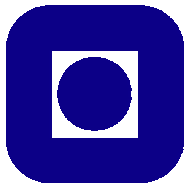
NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING



Developing mobile applications

The development of a fleet steering application with mobility support

Project assignment

Audun Simonsen

Autumn 2004

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING

PROJECT ASSIGNMENT

Student's name:		Audun Simonsen
Course:		TTM4700 Teleservices and Networks, Specialization
Title	:	**Developing mobile applications**
Text:		The goal is to design, implement and experiment with an application for the mobile user using the TAPAS platform. The focus will be on personal and terminal mobility. The project will comprise three phases. The first phase is the design of the service functionality, platform support and user interfaces. This phase will be based on UML diagram templates. The second phase is to implement the application itself and the third phase is the experimentation with the user environment, most preferably to conduct scenarios on different user mobility cases. The application chosen is an application for fleet steering of emergency vehicles. This application will make use of GPS to get the geographical location of the user and then report this to the emergency central in order to guide the vehicles efficiently. The central can also send new waypoints out to the vehicle and this will then be put in a car navigation system to guide the user to the desired position. In addition to this there shall be possible to create different user profiles, e.g. to let the users send and receive data files like pictures. For test purposes, the application will be implemented on a handheld PDA connected to a WLAN network. The central unit will be represented by a regular PC using Windows as platform.
Deadline:		November 29, 2004
Handed in:		
Carried out at:		Department of Telematics
Supervisor:		Mazen Malek Shiaa

Trondheim, ……2004

Finn Arve Aagesen
Professor

Preface

This report is the result of my project at the Norwegian University of Technology and Science (NTNU) in Trondheim during the autumn semester 2004, and documents the project assignment "Developing mobile applications". The work on this project was carried out at the Department of Telematics and has been a part of the TAPAS research project that is carried out there.

The project has been very challenging, partly because of the complexity of TAPAS and partly because of the development process involved in this project. Before this project my knowledge of TAPAS and on adaptable networking in general was practically non-existent, and I'm sure of that the knowledge and experience that I have gained will be of much value later on in life.

It has been a challenge to work with technologies that are relatively young and that are constantly undergoing further development. During the short period of time that this project has been going on a lot have changed, even during the last few days before handing in this report.

The supervisor for this project has been Mazen Malek Shiaa and the academic responsible has been professor Finn Arve Aagesen. I would like to thank them both for help and suggestions during this project and especially Mazen who has been available for answering my questions at all times. I would also like to thank the IT department at the department of Telematics for helping out with technical issues that have arisen and to my fellow students at the TAPAS lab for support during this project.

Trondheim, 29th of November 2004

Audun Simonsen

Contents

Figures

Tables

Abbreviations

API	Application Programming Interface
EGNOS	European Geostationary Navigation Overlay Service
ESA	European Space Agency
FAA	Federal Aviation Administration
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
ICT	Information and Communication Technology
IDE	Integrated Development Environment
J2ME	Java 2 Platform, Micro Edition
JNI	Java Native Interface
JVM	Java Virtual Machine
MicroTAPAS	Implementation of TAPAS for small handheld devices
NAVSTAR	NAVigation Satellite Timing And Ranging, official U.S. Department of Defence name for GPS
NMEA 0183	National Marine Electronics Association's standards for data communication between marine instruments (as used between a GPS and Autopilot, for example)
OS	Operating System
PDA	Personal Digital Assistant (electronic handheld information device)
PPC	Pocket PC, Microsoft OS for PDA
RI	Reference Implementation, a prototype for a JSR
SDK	Software Development Kit
TAPAS	Telematics Architecture for Play-based Adaptable Systems
JSR	Java Specification Request
TETRA	Terrestrial Trunked Radio, open standard by ETSI
TETRAPOL	Emergency communication from EADS Telecom
US DoD	United States Department of Defence
WLAN	Wireless Local Area Network

Abstract

The complexity of telecommunication systems is rapidly increasing, and the ongoing convergences in the ICT sector is opening up for new and innovative possibilities for services. These possibilities are also setting new demands to the services and applications developed with a view to development speed, robustness, wearability and so on. Basically, the users want to have the usability from the computer world combined with the performance of the telecommunication world. And preferably all integrated in one unit or gadget.

There are major challenges in handling this evolution, and an approach to these challenges is the Telematics Architecture for Play-based Adaptable Systems (TAPAS) project at the department of Telematics at NTNU. Several TAPAS architectures are developed, including a mobility architecture that supports personal mobility, terminal mobility and program or application mobility. This last type of mobility is in the TAPAS context referred to as role figure mobility.

The goal of this project was to develop, implement and experiment with an application for the mobile user using TAPAS and its mobility architecture. This was to be done with focus on personal and terminal mobility. For this purpose an implementation of TAPAS called MicroTAPAS was chosen, because this is designed for small, portable devices like Personal Digital Assistants (PDA) in a wireless environment. MicroTAPAS is implemented with Java technology.

The project comprised basically of three phases. The first phase was to design the service functionality, platform support and user interfaces. For this phase predefined UML diagram templates was used for support. The second phase was to implement the application itself and this was done be using standard developing techniques for implementing Java code. The third phase was to experiment with the user environment, and to try to conduct different scenarios with the use of the developed application.

The result of this project is an application for fleet steering called Mobile Interactive Navigation Tool. This application, or service, has proven that TAPAS is very much usable for developing complex systems with short time-to-market demands, but there are still issues that need to be addressed and investigated further. In order to get the desired functionality and mobility required by Mint, the MicroTAPAS implementation had to be extended with support of user mobility. The extensions made can also be used for further enhancements of mobility functionality in MicroTAPAS.

1 Introduction

This report will try to give a thorough description on how Telematics Architecture for Play-based Adaptable Systems (TAPAS) and the mobility support of this architecture can be used in making applications with enhanced mobility support.

The first couple of sections of the report will present a brief introduction to the technology that was used in this project and is meant to give the reader an understanding of the underlying components that was used. The result of the project is presented later on in this report.

1.1 Background

Today's development in the Information and Communication Technology sector creates a demand for new services that needs to be rapidly developed. The time between idea and deployment should be as short as possible, but the services still have to be robust, secure and have the same quality of service as services in the "plain old" telephony systems. The services developed should also be highly adaptable to changes and evolution of the surroundings and networks to withstand time. For this purpose the Telematics Architecture for Plug-And-play Systems or what is now known as Telematics Architecture for Play-based Adaptable Systems, (TAPAS) has been proposed as a framework.

To test out functionality regarded to mobility in TAPAS, this project was proposed. The main goal of this project was thus to develop, implement and experiment with an application for the mobile user. The application chosen to be developed in this project was an application for fleet steering of emergency vehicles. This application makes use of satellite navigation to get the geographical location of the user and then report this to the emergency central in order to guide the vehicles efficiently.

The major technology used in this project is then of course the TAPAS architecture. This architecture can be very complex to understand and is of such importance that it is

included as an own section in this report (chapter 3). Other related technologies to this project and to TAPAS in general are presented in chapter 2.

A little knowledge of the emergency communication network in Norway may also be needed, and this is presented in the following section.

1.1.1 Emergency communication network in Norway

For several years there have been discussions in Norway to replace the existing open, analogue emergency communication networks with a new more modern, digital and closed network. In November 2004, the Norwegian government finally decided to propose for the parliament to establish a joint digital public safety radio network for the fire department, the police and the health services [19]. This network is proposed to be completed in 2009 and is estimated to cost about 440 million Euros.

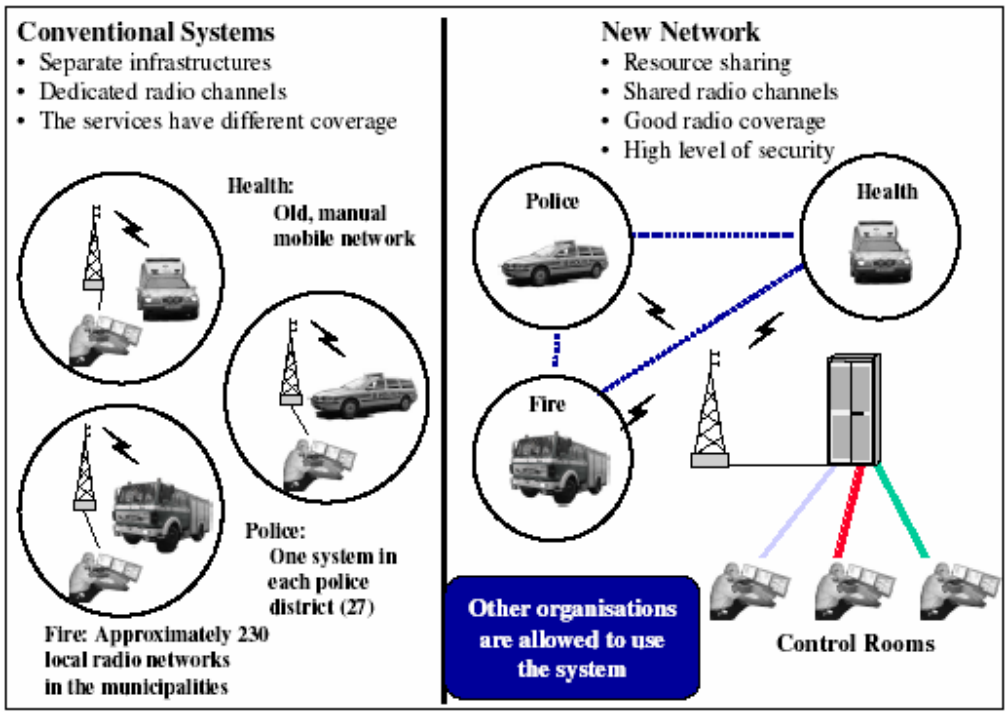


Figure 1: Changeover to shared public safety network [19]

Figure 1 shows the changeover from today's networks to the new network. It isn't chosen what kind of network technology to build, and during 2005 there will be a "beauty-contest" for developing this network. It is expected that there will be tenders with at least three different technologies and that seven or eight major companies will participate.

The governmental proposal gives guidance in the direction of TETRA or TETRAPOL technology, since these are recommended within Schengen, although the proposal principally is technology-neutral. The third alternative expected is a CDMA based service in the old NMT network (450 MHz), called CDMA 450. There are also other possibilities, like to make a hybrid network based on the mentioned networks or GSM/UMTS. The Norwegian newspaper "Teleavisen" [OL14] wrote on the 5th of November 2004 a comment on the public safety radio network. According to this comment, whatever network that is chosen, it will be outdated at the time of initiation. The major drawback will be the lack of possibilities to transfer data like images or video.

1.2 Approach

The goal for this project was in the first place to try to exploit the different kinds of mobility supported by the TAPAS architecture. Although there have been developed several different applications using TAPAS over the last couple of years, none of those have made use of all the types of mobility and aspects that is included in TAPAS and in the TAPAS mobility framework.

The task emphasizes on the different mobility aspects of the TAPAS mobility architecture and how this is carried out in the MicroTAPAS implementation. This required a study of the basic concepts of TAPAS as well as the support for mobility, and some effort was spent on studying this.

Related technologies that were used in the project were studied to get a more complete understanding of how the different aspects of the application that was going to be developed could be fulfilled. Technologies related to TAPAS was also studied in order to get a better understanding of what TAPAS is and what is tried to accomplish with it.

The next step was then to create models and scenarios of how the application was supposed to perform, as well as defining the requirements for the application, before the implementation process could begin. The implementation phase was approached with

code generation of the created models, and these were used as guidance for further implementation of functionality.

The last step was then to test the developed application and try to experiment with the user environment and conduct scenarios with different user mobility cases.

1.3 Demarcations

This report has no intentions of giving the reader a full introduction to the world of TAPAS and all the aspects of this architecture. The focus will be on the mobility architecture of TAPAS, and other aspects will only be briefly discussed if discussed at all. Focus will also be on one of the current implementations of TAPAS, MicroTAPAS that is designed for wireless PDAs.

The application made in this project is only made for testing purposes and certain parts of this application has not been prioritized to implement. This goes especially for the server or central side of the application where there is only implemented a text-based interface for performing the different aspects of the service.

The reader should be familiar with basic processes of developing applications and with the Java programming language in general. Some basic knowledge of the concepts of telecommunication systems can also be an advantage.

1.4 Structure of this report

Chapter 2, Related technologies: This chapter gives an overview of technologies related to this project and to TAPAS in general.

Chapter 3, The TAPAS Architecture: This chapter gives an introduction to the TAPAS architecture including TAPAS mobility architecture and a section about one implementation of TAPAS, called MicroTAPAS.

Chapter 4, The Mobile Interactive Navigation Tool (Mint): This chapter gives an introduction to the application developed during this project. Some scenarios are sketched out to give an impression of how Mint can be used.

Chapter 5, Design of Mint: This chapter gives a more detailed description of how the application was designed, and also includes some UML diagrams and XML file. Appendix E: CD will have more details on these diagrams.

Chapter 6, Implementation of Mint: This chapter describes the implementations process and implementation issues that occurred during the project.

Chapter 7, Testing and experimentation: This chapter describes some of the tests that were performed in order to test functionality. This is both the given functionality of the implementation of TAPAS and functionality of the Mint application.

Chapter 8, Discussion: This chapter tries to wrap up the report with an overview of faced challenges, an evaluation of the work and suggestions for further work.

Chapter 9, Conclusion: This chapter concludes the report and summarizes the project.

2 Related technologies

This chapter will give a brief introduction of some technologies that is used in this project and some that are related to TAPAS in general. Some of the technologies mentioned here are also given a more thorough introduction in appendixes. Section 2.3 and 2.4 tries to give the reader an overall view of the field that TAPAS is in, and will not go in details on specific technologies.

2.1 Satellite navigation

Mankind has always tried to figure out a way to find out where they are, and how to guide them to a destination and back again. Seafarers have used stars for navigation for as long as humans remember and when the Sputnik rocket was launched in 1957, it became clear that these "artificial stars" could also be used for navigation. The Americans figured out that they could locate Sputnik quite precisely just by tracking the radio signals that Sputnik sent. Then they figured out that this could be used the other way around, to get an exact position on the ground using radio signals from satellites. This resulted in what was known as the TRANSIT system, which was used by US submarines in the early 1960s. The TRANSIT system was not very accurate and had its flaws (e.g. the receiver could not move in order to get the location) and even though they tried to upgrade the system it was simply not precise enough. So in the early 1970s, the U.S. Department of Defense launched a program called Navstar Global Positioning System. The first satellite was launched in 1978 and in the mid-1990s the system was fully operative with 24 satellites. Although it took over 20 years to establish the system, the potential of the system was clear quite early on. Professor Bradford W. Parkinson wrote in 1980: "*With the quiet revolution of NAVSTAR, it can be seen that these potential uses are limited only by our imaginations*." [18]

2.1.1 Global Positioning System

The Navstar Global Positioning System is more commonly known just as GPS. GPS is, as stated above, controlled and operated by the U.S. Department of Defense and has therefore several issues is discussed in Appendix C: Satellite Navigation Systems. GPS

consists of three different segments; the space segment, a control segment and a user segment as described in Figure 2.

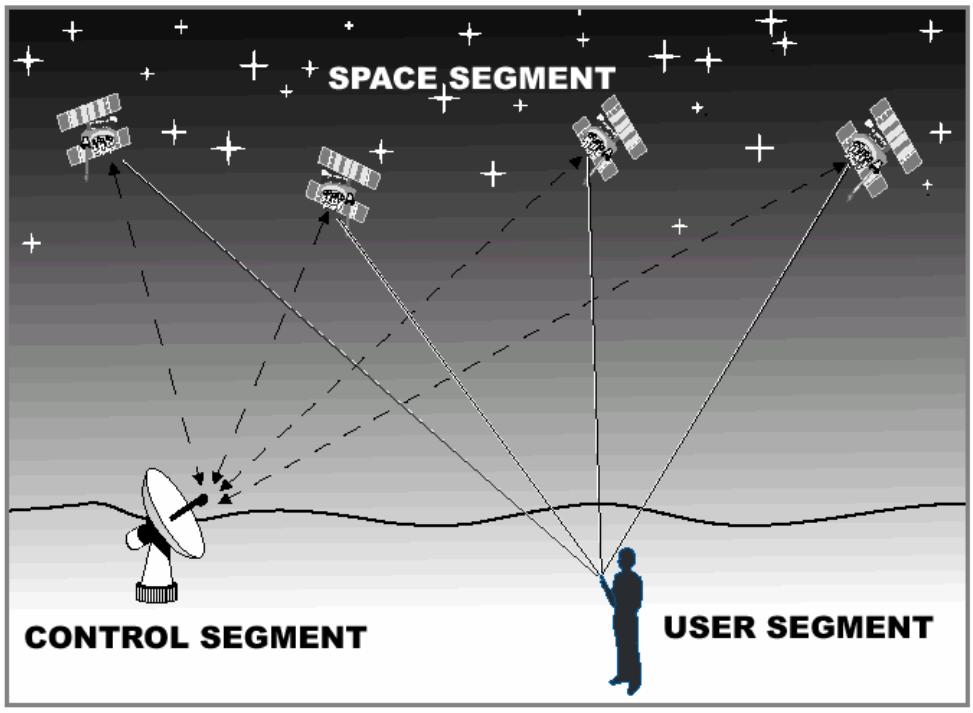


Figure 2: The GPS segments [3]

The space segment consists of 24 satellites, where 21 of them are active at all time and 3 are reserve in case of failure. These satellites transmit low power signals that allow any user with a GPS receiver in the user segment to determine their location on the surface of earth. The control segment is used to control and correct the data from the satellites. The signal from the satellites will pass through clouds, glass and plastic, but not through more solid objects like buildings and rocks.

There are also other systems used for satellite navigation, and these plus a more thorough description of GPS are given in Appendix C: Satellite Navigation Systems.

2.1.2 Car Navigation System

The increased use of satellite navigation systems has also entered the car industry. It is getting more and more common to buy a designated navigation system to be used with the car. In addition to telling the user the current location, car navigation systems also tell the user where to drive. The systems make use of roadmaps and extra info about the

roads (i.e. speed limits, one way traffic, etc.) and can therefore tell the user the best way to get to a desired destination.

Several designated systems are on the market today, often integrated with the car stereo or on more high tech cars, the cars multimedia platform. These systems are then mounted inside the car and can thus not easily be moved to another car or vehicle. Another solution is to use a PDA connected to a GPS receiver. Then the user can freely move the PDA to another vehicle or use it as a pedestrian. There are several programs that can be used on the PDA, and the most popular ones are OziExplorer [OL7] and TomTom Navigator [OL8].

2.2 J2ME

Java 2 Micro Edition (J2ME) is a version of Java for running Java code on devices with limited capabilities. Although J2ME is a quite new technology, some say that this is a return to the very origins of the Java technology. Java was originally developed for programming consumer electronic devices, but has evolved to being primarily used for desktop and server-based applications [OL9].

J2ME is divided into two different configurations; *Connected Limited Device Configuration* (CLDC), for devices with very little system resources like cell phones, and *Connected Device Configuration* (CDC) for devices that have a bit more resources like PDA or set-top boxes. The CDC configuration has focus on achieving as much compatibility as possible with Java 2 Standard Edition (J2SE) as possible, and this implies that CDC can be used to develop applications pretty much the same way as with J2SE. On the other hand, the CLDC configuration focuses on limiting the resource usage as much as possible and the code is very limited compared to J2ME.

Appendix D: J2ME gives a thorough introduction of J2ME and the process of developing new Java technologies.

2.3 Ubiquitous computing

"*Ubiquitous computing enhances computer use by making many computers available throughout the physical environment, while making them effectively invisible to the user*" [21]. Ubiquitous computing is called the third wave of computing, in the sense that it follows mainframe computing, that involved many people on one computer, and personal computing with one person on one computer. Ubiquitous means thus that there are one person and many computers, and is thought of being the post-PC world and computing for the 21st Century [20].

Ubiquitous computing is an approach to human-computer interaction, and is about distributing computation in the environment, as opposed to keeping it bottled in a desktop-bound personal computer. Ubiquitous computing can be seen to be roughly the opposite of virtual reality. In stead of putting people inside a computer generated world, ubiquitous computing forces the computers to live out here in the world with people. This omnipresent computing needs integration of human factors, computer science, engineering and social sciences.

In other words, ubiquitous computing means that the computer as we know it today disappears into everyday objects all around us with a natural interface. This will thus support everyday activity without the users actually thinking about that there are computers doing the work. This is no longer science fiction, and more and more devices are produced with the aim of being ubiquitous. The thought of hundreds of computers surrounding everyday life can also be frightening, and brigs up thoughts from George Orwell's book "1984". Several issues with protection of personal privacy have to be solved before the vision of ubiquitous computing can be fully implemented.

Xerox Palo Alto Research Centre (PARC) had a project from 1988 to 1994 that initially came up with the phrase 'ubiquitous computing' [21].

2.4 Adaptable networking

The phenomenal growth of today's information technology networks has led to exceptional levels of complexity. This growing complexity is leading to problems, and dealing with these problems "… *is the single most important challenge facing the IT industry*" [OL18]. The network growth, and thus the complexity, is rapidly outgrowing the growth of people that can manage the networks, and this calls for the need of networks that are adaptable or self-configuring. The main challenge is thus not to keep pace with Moore's Law any more, but to deal with the consequences of its decades-long reign.

Imagine a network that is capable of recovering from faults and actually 'learn' from the failure before any user notices that the network has been inaccessible. Or that Windows after installation actually will improve performance over time, without any human interaction! "*In the evolution of humans and of human society, automation has always been the foundation for progress*" [OL18] and this will most probably apply for moving forward in the ICT sector too.

Adaptable networking means thus that the network-based services are capable of handling dynamic changes. These changes can be in both time and position related to resources, users and changes in service requirements [1]. Adaptability basically implies flexibility. This flexibility can be related to software, services, configuration of resources and interoperability with different kinds of architectures and with issues of mobility.

IBM has a project called autonomic computing [OL19], which approaches this problem with inspiration from the human body and the autonomic nervous system. Within the human body, the autonomic nervous system controls the basic functionality that keeps us humans going. This includes functions like controlling heart rate, breathing and making the body recover from "breakdowns" or diseases, but without using our consciousness.

3 The TAPAS Architecture

Telematics Architecture for Play-based Adaptable Systems (previously known as Telematics Architecture for Plug-and-Play Systems) (TAPAS) is a research project at the Department of Telematics at the Norwegian University of Science and Technology (NTNU). At the first international conference on information and communication technology (ICT) in 2003 the project was presented with the aim of "… developing an architecture for network-based service systems with

A: flexibility and adaptability,

B: robustness and survivability, and

C: QoS awareness and resource control.

The goal is to enhance the flexibility, efficiency and simplicity of system installation, deployment, operation, management and maintenance by enabling dynamic configuration of network components and network-based service functionality." [1]. The project is under constant development and the different perspectives of the project are dealt with in several PhD and Master thesis as well as in scientific researches and projects. Results and reports are published at the TAPAS website [OL1]. TAPAS is basically a concept for developing services, and it is not fully implemented. There are at the moment two different models of TAPAS implemented. One fully scaled version called TAPAS Core Platform, and one reduced version for devices with limited resources like Personal Digital Assistants (PDA) called MicroTAPAS.

3.1 Functional architecture

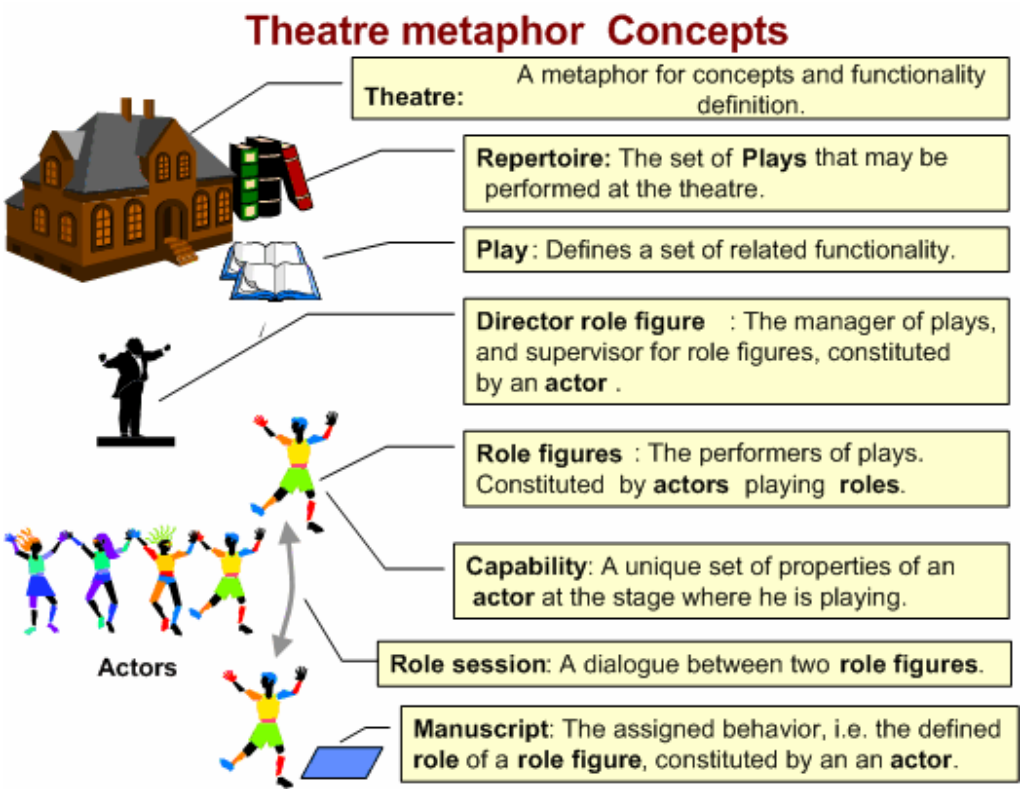


Figure 3: The TAPAS theatre metaphor [1].

The functional architecture of TAPAS is based on a theatre metaphor presented in Figure 3. This metaphor defines the functionality as plays with actors performing different roles. The actors are generic object that, dependant of its capability, can perform different role figures which are described in manuscripts. The different actors interact with each other in role sessions and a play is conducted by a director, which is an actor performing a specialized role for this purpose. One director supervising actors is also representing a domain. The repertoire is finally a collection of plays that can be performed at the theatre, which is a metaphor for the definition of concepts and functionality.

3.2 Basic architecture

The basic architecture covers the basis of TAPAS functionality. Figure 4 shows the object model of this basic architecture as explained in [1].

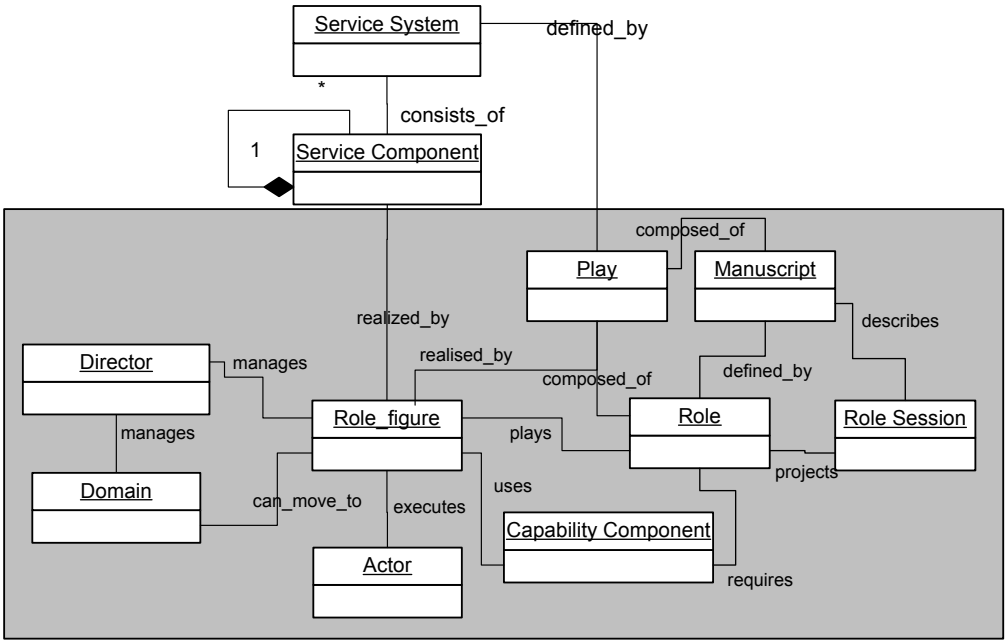


Figure 4: Object model of the TAPAS basic architecture [1]

Figure 4 shows an object model of the basic architecture of TAPAS. This architecture gives the basic support that is needed in TAPAS, and the mobility architecture that will be presented later on is based on this. The grey box is the play view of the architecture. The foundation in this model is the *Actor* component, which is a generic software component running on a node in the network. One Actor is modeled as an Extended Finite State Machine (EFSM) that can download a *Manuscript* that defines a *Role*, in order to play a certain *Role Figure*. This way the actor component can execute the different role figures. The Manuscript describes also the different *Role Sessions* in the play. The Role requires capabilities defined in the *Capability Component* in order to run and these capabilities have to correspond with the offered capabilities on the hosting node in order for the different Roles to be carried out. The *Director* manages the domain and the Role Figures that realizes the *Play*.

3.2.1 TAPAS layered model

From the functionality and support platform point of view, the TAPAS basic architecture consists of several layers. Figure 5 shows these layers and these are described further in [6] and [7].

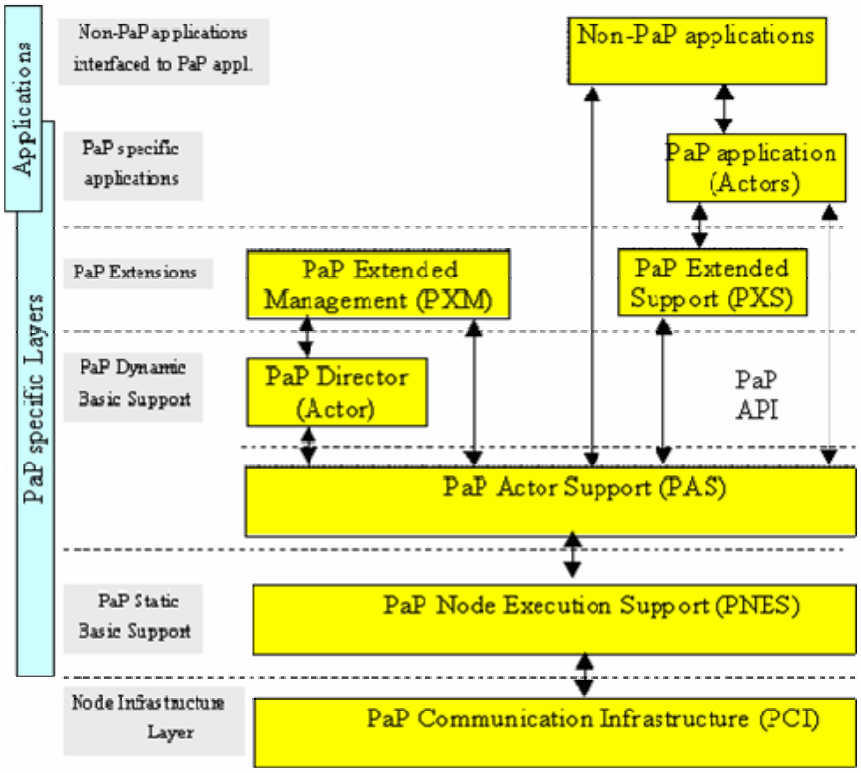


Figure 5: TAPAS layered model [6]

- **Plug and play Communication Infrastructure (PCI):** Lowest layer in the infrastructure. Constitutes of the Operating System (OS), network communication (e.g. TCP/IP) and a distributed system solution. Java RMI is used for current implementation of the TAPAS Core Platform.
- **Plug and play Node Execution Support (PNES):** Makes it possible to run TAPAS software on a node and to let actors on different nodes interact. A part of this layer is static in order to make the system boot on a node.
- **Plug and play Actor Support (PAS):** Makes it possible to create and execute actors within an operating system process (or thread). Each PAS is has a separate Java Virtual Machine instance.
- **Director:** Manages the plays as described earlier
- **Plug and play Extended Management (PXM):** Support for extended services that is not required for TAPAS basic support, but for specified extensions related to operational quality.

- **Plug and play Extended Support (PXS):** Required for the applications to utilize PXM functionality.
- **Plug and play applications:** The collection of application actors.
- **Non Plug and play applications:** Other functionality that is not defined according to ApplicationActor requirements but allowed to communicate with actors and to utilize TAPAS support functionality

Figure 6 shows an example of a TAPAS system distributed over 4 nodes.

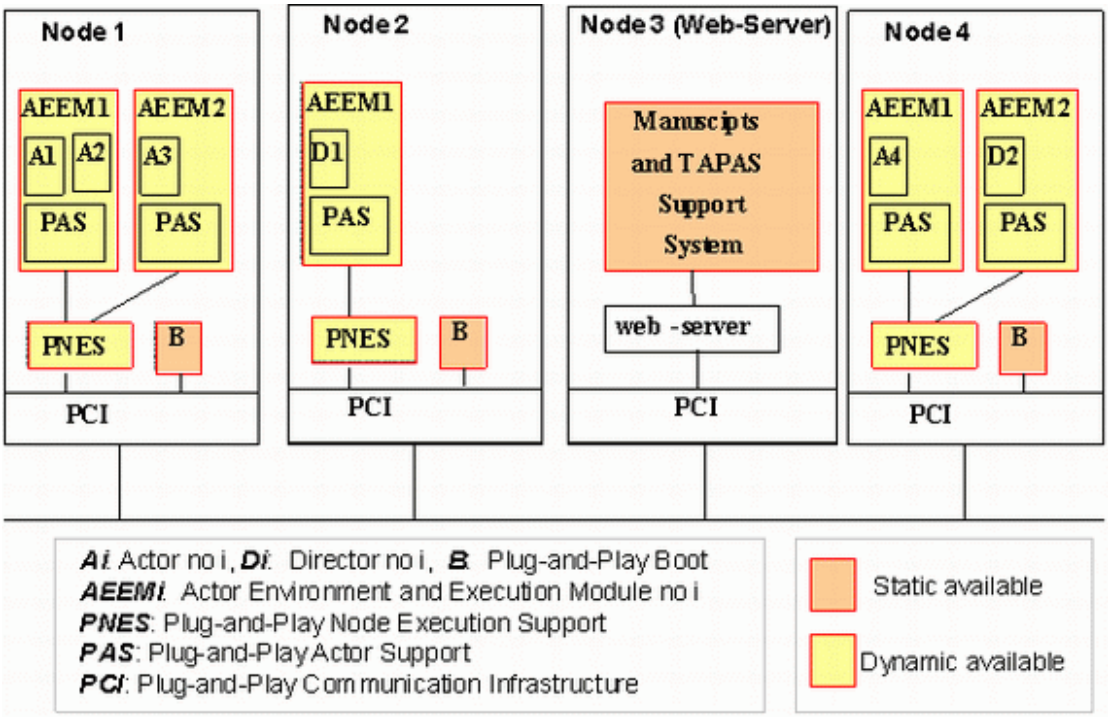


Figure 6: Example of TAPAS system [1]

The statically available bootstrap code on each node downloads the necessary code from the web server, and executes the PNES that is needed in order for the nodes to run TAPAS software and TAPAS-based services. AEEM corresponds here to a process or thread that executes a collection of actors with one associated PAS. The web server holds the manuscripts (e.g. the executable code for a service) and the code for the basic support system and these will be downloaded whenever needed.

Several defined procedures are introduced in order to give a basic set of functionality to TAPAS. These are *PlayPlugIn, PlayChangesPlugIn, PlayPlugOut, ActorPlugIn,*

ActorPlugOut, ActorBehaviourPlugIn, ActorChangeBehaviour, ActroBehaviourPlugOut, RoleSessionAction, ChangeActorCapabilities and *Subscribe*. The plug-in procedures are basically used for initializing the different dynamic components in Figure 6, and the plug-out procedures for removing the instances. For more details about these procedures, see [16].

3.3 The TAPAS Mobility Architecture

The TAPAS platform consists of several architectures in addition to the basic architecture; mobility architecture, dynamic configuration architecture, capability architecture and adaptive service architecture. These are meant to support and solve the different aspects of TAPAS described in chapter 3. For this project the mobility architecture was used in order to solve mobility issues.

Mobility is regarded as the most important feature to achieve adaptable and flexible systems [2]. This implies that the system then is capable of handling dynamic changes in availability of different kinds of resources and to handle users regardless of their physical location. The TAPAS mobility architecture is based on the basic architecture described in section 3.2, and adds components and functionality to enhance support for mobility issues.

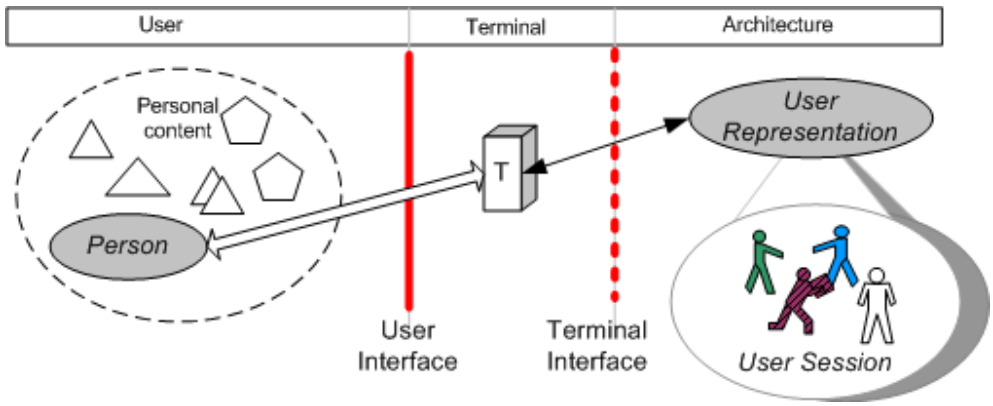


Figure 7: TAPAS Mobility Concept [2]

Figure 7 shows how mobility is handled in TAPAS, and gives a description of the terms used in TAPAS to describe mobility. The terminology framework and definitions are described in details in [2]. As a result of using two interfaces, a user interface and a terminal interface, TAPAS gets a flexible way of describing users and terminals independently. This concept is a quite normal way of distinguish users and terminals and is used commonly in systems that require user or personal mobility. By using this concept, there is room for the different kinds of mobility that is supported by the TAPAS mobility architecture; personal, terminal and role figure mobility. These types of mobility

are described briefly in section 3.3.1 to 3.3.3 later on in this report and in more detail in [5].

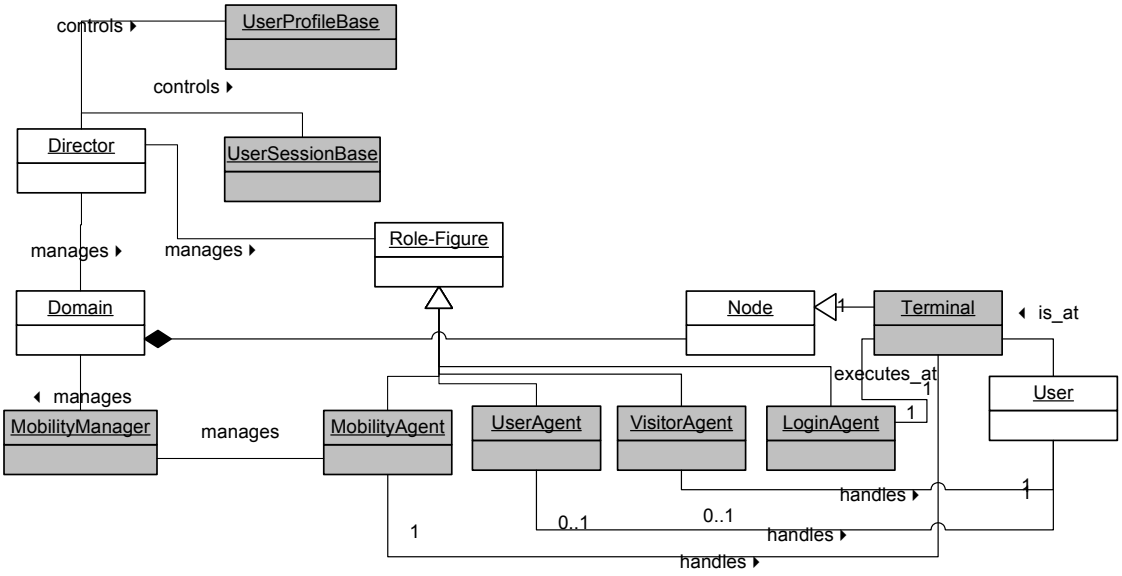


Figure 8: Object model of TAPAS mobility platform [2]

The TAPAS mobility platform can be described with an object model like Figure 8. This resembles very much on the basic architecture described in 3.2, but the main difference is the colored objects. The Director is now also in charge of handling user profiles and user sessions which are stored in *UserProfileBase* and *UserSessionBase* respectively. These two objects represent a knowledge base or data repository, and can be implemented as one or several databases or any other way of storing data. The current implementations of TAPAS use XML files to store the info.

A couple of new role figures are also introduced. *MobilityManager* is a role figure that manages the mobility in a domain. This manager can have contact with several *MobilityAgents*. *MobilityAgent* is controlling the movement and connectivity of a Node, and there are one MobilityAgent per Node in the net and one MobilityManager per Domain. The different MobilityAgents keeps contact with the MobilityManager, in order for this to known whether the Nodes are available, connected or have changed location (i.e. changed addressable location, e.g. IP address). The MobilityManager must thus be running at a location known to all other nodes in that domain.

The *LoginAgent* is functionality to support entry of a user to a domain. *UserAgent* and *VisitorAgent* is supporting the User in the user's home domain and visiting domain respectively. The home domain of a user is defined as the domain that holds the profile for that user. For more details on the objects, see [2].

Figure 9 shows the different parts of the mobility architecture within one domain with four nodes, two terminals and a web server. The users entering a domain are first met by a LoginAgent that handles authentication of the user based on the user profile and other preferences that are stored in the UserProfileBase. If the user is entering his or hers home domain (like on TerminalA), the user gets assigned a user agent that handles the interactions with the system, service requests and administration of the user profile and sessions. Is this not the home domain for the user (TerminalB), a visitor agent gets assigned instead to handle the interactions with this visiting domain as well as the users home domain.

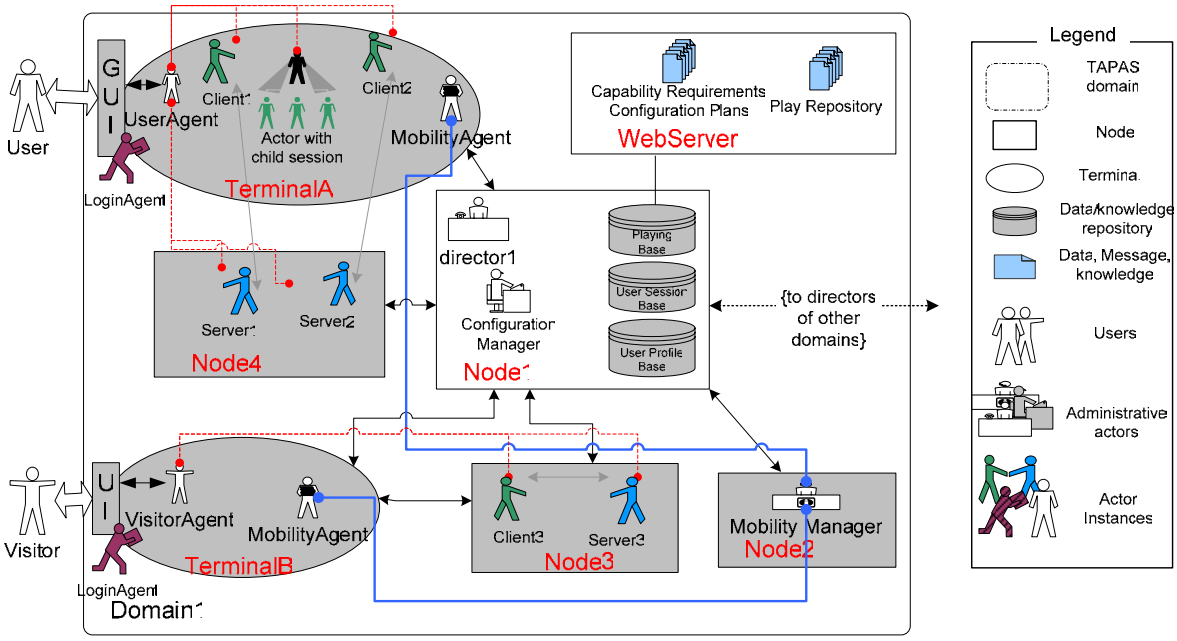


Figure 9: Engineering model of TAPAS mobility architecture [2]

The MobilityAgents running on the terminals keep contact with the MobilityManager in order to keep this updated with the status of the terminal. Based on the services that the user requests, service instances can be set up and plays and manuscripts be downloaded an executed. These are distributed on the terminals and nodes. The domain management,

which comprises the administrative actors (director1, Configuration Manager and Mobility Manager) supervises the whole process and handles requests from all the user entries that are within the domain, as well as requests from other domains. This includes the different mobility aspects that are supported by the domain. The management of services in TAPAS is based on the main concepts of TINA (Telecommunication Information Networking Architecture) service architecture [8].

3.3.1 Personal mobility

Personal mobility is in the TAPAS context described as functionality and support for giving an end-user orientation for the applications or services that is running on the TAPAS platform. This means that the user shall be able to personify the environment and that the sessions of the user at any time can be stored or suspended and at a later time be resumed again. All this is independent of physical location, terminal or equipment. In [2], personal mobility is separated into two kinds of personal mobility; user session mobility and user mobility.

3.3.1.1 User Session mobility

User session mobility describes the functionality for suspending and resuming a session, or to continue an executed service at a later time. The user's current session is regularly updated by the UserAgent and the session is maintained by the Director. This way the UserAgent can send a message to the Director that it shall suspend the session whenever needed (e.g. the user logs out). Figure 10 shows an example of user session mobility. UserA is first logged in at TerminalA and the UserAgent there handles the *update_session* requests to the Director (director1). The user has two services defined by Play1 and Play2, which involves Client1, Client2, Server1 and Server2. The red dotted lines indicates the connections in one user session, and note that Server2 is not maintained by this session, but by an other session.

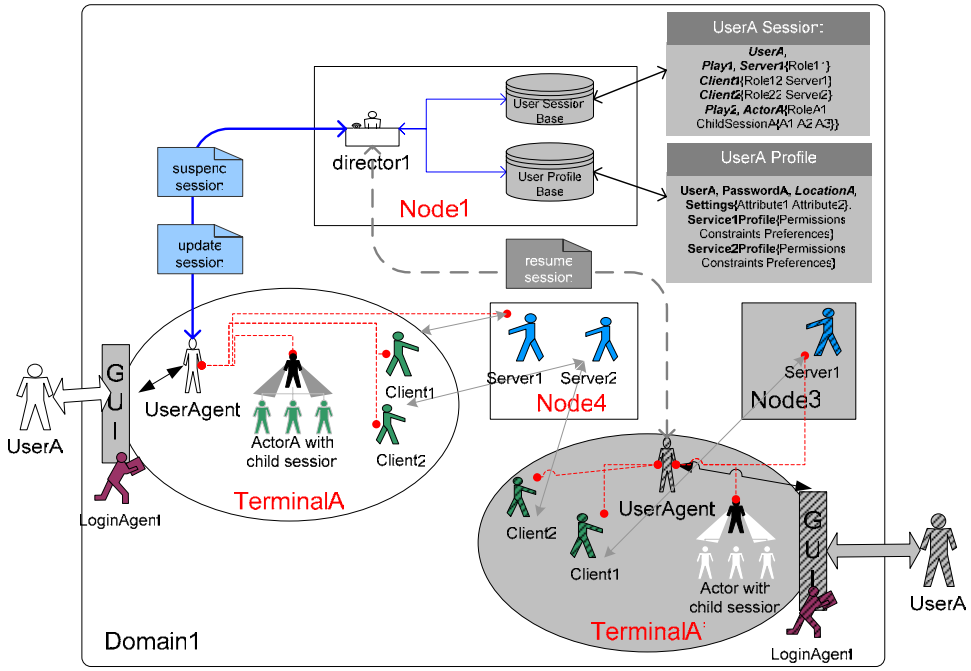


Figure 10: User Session Mobility [2]

The user suspends its session and moves to another terminal (TerminalA'). Here the user resumes the sessions via the new UserAgent and the ongoing services are resumed. Since Server2 was maintained by another user session, this instance was not terminated and therefore the connection is resumed. Client1, Client2 and Server1 instances as well as the actor with child sessions is instantiated in order to resume the sessions.

3.3.1.2 User mobility

User mobility is providing the user the ability to roam within different domains and still get access to its subscribed services. This is where user profiles and maintaining them comes in to play. The UserAgent or VisitorAgent is responsible for retrieving and updating the profile via the director in the home domain to the user.

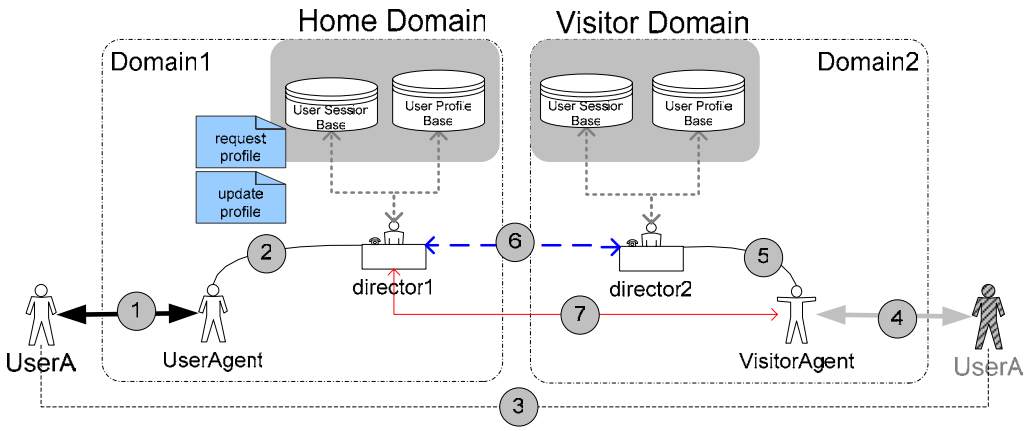


Figure 11: User mobility [2]

Figure 11 shows a scenario where UserA is logged in at the home domain for this user, Domain1. The UserAgent is capable of requesting the user profile and update the same profile with new info if necessary. It is the director of the domain, here called director1 that handles and updates the profile. When the user roams to another domain, Domain 2, the user logs on and gets a VisitorAgent assigned for this session since there is no user profile for this user in this domain. When the user wants to access the subscriptions in the home domain, the VisitorAgent contacts the director of the visiting domain, director2. This director makes contact with the director of the user's home domain, and these two negotiates the access and authenticates the user so that the VisitorAgent can access the user profile via the director in the home domain of the user.

Now the user has access to the same services as when the user is in his or hers home domain.

3.3.2 Role Figure mobility

Role Figure mobility in TAPAS means the capability of moving an already instantiated actor, i.e. to move running code. Role figures are, if we recollect from section 3.1, actors performing defined roles. Role Figure mobility can be performed between terminals, nodes or even between different domains. All the properties, capabilities and role sessions that the moving role figure has, have to be moved as well so that the complete movement is transparent for the rest of the system. This type of mobility is also known as service mobility or program mobility.

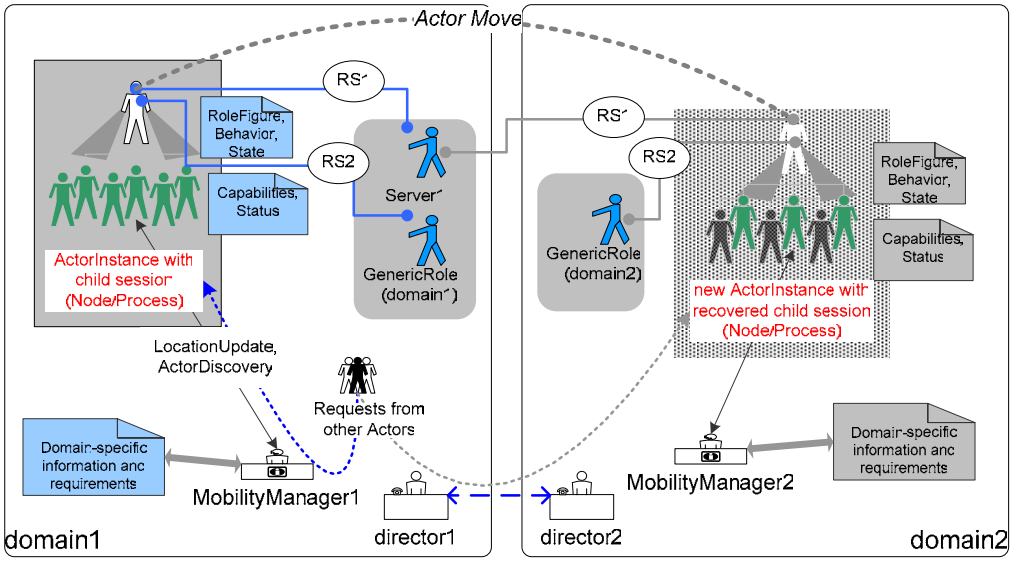


Figure 12: Role Figure mobility [2]

Figure 12 shows an example of role figure mobility where a role figure is moved between two different domains. The role figure has two ongoing role sessions, RS1, with a server called Server1, and RS2 with another role figure that is specific for domain1 (this can be a type of domain name server, or other domain specific role). When the role figure moves it notifies the MobilityManager of the domain, MobilityManager1, about its new location. When other actors or role figures send requests to the moved role figure (or any actor for that matter), they first check with the MobilityManager to get the location of the role figure.

When the role figure arrives at the new location, the behavior, capabilities and role sessions are recovered if possible. In Figure 12, the role session with Server1 was recovered and a new role session to a new GenericRole (that is domain specific for domain2) was created. Not all of the child sessions of the role figure could be recovered, and these are marked with black color in Figure 12. This may be because of different capabilities at the new location or that they are no longer relevant for the moved role figure.

There are several reasons to initiate movement of role figures. It can be because of changed capabilities at the hosting node or terminal, so that the role figure has to move in order to continue, or because of user input. An example can be that a user has an ongoing

service at his stationary terminal at his office. The user wants to continue this service after work and can thus move the role figure that is performing the service to the his handheld PDA or cellular phone.

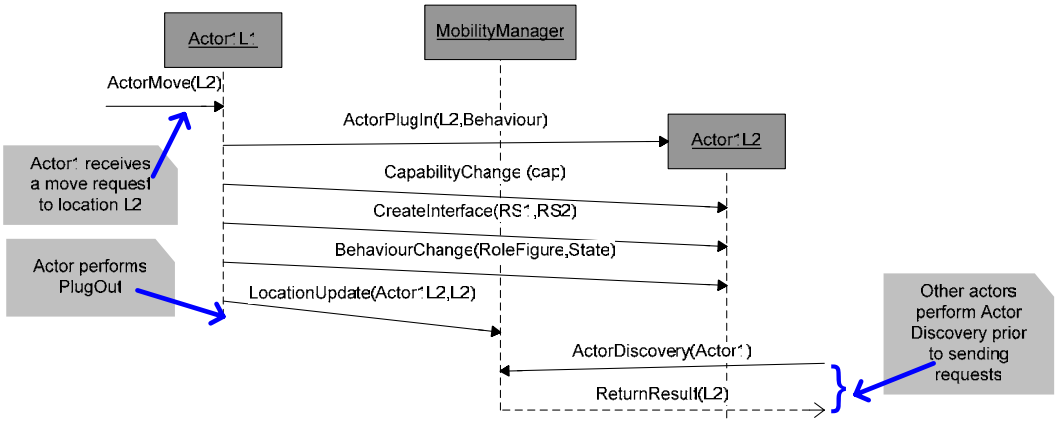


Figure 13: Sequence chart of Role Figure mobility [2]

The actual movement of an actor is possible with the use of several requests. Figure 13 show a possible sequence diagram for moving an actor. When the actor at location1 receives an ActorMove request, it tries to plug in a new actor at the new location before it transfers its capabilities, interface (where info about this actor can be stored) and the current behavior to the newly plugged in actor. When this is done, the actor notifies the MobilityManager with a *LocationUpdate* request before it terminates itself by performing a plug out request. As described earlier, other actors and role figures perform an ActorDiscovery procedure with the MobilityManager in order to get the location of the actor.

3.3.3 Terminal mobility

The last type of mobility described in TAPAS Mobility Architecture, is terminal mobility. This is when a terminal changes its location and/or connection point to the network, but still maintains the services that are running on the terminal. This includes that the terminal can loose connectivity with the network for some time and returns to the same connection and that the terminal is moved without loosing connectivity, but that the actual address to the terminal is changed. This can be done within the same domain (i.e. with the same director) or between domains.

Each terminal executes a MobilityAgent in order to keep track of the current location of this terminal as described in Figure 8 earlier on. Both the MobilityManager and the MobilityAgent must be able to discover changes in the connectivity, so that if a terminal goes out of coverage or looses connectivity in any other way it is picked up by the system. When a MobilityAgent discovers that it has changed location, it sends an LocationUpdate to the MobilityManager with the new location.

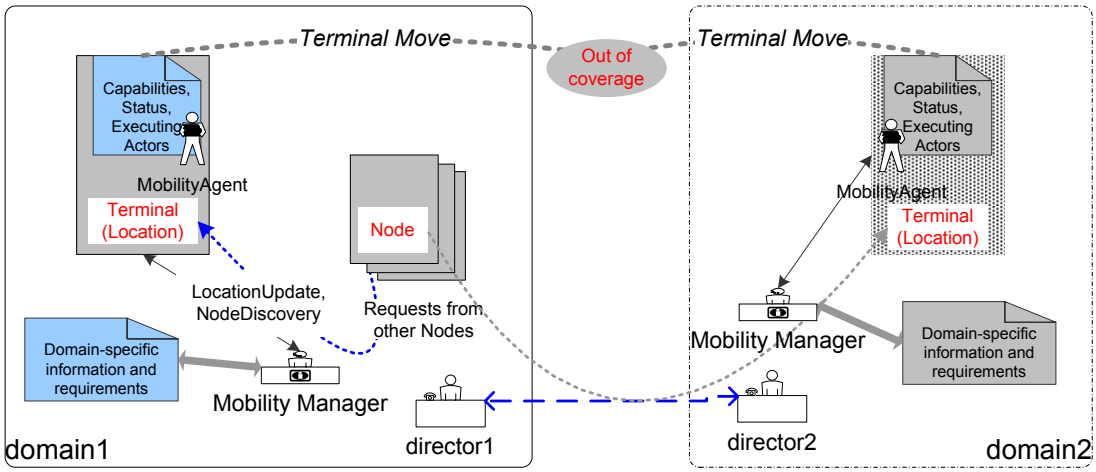


Figure 14: Terminal mobility [2]

Figure 14 shows an example of terminal mobility where the terminal moves from one domain to another (from domain1 to domain2) and in the time between is out of coverage for some time. The MobilityAgent keeps the MobilityManager updated until it reaches the boundary of its domain, and then they loose connectivity and the terminal is marked as inaccessible. When the terminal enters domain2, it may be allowed to access certain services that are negotiated between the directors like described with user mobility in section 3.3.1.2. The MoblityAgent will then execute a LocationUpdate to its home domains MobilityManager so that other nodes can reach it. The other nodes performs NodeDiscovery procedures (and ActorDiscovery like described in section 3.3.2) in order to contact the moved terminal and its actors.

3.4 MicroTAPAS

MicroTAPAS is an implementation of TAPAS for devices with limited resources, like PDAs. As all current implementations of TAPAS, MicroTAPAS is realized with the use of Java as programming language, but some parts of the code had to be made differently because of limitations in the targeted devices. MicroTAPAS was implemented by Eirik Lühr as a project at NTNU [4] and further developed with some mobility support in his master thesis [9]. Minor changes have been done by others later on, but this project is based on the work done in [4] and [9].

3.4.1 Differences to TAPAS Core platform

MicroTAPAS is implemented using Java 2 Micro Edition (J2ME) with the Connected Device Configuration (CDC), which is briefly described in chapter 1.1 and in Appendix D: J2ME. The TAPAS Core platform was implemented using Java 2 Standard Edition (J2SE) and made use of Java Remote Method Invocation (RMI) for communication. Java RMI is an optional package to the CDC profiles, and some features are removed in order to make it less demanding for the device. Some of these removed features was used in the TAPAS Core platform, and this implicated that RMI could not be used in MicroTAPAS. Socket communication was used instead for the communication between the different nodes.

In order to not use more system resources than necessary on the devices, it was decided to be a limitation of only running one instance of a virtual machine at any given time on a device. This opposed to the TAPAS Core platform where it is possible to run several instances at the same time.

Other major differences in this implementation are within the layered model. It was decided that there should be only one PaP Actor Support (PAS) (see section 3.2.1) instance for each node. This led to the decision of merging the PAS layer and PNES layer to one layer called MicroPNES, and the MicroTAPAS layered model would then look like Figure 15.

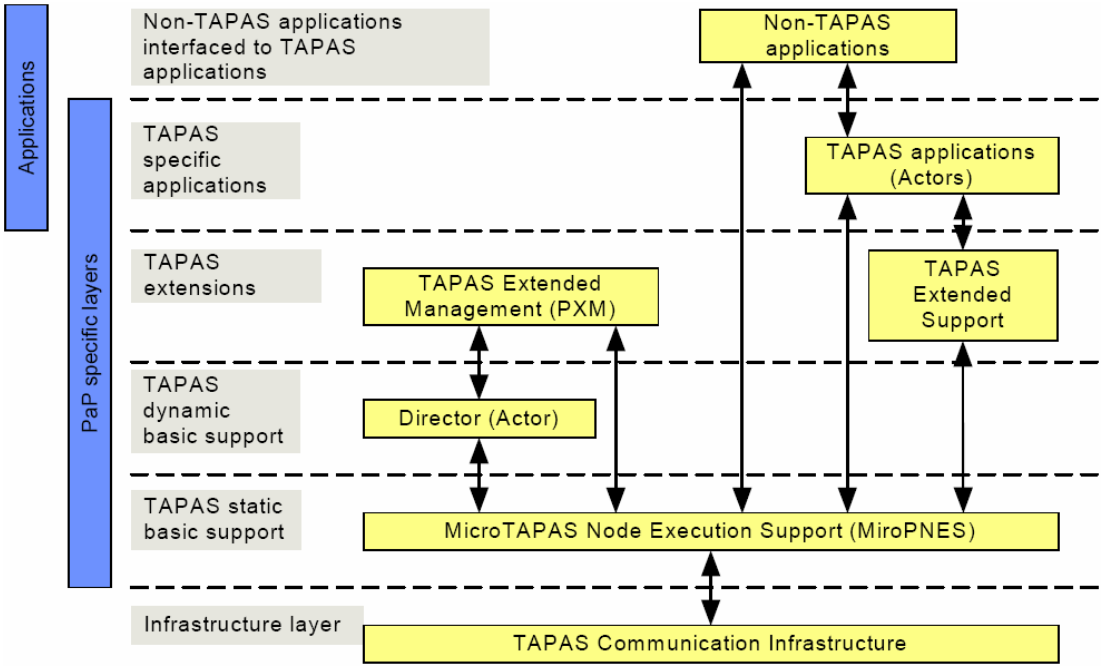


Figure 15: MicroTAPAS layered model [4]

A modified version of Figure 6 would then be like Figure 16. This figure also displays the use of mobility features like MobilityManager and MobilityAgents.

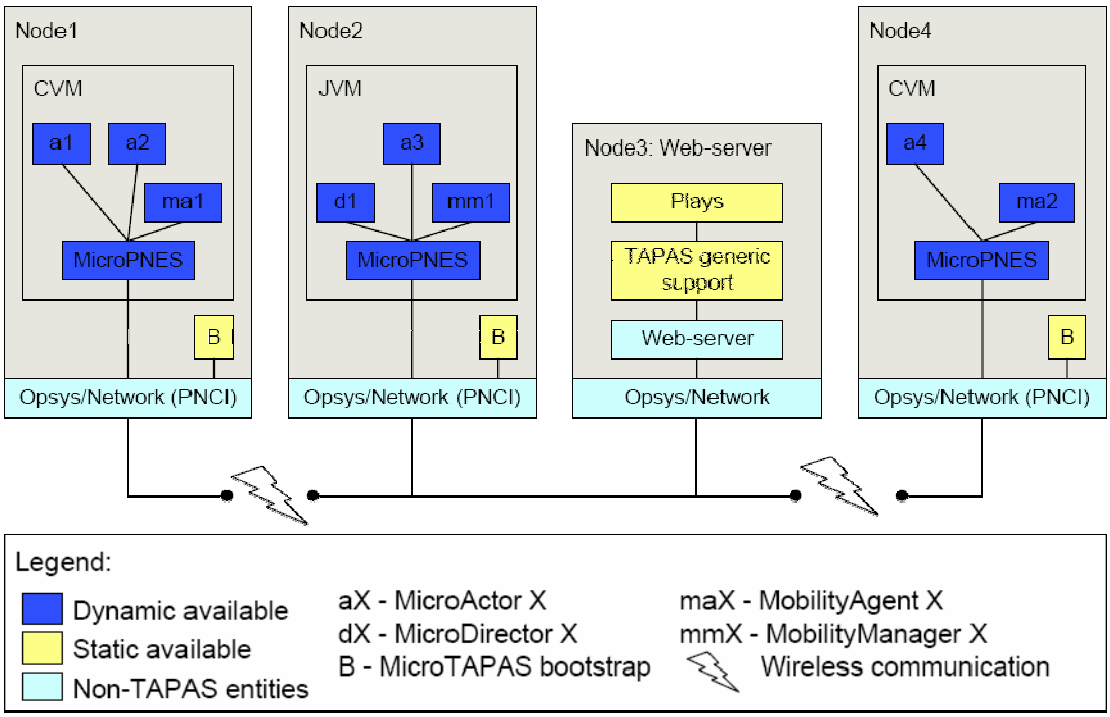


Figure 16: MicroTAPAS example system [9]

Node 1 and 4 can for example be PDAs running a CDC JVM, and these are connected to the network via wireless communication. Node 2 is here an ordinary computer running a normal JVM. Node 2 is also running the Director and the MobilityManager for this domain, and Node 1 and 4 is running a MobilityAgent each that is in contact with the MobilityManager.

3.4.2 Implementation issues

MicroTAPAS is implemented based on the use of PDAs as devices and WLAN as network technology. This determined how the mobility management mechanisms were worked out. These features were realized by using plain socket routines and pinging to check for connectivity. The implementation from [9] is supporting terminal mobility and role figure mobility, but no support for personal mobility is implemented. Tests of the existing mobility features were performed and some are explained in chapter 7. Actor mobility was tested successfully, but terminal mobility failed with the use of PDA.

In [13], Marius Dalsmo addresses an improvement of connectivity by solving a timeout problem with the java.net.Socket implementation. This code was also found with the UML specifications described in [14], but execution of this code resulted in an error and was not used further in the project.

4 The Mobile Interactive Navigation Tool (Mint)

This project's main task was to develop a mobile application using TAPAS and its mobility architecture. The development of this application comprised of three phases; design phase, implementation phase and at last a testing phase. This is the normal lifecycle for developing services or applications, and these phases are described in chapter 5 to 7.

The application chosen to be developed was an application for fleet steering of emergency vehicles, and was named Mobile Interactive Navigation Tool (Mint). The emergency services have the need for knowing the whereabouts of their emergency vehicles to be able to efficiently guide them. They also need an easy way to guide the vehicles to their destination, e.g. with the use of car navigation systems. The emergency services also have the need for transferring different types of documents between the users and the communication central to keep an updated situational report.

Although this application was developed with the emergency services in mind, there should be no problems for other services, e.g. transport or delivery services, to use this application as well.

It is important to notice that this application is meant to be a supplement to the communication system chosen by the emergency services, and functionality like voice transmission is not covered here at all. Issues like securing the transmissions and encryption isn't covered either, and must be handled by the chosen network.

Since the technology that is going to be used for the emergency network isn't decided yet, this project used PDAs as terminals and WLAN as the transport medium, but there should not be any big problems porting this to whatever type of terminal as long as it supports Java, J2ME and preferably CDC.

The implementation made is also primarily made for testing purposes, and improvements might be needed if this is going to be implemented in full scale. But the major functions

and functionality should easily be shown from this implementation. Some features are probably not implemented the most efficient way, e.g. the document sending part which is further described in section 6.2, but the main reason for this is to utilize most of the already implemented functionality in MicroTAPAS.

4.1 Scenarios

Scenarios to describe the use and need of the Mint service can be useful for understanding how the Mint application is supposed to work. The following scenarios were created early in the process, and have been extended and adjusted during the work on this project. This leads up to the design phase that is more thoroughly described in chapter 5.

4.1.1 Speeding scenario

The first scenario involves a speed check where a police officer tries to stop a car that is driving too fast. The officer reports that he or she now is involved in a car chase to the central. The central gets the geographical location of the officer and of other users, and can relocate the other users to intercept the speeding car.

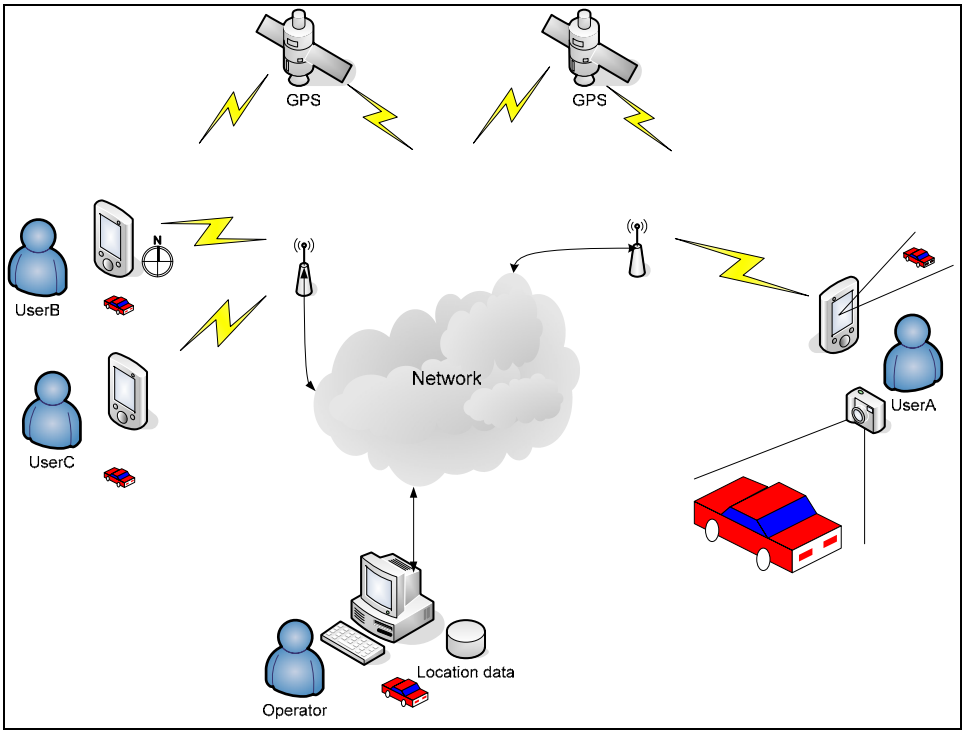


Figure 17: Speeding scenario

The scenario is also shown in Figure 17, where UserA is the patrolling officer. This figure also shows that if UserA has a camera and takes a snapshot of the car he can send this picture to the central, which can distribute it to the other users that is en route, here called UserB and UserC.

4.1.2 Busy user scenario

The next scenario is that the mobile user is busy and doesn't have time to update the situation with documents or pictures to the central. The central has thus the opportunity to get updated files, pictures or other documents from the mobile user's terminal. This can for example be in the case of a fire, where the firemen needs to concentrate on putting out the fire or any other type of scenario where the user can't prioritize to update the central unit.

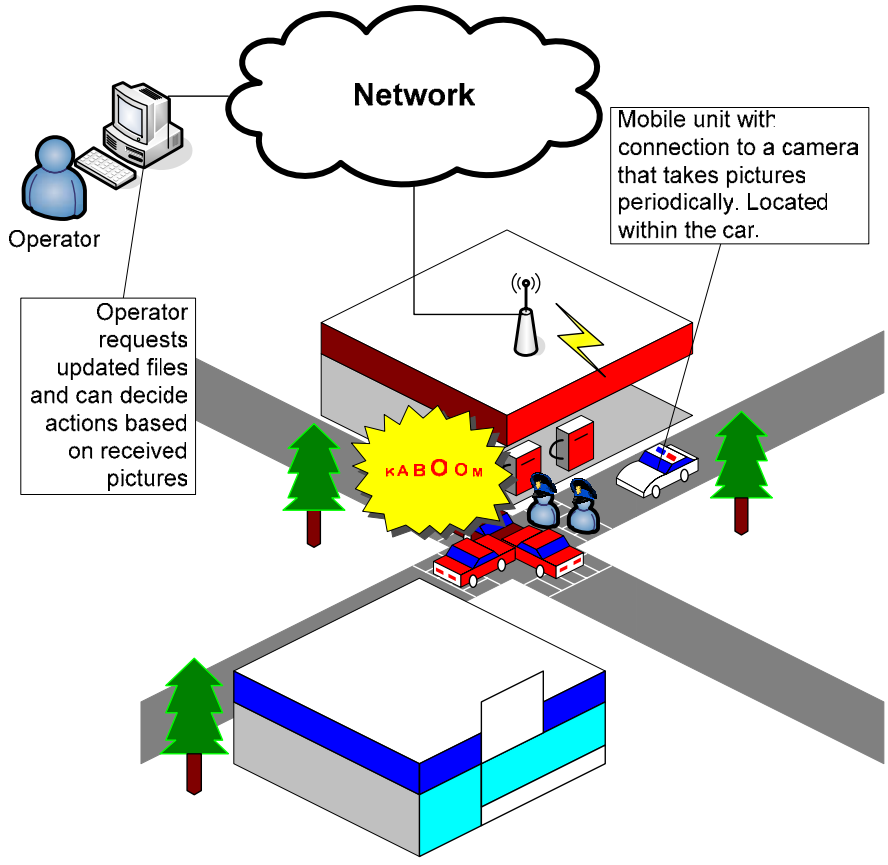


Figure 18: Accident scenario

Figure 18 shows a car accident where the two police officers at the scene are busy with handling the situation. In their patrol vehicle, they have mounted a camera that takes

pictures periodically. This camera is connected to a PDA running the Mint application, and saves the pictures at a given location on the PDA. When needed, the operator can request updated pictures, and can thus decide whether e.g. a fire truck or an ambulance is needed for assistance.

4.1.3 Disaster scenario

Disaster is here understood as a major accident, where several emergency units are involved. At this time there is no need for geographical location updates, and these updates will only use resources and bandwidth that might be needed for other purposes. Since the central already knows where the units are, it can tell the mobile units to not transmit location updates any more, or to change the frequency of the updates.

At a later time, if the users e.g. are moving away from the disaster area, the communication central might find it useful to start the geographical updates again or change the update frequency.

At a major disaster, like terrorist attack or train accident, the rescue work will be organized with some sort of local administration. At this time the local administrator have more need for updates on the situation, and the other users in the disaster area can thus send updated info like pictures, video and other information directly to the administrator or to someone else in the administration (e.g. there might be assigned a special communication officer for this task).

5 Design of Mint

The design of Mint was carried out with the use of the UML templates made in [14]. These templates were for some strange reason made using Rational Rose RealTime. This tool is a powerful, expensive and proprietary tool for developing real time applications in Java or C++. In addition to normal UML diagrams, Rational Rose RT contains functionality for using State Diagrams and other diagrams used for real time system development. It also includes functionality for code generation and to compile the code. None of these 'extra' functionalities can be used for development of TAPAS applications. The department of Telematics at NTNU does not have licenses for using this tool either, so this had to be borrowed from the department of Engineering Cybernetics. This department does not have license for the latest version of Rational Rose RT (now named IBM Rational Rose Technical Developer) and the version used was version 6.4 from 2002.

The system will consist of two different parts that are distributed over an unknown geographical area. One part is the mobile unit, which can be located in an emergency vehicle, and the other part is a central unit which works as the server and can be located within some sort of communication central.

5.1 Functional requirements

There will be different requirements for the mobile unit and the central unit, so these requirements are divided in to two sections.

5.1.1 Mobile unit

The mobile unit will typically be placed within an emergency vehicle. The main function of this unit will be to send its geographical location to the server, receive a destination position from the server and handle other features. This unit should either contain a car navigation system or be in touch with a map program to be able to show the desired route to the destination. The geographical info can be sent periodically or by initiation from either the central user or the mobile user.

Other features of this unit can be sending or distributing different kinds of data like pictures, video clips or written documents and reports. These can be sent to the central unit to give an overview of the situation of e.g. an accident. The central unit can then forward relevant material to expertise for help, to another emergency unit which is currently en route to the accident, to a hospital for preparing the personnel or similar cases. The mobile units may also have the need for exchanging data directly if required.

To get these kinds of services there has to be possible to log on with different user profiles. There could e.g. be a default/basic user profile that handles the position functions, a more advanced profile for sending/receiving data and another for local administration of an accident, which of course needs the have more or less the same info as the central. These profiles have to be configurable to fit in with different organizational structures.

5.1.2 Server/Central unit

The server unit will typically be co-located with the communication central. This unit is not mobile and should contain functionality for displaying the positions of the mobile units, sending coordinates or destination addresses to the mobile units, manage user profiles and handle data from the mobile units.

This unit shall be capable of requesting the geographical position of a mobile user at any time. The central user can also send a waypoint to the mobile user to get the mobile users car navigation system or other navigational tool to guide the mobile user to a destination.

The central user shall also be capable of requesting documents from the mobile user if needed. For example if the mobile user is busy with handling the situation at e.g. an accident, the central shall be capable of getting updated documents like pictures from the mobile user without the mobile users interaction.

5.2 Non-functional requirements

Some non-functional requirements are also set for the Mint application.

- The user profiles shall be easy to maintain in order to change settings for a user.

- The central must be able to control the periodic transmission of GPS locations from the mobile users. This can be if the central knows that the mobile user is at a known accident and the central decides to save some bandwidth for other more important traffic.
- The user interface of the mobile unit has to be intuitive and easy to use, preferably with working gloves and a fireman's outfit.

Figure 19 shows a Use Case diagram for the wanted functionality of the Mint application.

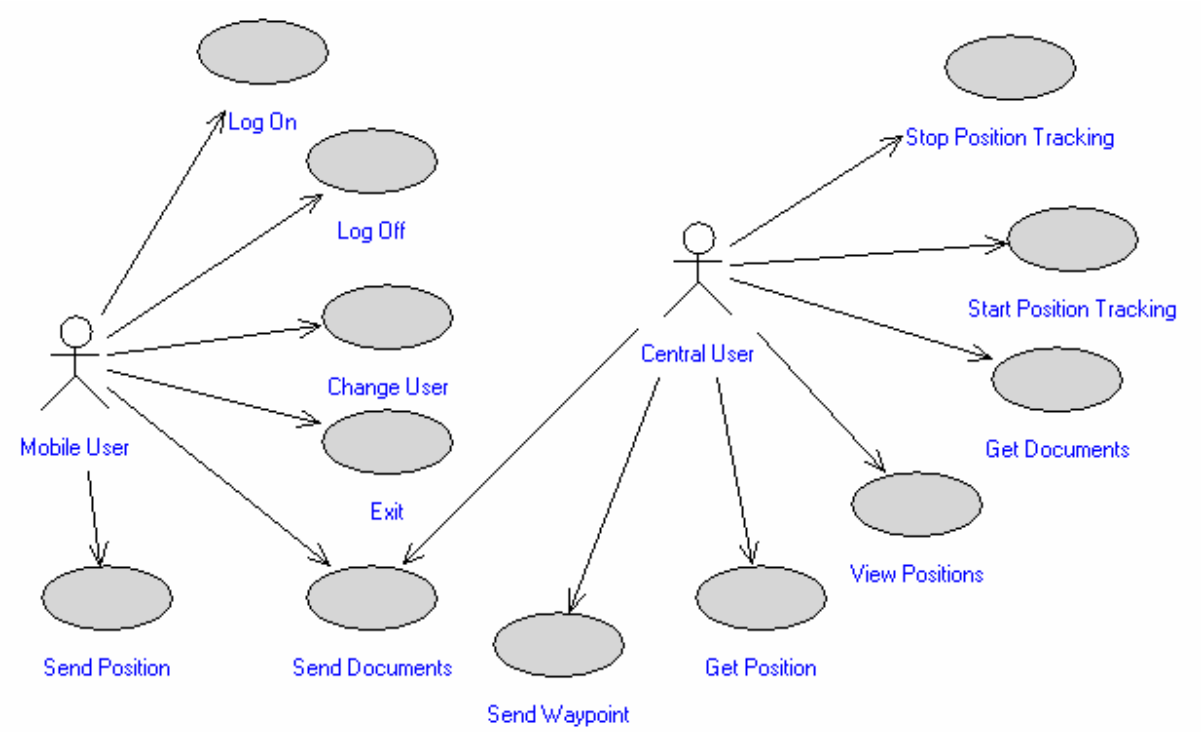


Figure 19: Use Case for Mint

The *Mobile User* represents here a user with a mobile unit and the *Central User* is within the communication central.

5.3 Class diagrams

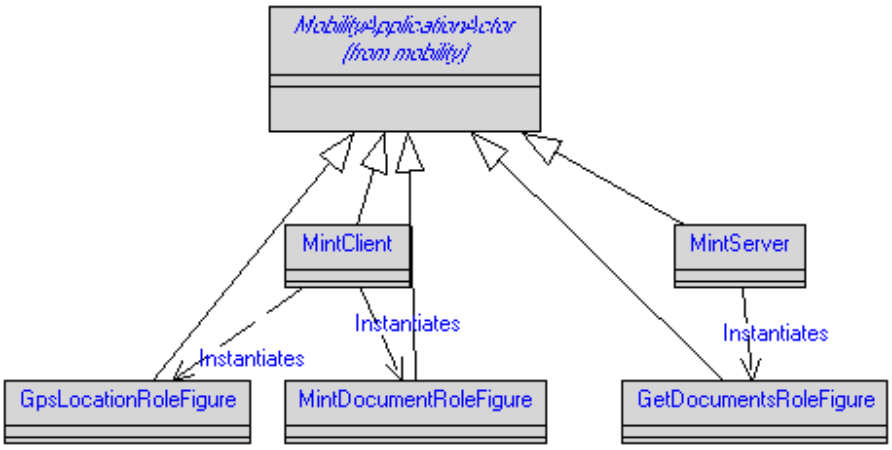


Figure 20: Class diagram of Mint

Figure 20 shows a class diagram for Mint. The mobile unit is represented by three classes or role figures, *MintClient*, *GpsLocationRoleFigure* and *MintDocumentRoleFigure*, and the central unit is represented by two, *MintServer* and *GetDocumentsRoleFigure*. All role figures are extended from MobilityApplicationActor, which is in the MicroTAPAS.mobility package.

The MintClient role figure is the basic starting point for the application at the mobile unit. This role figure takes care of the login process for the user and has the basic functionality for Mint. According to the response of the login request to the director, this role figure can instantiate GpsLocationRoleFigure and/or MintDocumentRoleFigure.

MintServer is the role figure that handles server functionality, and is located within the communication central. This role figure will most probably be running on the same node as the director and the MicroTAPAS domain management. MintServer can instantiate GetDocumentsRoleFigure in order to get documents from a client's terminal. This role figure will actually be instantiated at the client's terminal, but will request an actorMove procedure in order to move back to the server with the documents requested.

Figure 21 shows a more detailed class diagram for the MintClient role figure. This role figure has two windows for Graphical User Interface (GUI), one logon window to get the

users name and password from the user and one window for handling the rest of the functionality for the Mint application.

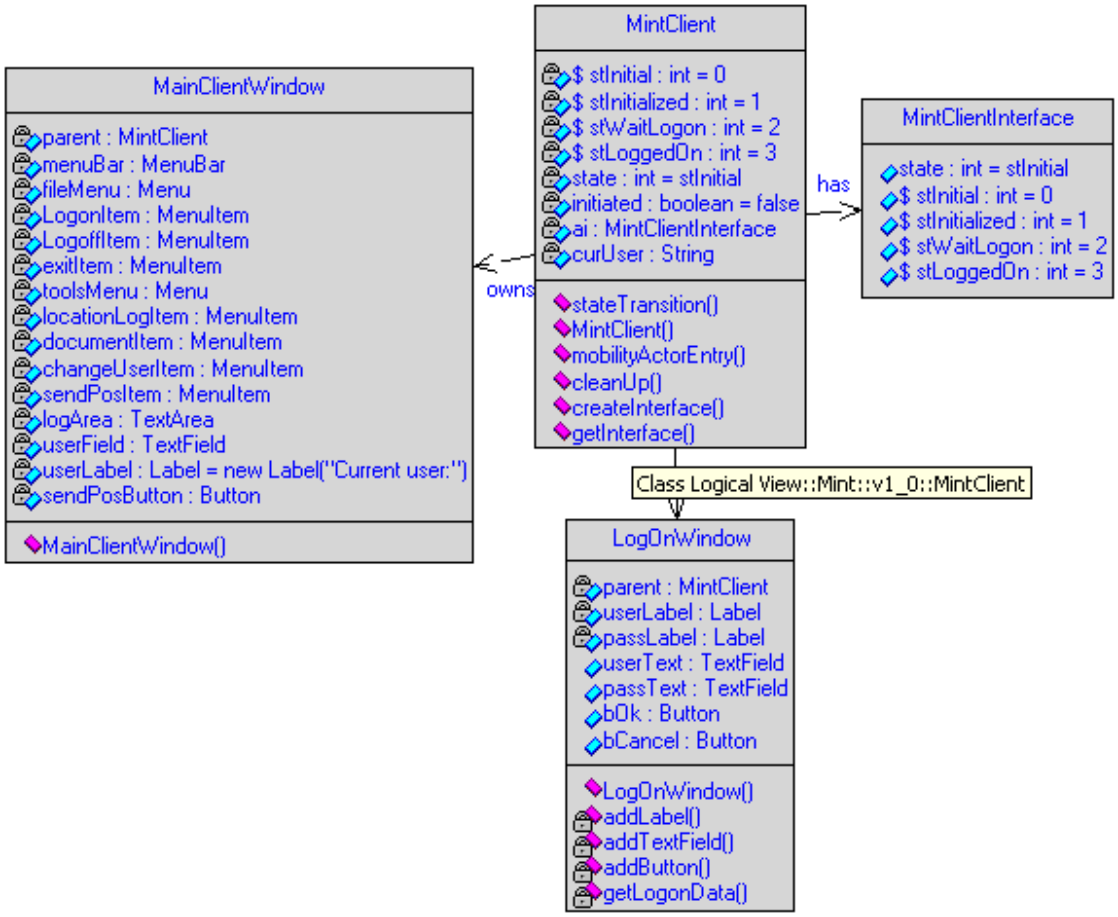


Figure 21: Class diagram of MintClient

For more detailed class diagrams of the other classes, see Appendix E: CD.

5.4 State diagrams

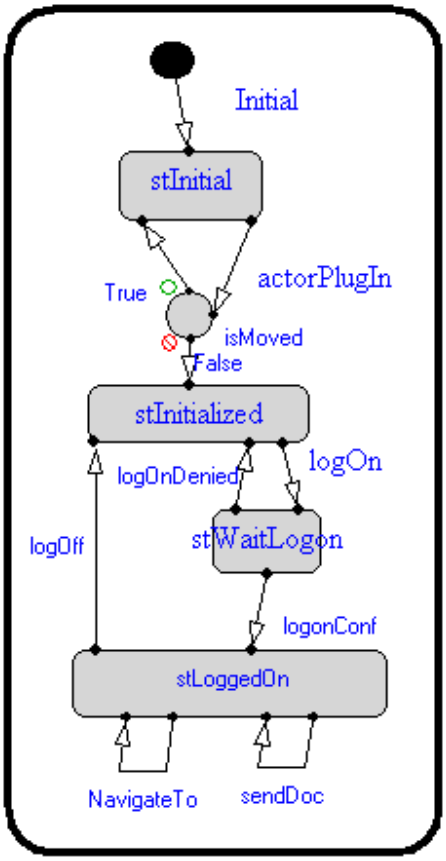


Figure 22: State diagram for MintClient

Rational Rose RT gives the opportunity to model state diagrams. These are used by the program to generate real-time code, and can thus not be used directly to generate TAPAS or MicroTAPAS code. But still it can be used to show some functionality. Figure 22 shows the state diagram created for MintClient. The *isMoved* test is used to not initiate the role figure if it is moved, but wait for the creation of the interface that contains the current state.

5.5 Sequence diagrams

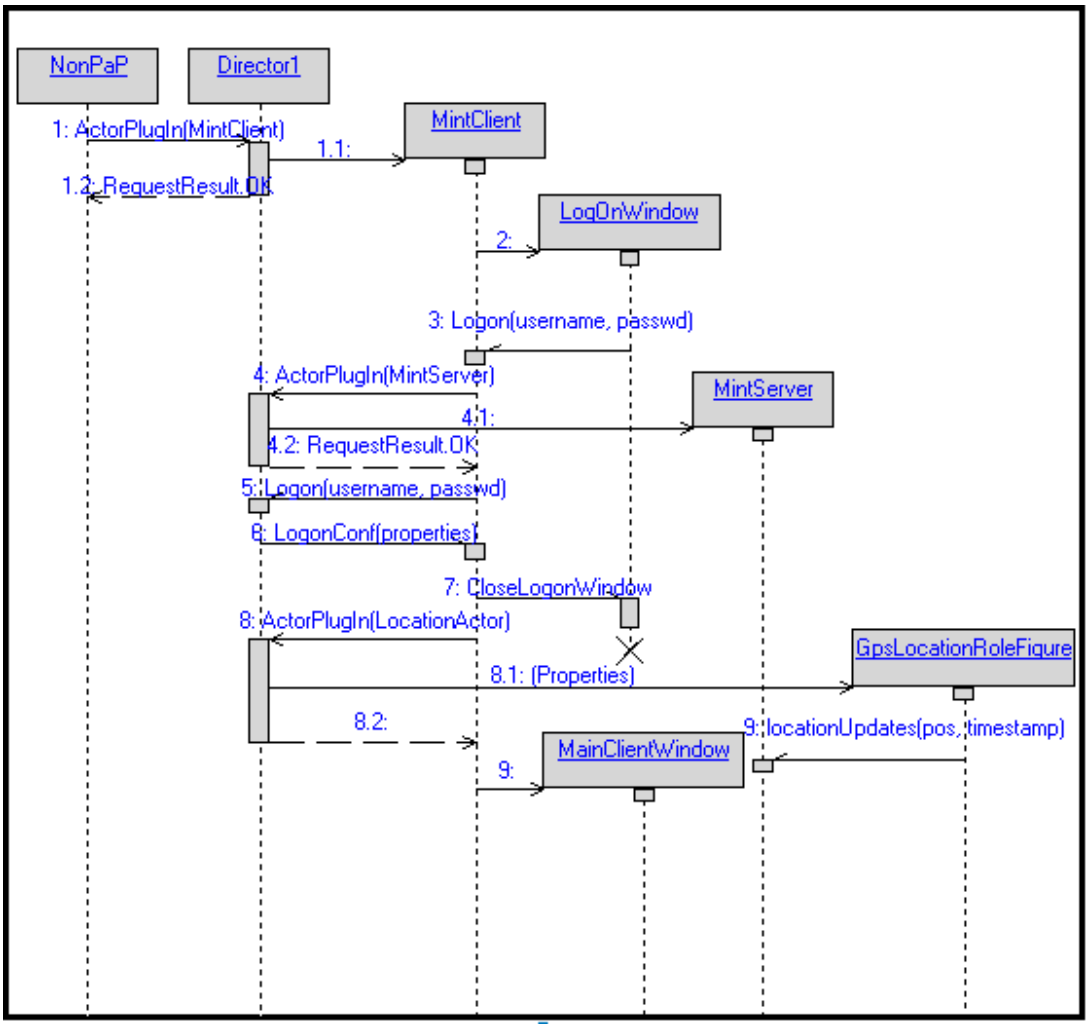


Figure 23: Sequence diagram for logging on

Figure 23 shows a sequence diagram for starting up the application at the mobile unit. When the MintClient is plugged in, it opens the LogOnWindow in order to log on the user to the system. The client tries to plug in the MintServer role figure if this is not already running, and sends the username and password to the director in a *LoginRequest*. If the login is correct, the director returns the user's properties that are retrieved from the user profile base. If these properties say that the user is supposed to send GPS positions, the MintClient tries to plug in the GpsLocationRoleFigure for handling this. The GpsLocationRoleFigure will then send locationUpdates to MintServer according to the profile. The login process ends with the MintClient opening the MainClientWindow, which is the main interface towards the user.

5.6 XML file

In order to handle user mobility, different user profiles was created. These profiles are stored in an XML file called UserProfiles.xml that is located in the bootstrap root directory for the director. See also Appendix E: CD.

```
<UserProfiles>
  <User>
    <Username>user1</Username>
    <Password>1</Password>
    <BackgroundColor>102 51 255</BackgroundColor>
    <WindowLocation>422 55</WindowLocation>
    <WindowSize>522 358</WindowSize>
    <SendDoc>false</SendDoc>
    <SendGpsPos>true</SendGpsPos>
    <SendGpsAutoTimer>10</SendGpsAutoTimer>
  </User>
</UserProfiles>
```

Table 1: Example of user profile

Table 1 shows an example of a user profile used by Mint. The user with the username "user1" has properties for the background color, window location and window size. (Note that window location and window size is not used on PDAs, since these always show the windows in full screen). The user isn't authorized to send documents, but is supposed to send its GPS location and this is by default set to be sent periodically with 10 seconds intervals.

6 Implementation of Mint

The UML diagrams that were designed in chapter 5 were used to generate "code shells" of the classes using Rational Rose RT, and these classes was later used as a framework for developing the application. These shells were then imported in to NetBeans, which is an open source, free Java IDE sponsored by Sun that can be downloaded from the NetBeans web page [OL16].

Although the application was mostly implemented using a normal J2SE IDE, the application was targeted for J2ME, CDC configuration with the Personal Profile (see Appendix D: J2ME for information about J2ME).

Some parts of the program were implemented using IBM WebSphere Studio Device Developer [OL4], which was useful because the JVM used on the targeted PDA was IBM's J9. WSDD has also the capability of porting the code directly to the PDA for testing. This was useful when the GPS support code was developed.

A dynamic link library file was made in C++ for interfacing TomTom Navigator. This dll was made using Microsofts eMbedded Visual C++ 3.0 [OL21], but due to problems described in section 6.3, this was never finished.

Some issues that occurred during this phase and solutions made are described in more detail in this chapter.

6.1 Location API (JSR 179)

There are several optional packages to J2ME. One of these is the Location API or JSR 179. This is primarily developed for the CLDC configuration, but it is designed to also work with CDC. The only problem with this API is that it only contains classes with abstract methods, which means that the functionality of this API is to make a standard interface. It is up to the user to implement how the location is retrieved, but by using this API the implementations will have the same interface and methods.

Nokia was the prime mover behind this JSR, and Nokia have developed a Reference Implementation (RI) to JSR 179 that includes an example using GPS. It was hoped that this could be used in this project but this example used the MIDP profile of CLDC, and this proved to be impossible to use with CDC. The alternative was now to make the code from scratch, getting the input directly from the serial port and parsing the info.

Instead, it was chosen to use an existing NMEA parser and serial port driver made by James Nord [OL17]. These classes use a dynamic link library (dll) to get the serial port data, and this meant that this dll had to be ported to the PDA. The different classes made by Nord are also made in separate Java packages, and this proved to be difficult to import "the TAPAS way", i.e. to download it from the web server. It is believed that this is because of the way that package names are used when plugging in a play. Instead the needed classes was ported to the targeted PDA, and stored in the classpath for the JVM on the device.

6.2 Sending and getting documents

MintDocumentRoleFigure and GetDocumentsRoleFigure were implemented with the use of role figure mobility. This was mainly to show that it is possible to use role figure mobility for this type of transfer, but this choice also sets some restrictions. Using role figure mobility proved to be very easy and quick to implement, because this then lets MicroTAPAS handle all the transfer. There is no need to set up a socket connection between the two peers to transfer the files, because this is done by MicroTAPAS when transferring the interface of the role figure (see Figure 13 for how this transfer is performed).

The drawback is that the files have to be read in to memory and stored as a variable in the interface of the role figure prior to the movement. This implies that transfer of large files can take up a lot of memory, and cause problems on a device with limited resources. The JVM will probably prevent this from doing any damage to the system, but it is important to know about this issue.

MintDocumentRoleFigure is implemented so that the user selects the documents he or she wants to send, while GetDocumentsRoleFigure searches for documents with given parameters (i.e. folder, file type and/or date modified) and this search will include files in subfolders.

6.3 Sending waypoint

This functionality was meant to be implemented with the use of a car navigation system for the PDA. For this purpose, TomTom Navigator [OL8] was chosen to be the targeted system. TomTom Navigator has implemented an interface for third party programs through an extra package called TomTom Navigator SDK. This SDK only has interface for C++ or Visual Basic, but this could have been solved with Java Native Interface (JNI).

6.3.1 Problem #1

TomTom Navigator 3 was the newest version available at the start of this project, but TomTom hadn't released any SDK for this version. This was due to be released in mid November, and would have been too late for this project. The available SDK was for TomTom Navigator 2, so this was ordered.

6.3.2 Problem #2

TomTom Navigator SDK 2 was only available to purchase online, via handango.com or amazon.com. To order online from these web sites, Visa or other credit cards have to be used. This isn't easy for an institution like NTNU, and the order was stopped due to different names on the credit card and the purchaser. The IT department at the institute of Telematics has been working to solve this, but unfortunately without luck.

There was, however, made a JNI class and a simple dynamic link library (dll) to test if MicroTAPAS was able to use C++ code, and this worked fine. The current implementation just prints to the mobile user that is should navigate to certain coordinates, and the user now has to put this coordinates in to the navigation system manually.

6.4 MintServer

To get a running server for the Mint application, there is need for dynamically updated maps to show the presence of the different mobile users. This task would have included a lot of GUI programming, and this was thought to be little bit on the side of the purpose of this project. A simple, text-based role figure was implemented instead, using MicroTAPAS's own debug window for this. All the necessary commands have been implemented in MintServer's drop-down menu for commands.

6.5 Screenshots

This section will show some screenshots of the Mint application, as it is viewed at the targeted PDA. Other screenshots are provided in Appendix B: Screenshots of Mint and on the CD supplied as Appendix E: CD.

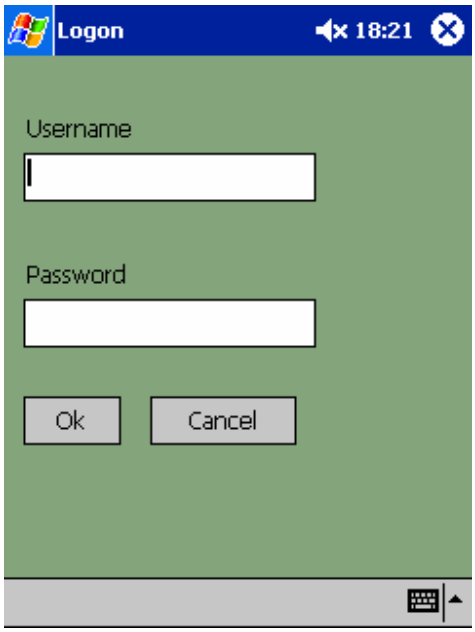


Figure 24: Login window

Figure 24 shows the logon window of the MinClient as it is displayed the first time a user logs on. If the typed username or password is incorrect, an error message will be displayed. If the user presses "Cancel", the MintClient will try to perform an actorPlugOut procedure and exit.

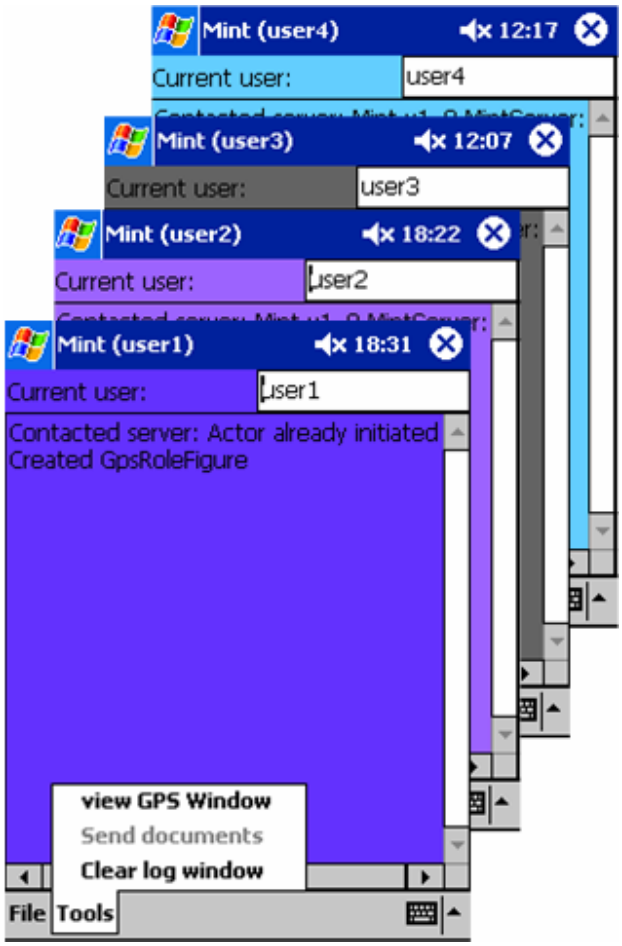


Figure 25: Four different users, with different profiles

Figure 25 shows that the different users have different background color set in their profile. User3 was maybe not too lucky with its choice of color, though.

7 Testing and experimentation

Testing and experimentation of functionality is a major part of any programming cycle. During the process of making the Mint application described in chapter 4, and after the completion of the service, several tests were conducted. After installing MicroTAPAS on both a PC and a PDA (see Appendix A for info about MicroTAPAS on PDAs), the first test was to be carried out. This was a simple test of terminal mobility that was already supported by MicroTAPAS.

7.1 Terminal mobility test

This test was basically a recreation of a test described in [9], and was tested with moving a PDA between different WLAN access points on different subnets. There was only one director and MobilityManager involved, so the test was just to check how MicroTAPAS handled changes of IP address and connectivity. Two APs were configured in addition to an open AP at the department of Telematics, and these were on separate IP subnets in order to make the PDA change IP and not just roam between the APs.
The test was set up with the following configuration:

#	Hardware	Connection	Virtual Machine	Role
1	PC, Windows XP	LAN	Sun JVM 1.4.2_05-b04	Director/MM
2	PDA, iPAQ 3870, PocketPC 2002	WLAN	IBM J9	MA
3	Laptop, WindowsXP	WLAN	Sun JVM 1.4.2_04-b05	MA
4	Student web server, Linux	LAN		Web server
5	AP, tapas	LAN/WAN		Subnet 200.X
6	AP, tapas2	LAN/WAN		Subnet 209.X
7	AP, itemwlan	LAN/WAN		Subnet 66.X

Table 2: Configuration of terminal mobility test

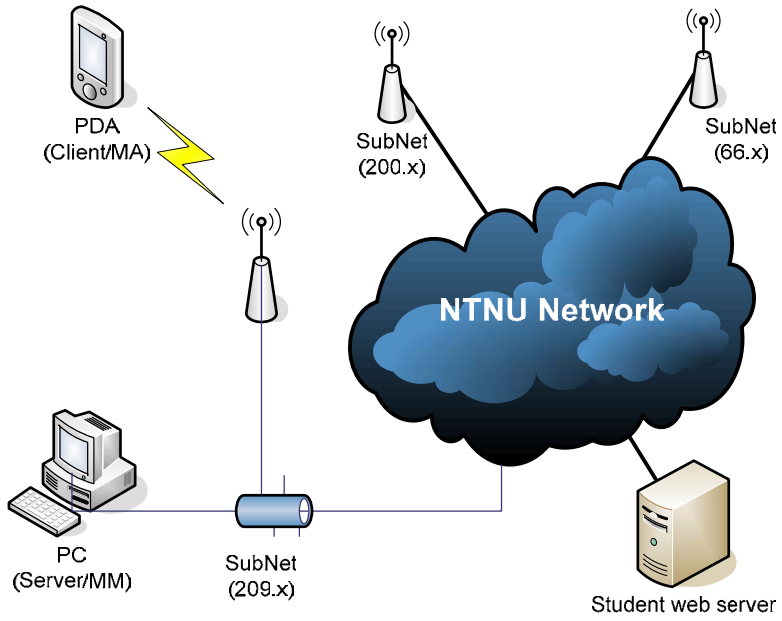


Figure 26: Terminal mobility test

Figure 26 shows the setup for this test. The PDA did not automatically change IP address when entering a new subnet, but this was solved with using a small tool called PocketLAN [OL20] that was downloaded from Internet. Manually changing the IP was also tested, like described in [9, section 6.5.2].

7.1.1 Test results

The PDA reported to the consol window when it lost connection, and the MM set the node status to "not connected" like expected. When the IP address was changed, the MA did not report the expected TerminalMove, but only that the connection was reestablished to the director. (This was reported by the MicroPingClient). On the other hand, the MM did not report any changes, so any actorPlugin requests was not successful and the director reported that the receiver was not registered. The test was performed several times, trying to change between different APs, but the result was the same. The change of IP address was confirmed by the PocketLAN tool and also by accessing www.whatismyip.com on the Internet.

On the itemwlan network (66.x), there was not made any successful connection at all. There is apparently hard coded somewhere in the MicroTAPAS code that the 200.x and

208.x/209.x networks can be used by the MobilityManager, but this code has not been found.

The same test was performed with a laptop on the same WLANs. This time the PNES on the laptop reported a TerminalMove request, and the MobilityManager also reported the change of IP. However, trying to perform a role session action with existing role figures on the moved terminal failed with the error message that the terminal was not registered. Plugging in new actors, however, worked fine and these were registered as normal.

The conclusion of this test must be that terminal mobility is not yet fully implemented and supported by MicroTAPAS, and the problems that occurred needs further study.

Because of this result, further extensive tests including terminal mobility could not be performed.

7.2 User mobility test

This test was to ensure that the basic functionality of the application worked and the different profiles was retrieved correctly. For this test four user profiles were saved in UserProfiles.xml with the profiles given in Table 3.

Username	Background color	GPS	Send documents
user1	R:102 G:51 B:255	True	False
user2	R:153 G:102 B:255	False	True
user3	R:100 G:100 B:100	False	False
user4	R:100 G:200 B:255	True	True

Table 3: User profiles

The test was just to log on with the different users, log off and to try to change user while logged on. The configuration was the same as in Table 2 and Figure 26, but only connected to one subnet.

7.2.1 Test results

The log on procedure of the different users worked as expected, and screenshots of windows with different background color and different functionality can be seen in

Appendix B: Screenshots of Mint. If the username or password was mistyped, an error message occurred in the Logon window. With the proper username and password, the MintServer role figure was instantiated on the node hosting the director of the domain if this was not already running, and the main window of the client was opened.

Changing the user while logged on resulted in an error when the currently logged on had the GPS functionality set to true (user1 and user4). Further investigation of this problem revealed that the GpsLocationRoleFigure received an exception while trying to plug out. It was tried to fix this problem, but without success. It is believed that this error is because of the NMEA parser and serial port driver used, probably because some process is trying to communicate with the parser or driver while plugging out.

The conclusion of this test is that user mobility is supported by MicroTAPAS, but an unwanted error occurred while plugging out the GPS location role figure. The reason for this error was not resolved.

7.3 Geographical position test

This test was performed to test the functionality of sending the GPS position from a mobile client to the server. The test was performed with user1 from Table 3 adding a timeout parameter for periodically sending of the GPS location. This parameter was set to 10 seconds. The test was performed with the same configuration as in Table 2 and Figure 26, with the client only connected to one WLAN.

7.3.1 Test results

The test was performed with the following procedure and result:

Test	**Result (time* in seconds)**
1. Wait until the timeout occurred	Update received ~ every 12 sec.
2. Send the position explicitly from the client	2.8s, 2.6s and 2.6s
3. Request the position explicit from the central	Server reported 0.891s, 0.969s and 1.109 s to request was sent.

	Total time was 3.1s, 3.3s and 3,4s.
4. Stop the periodic transmission by request from the server	Server reported 0.969s, timed to 1,9s (received at client)
5. Send the position explicitly from the client	2.4s, 2.3s, and 2.3s
6. Request the position explicit from the central	Server reported 0.937s, 0.766s and 0.875s to request was sent. Timed to 3.4s, 3.3s and 3.3s
7. Start the periodic transmission by request from the server with new timer of 20 sec.	Server reported 0.892s to request was sent. Timed to1.3 s (received at client). Update received ~ every 23 sec.

Table 4: GPS location test

* If not explicitly said otherwise, the timing was done manually with a stop watch.

Item number 3 and 4 was included to the test to see if it still was possible to send and request the position when the periodic timer was switched off. Every test, except switching on and off the timer, was performed three times in order to get some variation. Preferably each tests should have been performed a number of times, but the main purpose of this test was to get an impression of the time consumed and to check that the specified functionality worked.

The reason for the timers being a couple of seconds after what the timer was set to is believed to be because of processing time at the client. The client first processes the timeout before it starts a new timer.

The conclusion of this test is that the GPS functionality specified is fully supported in Mint.

7.4 Document transfer test

This test was performed to test the document features that are implemented in Mint. The test was performed with user2 from Table 3. This user is only allowed to send documents according to the user profile. The test was performed with the same configuration as in Table 2 and Figure 26, with the client only connected to one WLAN.

The files used for these test is displayed in Table 5.

Files	Type	Size
Pocket PC Note	Pwi (may be opened with Word)	2 KB
Four bitmap files	Bmp	151 KB
Jpeg file	Jpg	21 KB
Mobile video	3gp	1811 KB
Total	*7 files*	*2.37 MB*

Table 5: Files used in document transfer

These files were with different timestamps, and the mobile video clip was saved the same day as the test was performed. The Jpeg image was stored in a subdirectory, to test the get documents functionality, which scans subdirectories.

7.4.1 Test results

Test	Result (time* in seconds)
1. Send all documents from client to server	18.1, 17.8, 15.3
2. Send all documents from server to client	14.3, 14.5, failed**
3. Get all documents in folder	18.9, 18.7, 18.8
4. Get all jpeg files in folder	13.0, 12.9, 13.2 (one file)
5. Get all bitmap files in folder	14.9, 14.9, 14.6 (four files)
6. Get documents newer than one day	17.2, 18.0, failed**

Table 6: Document transfer test

* If not explicitly said otherwise, the timing was done manually with a stop watch.
** MicroTAPAS experienced a crash, reason unknown

Every test was performed three times in order to get some variation. Preferably each tests should have been performed a number of times, but the main purpose of this test was to get an impression of the time consumed and to check that the specified functionality worked. Some failures were experienced during this test. The reason for this was not resolved and can be because of general instability of Pocket PC (e.g. the WLAN card stopped to work after a soft reset), instability of running Java, instability of MicroTAPAS as well as failure in the tested functionality. On at least one occasion, the JVM on the server reported that it failed to connect to the client (java.net.SocketTimeoutException:

Read timed out). It was for some time believed that the timeout was because of the code for GetDocumentsRoleFigure, which does the entire file reading process and actorMove procedure during the actorPlugin procedure. Modifications of the code to separate this into two different processes were tried out, but without success.

The conclusion of this test is that the document transfer functionality is working, although some failures were experienced. The reasons for these failures were not revealed.

8 Discussion

The Mint application developed in this project has proven that TAPAS is useful for developing quite complex distributed systems with extended support for mobility. However, the process of getting there was not so easy. Several challenges were faced during this process, and this chapter will try to address them, evaluate the work done and come with suggestion for further work.

8.1 Faced challenges

One major problem in developing applications is always documentation. In Java there is provided functionality for generating documentation of the code, called Javadoc. This documentation is very good in explaining the different classes and methods created, but it lacks functionality to describe *how* to use the classes and methods. This problem impairs with the complexity of the documented code.

This is thus also a major problem for TAPAS and MicroTAPAS, where up to 80 classes are involved and these classes are supposed to be used by others as a framework for developing new applications and services. Although that it is simple and quite easy to implement role figures when the procedure is known, there would have been quite helpful to have some sort of how-to, tutorial or user manual. What is meant by a how-to is a simple document that tells the user which classes and methods to use for making new role figures, in which order the different methods are accessed and what the consequences of different choices are.

The UML diagrams in [14] are also lacking documentation on how to use them. First of all, it was discovered that the diagrams were not usable as templates in Rational Rose RT (that is in the computer term of the word template), but could only be copied and this resulted in some unwanted linking in the UML diagrams. Secondly, there was no description in how to use the classes supplemented. A lot of time was used on figuring out how to use the diagrams and how the code worked.

Another problem for the development was that none of the people that have developed the implementations is in the project today, at least for MicroTAPAS where all the work has been done by students. There are simply none that have in-depth knowledge of the functionality that could easily be asked when problems and questions arose.

It took some time before it was understood that enhancements and additions to the platform (MicroTAPAS) had to be made. This was not explicitly specified in the task, and it was for a long time thought that only already implemented functionality was to be used in the application created.

When the proper knowledge of MicroTAPAS and the inner workings were sufficient, it became clear that it was surprisingly easy to create role figures and functionality with the use of the MicroTAPAS platform. This indicates the importance of good documentation and of good models, and a lot of time could probably have been saved here.

8.2 Evaluation of the result

Not all of the listed requirements for the Mint application given in chapter 5 were completed in the final application. The part with a car navigation system for guiding the user to the right position, stranded due to administrative problems (see section 6.3 for more info). Other than that, the functional requirements given were completed.

The solution of using role figure mobility for transferring documents can be discussed. This procedure leaves it to the MicroTAPAS support system to handle the actual transfer of the files when it sends the interface to the newly created role figure. But this limits the possibility of sending large files. Video clips made by a cellular phone were tested and the clip used when testing file transfer (see section 7.4) was 3 minutes long and there were no problems transferring this, but lager files may cause problems. The proper way of handling file transfer is thus to read small parts of the file and send these on the fly.

There are of course issues about non-functional requirements. Time can be essential for the emergency systems, and MicroTAPAS takes up to over one minute just to start up. If the PDA had been on and working for some time, things also started to slow down until

they eventually stopped working. This is a problem with instability in the PocketPC OS or the PDA, and most of these problems will presumably be solved when newer and more powerful devices are developed.

8.3 Changes made to the MicroTAPAS implementation

During the process of making the Mint application described in chapter 4, it became clear that some additions and some changes to existing code were needed in order to get the mobility support that was needed. These changes are given in Table 7 with a brief description. For further description of the classes and methods used, see the updated Javadoc for MicroTAPAS in Appendix E: CD.

Class/File	Package	Description
UserProfiles.xml		XML file that stores user profiles. This xml file has to be in the TAPAS bootstrap root directory of the director.
LoginRequest	MicroTAPAS	This is a special request used with RequestPars to handle login requests. Modified version from the one in [15].
MicroDirector1	MicroTAPAS	Reads now UserProfiles.xml and gets the user profiles. Method directorActorEntry updated to handle LoginRequests and check this with the user profiles.
TimerMessage	MicroTAPAS	A simple timer in order to make automatic or periodic actions.
Session	MicroTAPAS.util	The same is in [15], but currently not used.
UserProfile	MicroTAPAS.util	Modified version of the one in [15]. Used to store a user profile.
UserProfileBase	MicroTAPAS.util	Modified version of the one in [15]. Used to store a collection of user profiles. Used by MicroDirector1
XMLFileUtil	MicroTAPAS.util	File utility that can read and write xml files. This

		class handles UserProfiles, SessionDescriptions and RoleList xml files, and is basically the same as in [15]

Table 7: Changes and additions to MicroTAPAS

8.4 Further work

8.4.1 Implementation of lacking mobility features

The additions to MicroTAPAS that is listed in Table 7 are mostly taken from Lars Erik Liljebäcks master thesis [15], and functionality from that thesis that is not currently implemented in MicroTAPAS (i.e. the use of LoginAgents, VisitorAgents, RoleLists, SessionDescription etc.), can easily be implemented with the use of these additions.

Terminal mobility also failed to work as specified in the test performed in section 7.1. This needs to be investigated further, because this type of mobility is very important for mobile systems. In fact, this can bee seen as the basis for a system being mobile at all.

8.4.2 Handling of Non-TAPAS code

There were experienced problems when trying to use Java classes that used other packages than the play names during this project. There should absolutely be implemented support for this, in order to enhance the usability of MicroTAPAS. Whether this can be done by doing a playPlugin with the full package name has not been tested out during this project.

8.4.3 Better model support for developing role figures

It should be possible to use even better model support for developing role figures in TAPAS. With the use of e.g. Model Driven Architecture (MDA) or supplements to UML 2.0, it may be possible to generate developed code directly, generate XML or any other machine interpretable language. This will probably be a very demanding task, but it would also have been of great help for developing new applications with TAPAS.

9 Conclusion

This project has proven that it is possible to develop an application with quite complex functionality in a relatively short period of time with the use of the TAPAS and its mobility architecture. The idea of using code on demand enables a simple way of distributing new or updated code without having to update every terminal, but it has been shown in this project that it also can cause problems and limitations. This regarding the way that code on demand is performed in TAPAS. One major paradox when trying to make things simpler is that the complexity of the new technology often increases, although the end result for the users is much simpler.

The Mobile Interactive Navigation Tool (Mint) application was developed to make use of the mobility support within TAPAS, with the use of the MicroTAPAS implementation. This project has been the first to try to make an overall and comprehensive application design, specification, implementation, and testing work using all the features of the TAPAS/MicroTAPAS mobility framework. This includes both the architectural concept and the support framework.

It is very important to notice that neither Mint nor the MicroTAPAS implementation are finished and ready to use. They are just used to show *how* the overall ideas of TAPAS may be implemented.

References

[1] Finn Arve Aagesen, Bjarne E. Helvik, Chutiporn Anutariya and Mazen Malek Shiaa, ”On Adaptable Networking”, The First International Conference on Information and Communication Technologies, ICT 2003, Assumption University Thailand, April 2003.

[2] Mazen Malek Shiaa, “Mobility Support Framework in Adaptable Service Architecture”, IFIP - IEEE Conference on Network Control and Engineering for QoS, Security and Mobility, NetCon'2003, Muscat-Oman, October 2003.

[3] Garmin Corporation, “GPS guide for beginners”, Part Number 190-00224-00 Rev. A, December 2000.

[4] Eirik Lühr, “TAPAS for wireless PDA”, Project at department of Telematics, NTNU, Spring 2003.

[5] Mazen Malek Shiaa and Finn Arve Aagesen, “Mobility management in a Plug and Play Architecture”, IFIP TC6 Seventh International Conference on Intelligence in Networks, Saariselka, Finland, April 2002. Published by Kluwer Academic Publishers.

[6] Finn Arve Aagesen, Bjarne Helvik, Ulrik Johansen and Hein Meling. ”Plug and Play for Telecommunication Functionality -- Architecture and Demonstration Issues”, The International Conference on Information Technology for the New Millennium (IConIT), Thammasat University, Bangkok, Thailand, May 2001

[7] Mazen Malek and Finn Arve Aagesen. “Mobility management in a Plug and Play Architecture”, IFIP TC6 Seventh International Conference on Intelligence in Networks, Saariselka, Finland, April 2002. Published by Kluwer Academic Publishers.

[8] Berndt H., Darmois E., Dupuy F., Inoue Y., Lapierre M., Minerva R., Minetti R., Mossotto C., Mulder H., Natarajan N., Sevcik M., and Yates M., “The TINA Book”, Prentice Hall Europe, 1999.

[9] Eirik Lühr, “Mobility support for wireless devices – within the TAPAS platform”, Master thesis at department of Telematics, NTNU, 2004.

[10] Philip Yam, “Everyday Einstein”, Scientific American, Volume 291, Number 3, September 2004.

[11] European Union Council Conclusions on GALILEO, 7282/02, 25./26. March 2002.

[12] David Last, “GPS and Galileo: where are we headed?”, European Navigation Conference, GNSS2004, May 2004.

[13] Marius Dalsmo, “Plug-and-Play services for PDA and Java-enabled phones”, Project at department of Telematics, NTNU, Autumn 2003.

[14] Fred Inge Henden, “Developing Role Figure Model based on UML Specification”, Master thesis at department of Telematics, NTNU, 2004.

[15] Lars Erik Liljeback, “User and Session mobility in a Plug-and-Play architecture”, Master thesis at department of Telematics, NTNU, 2002.

[16] Finn Arve Aagesen, Bjarne Helvik, Vilas Wuwongse, Hein Meling, Rolv Bræk and Ulrik Johansen, “Towards a Plug and Play Architecture for Telecommunications”, IFIP TC6 Fifth International Conference on Intelligence in Networks, Bankok, November 1999, Kluwer Academic Publishers, ISBN 0-7923-8691-4.

[17] Ronald Ashri, Steve Atkinson, Danny Ayers, Marten Haglind, Bill Ray, Rob Machin, Nadia Nashi, Richard Tatlor, Chanoch Wiggers, “Professional Java Mobile Programming”, Wrox Press Ltd, ISBN 1-861003-89-7, 2001

[18] Bradford W. Parkinson and James J. Spilker Jr, “Global Positioning System: Theory and Applications, vol. I”, American Institute of Aeronautics and Astronautics, ISBN 1-56347-106-X, 1996

[19] Proposition No.1 to the Storting – Supplement No.3, 2004 – 2005, For the budget period 2005, “Future radio communication for the emergency and preparedness services”, 5 November 2004.

[20] Mark Weiser, “The Computer for the 21st Century”, Scientific American, Volume 265, Number 3, September 1991.

[21] Mark Weiser, “Some computer science issues in ubiquitous computing”, Communications of the ACM, Volume 36, Issue 7, pp. 75 – 84, July 1993

Online references

[OL1] Telematics Architecture for Play-based Adaptable Systems (TAPAS), http://tapas.item.ntnu.no/, [Accessed September 2004]

[OL2] GlobalSecurity.org, Beidou (Big Dipper), http://www.globalsecurity.org/space/world/china/beidou.htm, [Accessed September 2004]

[OL3] Java Technology, Sun Microsystems, http://java.sun.com/, [Accessed November 2004]

[OL4] IBM Workplace Client Technology, Micro Edition, http://www-306.ibm.com/software/wireless/wctme_fam/, [Accessed October 2004]

[OL5] Federal Aviation Administration, Satellite Navigation Product Team, http://gps.faa.gov/Library/waas-f.htm, [Accessed November 2004]

[OL6] Official Galileo Website, EUROPA - Energy and Transport – GALILEO, http://europa.eu.int/comm/dgs/energy_transport/galileo/index_en.htm, [Accessed October 2004]

[OL7] OziExplorer, http://www.oziexplorer.com/, [Accessed November 2004]

[OL8] TomTom, http://www.tomtom.com/, [Accessed November 2004]

[OL9] Enrique Ortiz, "A Survey of J2ME Today", dated October 2004, http://developers.sun.com/techtopics/mobility/getstart/articles/survey/, [Accessed November 2004]

[OL10] Sun, Whitepapers on J2ME, http://java.sun.com/j2me/reference/whitepapers/, [Accessed November 2004]

[OL11] Sun, "J2ME Technologies Overview Data Sheet", http://java.sun.com/j2me/docs/j2me-ds.pdf, [Accessed November 2004]

[OL12] Sun, "CDC: An Application Framework for Personal Mobile Devices", http://java.sun.com/products/cdc/wp/cdc-whitepaper.pdf, [Accessed October 2004]

[OL13] Norwegian Public safety radio project, http://www.nodnett.no/english/index.htm, [Accessed November 2004]

[OL14] Teleavisen AS, http://www.teleavisen.no/, [Accessed November 2004]

[OL15] Russian Federation, Ministry of Defence, Coordinational Scientific Information Center, GLONASS, http://www.glonass-center.ru/, [Accessed September 2004]

[OL16] NetBeans.org, http://www.netbeans.org/downloads/index.html, [Accessed September 2004]

[OL17] Teilo, http://www.teilo.net/, [Accessed November 2004]

[OL18] Autonomic Computing: IBM's Perspective on the State of Information Technology, IBM Corporation (2001), http://www-1.ibm.com/industries/government/doc/content/bin/auto.pdf, [Accessed November 2004]

[OL19] IBM Research, Autonomic Computing, http://www.research.ibm.com/autonomic/, [Accessed November 2004]

[OL20] PocketLAN, download page, http://www.pocketgear.com/software_detail.asp?id=2825, [Accessed November 2004]

[OL21] Pocket PC 2002 SDK, downloadable from http://www.microsoft.com/downloads/, [Accessed September 2004]

Appendix A: Running TAPAS on a PDA

In [3, Appendix A.2] Lühr gives a short explanation of how to create a link for Pocket PC 2000. Due to the fact that running Java applications on PDAs can be very troublesome, the following appendix will give a bit more extensive guide on how to get your java application running.

A.1 JVM for PDA

One idea behind Java as a programming language is that the created code is platform[1] independent. In order to make this work, there has to be an interpreter on the platform that translates the Java code to machine code that the OS understands. This interpreter is for the Java environment called Java Virtual Machine (JVM). This "virtual machine" is created in some kind of natural code (usually C++), and cannot be moved seamless between different hardware and operating systems, although the code that they are interpreting can.

That the Java code is platform independent is actually a truth with many modifications, due to the fact that different kinds of hardware have different runtime demands, and that there are several different vendors out there that create JVMs. This problem comes especially to place when we talk about J2ME, which is designed for devices with limited resources. Sun Microsystems, which is the sponsor for Java, have a webpage [OL3] where people can find information about Java and the Java Technology. Here you can find several JVMs for different uses, but for the CDC Technology, which is used in MicroTAPAS, they have only released a version for Linux OS and not for Microsoft Pocket PC.

The task of finding a proper JVM can be demanding, especially since the vendors tend to demand fees for their implementation. Several PDAs are delivered with JVM that can be installed, like Jeode from Insignia or Jbed CDC from esmertec, but none of these are free for users that have not bought a PDA with the software. IBM has developed a JVM that

[1] Platform is in this context viewed as the operating system

free for development purposes called J9, and this can be downloaded from IBM together with IBM WebSphere Studio Device Developer (WSDD) [OL4]. There is no need for using WSDD, only run the installation and select “Install Micro Environment” and follow the instructions given on the screen and J9 will be installed on the PDA.

A.2 Creating link for Pocket PC

To be able to run java applications in any environment, the JVM has to be executed first with the application to run as an argument. This can be fixed using links or scripts, i.e. .bat-files in the windows environment. This is the same for PPC, but here links is the appropriate way. Creating links for PPC can be tricky, and has to be done on a PC and transferred using Active Sync to the PPC if there has to be additions, like arguments, added to the link.

To make the link, just open a text editor like notepad and type in the file path and arguments. The first characters declare how many characters to be used. Maximum is 255 and by setting this to 300, there is no need to count the characters. Table A1 shows an example of a link created for starting MicroTAPAS.

```
300#"\Program Files\J9\PPRO10\bin\j9w.exe" "-jcl:ppro10" "-cp"
"\MicroTAPASBoot\StartMicro" "MicroTAPAS.startMicroTAPAS"
"MicroTAPAS.MicroPNES" "tapas.cfg"
```

Table A1: Example of link file for Pocket PC

The options used here are –jcl, which indicates what profile to use (here Personal Profile 1.0) and –cp which is indicates the classpath, i.e. the path to where the java classes are located.

When the link is placed on the PPC, only tap the link and MicroTAPAS will load according to the specified configuration file. Java code takes some time to load and especially when starting MicroTAPAS due to the fact that it has to download code from the web server, so patience is important. If there are problems with getting the service started correctly, a soft reset of the PDA might be needed. Also note that running PPC can be quite unstable and you can get many strange errors with programs that normally work ok.

Appendix B: Screenshots of Mint

Figure B1: Different profiles have different functionality

Figure 25 on page 47 shows that User1 does not have the privileges of sending documents and here Figure B1 shows that User2 does not has this privilege, but GPS functionality. User4, on the other hand, has the privileges of both GPS and sending documents.

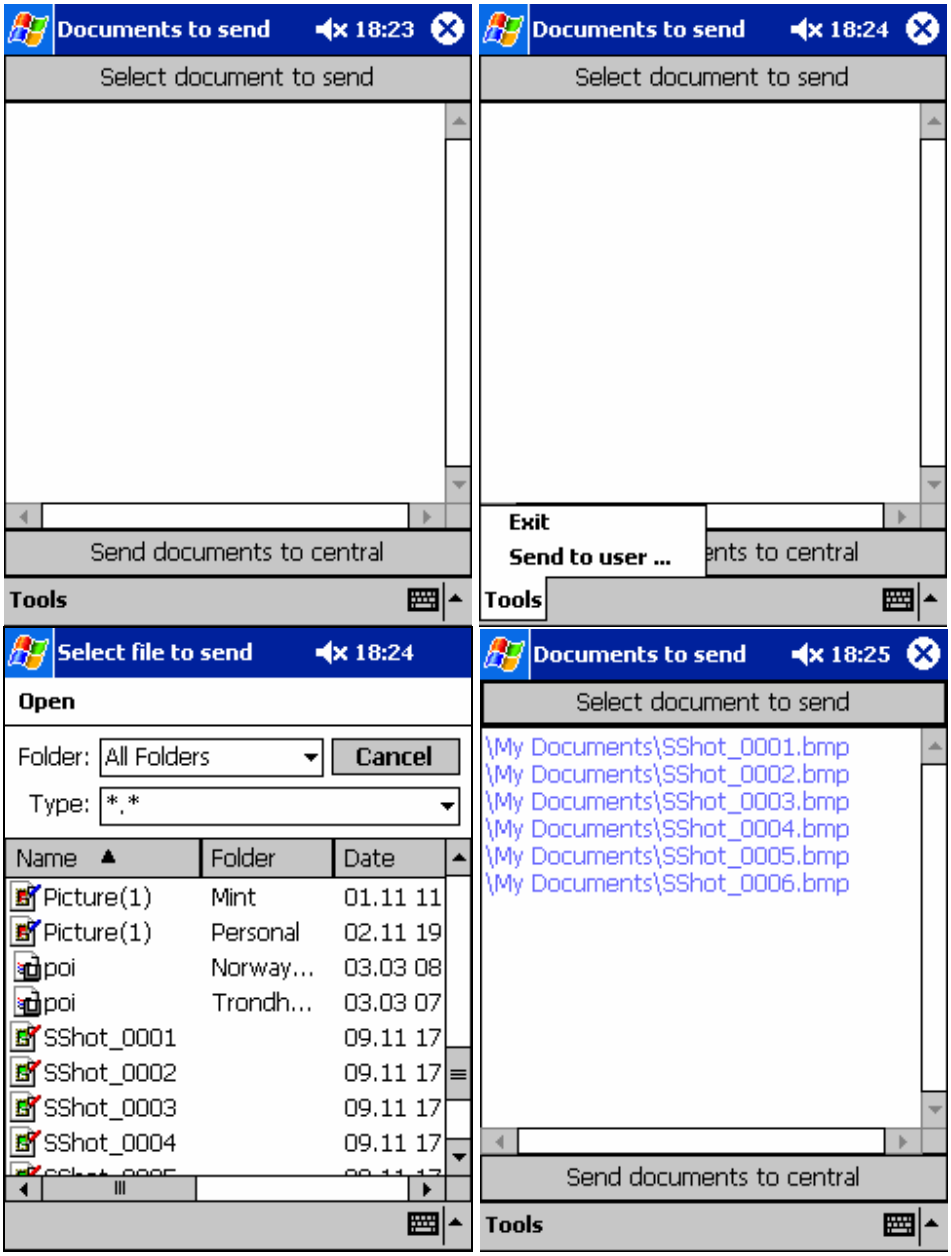


Figure B2: Sending documents from mobile client

Figure B2 shows screenshots of when a user wants to send documents. The top-left window shows the basic window with two buttons, one for selecting files, and one for sending to the central. To send to another user, the user can choose this from the menu (top-right window). Selecting files is done by clicking on the wanted file (bottom-left). The text area displays the selected files (bottom-right).

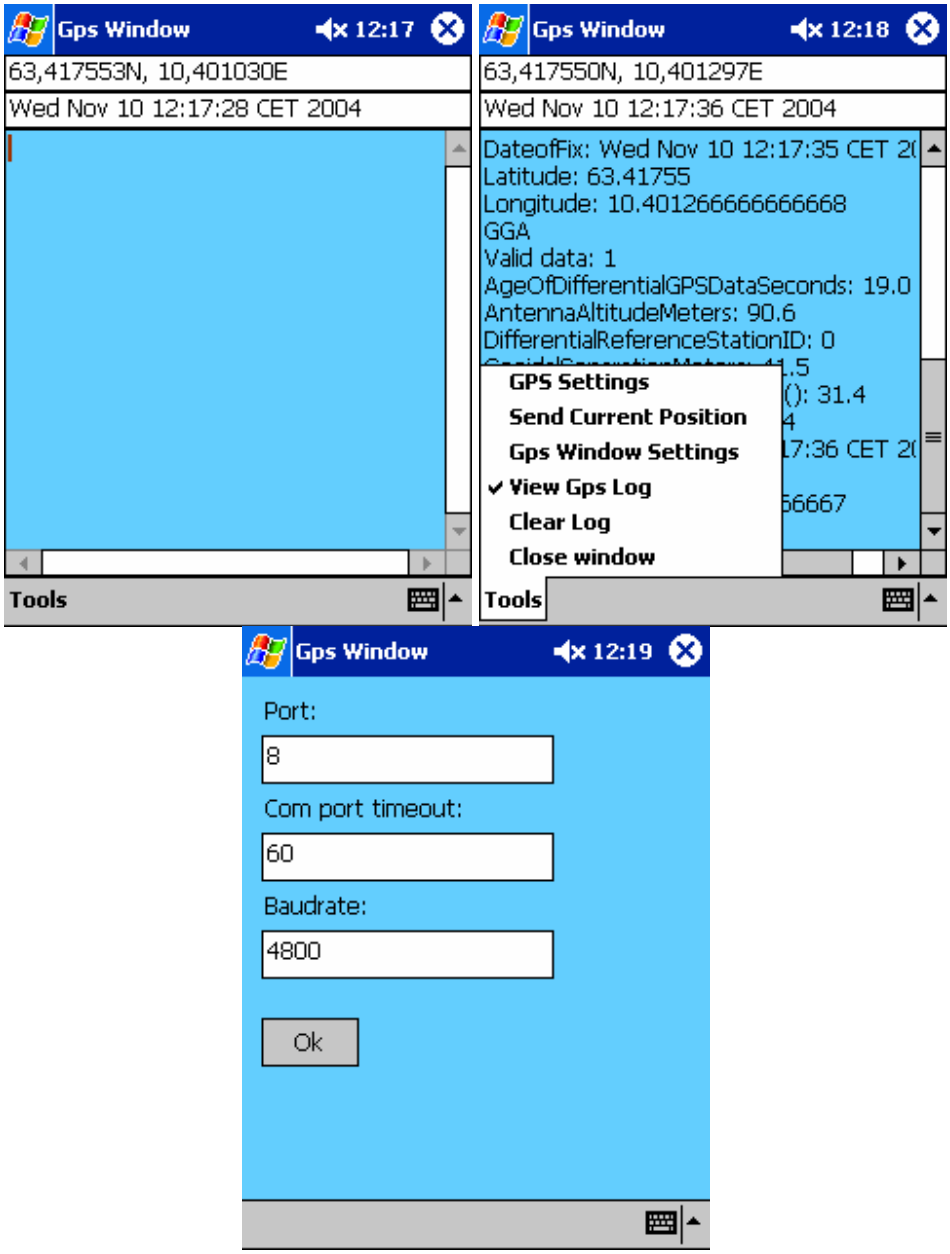


Figure B3: GPS window

The GPS window can be opened from the menu at the MainWindow (Figure B1). This window displays the current GPS location of the terminal with an associated time-stamp (top-left). The user can toggle to view the GPS log and settings, as well as sending the current position to the central from the menu (top-right). The settings are displayed and can be changed in a new dialog (bottom).

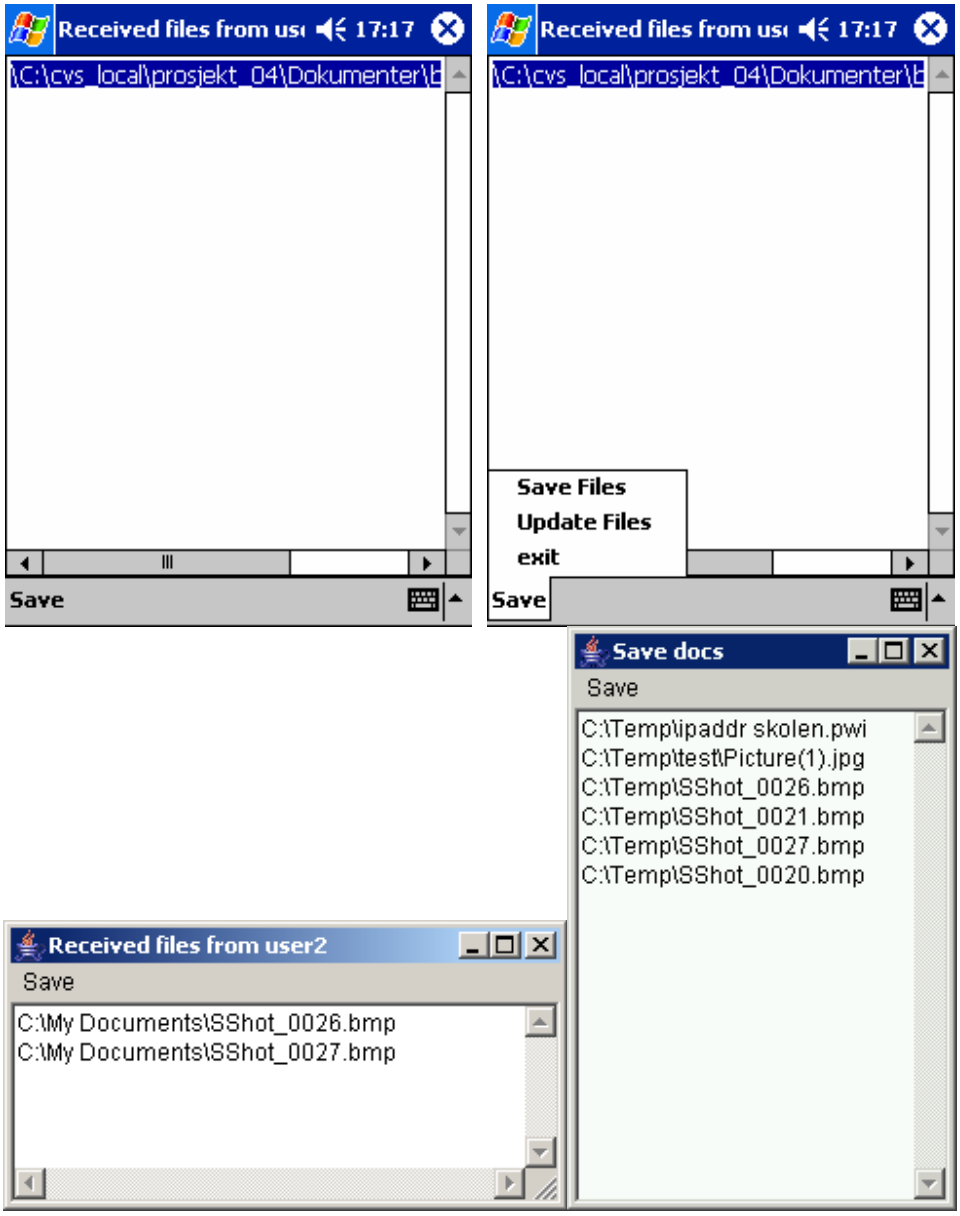


Figure B4: Received files

Figure B4 shows windows for received files. The windows on the top show the window on the PDA, and the menu for saving (Top right). On a PC (central unit) the received files windows is like bottom left, and bottom right window shows the result of the get documents transfer test in section 7.4

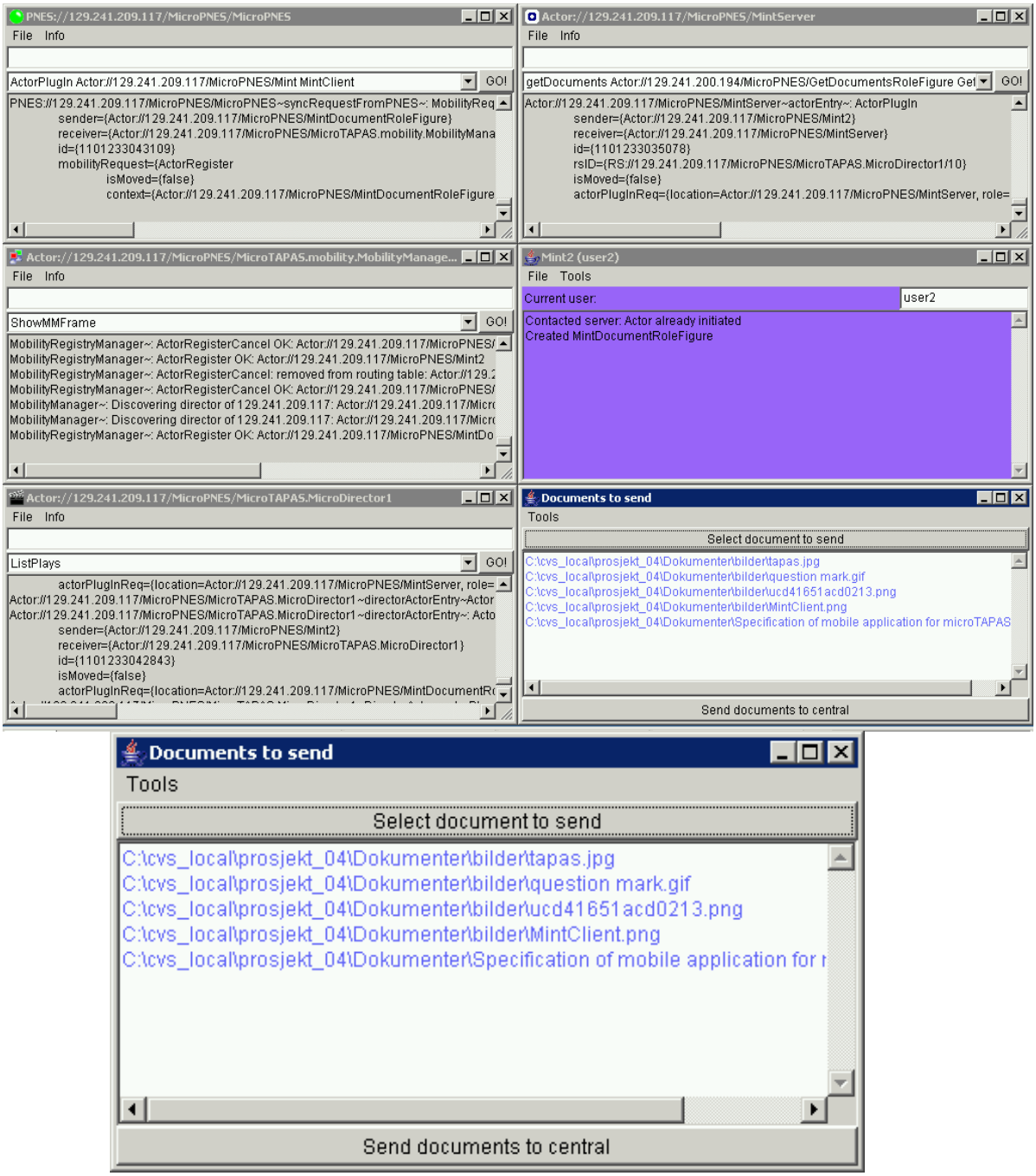


Figure B5: Screenshots from PC

The windows on the left side in the top picture, shows the debug windows for (from the top) MicroPNES, MobilityManager and MicroDirector1. On the right side is MintServer, MintClient (with user2 logged in) and the last window is for sending documents (also given at the bottom.

Appendix C: Satellite Navigation Systems

This appendix is provided to give a more thorough introduction to satellite navigation than the one given in section 2.1.

C.1 Global Positioning System

The Navstar Global Positioning System is more commonly known just as GPS. GPS is controlled and operated by the U.S. Department of Defense and has therefore several issues which will be discussed later on. As described in Figure 2 on page 8 of this report, GPS consists of three different segments.

The satellites in the space segment orbit the earth in 6 different orbital planes in an intermediate circular orbit (ICO), or medium earth orbit (MEO), which means at an altitude of 20200 km above the surface, and each satellite circle the earth twice a day. The satellites are spread out so that a receiver at any time anywhere on the surface is able to receive signals from at least 4 satellites, given that there are no obstacles between the receiver and the satellites.

Each satellite carries an atomic clock and constantly transmits the precise time according to their clock along with some administrative information, and the receiver uses this information to calculate its latitude, longitude and elevation. The radio signals is transmitted on 1575.42 (L1) and 1227.60 MHz (L2) and the signals follows thus 'Line of sight'. The L2 frequency is mainly used to transmit an encrypted signal for military use, but several vendors have developed techniques to utilize this signal even without the encryption key.

The control segment controls the satellites by tracking them and providing them with correct orbital data and clock information. In fact, according to Scientific American [10], GPS proves Einstein's Theory of Relativity. Due to the speed the satellites are traveling in, the onboard clocks run about seven microseconds slower per day than ground clocks. On the other hand, the weaker gravitational pull makes the clocks run 45 microseconds

faster, so in order to correct the relativistic errors each onboard clock has to be turned back 38 microseconds per day. There are five control stations located around the world, four that are unmanned and one master control station that controls the other ones. The master station monitors the satellite position and collects data from the satellites and corrects it if necessary via a couple of uplink stations.

The user segment is then of course all the users with a GPS receiver, both military and civilian users.

C.1.1 How GPS works

In order for the receiver to know its whereabouts on the earth surface, it has to know two things. It has to know where the satellites are (the satellites location) and how far away from the receiver they are (the satellites distance).

To know the location of the satellites, the GPS picks up two kinds of coded information from the satellites. The first is called the “almanac” data, and contains an approximate location of the satellites. This data is constantly transmitted and stored in the memory of the receiver, so that it will know the orbits of the satellites. This almanac data is updated periodically with new information.

The second kind of coded information the receiver needs is called the “ephemeris” data. This data is correction data due to leeway of the satellites. Any satellite travels slightly out of orbit so the control segment keeps track of the leeway and the master station sends this ephemeris data up to the satellites. The ephemeris data is valid for about four to six hours.

With the almanac and ephemeris data, the GPS receiver knows the location of the satellites at all times.

The next thing to figure out for the receiver is the distance to the satellites in order to determine its position on earth. The formula used for this is the simple:

Velocity x Travel Time = Distance

The radio waves travel with the speed of light minus any delay of the signal due to the earth's atmosphere. The travel time is calculated from coded signals from the satellites, which is coded with pseudo-random code. The receiver generates the same code and tries to mach it with the code from the satellite. By comparing the two codes, it knows how much it needs to shift (or delay) its own code to make a match and this is the travel time.

With the knowledge of distance and location to a satellite, the receiver knows that it is somewhere on an imaginary sphere surrounding the satellite. With knowledge of two satellites, it is somewhere on a common circle where the two spheres are intersecting (Figure C.2).

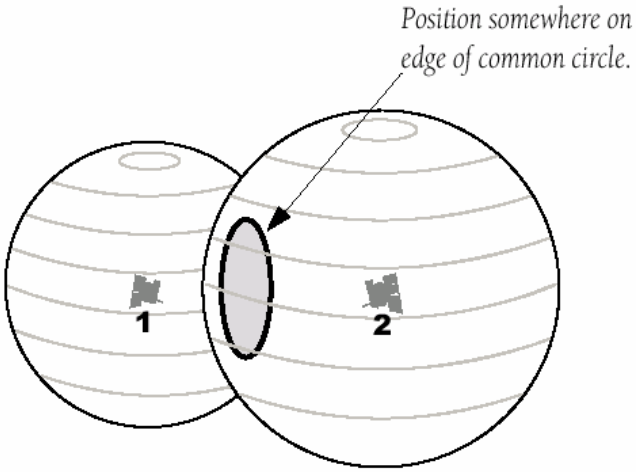


Figure C1: Determination of position, part 1 [3]

With a third satellite, the receiver will now have two points where all three spheres are intersecting. These two positions differ greatly, but if the user adds an approximate altitude, it is possible to calculate the latitude and longitude of the position (Figure C.3).

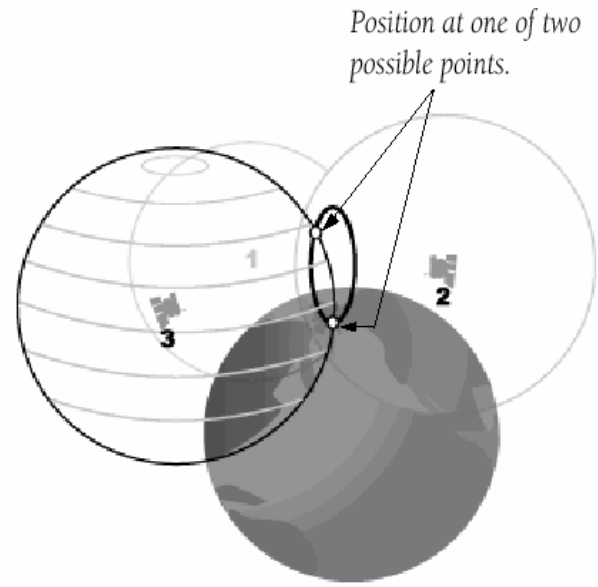


Figure C2: Determination of position, part 2 [3]

With a fourth satellite there will only be one common position, and the GPS receiver can thus calculate a three-dimensional position (latitude, longitude and altitude).

There are several sources for getting errors or position with low accuracy. Examples of these are signal multi-path due to buildings, structures or rock surface, receiver clock error, number of satellites visible or intentional degradation of the signal. Intentional degradation was originally used by the US military in order to prevent military adversaries to get an accurate GPS signal. This is now turned off, but it is fully possible for the US DoD to turn on this functionality if they find the need for it. And also, since the GPS system is fully operated by the US DoD, they have the possibility to only transmit encrypted signals and blocking out other users if needed.

C.1.2 Differential GPS (DGPS)

In order to get a more accurate position, several services have arisen. Differential GPS is a service that makes use of a known, fixed location. If two GPS receivers operate in the same area, many of the same errors apply to both. If one of the receivers is at a known, fixed location, it is possible to determine errors in the received signal from the satellites. The differences for each satellite is collected and transmitted to DGPS receivers via radio transmitters on the surface. By using this “differential correction”, the receiver can now

remove errors and improve accuracy. DGPS is normally limited to separations between users and reference stations to approximately 100 km. To carry out the same over a wider area, wide area augmentation service (WAAS) was evolved.

C.1.3 Wide Area Augmentation Service (WAAS)

Another way of improving accuracy has been implemented by the US Federal Aviation Administration (FAA), and it is called WAAS. This system is according to [OL5], intended to improve the safety and capability for civil aviation. The system consists of several reference stations distributed over a wide area that collects the GPS information and relays it to a WAAS wide area master station. These master stations use the info to develop corrections, and this is then sent to an uplink station that transmits the corrections to a geostationary satellite. This satellite broadcast the signal at the same frequency as GPS and the FAA claims that it improves the accuracy from 20 to approximately 1.5 - 2 meters for GPS/WAAS receivers. Users, on the other hand, report the improvement to be less.

C.2 Galileo

Due to the fact that GPS is owned and operated by the American military and to ensure European economies independency from other states systems, the European Union (EU) has founded a project called Galileo [11]. It is designed to be the first satellite positioning and navigation system designed for civilian purposes. Galileo is a cooperative project between the EU and the European Space Agency (ESA), where EU is responsible for the political dimension and high level mission requirements and ESA handles definition, development, and in-orbit validation of the space segment and related ground element.

The EU and ESA want to ensure that Europe has a full role in the development of the next generation of Global Navigation Satellite System (GNSS), and Galileo is their contribution. Galileo will consist of totally 30 satellites, and is according to the plans going to be more accurate than GPS. The plan is to make Galileo interoperable with both GPS and the Russian GLONASS system explained later on, and this will further increase the accuracy of the system.

As a precursor to Galileo, the European Geostationary Navigation Overlay Service (EGNOS) has already been launched. EGNOS consists of three geostationary satellites and a network of ground stations and augments the existing military satellite systems of the US and Russia. In addition to transmit corrected signals to both GLONASS and GPS, the EGNOS system informs the users of errors in position measurements. EGNOS will also do the same thing to correct Galileo signals when this system is operable.

The Galileo project also includes nations outside of Europe and the EU. India, China and Canada are among the contributors [12], and this makes the project outward-looking compared to GPS. There are other differences between Galileo and GPS too. There will be some differences in modulation, frequencies used and codes, but the main difference will be on operational level. GPS is the property of a single nation, military operated and currently free of charge. Galileo will be multinational, operated by a public-private partnership and will also include a commercial service, where the users can pay to get higher accuracy, higher data rates or extra messages.

According to the Galileo website [OL6], Galileo is to be fully operable in 2008, and the first transmissions will according to plan take place in 2005.

C.3 Other systems

The Russians have developed their own satellite navigation system, called GLONASS [OL15], and launched the first operational satellite in 1983, five years after the US. The system has been suffering from lack of foundation from the Russian government and in the time of writing (September 2004) only 10 out of 24 satellites are operational. The situation in Russian economy is improving, and there are plans to have the system back in full operation in 2007.

China has also developed an independent satellite navigation system called the Beidou navigation system. The first satellite was launched in 2000 and the third in 2003. This system differs from the other systems by not being global and that it is covered by only

three geostationary satellites. GlobalSecurity.org [OL2] speculates that this system is compatible with the WAAS system mentioned in 0, and this could enable China to continue to use the American GPS system, even in the face of American efforts to deny GPS to adversaries in wartime.

Appendix D: J2ME

This appendix is provided to give a more thorough description about Java 2 Micro Edition and the process behind the development of this technology.

J2ME as a technology was launched in 1999, but the evolution has been going on since the early 1990s in different projects. Now J2ME is one of three editions of the Java 2 platform as seen in Figure D.1.

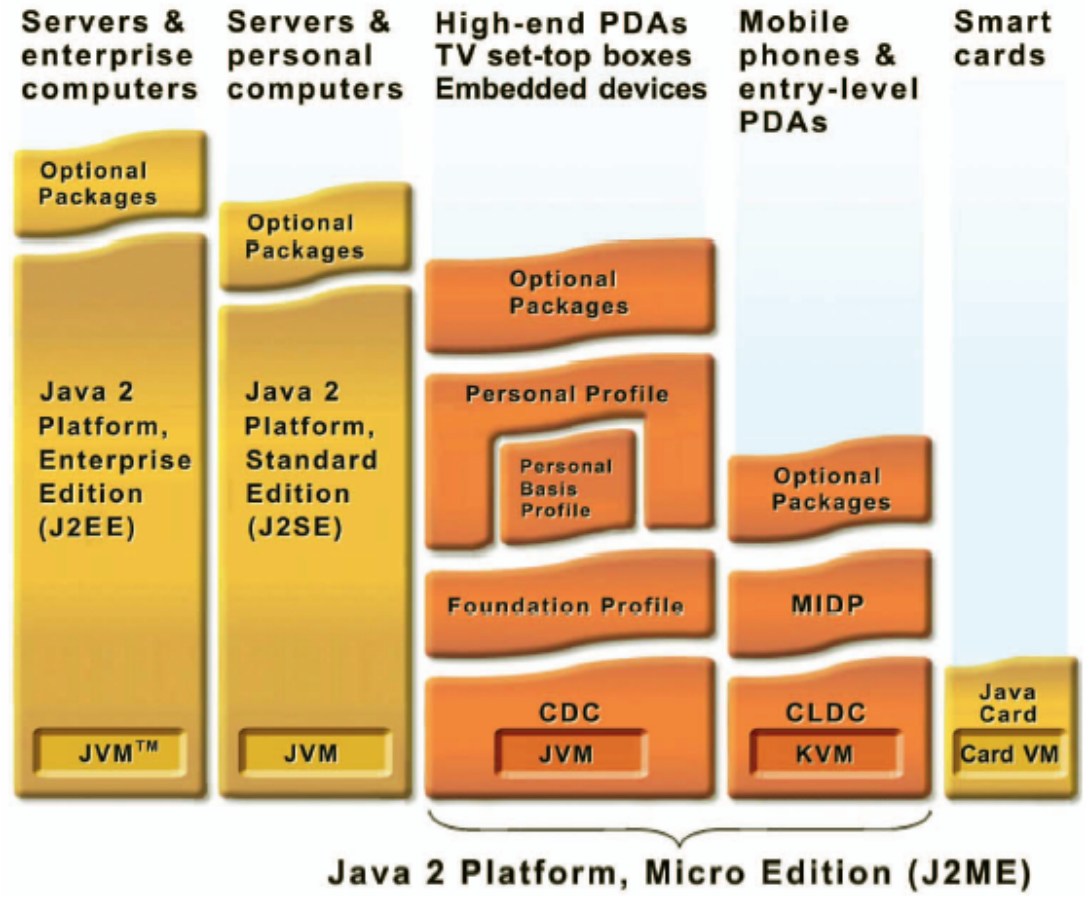


Figure D1: Java editions [OL11]

The fourth technology in the figure, Java Card (actually an older technology, launched in 1996), is an independent Java platform based on smart cards.

As seen in Figure D.1, J2ME is divided in two separate configurations [17], [OL12]:

Configuration	Device Examples
Connected Limited Device Configuration (CLDC)	Cell phones and simple, low-end PDAs. These devices typically have 16- or 32-bit CPU and somewhere between 128 KB and 512 KB memory available for the Java implementation.
Connected Device Configuration (CDC)	More capable devices with better network connectivity, e.g. TV set-top boxes, in-vehicle telematics systems and high-end PDAs. These typically have 32-bit CPUs and at least 2 MB memory available for the Java implementation.

Table D1: J2ME configurations

The difference between CLDC and CDC is that CLDC was designed around limited memory requirements and CDC was designed to achieve as much J2SE compatibility as possible within constrained device resources. CDC 1.0a is based on J2SE 1.3.1 and CDC 1.1 is based on 1.4.2.

The KVM in the CLDC configuration stands now for Kilobyte Virtual Machine and indicates that it only uses a couple of kilobytes of memory. Originally the 'K' stood for Kuaui, which was a codename for the project, but it was changed to Kilobyte since this was the design goal for use of memory.

This project used the CDC configuration of J2ME, and this will thus be the focus in this report. J2ME and especially CLDC is evolving rapidly as new technologies arrive, so for info on this configuration, see [17] (or preferably a newer book..) or online references like [OL3], [OL9] and [OL10].

D.1 Connected Device Configuration (CDC)

CDC was, as mentioned earlier, designed around two major goals; J2SE compatibility and support for resource constrained devices. This allows developers to leverage their investments in J2SE technology, including libraries, tools and skills, while still allow device vendors to offer a feature-rich Java application environment that can support

mobile enterprise applications with security [OL12]. The CDC configuration is backward compatible with CLDC, which means that applications designed for CLDC also can run on CDC, but not necessarily the other way round.

CDC uses class libraries that have been optimized for small memory environments, and to save resources some J2SE-based class libraries have modified interfaces and some have been removed entirely. This means that it is possible to use an ordinary J2SE Integrated Development Environment (IDE) tool and a regular java compiler (e.g. javac from Sun). But the developer has to pay close attention to the API for the profile that is targeted, to not use a class that is not supported.

Some annoying differences might be useful to know about. In the java.awt.Color class, for example, the static Color objects have for some reason changed from uppercase to lowercase (e.g. Color.BLACK is equivalent to Color.black). This can lead to some unwanted errors and hours of debugging, and the J2SE compiler does not compile the lowercase version.

Sun has now released a fully compliant Java Virtual Machine, called Connected Device Configuration HotSpot Implementation. This replaces the earlier virtual machine known as CVM, and indicates that this now is a fully part of the HotSpot VM family of Sun known from J2EE and J2SE. However, this JVM implementation is currently only available for Linux, Solaris and VxWorks, which means that this cannot be used by i.e. Microsoft's Windows CE PocketPC operative systems. See appendix A.1 JVM for PDA, for more info about JVM and PDA.

D.1.1 Profiles

Figure shows the available profiles for the CDC configuration, and these are described in Table D.1 [OL12]:

Profile	Description
Foundation Profile	Foundation Profile is the most basic profile. In combination with the class library provided by CDC, Foundation Profile

	provides basic application-support classes such as network support and I/O support. In particular, it does not include any support for graphics or GUI services.
Personal Basis Profile	Personal Basis Profile provides a structure for building lightweight component toolkits and support for the Xlet application programming model. In addition, Personal Basis Profile includes all of the Foundation Profile APIs.
Personal Profile	Personal Profile provides full AWT support, applet support and limited bean support. In addition, Personal Profile includes all of the Personal Basis Profile APIs. Personal Profile also represents the migration path for PersonalJava technology.

Table D2: CDC profiles

These profiles give a product designer a flexible way to design a product the most efficient way according to what the product is supposed to do, instead of providing separate APIs.

Optional packages are also included under the CDC technology to give additional choices for supporting specific technologies. Currently these packages are for RMI and JDBC. Several optional packages that are made for CLDC can also be used with CDC.

D.2 Java Community Process (JCP)

The development in the Java world is lead and sponsored by Sun Microsystems, and the Java technology is evolved though the Java Community Process (JCP). JCP is an open, community-based standards organization with a formal process for defining and revising Java technology specifications [OL9]. While concentrating on the process of how J2ME will be developed, Sun allows industry leaders and experts to create flexible groups of like-minded individuals and companies to develop and define general configurations, and specific profiles, of J2ME [17].

JCP is “by-invitation-only”, meaning that anybody that wants to join the process can’t, only those invited by Sun. This of course leaves Sun in control of how the process

evolves and Sun Microsystems retains all Java-related trademarks, and remains the ultimate authority of the Java platform. On the other hand, this way of selecting members keeps the process controlled and ensures that people with relevant expertise develop the product.

There are four main steps in the JCP [17]. Firstly a specification is submitted by members of the Java Development Community to the Process Management Office (PMO) within Sun. Is the specification approved by the Executive Committee (EC) for development it will receive a Java Specification Request (JSR) code. Secondly, an expert group is formed to develop a first draft. This expert group gets feedback from the Java community on the draft and can update the draft if needed. Together with the EC the expert group decides whether the specification is ready to proceed to public draft. Thirdly, the draft specification is reviewed by the public. That means that everyone with Internet access and wishes to comment or participate on the draft can do so. The expert group uses this feedback to further refine the specification. Once the EC is satisfied, the specification receives a final approval and the expert group is disbanded. The fourth step is maintenance, and this means that the specification, reference implementation and compatibility tests are maintained and developed if needed.

This dynamic approach has proven to be more efficient than the traditional standards development cycle. The whole process is set (by Sun) to take between 150-250 days, but still some are worried that this is to slow and that Java can loose terrain to other technologies, like Microsoft's .Net that is totally controlled by one corporation.

Appendix E: CD

Contents:

Folder	Description
/Documents	This report in word and pdf format, the project assignment and a project plan made early in the process.
/Javadoc	Java documentation of the code.
/MicroTAPAS	Bootstrap directories for PC and PDA and Java classes to be ported to a webserver.
/ Screenshots	Screenshots used in this report, plus additional screenshots and mobile video.
/Source_code	Source code for both java and C++ code.
/UML	Rational Rose RT UML files for this project.