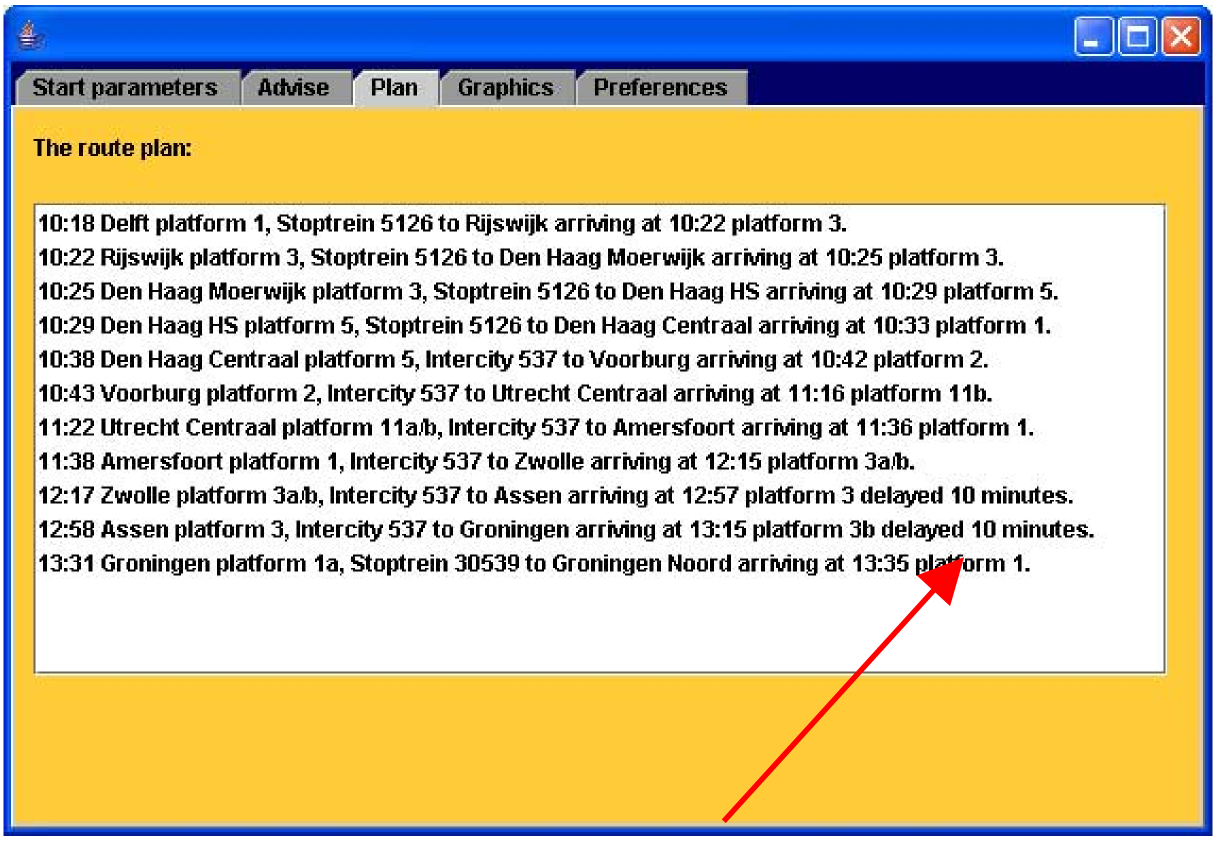
Delay minutes: 10

Figure 7.6 The introduction of an interregional delay for Intercity 537

Since Intercity 537 crosses the region boundaries (station Utrecht is in the Maastricht region and station Amersfoort is in the Groningen region), the delay will be synchronised over all servers and the personal agent will get a notification of the Maastricht server. After receiving this notification, the personal agent will update the travel plan with the new delay information. The new route plan is shown in figure 7.7.

Since this delay still makes it possible to arrive at station Groningen before the departure of Stoptrein 30539, the personal agent will only update the route plan with the delay and not take actions to reschedule a new route to the destination.

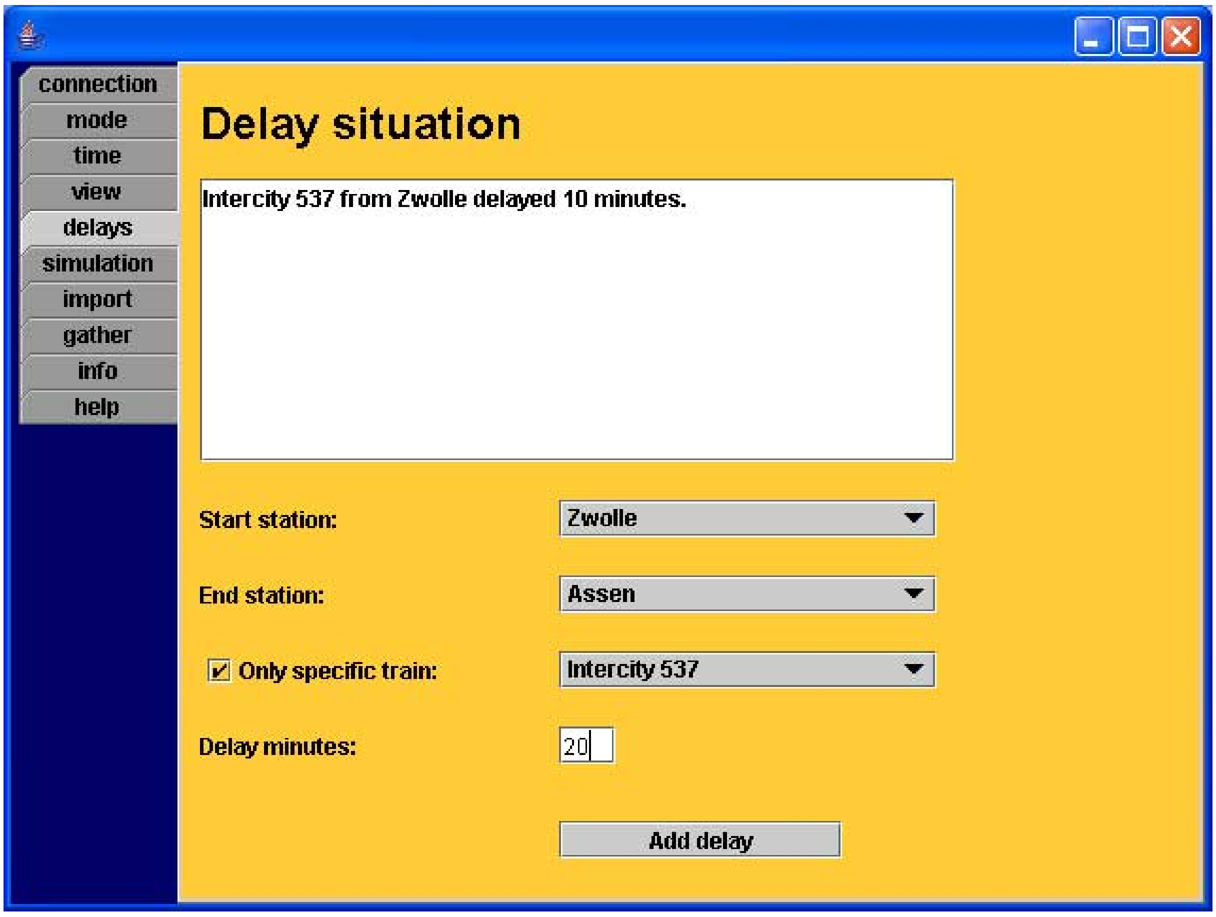
While the user travels from Voorburg to Utrecht, the delay is updated to 20 minutes, which no longer makes it possible to have the connection to Groningen Noord with Stoptrein 30539. Figure 7.8 shows the server management interface which is used to update the delay to 20 minutes.



The route plan:

```
10:18 Delft platform 1, Stoptrein 5126 to Rijswijk arriving at 10:22 platform 3.
10:22 Rijswijk platform 3, Stoptrein 5126 to Den Haag Moerwijk arriving at 10:25 platform 3.
10:25 Den Haag Moerwijk platform 3, Stoptrein 5126 to Den Haag HS arriving at 10:29 platform 5.
10:29 Den Haag HS platform 5, Stoptrein 5126 to Den Haag Centraal arriving at 10:33 platform 1.
10:38 Den Haag Centraal platform 5, Intercity 537 to Voorburg arriving at 10:42 platform 2.
10:43 Voorburg platform 2, Intercity 537 to Utrecht Centraal arriving at 11:16 platform 11b.
11:22 Utrecht Centraal platform 11a/b, Intercity 537 to Amersfoort arriving at 11:36 platform 1.
11:38 Amersfoort platform 1, Intercity 537 to Zwolle arriving at 12:15 platform 3a/b.
12:17 Zwolle platform 3a/b, Intercity 537 to Assen arriving at 12:57 platform 3 delayed 10 minutes.
12:58 Assen platform 3, Intercity 537 to Groningen arriving at 13:15 platform 3b delayed 10 minutes.
13:31 Groningen platform 1a, Stoptrein 30539 to Groningen Noord arriving at 13:35 platform 1.
```

Figure 7.7 The updated route plan, containing a 10 minute delay from station Zwolle



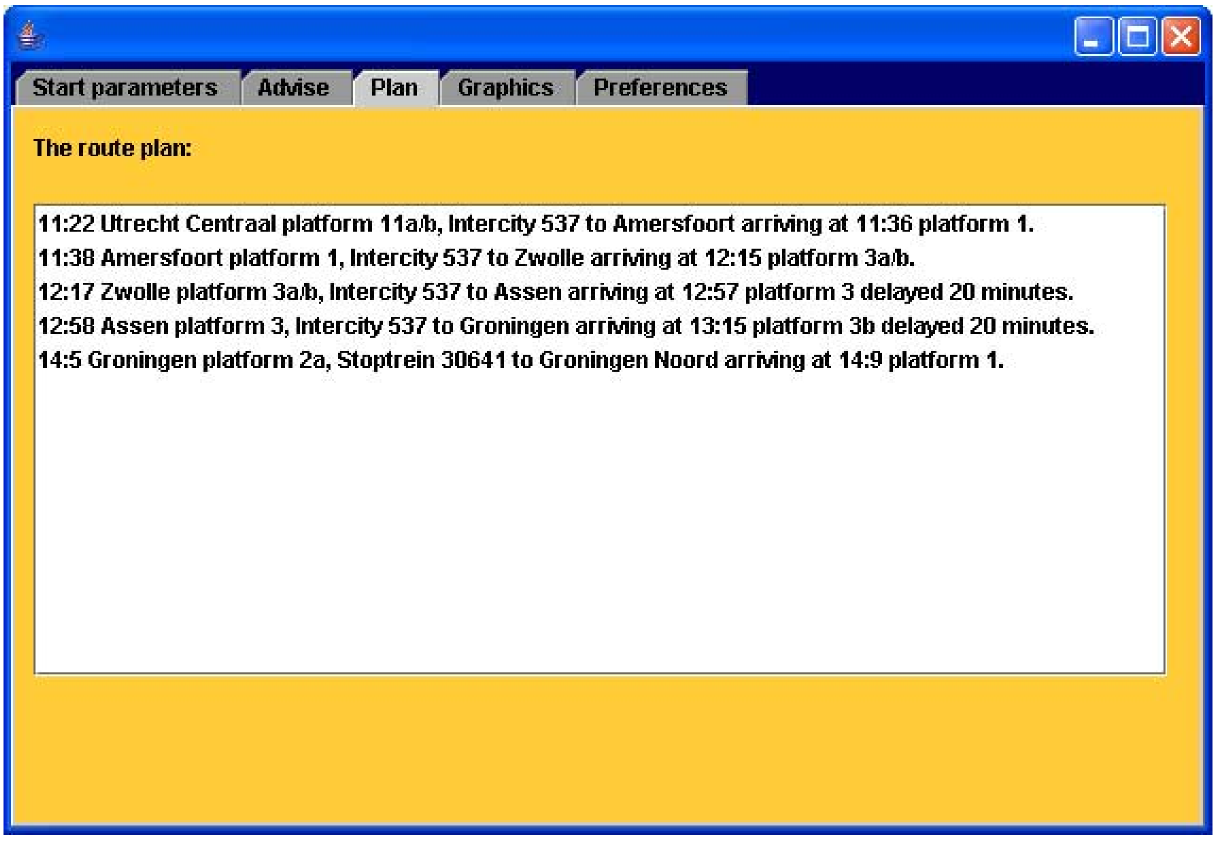
Delay situation

```
Intercity 537 from Zwolle delayed 10 minutes.
```

Start station:	Zwolle
End station:	Assen
☑ Only specific train:	Intercity 537
Delay minutes:	20

Figure 7.8 Updating the delay to 20 minutes

Because the delay affects an interregional train, the personal agent is notified after updating the delay. Because this time it is no longer possible to reach the destination with the updated plan, the route is rescheduled from station Utrecht Centraal (as the user is currently in the train from Voorburg to Utrecht Centraal).



The route plan:

11:22 Utrecht Centraal platform 11a/b, Intercity 537 to Amersfoort arriving at 11:36 platform 1.
11:38 Amersfoort platform 1, Intercity 537 to Zwolle arriving at 12:15 platform 3a/b.
12:17 Zwolle platform 3a/b, Intercity 537 to Assen arriving at 12:57 platform 3 delayed 20 minutes.
12:58 Assen platform 3, Intercity 537 to Groningen arriving at 13:15 platform 3b delayed 20 minutes.
14:5 Groningen platform 2a, Stoptrein 30641 to Groningen Noord arriving at 14:9 platform 1.

Figure 7.9 The new route plan to Groningen Noord from Utrecht Centraal

As can be seen in figure 7.9, the new route from station Utrecht Centraal still uses the delayed Intercity 537. For this advise we used a profile with a change_value of 10 minutes. Table 7.1 shows which options the algorithm has compared for travelling to Groningen. Because of the 10 minutes change_value, the search algorithm prefers to remain on the same train and find a new connection to Groningen Noord.

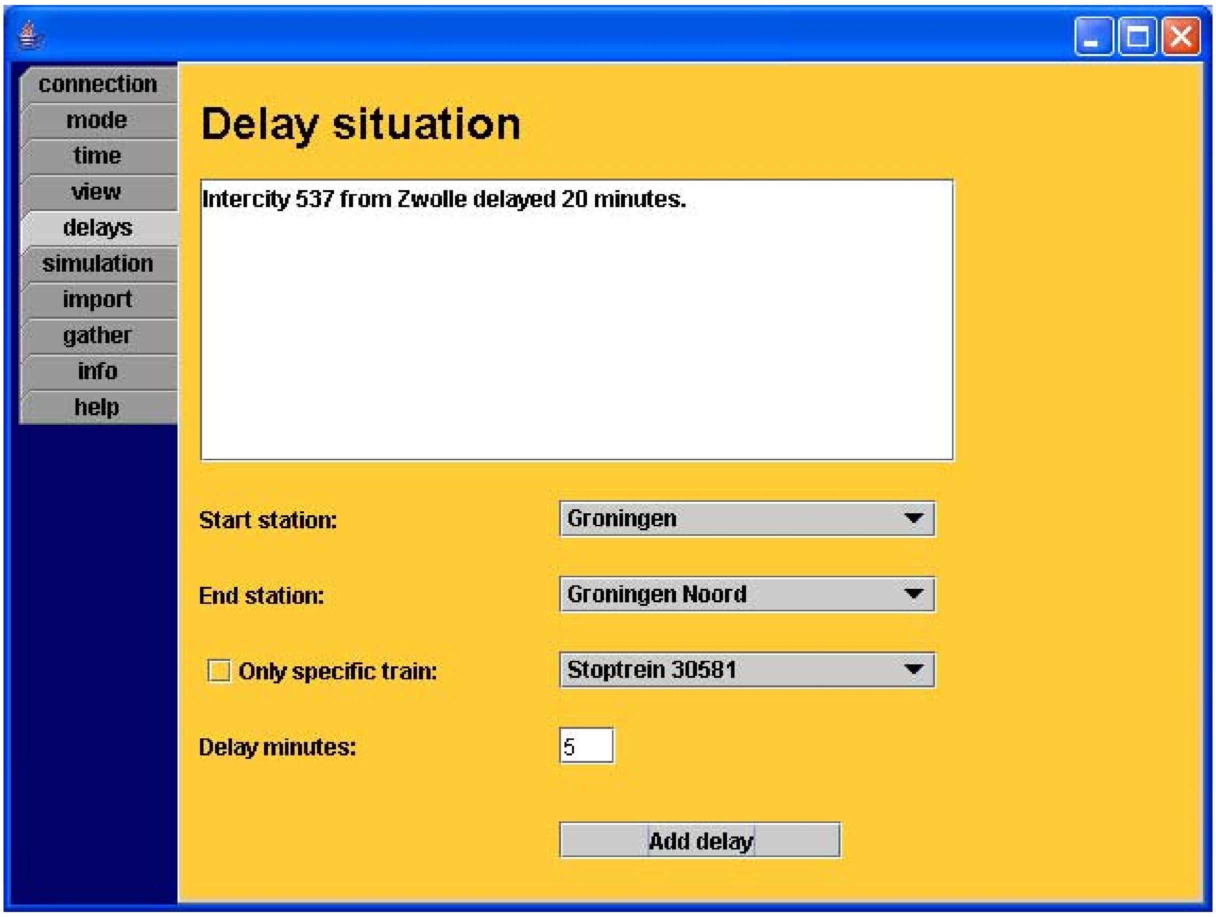
Table 7.1 The two options considered by the algorithm for reaching Groningen

Train:	Intercity 537	Stoptrein 9137
Departure time Zwolle:	12:17	12:26
Arrival time Groningen:	13:15	13:31
Delay minutes:	00:20	00:00
change_value penalty:	00:00	00:10
Corrected arrival time:	**13:35**	**13:41**

Next we will introduce a regional delay in the region Groningen.

7.4.2 Regional delay

As regional delays are distributed in another way as interregional delays, it is interesting to look at the effect of introducing regional delays. The traveller currently travels from Voorburg to Utrecht Centraal in Intercity 537. We will delay all trains on the track Groningen to Groningen Noord for 5 minutes. This can be done by selecting the track and leaving the checkbox for “Only specific train” unchecked, as is shown in figure 7.10.



Delay situation

Intercity 537 from Zwolle delayed 20 minutes.

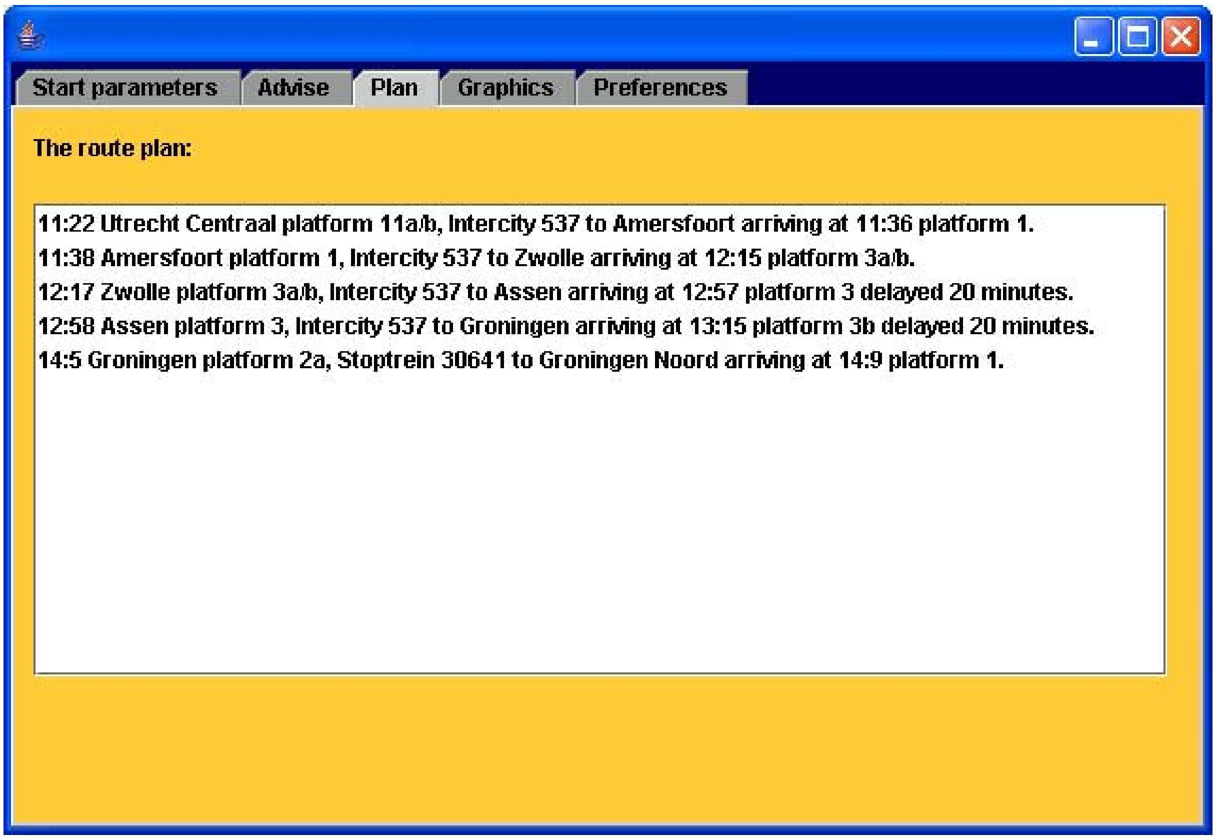
Start station: Groningen

End station: Groningen Noord

☐ Only specific train: Stoptrein 30581

Delay minutes: 5

Figure 7.10 Introducing a regional delay for all trains between Groningen and Groningen Noord



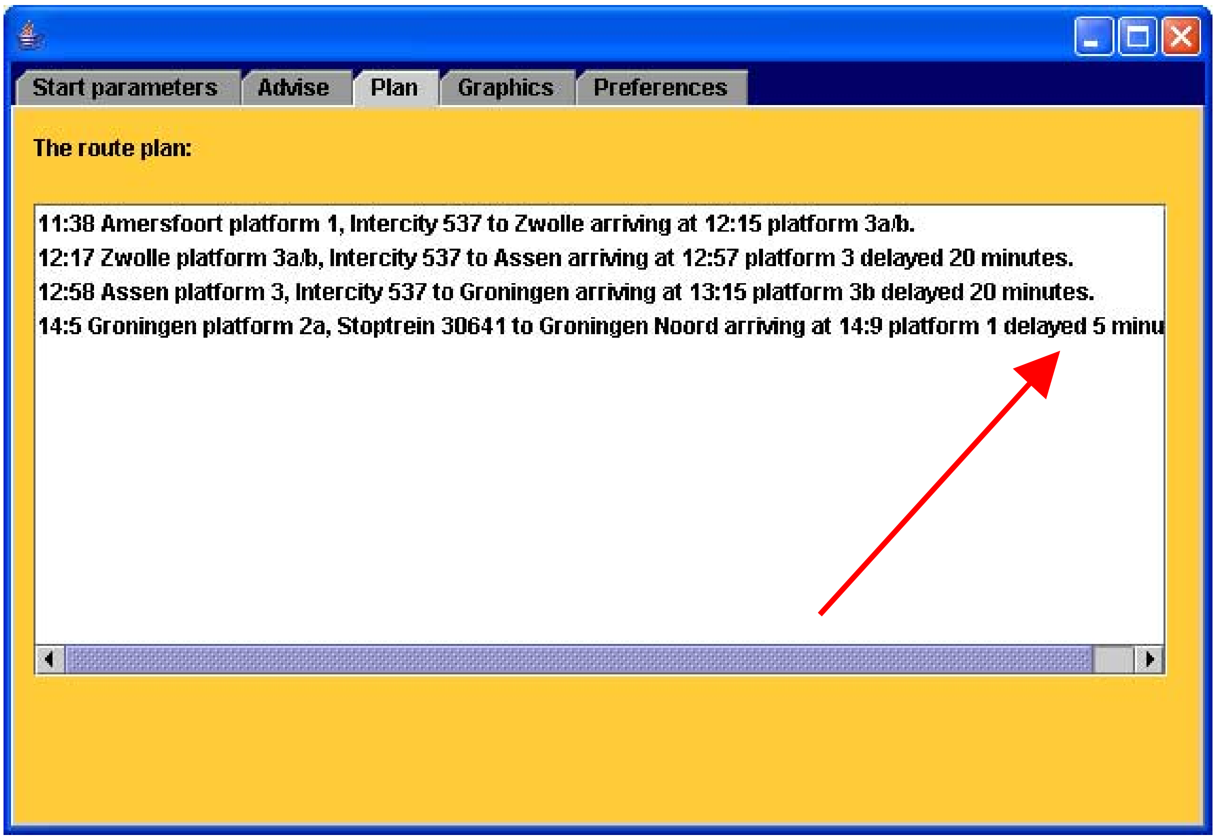
The route plan:

```
11:22 Utrecht Centraal platform 11a/b, Intercity 537 to Amersfoort arriving at 11:36 platform 1.
11:38 Amersfoort platform 1, Intercity 537 to Zwolle arriving at 12:15 platform 3a/b.
12:17 Zwolle platform 3a/b, Intercity 537 to Assen arriving at 12:57 platform 3 delayed 20 minutes.
12:58 Assen platform 3, Intercity 537 to Groningen arriving at 13:15 platform 3b delayed 20 minutes.
14:5 Groningen platform 2a, Stoptrein 30641 to Groningen Noord arriving at 14:9 platform 1.
```

Figure 7.11 The advise is not updated, as the traveller is still in the Maarstricht region

Because regional delays are only distributed to the servers which have the responsibility for the region in which the train rides, only the Groningen server knows of this regional delay. The traveller is still travelling from Voorburg to Utrecht and therefore in the Maastricht region. This means the route plan is not updated, as can be seen in screenshot 7.11.

However, as soon as the traveller reaches the Groningen region by travelling from Utercht Centraal to Amersfoort in Intercity 537, the personal agent contacts the Groningen server and gets the information on the delay of the Groningen to Groningen Noord track. This means the route plan is updated as can be seen in figure 7.12.



The route plan:

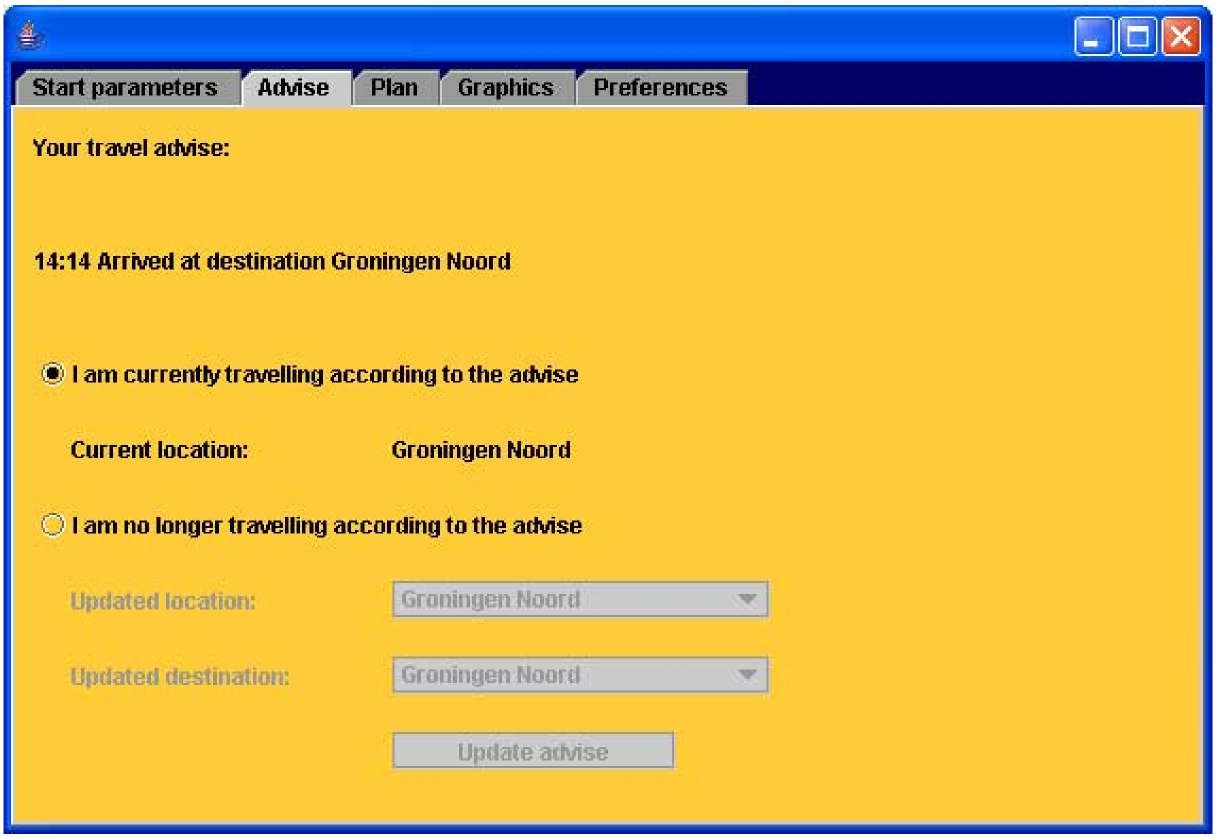
11:38 Amersfoort platform 1, Intercity 537 to Zwolle arriving at 12:15 platform 3a/b.
12:17 Zwolle platform 3a/b, Intercity 537 to Assen arriving at 12:57 platform 3 delayed 20 minutes.
12:58 Assen platform 3, Intercity 537 to Groningen arriving at 13:15 platform 3b delayed 20 minutes.
14:5 Groningen platform 2a, Stoptrein 30641 to Groningen Noord arriving at 14:9 platform 1 delayed 5 minu

Figure 7.12 The updated route plan from Amersfoort to Groningen Noord with the regional delay included

7.4.3 Reaching the destination

The traveller will eventually travel with Intercity 537 to Groningen, reaching Groningen at 13:35. Then waiting for the departure of Stoptrein 30641 at 14:05, the destination Groningen Noord will be reached at 14:14. Switching back to the Advise tabpane, screenshot 7.13 was made after the traveller succesfully arrived at destination Groningen Noord.

This showed a scenario containing some important aspects of the system: interregional delays, regional delays, rescheduling after a delay and rescheduling after switching to another region. Throughout the scenario, the sytem was rescheduling and communicating with the backend, while the user only had to follow the given advice and could trust the system to relieve him from the travelling problems. By adequately informing the traveller, he (or she) can focus on other things and only has to follow the advice at train changes.



Your travel advise:

14:14 Arrived at destination Groningen Noord

[✓ I am currently travelling according to the advise]

Current location: Groningen Noord

I am no longer travelling according to the advise

Updated location: Groningen Noord

Updated destination: Groningen Noord

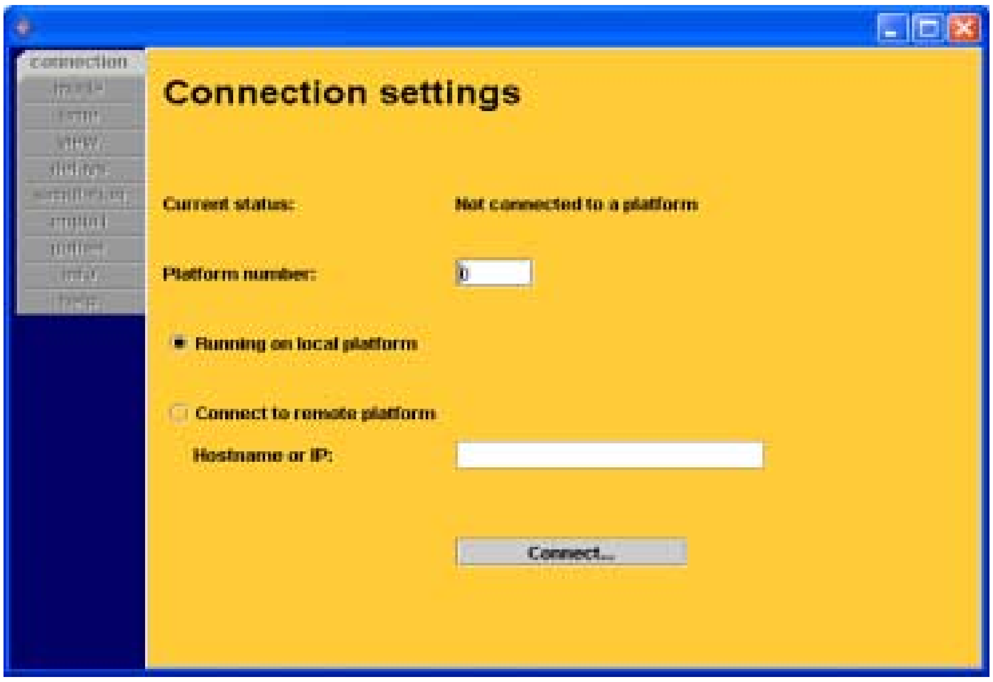
Figure 7.13 The traveller has reached the destination Groningen Noord at 14:14

Chapter 8: Simulation environment

In this chapter the simulation environment which was developed as part of this project is described. First the simulation possibilities of the server management interface are described, followed by the NS delay agent. Then the simulation options of the personal agent interface are described. Finally a visualisation applet which was developed as part of this project is described.

8.1 Server management interface

As described in previous chapters, the server management interface offers different functionality in order to perform simulation experiments. The first option the interface offers is the adjustment of the system time, as shown in chapter 7. Also shown in chapter 7 is the possibility to view the current region division among the distributed servers of the backend and the possibility to manually introduce delays to the system. However, an important part of the functioning of the server management interface was not yet discussed. As the server management interface is started, it is necessary to connect it to a single backend server as shown in figure 8.1.



Connection settings

Current status: Not connected to a platform

Platform number: 0

[✓ Running on local platform]

Connect to remote platform

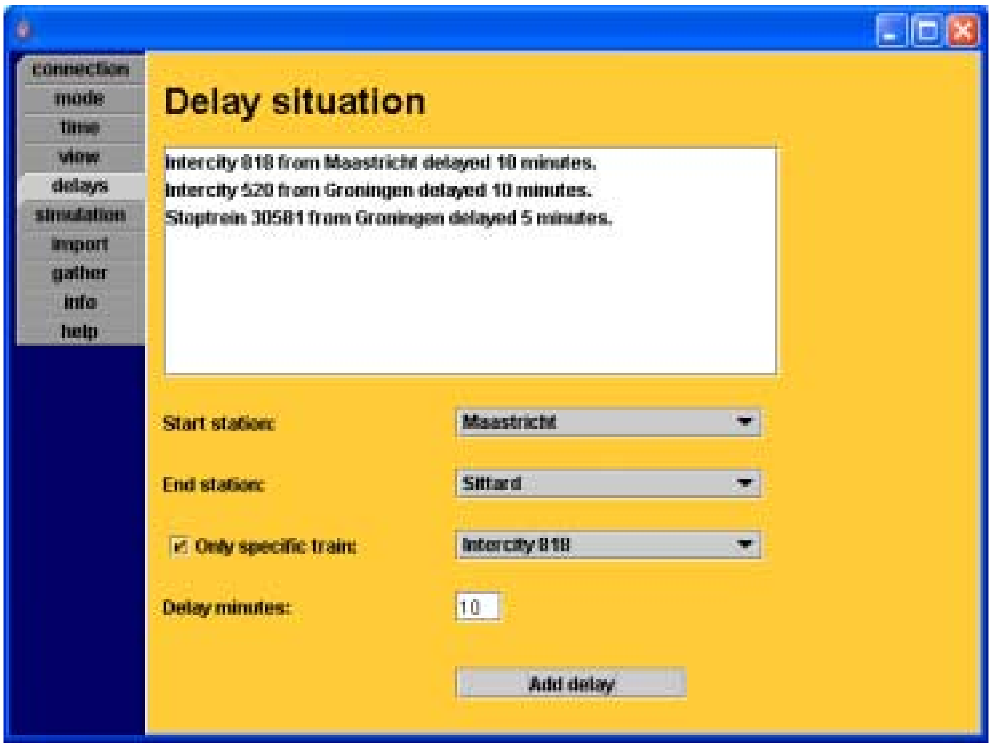
Hostname or IP:

Figure 8.1 Connecting the server management interface to a PITA server platform

As can be seen, it is possible to connect to a platform running on the local server, or to connect to a platform on another server. After connecting to a server, the time is synchronised with the system time of this server and the delays which are known to this particular server are loaded into the system. This is a very important aspect, because this also shows the distribution model of the delay data. For instance, if the region division is the same as described in section 7.1, sending a delay for a regional train in the region Groningen to the Maastricht server, will only add this delay to the Groningen server. To illustrate this, we used the same region division and added the following delays:

Maastricht – Maastricht Randwyck – Stoptrein 5394 – 5 minutes delay (regional)
Groningen – Groningen Noord – Stoptrein 30581 – 5 minutes delay (regional)
Groningen – Assen – Intercity 520 – 10 minutes delay (inter-regional)
Maastricht – Sittard – Intercity 818 – 10 minutes delay (inter-regional)

After these delays were added to the system, the delay situation on each of the servers looked as is shown in figure 8.2 and 8.3 for each server.



Delay situation

Intercity 818 from Maastricht delayed 10 minutes.
Intercity 520 from Groningen delayed 10 minutes.
Stoptrein 30581 from Groningen delayed 5 minutes.

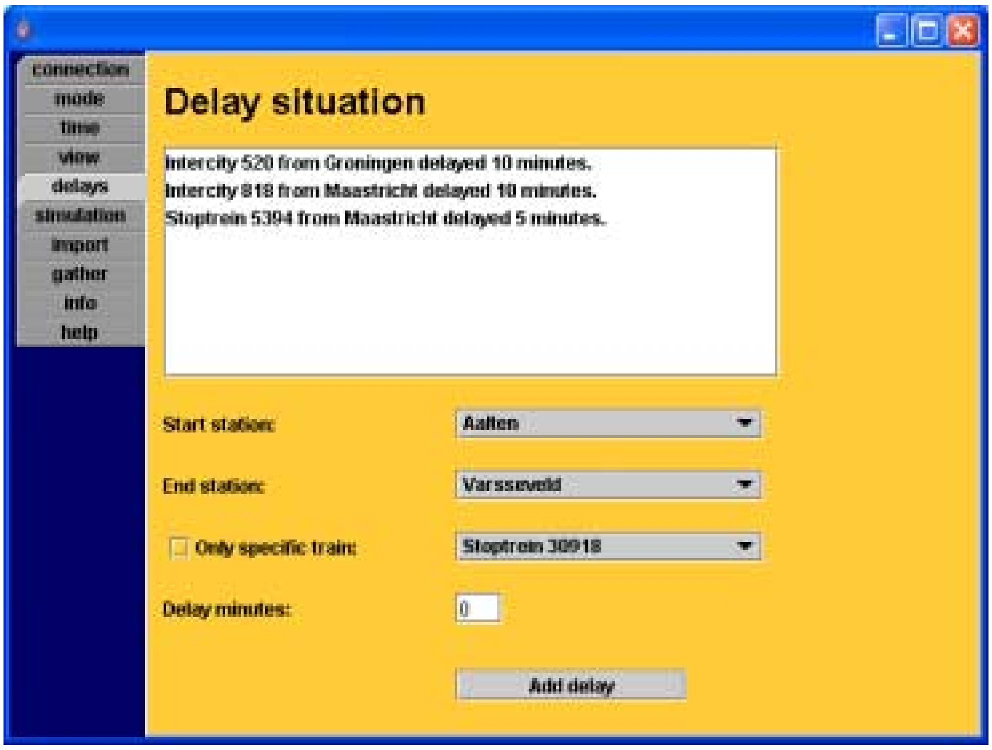
Start station: Maastricht

End station: Sittard

[✓] Only specific train: Intercity 818

Delay minutes: 10

Figure 8.2 The delay situation on the Groningen server after the addition of the delays



Delay situation

Intercity 520 from Groningen delayed 10 minutes.
Intercity 818 from Maastricht delayed 10 minutes.
Stoptrein 5394 from Maastricht delayed 5 minutes.

Start station: Aalten

End station: Varsseveld

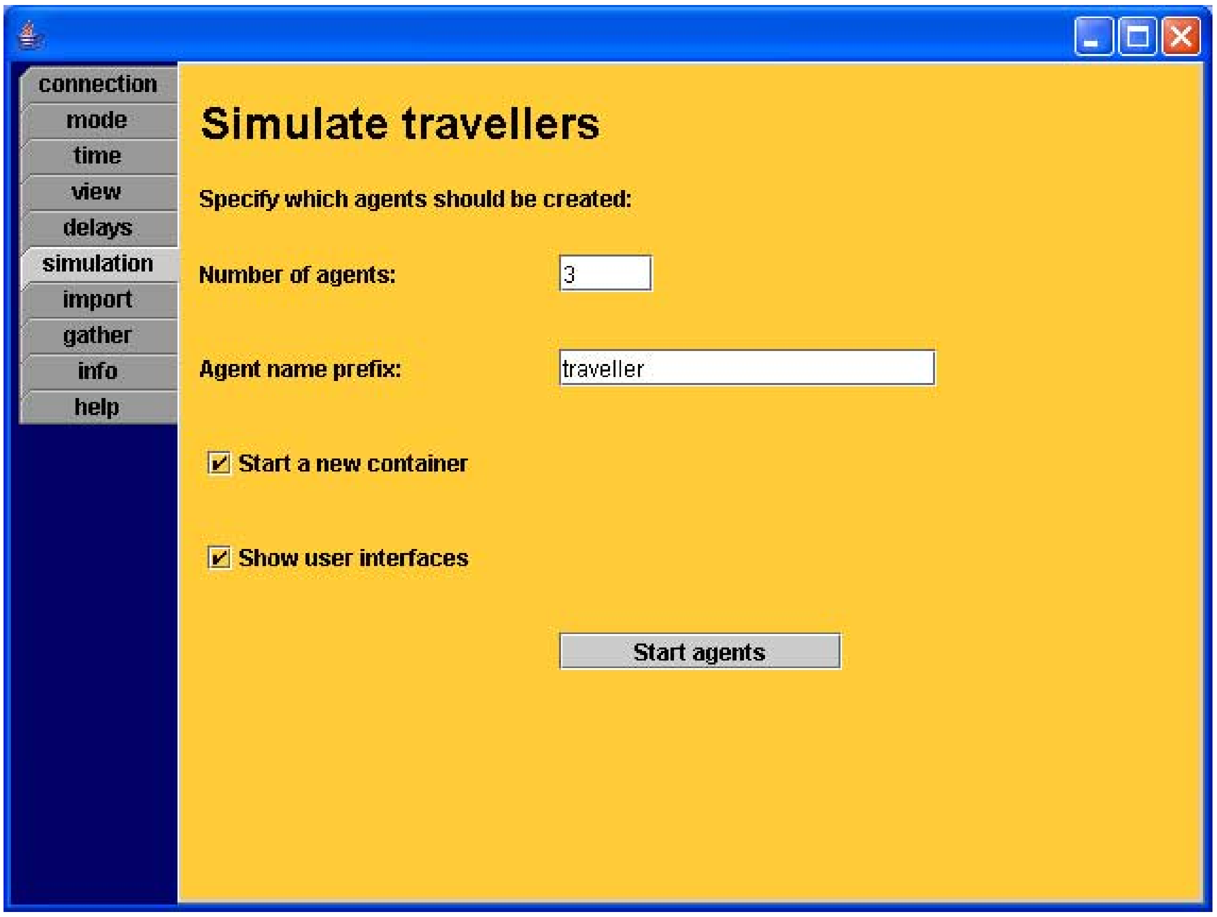
Only specific train: Stoptrein 30918

Delay minutes: 0

Figure 8.3 The delay situation on the Maastricht server after the addition of the delays

As can be seen from the screenshots, the interregional delays are synchronised across both servers, while the regional delays are only known to the regional servers. Using these backend configurations, experiments can be done on the influences of this distribution model on the routing of different travellers. By making the regions very small, this way a simulation can be done for the scenario in which the full application runs on the handheld, as described in section 5.2.6, and only knows about delays within a small region. This is useful for testing the effects of distributing this information and to estimate the chances of realising a distributed handheld application.

Lastly the server management agent can be used to simulate randomly travelling personal agents. Figure 8.4 shows the interface, used for starting simulated travellers.



Simulate travellers

Specify which agents should be created:

Number of agents: 3

Agent name prefix: traveller

☑ Start a new container

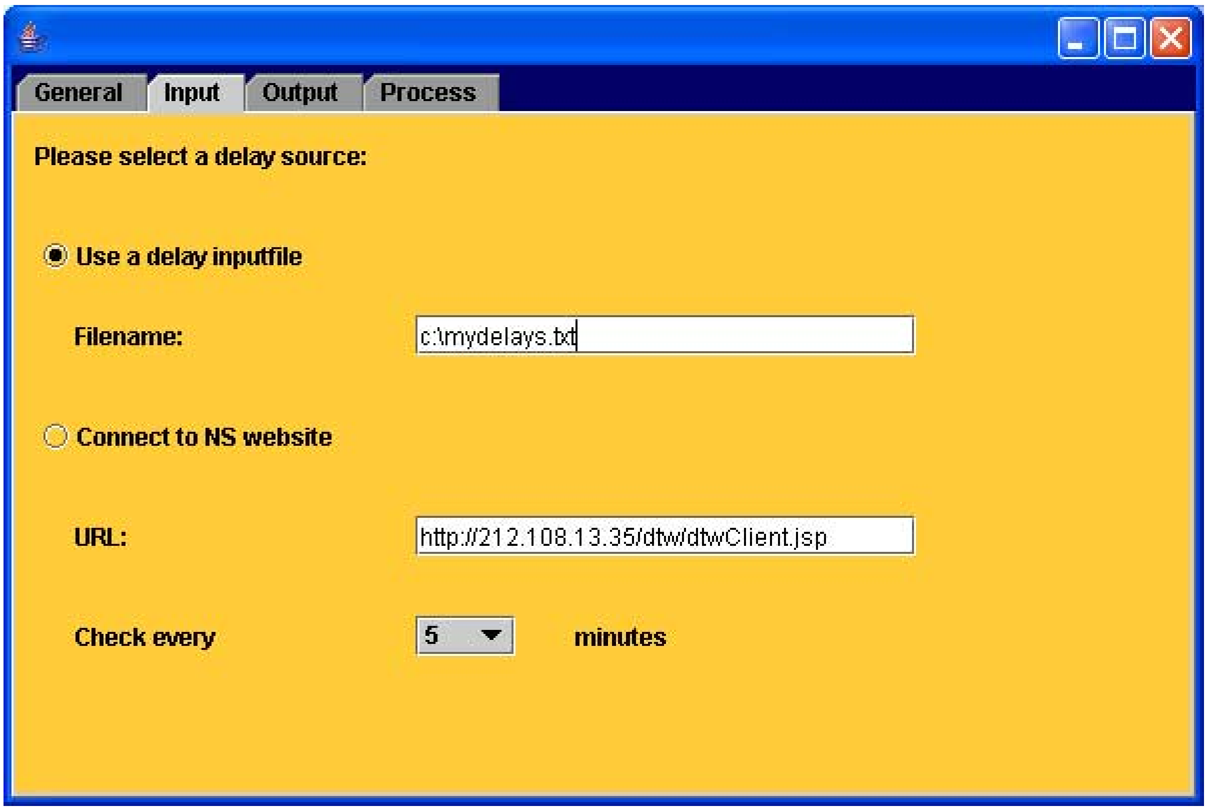
☑ Show user interfaces

Figure 8.4 Random travellers can be introduced to the system

The number of personal agents to start can be specified, followed by a prefix which should be unique, to form the agent names. Next it is possible to select whether the agents should be launched in a separate container and whether or not the user interfaces of the agents should be started. After creation, the agents will be initialized with a random location and destination and start travelling towards the destination.

8.2 NS delay agent

Just like the server management interface connects to a PITA server platform, the NS delay agent also connects to a platform as was already shown in figure 5.21. After the connection to a platform is established, the input for the NS delay agent can be selected on the Input tab pane as shown in figure 8.5.



Please select a delay source:

(●) Use a delay inputfile

Filename: c:\mydelays.txt

( ) Connect to NS website

URL: http://212.108.13.35/dtw/dtwClient.jsp

Check every 5 minutes

Figure 8.5 Selecting the input for the delay simulation

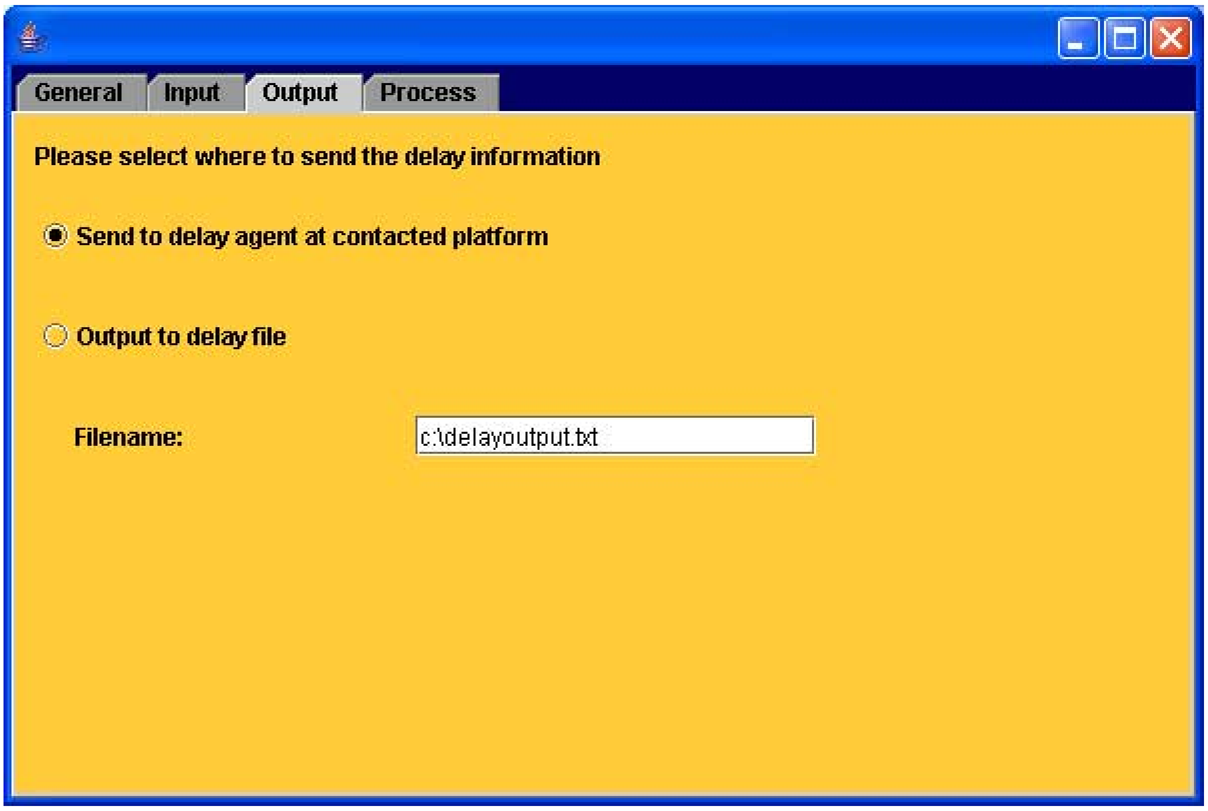
The delays can be gathered either live from the NS website ("actuele vertrektijden") or from an input text file, containing delays in the following format:

```
10:16,rta,Intercity,21739,10
13:18,hgv,Intercity,739,5
```

The first time is the time at which the delay has become known to the system (so this is the system time). Next is the identifier of the station from which the train is delayed. The next field contains the train type, then the train number and finally the number of minutes of delay are specified.

Gathering the delays from the NS website can be done in intervals of 5, 10 or 15 minutes, because this generates a lot of request (about 83) to the NS webserver everytime.

After specifying the input, the output needs to be specified as shown in figure 8.6. Either the gathered delays can be send to the server to which the NS delay agent has been attached, or the delays can be written to a file, in which case they can be used for later simulations. After pressing the process button, the delays will be processed from the selected input to the selected output.



Please select where to send the delay information

(●) Send to delay agent at contacted platform

( ) Output to delay file

Filename: c:\delayoutput.txt

Figure 8.6 Selecting the output for the delay data

8.3 Personal agent interface

Chapter 7 already intorduced the personal agent interface, however the visualisation tab pane of the interface was not yet shown. Figure 8.7 shows a graphic representation of the route from Delft to Groningen Noord. The server backend consisted of 3 servers: one in Maastricht, one in Groningen and one in Amsterdam. The black line shows the route plan from the start to the destination. The magenta cross shows the current position of the user.

To determine the position of the user, a linear interpolation using the total travelling time, the current time and the positions of the start and end location is used, as follows:

Progress = (Currenttime – Starttime) / (Endtime-Starttime)
Currentposition = (1-Progress) x Startlocation + Progress x Endlocation

Of course, the best solution is to use real-time GPS data for this visualisation.

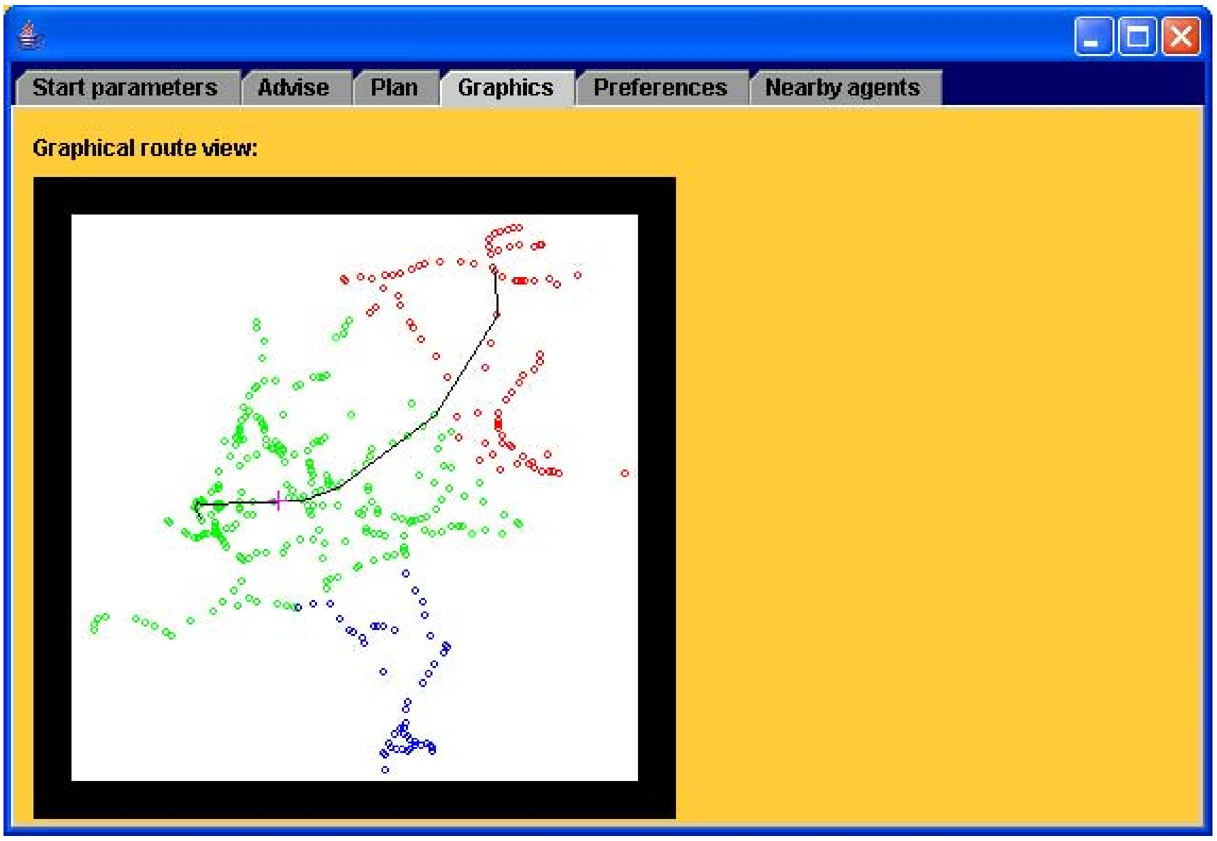
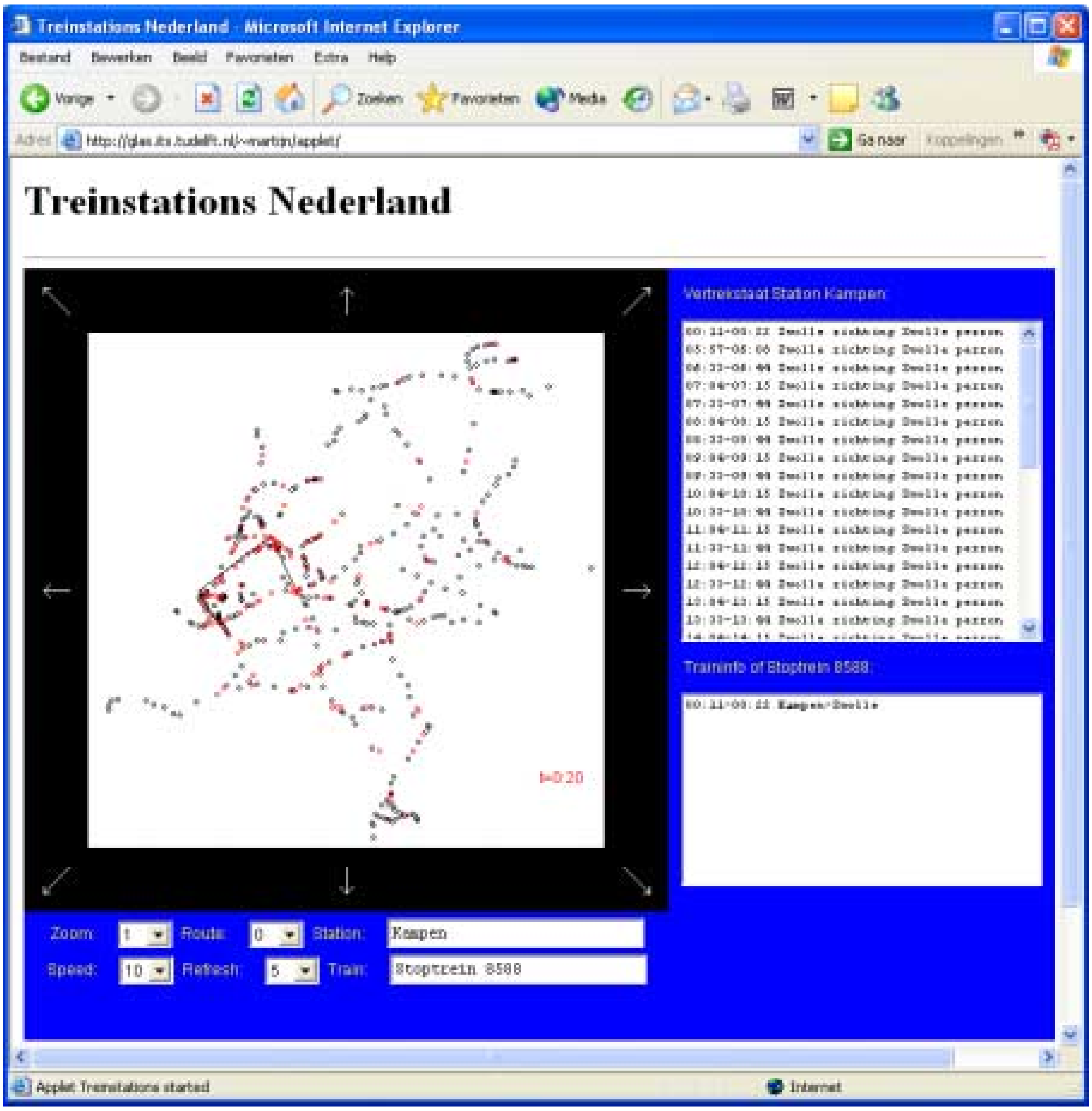


Figure 8.7 Visualisation of the route plan and current position

8.4 *Visualisation applet*

The last tool which was developed as part of this project is a visualisation applet. Figure 8.8 shows a screenshot of the applet running in a browser. Black pixels represent the trainstations, while red pixels represent trains. The applet has the possibility to zoom in on certain regions and to inspect the departure times at stations by clicking on them. Also individual trains can be clicked in order to show more information on the train. Using the comboboxes it is possible to speed up or slow down the simulation.

Obviously the colors of the visualisation applet do not match with the rest of the system. Because this applet was developed in order to test whether the information gathered from the NS website was representative for the situation on the Dutch railways and to find interesting situations or look up specific trains, this applet was developed at the start of the project. The goal was to integrate the visualisation into the rest of the system, but because of a lack of time this was unfortunately not completely realised.


Treinstations Nederland - Microsoft Internet Explorer
Bestand Bewerken Beeld Favorieten Extra Help
Vorige
Zoeken
Favorieten
Media
Adres http://glas.its.tudelft.nl/~martijn/applet/
Ga naar
Koppelingen
t=0:20
Applet Treinstations started
Internet

Treinstations Nederland

Vertrekstaat Station Kampen

```
00:11-00:22 Zwolle richting Zwolle perron
05:57-06:06 Zwolle richting Zwolle perron
06:33-06:44 Zwolle richting Zwolle perron
07:04-07:15 Zwolle richting Zwolle perron
07:33-07:44 Zwolle richting Zwolle perron
08:04-08:15 Zwolle richting Zwolle perron
08:33-08:44 Zwolle richting Zwolle perron
09:04-09:15 Zwolle richting Zwolle perron
09:33-09:44 Zwolle richting Zwolle perron
10:04-10:15 Zwolle richting Zwolle perron
10:33-10:44 Zwolle richting Zwolle perron
11:04-11:15 Zwolle richting Zwolle perron
11:33-11:44 Zwolle richting Zwolle perron
12:04-12:15 Zwolle richting Zwolle perron
12:33-12:44 Zwolle richting Zwolle perron
13:04-13:15 Zwolle richting Zwolle perron
13:33-13:44 Zwolle richting Zwolle perron
```

Traininfo of Stoptrein 8588

```
00:11-00:22 Kampen-Zwolle
```

Zoom: 1 Route: 0 Station: Kampen

Speed: 10 Refresh: 5 Train: Stoptrein 8588

Figure 8.8 A screenshot of the visualisation applet

Chapter 9: Conclusions and recommendations

This chapter presents the conlusions of the project which was started with the goals as described in chapter 1. Finally the recommendations and suggenstions for future developments are presented.

9.1 Conclusions

This section follows the structure of the goals presented in chapter 1 to discuss the realisation of each of the goals individually.

9.1.1 Literature survey

Concerning the literature survey, the current agent platforms are described in chapter 2. For this project COUGAAR was not used, but the combination of the distributed blackboard model of COUGAAR with the stable JADE platform looks promising. The choice for JADE combined with Jadex, giving the possibility to design and implement BDI agents, has proven to be a good choice, which has eventually realised the working prototype. Mainly because of the available userbase of these tools, the problems occuring from this platform were minimal or were solved after some email converstations (also with TU Delft specialists).

9.1.2 Design of a distributed system

An important part of the project was the design of a distributed system. Besides a design of a distributed backend with a suitable distribution model for the delay information, a personal agent was designed which can run on available handhelds. By combining ad-hoc networks with mobile Internet connections to aquire relevant information, the real power of the PITA system can be unleashed. By having autonomously functioning personal agents communicating with each other and the backend system, the system can distribute its own data across the different available networks and store and share information between handheld devices. As suitable routing algorithm a slightly adapted version of the DYNET algorithm from Eduard Tulp [Tulp91] was used. Finally an UML model was designed which was used to implement a first prototype of the system.

9.1.3 Implementation of a first prototype

Based on the design a first prototype was developed in Java, based on Jade combined with Jadex. First the relevant information from the NS website was acquired. Next the distributed backend was implemented, which also included the designed distribution model for the train delay data. Several BDI agents were developed of which the personal agent plays a central role in communicating with the different parts of the system. The DYNET algorithm was implemented in Java, which provides the system with a customisable routing algorithm, which fits the user centric PITA system very well. A lot of time of the project is used for implementation of the prototype. The implemented personal agent can run on the Sharp SL-C860 Zaurus handheld as well as on a desktop system.

9.1.4 Design and implement a PITA simulation environment

At the end of the project, a simulation environment is realised which gives the possibility to gather real time delay information from the NS website and store this in files, which can be used as input for simulations. Also the delay situation on each of the distributed servers can be inspected and it is possible to manually add or change delays using the server management interface. Besides this, a visualisation applet has been realised giving an overview over the situation on the dutch railways. However, because of a lack of time the visualisation was not fully integrated into the rest of the system.

9.1.5 Perform PITA test experiments

Throughout the implementation fase continually tests were performed to test the functioning of the different components of the system. As it is very hard to implement (and debug) a distributed agent application this was more than necessary in order to incrementally build the system. In chapter 7 a user scenario was described which shows the functioning of several different important aspects of the system.

9.2 Recommendations and future developments

This section describes the recommendations and possibilities for future developments, which were not researched or implemented as part of this project.

9.2.1 User interface

The currently provided interfaces are prototype interfaces only, showing which functionality is possible. However for large scale use of the PITA system, it is necessary to further research and develop a multi modal userinterface, which should be thouroughly tested on users as to whether the usability is acceptable. Therefore a speech recognition interface will have to be developed which runs on a handheld device, an iconic interface has to be designed and developed and research has to be done into the dialogue which the user has to have with the system. Easy ways of specifying a location and destination such as using a combination of clicking and speech recognition look very promising. Also further development has to be put into the automatic detection of the user position and how to translate this into delay information.

Currently the PITA application also runs stand-alone. Integration of the application with planning software or meeting schedulers would provide the user with a complete solution for his scheduling problems. This way the user is relieved of scheduling appointments, calculating travelling times, planning and replanning of routes, informing others of experienced delays etc., which really relieves him of a lot of problems. Also integration with route guidance software for cars can be interesting, in order to develop a system like KRIS of Gerritjan Eggenkamp [Egge01], which is able to switch between public transportation and car transportation to reach the destination as fast as possible. Finally the visualisation interface as developed as part of this project, has to be fully integrated into the current PITA system, to optimally use the system for simulation purposes.

9.2.2 Delay distribution

Another interesting notion of the thesis of Gerritjan Eggenkamp is the use of an expertsystem in order to reroute travellers. The expertsystem approach could also be used in the PITA system for the distribution of delay data. If an expertsystem which knows which delays affect which routes and is able to generate alternatives for this route is designed and implemented, the current delay distribution system can be extended and personal agents can be informed of delays and rerouting information at the same time, giving a nice solution which requires only minimal bandwidth usage.

The scenario of the distributed handheld application, as described in section 5.2.6, is also very interesting to research. As bandwidth for mobile Internet connections becomes cheaper, this scenario is realisable. However it is very important to design a model for communication and distribution in the fully distributed situation in order to have a functioning system. Otherwise there are chances that users might not accept the experimental technology. Another way to minimise the costs of the application for the customer is to use compression techniques to minimise the bandwidth usage over the mobile internet connection.

9.2.3 The route algorithm

As described, two route algorithms were used in this project. The DYNET algorithm shows very good performance, also because it can be adapted to the user preferences. However a lot of research is done in this field, so all sorts of optimisations can be applied to the current implementation in order to speed it up or improve the found routes, for instance by using heuristics. The current implementation in Java is probably not the best solution in case a lot of users would be using the system. Probably a natively compiled algorithm, running on a separate server is necessary in order to provide a large user base with optimal routes within an acceptable time.

The current implementation uses a penalty for train changes in order to find the best route, however a system which determines the overall risk of a certain route can be developed. Very tight connections give a high risk value to a route, while trains that realise a certain connection 99,9% of the time, can be calculated as a certainty. This could be an interesting extention for the algorithm in order to further maximise the travelling comfort for the user.

9.2.4 New technologies

As described in chapter 2, the combination of COUGAAR with JADE looks very promising. This could provide the PITA system with a distributed blackboard on which delay and route information is shared across ad-hoc networks. Ad-hoc networks should be integrated into the personal agent, to detect nearby agents autmotically. Also as handhelds are becoming almost as powerful as desktop systems, it should be possible to run speech recognition software on the handheld, enabling a full speech interface, for instance using Sphinx from Carnegie Mellon University. Also other voices (for instance of a Dutch female) could be compiled for the Festival Lite speech synthesis engine. Finally the replication of personal agents over multiple systems could be investigated, in order to be able to restore a personal agent after a handheld has crashed.

Bibliography

[BCPA02] F. Bergenti, G. Caire, R. Pels, C. van Aart, *Creating and using ontologies in agent communication*, in: EXP, volume 2, number 3, Telecom Italia Lab, Turin, Italy, September 2002, pages 110-121. See also gaper.swi.psy.uva.nl/beangenerator.

[BCPR03] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa, *JADE – A white paper*, in: EXP - In Search of Innovation (Special Issue on JADE), volume 3, number 3, Telecom Italia Lab, Turin, Italy, September 2003, pages 6-19. See also jade.cselt.it.

[CACTUS] The Context Aware Communication, Terminal and User project at TU Delft, see www.cactus.tudelft.nl.

[Combined] The Combined systems project of DECIS, see www.decis.nl.

[Dovl03] A. Dovlecel, *PITA, Web Service perspective*, TU Delft, january 9, 2003.

[Eclipse] The Eclipse IDE and Visual Editor Project, see www.eclipse.org and www.eclipse.org/vep.

[Egge01] G. Eggenkamp, *Dynamic Multi Modal Route Planning, An Artificial Intelligence Approach*, MSc thesis, TU Delft, July 2001.

[FIPA23] Foundation for Intelligent Physical Agents, *FIPA Agent Management Specification*, Geneva, Switzerland, 1996-2002. See also www.fipa.org.

[FIPA61] Foundation for Intelligent Physical Agents, *FIPA ACL Message Structure Specification*, Geneva, Switzerland, 1996-2002. See also www.fipa.org.

[FIPA-OS] FIPA-OS, see www.emorphia.com/research/about.htm.

[KLT03] K. Kleinmann, R. Lazarus, R. Tomlinson, *An Infrastructure for Adaptive Control of Multi-Agent Systems*, IEEE KIMAS'03 Conference Paper, october 2003. See also www.cougaar.org.

[LEAP] Lightweight Extensible Agent Platform, see leap.crm-paris.com.

[MCM99] D.L. Martin, A.J. Cheyer, D.B. Moran, *The Open Agent Architecture: A Framework for Building Distributed Software Systems*, Applied Artificial Intelligence journal, volume 13, number 1-2, pages 91-128. See also www.ai.sri.com/~oaa.

[PBL03] A. Pokahr, L. Braubach, W. Lamersdorf, *Jadex: Implementing a BDI -Infrastructure for JADE Agents*, in: EXP - In Search of Innovation (Special Issue on JADE), Vol 3, Nr. 3, Telecom Italia Lab, Turin, Italy, September 2003, pages 76-85. See also vsis-www.informatik.uni-hamburg.de/projects/jadex.

[Protégé] Protégé-2000 ontology and knowledge base editor, see protege.stanford.edu.

[RN95] S. J. Russell, P. Norvig, *Artificial Intelligence, A Modern Approach*, Prentice Hall, International, Inc., 1995.

[SHLPSZ98] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, V. Zue, *GALAXY-II: A reference architecture for conversational system development*, proceedings ICSLP 98, Sydney, Australia, November 1998. See also www.sls.csail.mit.edu/sls/technologies/galaxy.shtml and communicator.sourceforge.net.

[TRAIL] The Netherlands Research School for TRAnsport, Infrastructure and Logistics, see www.trail.tudelft.nl.

[Tryllian] Tryllian Agent Development Kit. See www.tryllian.com.

[Tulp91] E. Tulp, *Searching Time-table Networks*, PhD thesis, Vrije Universiteit van Amsterdam, Drukkerij Elinkwijk BV, Utrecht, oktober 31, 1991.

[Vark] R.J. van Vark, PhD thesis, to be published.

[Yild00] E. Yildiz, *DTGS, een ontwerp voor een dynamisch reisbegeleidingssysteem*, BSc thesis, TH Rijswijk/TU Delft, oktober 14, 2000.

Appendix

First a description of the functionality of handhelds necessary to realise the full PITA functionality is given, followed by a description of the source code developed as part of this project. Finally the source code of the C wrapper for Festival Lite is included.

Appendix A: Handheld requirements

In order to be able to use all functions as described in chapter 4, the handheld devices have to be able to fulfill several requirements. This section lists the requirements for a PITA handheld device, capable of all functions as described in chapter 4:

- **Wireless ad-hoc networking**
  The ad-hoc Wireless networking functionality has to be available either built-in or by installing extra software to make ad-hoc connections with nearby devices. Currently at TU Delft research is done in this area.
- **UMTS / GPRS or other mobile Internet connection**
  Some connection to the Internet is necessary to update the information (for instance on delays). An always-on connection is preferred, having acceptable prices for bandwidth usage. Currently these types of connections are available but the costs are high.
- **GPS or other positioning functionality**
  Some positioning functionality has to be built-in in order to locate the traveller. Another solution would be to have a separate Bluetooth device for this function. Important is the possibility to connect to the Internet and the positioning device at the same time. Some connection managers do not allow this. GPS receivers cost about 100 euro currently and are also built into handheld devices.
- **Supporting Java with Bluetooth or Location API**
  The PITA application is developed in Java, which requires a Java runtime to be available for the handheld platform. Currently specifications for a Java Bluetooth and Location API are finalised and these API's have to be supported by the handheld in order to be able to determine the position of the traveller from the application.
- **Powerful enough to run a Java routing application and speech interface**
  A well known disadvantage of Java is its demand on processing power and available memory. The handheld device needs to be able to store the static train schedule in memory and have enough processing power to find the best route and interact with the user, for instance using a speech interface. With the never ending increase in processing power and memory, this requirement will be fulfilled in the near future as handhelds are becoming almost as powerful as desktop computers.

Currently smart-phones are available (for instance PalmOne Treo 600), having GPRS connections, but lacking Bluetooth and Wireless LAN connections and positioning functionality. Handhelds capable of Bluetooth and Wireless LAN (such as iPAQ 5550) do not support Java and the Java Bluetooth API. Currently the best handheld option is the Sharp SL-C860 Zaurus, which supports Wireless LAN and is capable of running (not to fast) the Java 2 Standard Edition runtime, however this device does not support Bluetooth or positioning functionality.

Appendix B: The Java source code

The next table shows the structure of the Java code and some statistics of the input data:

package pitasystem			
Class:	**lines:**	**source (kb):**	**compiled (kb):**
DelayAgentPlan	312	11	7
DelayInformationPlan	351	11	7
DelaySubscriptionPlan	143	5	4
DienstRegeling	335	11	7
ManagementAgentPlan	584	17	12
NSDelayAgentPlan	592	18	12
PersonalAgentPlan	743	22	16
RouteAgentPlan	339	12	9
ServerManagementPlan	662	23	13
TravelPlan	554	19	8
UpdatePositionPlan	298	12	5
	4913	161	100
package pitasystem.interfaces (used Eclipse)			
Class:	**lines:**	**source (kb):**	**compiled (kb):**
DelayFrame	795	18	15
drawPanel	305	10	5
ManagementFrame	1673	39	38
TravellerInterface	1311	31	34
	4084	98	92
package pitasystem.moore			
Class:	**lines:**	**source (kb):**	**compiled (kb):**
RouteFinder	273	8	4
RoutePossibility	81	2	3
SearchNode	364	12	6
	718	22	13
package pitasystem.onto (used Protégé 2000/beangenerator)			
Class:	**lines:**	**source (kb):**	**compiled (kb):**
AlternativeJourney	47	2	2
Delay	48	1	1
FindRoute	48	1	1
IsAnswer	26	1	0
Journey	48	2	2
OVRouteOntology	205	12	8
OVTime	325	6	3
Route	59	2	0
RouteMap	36	1	2
RouteResult	26	1	1
Station	59	2	1
Train	281	8	6
	1208	39	27

package pitasystem.search			
Class:	**lines:**	**source (kb):**	**compiled (kb):**
Frontier	77	2	2
Node	258	7	5
PartialPath	430	16	8
SearchParameters	79	2	1
TulpAlgorithm	469	19	9
	1313	46	25
package pitasystem.util			
Class:	**lines:**	**source (kb):**	**compiled (kb):**
OVConstants	30	1	0
OVFunctions	59	2	2
SysTime	74	2	2
	163	5	4
Totals:	**12399**	**371**	**261**
Data files:	**lines:**	**source (kb):**	
stations.xml	387	40	
dienstregeling.xml	41301	17250	
	41688	17290	

Appendix C: The C source code of the Festival Lite wrapper

```
// file: myvoice.c
// uses Festival lite: see http://www.speech.cs.cmu.edu/flite/
// author: Martijn Beelen
// free to use, no warranties
// function: opens a socket on the specified port, by outputting text to this port, the system
will use Festival Lite to output the speech
// limitation: a buffer of 1000 characters is used, please make sure to cut the text in parts
of less than 1000 characters on whole words,
//              in order to prevent cuts in the middle of a word (which will be done by this
code)

#include <stdio.h>
#include <unistd.h>
#include "flite.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <errno.h>

cst_voice *register_cmu_us_kal();

int readpacket(int sck, char *buf, int size)
{
	int eolnfound = 0;
	int nrread = 0;

	int sizeleft = size-1;
	int i;

	while ((eolnfound == 0) && (sizeleft > 0))
	{

		nrread = recv(sck,buf,sizeleft,0);
		sizeleft = sizeleft - nrread;
		if (nrread > 0)
		{
			i = 0;
			while (i < nrread)
			{
				if (buf[i] == '\n')
				{
					eolnfound = 1;
				}
				i++;
			}
			buf += nrread;
		} else {
			shutdown(sck,0);
			eolnfound = 1;
			return 0;
		}

	}
	buf[0] = '\0';
	return 1;
}

int parsemessage(char *buf, int size, cst_voice *v)
{
	int li = 0;
	int cnt = 0;
	int i = 0;
	while ((cnt < size) && (buf[cnt] != '\0'))
	{
		if (buf[cnt] == '\n')
		{
			buf[cnt] = '\0';
			flite_text_to_speech(&buf[li],v,"play");
			li = cnt + 1;
		}
		cnt++;
	}
```

```
        if (li > 0)
        {
                // li is nu de laatste index die nog moet
                // laatste stuk nu weer naar voren kopieren
                while ((li < size) && (buf[li] != '\0'))
                {
                        buf[i] = buf[li];
                        i++;
                        li++;
                }
                return i;
        } else {
                // no complete messages found, output current message
                flite_text_to_speech(buf,v,"play");
                buf[0] = '\0';
                return 0;
        }
}

int main(int argc, char **argv)
{
        cst_voice *v;
        int socketfd;
        int acceptfd;
        int err = 0;
        int retval = 0;
        char bf[1024];
        int k = 0;
        int one = 1;

        struct sockaddr_in listensock;
        struct sockaddr_in acceptsock;
        socklen_t accept_addr_len = sizeof(struct sockaddr_in);


        if (argc != 2)
        {
                fprintf(stderr,"Syntax: myvoice <portnr>");
        } else {
                flite_init();
                v = register_cmu_us_kal();
                if ((socketfd = socket(PF_INET,SOCK_STREAM,0)) < 0)
                {
                        fprintf(stderr,"no socket");
                        return -1;
                }
                listensock.sin_family = AF_INET;
                listensock.sin_port = htons(atoi(argv[1]));
                listensock.sin_addr.s_addr = INADDR_ANY;

                if (bind(socketfd,&listensock,sizeof(struct sockaddr_in)) < 0)
                {
                        char buf[10];
                        fprintf(stderr,"bind failed");
                        sprintf(buf,"%d",err);
                        fprintf(stderr,buf);
                        return -1;
                }
                if (listen(socketfd,8) < 0)
                {
                        fprintf(stderr,"Cannot listen");
                        return -1;
                }
                while (retval == 0)
                {
                if ((acceptfd = accept(socketfd,&acceptsock,&accept_addr_len)) < 0)
                {
                        fprintf(stderr,"Accept failed");
                        return -1;
                }
                fprintf(stderr,"Accept succesful\n");

                setsockopt(acceptfd,IPPROTO_TCP,TCP_NODELAY,(void *)&one,sizeof(one));

                while (retval == 0)
                {
                        if (readpacket(acceptfd,&bf[k],1000-k))
                        {
                                k = parsemessage(bf,1000,v);
                        } else {
                                retval = 1;
                        }
                }
                retval = 0;
                }
        }
        return 0;
}
```