Foreword

This report documents the project assignment "Plug-and-Play services for wireless PDA and Java-enabled phones". The project is performed at the Norwegian University of Science and Technology (NTNU) from September to December 2003.

The work with this project has been challenging, partly because of the complexity of TAPAS, and due to work with new and emerging technologies. However, I have gained much experience and knowledge from this work that I will have advantage of in the future.

I would like to thank my supervisor Mazen Malek Shiaa, and professor Finn Arve Aagensen for their advice and assistance through the project.

Trondheim, December 2003.

Marius Dalsmo

Contents

Figures and tables

Abbreviations

2G	Second generation mobile communication technologies
3G	Third generation mobile communication technologies
API	Application Programming Interface
AWT	Abstract Window Toolkit
CDC	Connected Device Configuration
CDMA	Code Division Multiple Access
CLDC	Connected Limited Device Configuration
COD	Code on demand
DARPA	Defence Advance Research Agency
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
FA	Foreign Agent
GPRS	General Packet Radio System
GSM	Global System for Mobile communications
GUI	Graphical User Interface
HA	Home Agent
HOL	Head-Of-Line
IEEE	Institute for Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IN	Intelligent Network
J2ME	Java 2 Platform, Micro Edition
J2SE	Java 2 Platform, Standard Edition
Java RMI	Java Remote Method Invocation
MA	Mobile Agent
MIDP	Mobile Information Device Profile
PaP	Plug-and-Play
PAS	PaP Actor Support
PNES	PaP Node Execution Support
QoS	Quality of Service
REV	Remote Evaluation
SDK	Software development kit
SMS	Simple Messaging Service
TAPAS	Telematics Architecture for Plug-and-Play Systems
TINA	Telecommunication Information networking Architecture
UMTS	Universal Mobile Telecommunications System
VM	Virtual Machine
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network

Summary

The task of this project comprises investigation and experimentation of Plug-and-Play support functionality and services using J2ME for wireless PDA and mobile phones. The background for this task is the TAPAS project, which has elaborated an architecture concept providing Plug-and-Play support functionality for network-based service systems. The TAPAS concept is based on a theatre model, where generic actors are able to perform roles defined in corresponding manuscripts. The concept must be suitable for next-generation wireless systems in order to be applicable for the future.

The reliability and flexibility of the TAPAS concept for nowadays and near-future services and applications is studied. The available TAPAS platform for wireless devices has been used evaluating the concept. Wireless environments impose the need for reliable and flexible mobility management schemes and handling of the highly dynamic connections. Proposed solutions have been implemented, trying to improve the performance of the platform according to key functionalities needed in dynamic wireless environments. Demonstration and tests of the improvements have been carried out in order to show the suitability for these proposals.

1 Introduction

This chapter will give an introduction to the project. Its background, approach, goals and demarcations will be defined. At the end of the chapter the structure of the rest of the report will be given.

1.1 Background

This project deals with subjects strongly related to TAPAS (Telematics Architecture for Plug-and-play Systems). TAPAS is a research project that is supported by The Norwegian Support Council and The Norwegian University of Science and Technology, and has been running at the Department of Telematics since 1997. The vision of the TAPAS project aims at developing an architecture concept 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 [AAGE03].

Another objective of the TAPAS project is to demonstrate the feasibility of the results by use of trial implementations of the architecture. By implementing various features of the architecture concept, it will show the implementation possibility and validate the feature applicability. The objective is not to develop a complete executing platform, but to gain experience and knowledge of the concept by setting the various features coming from the specified requirements in a context related to totality.

The project has so far dealt with many of its predetermined challenges and many of its goals have been achieved. The work has resulted in four main architectures: the basic architecture, the mobility handling architecture, the dynamic configuration architecture and the adaptive service architecture [MALE03]. All these architectures require a support system necessary for software deployment, deployment, execution and management. Also, generic user functionality is required, to enable the flexibility features of the system. This support system is denoted the TAPAS platform.

The support system has been implemented in Java 2 Standard Edition (J2SE) with the use of Java Remote Method Invocation (RMI), and a scaled down version has been implemented in Java 2 Micro Edition (J2ME) with the use of communication sockets. The J2ME implementation is intended for small devices with limited capabilities, such as memory and processing power. Accordingly, TAPAS functionality is available for PCs using J2SE and wireless handheld devices using J2ME.

PhD studies and Master degree theses have dealt with different aspects within the TAPAS concept and a number of publications have been produced that have brought solutions and proposes to the existing architecture.

1.2 Approach

The task of this project assignment comprises investigation and experimentation of Plug-and-Play (PaP) support functionality and basic services using J2ME for wireless PDA and mobile phones. The task emphasizes on reliability, flexibility and

performance issues of wireless systems, and how suitable the TAPAS concept might be for the future.

Before any real work with the support functionality could be started, a basic understanding of TAPAS was needed. Therefore, some effort was spent on studying the TAPAS concept and its support for wireless devices. Special attention was paid at support functionality that is necessary in wireless environments (e.g. terminal mobility).

Related technologies has also been studied to be able to get a more complete understanding of what aspects that has to be considered when evaluating the suitability of the TAPAS concept. The focus has been on solutions similar to TAPAS, wireless technologies, the J2ME platform, and next-generation mobile applications.

With this background information, the investigation and experimentation was started. The developed TAPAS support platform for small devices was used as reference when studying the support functionality. The focus has been on functionality that should be available when operating in wireless environments. Demonstration and evaluation of already available functionality has been done, and improvements have been proposed.

A test was carried out in order to demonstrate the suitability of the proposed improvements. Different configurations and wireless environments have been used to show how these affect performance issues like overhead traffic.

1.3 Demarcations

This report has no intention of giving the reader a full introduction to all aspects of the TAPAS architecture. The focus is on mobility and the handling of wireless and small devices.

The reader should be familiar with the basic concepts of telecommunication systems and the Java programming language.

1.4 Structure

Chapter 2 gives an overview of related technologies to TAPAS and the project assignment in particular. Some examples of next-generation mobile applications are given.

Chapter 3 introduce the TAPAS concept and its support functionality. The TAPAS platform using J2ME is also presented and used as a basis for further discussion.

Chapter 4 presents a detailed description of the work and achievements from the investigation of TAPAS and its support for wireless environments. In the end of this chapter, some of the faced problems and challenges during this work are described.

Chapter 5 describes some test scenarios and its results, which were carried out in order to show the suitability of some of the work that is described in this report.

Chapter 6 concludes the report by summarizing the overall results and achievements. Some proposals for future work are also given.

2 Related technologies and trends

This chapter will introduce some technologies and trends that are related to TAPAS, and the project assignment in particular. First, a brief introduction to the area of interest is given, and then some architectures and solutions having similarities to TAPAS are discussed. Thereafter, some wireless technologies are presented and compared with each other. Issues concerning wireless devices and a software development platform (J2ME) for such devices and its current features are described. In the end of the chapter, some trends in next-generation mobile applications will be given.

2.1 Introduction

The increasing heterogeneity and complexity in today's tele-services and networks systems complicates development, installation, deployment, operation, management and maintenance. This is not only due to technology, but also due to the large number and widely differing participants in the area. Without a supporting architecture, the given tasks can be time-consuming and performed in an inefficient way.

Plug-and-Play (PaP) for telecommunications means that the hardware and software "parts", as well as complete network elements, that constitute a communication system, have the ability to configure themselves when installed into a network (to plug) and then to provide services (to play) according to their own capabilities, the service repertoire and the operating policies of the system [AAGE99].

The task of the project assignment is focused on such support in wireless environments in particular. Wireless technologies have recent years become more common and widely used, and supporting architectures should therefore apply for such environments as well. The handling of wireless components implies use of additional support functionality because of the nature of the environments where these components operate (e.g. mobility, variable radio recourses, delays, etc.).

TAPAS aims at developing an architecture concept that provides solutions to these problems and is explained in more detail in the chapter 3. In the following sections, some related architectures and solutions are presented. Also, related issues concerning wireless components and technologies are presented.

2.2 Similar architectures and solutions

PaP functionality increases network intelligence, here defined as *flexibility* in tele-service execution and introduction of new tele-services. Network intelligence is needed to cope with the increased accessibility and a richer supply of services that impose stringent demand on the network. Several architectures and technologies have been introduced to improve network intelligence. Examples of such solutions are Intelligent Network (IN), Telecommunication Information Networking Architecture (TINA), Mobile Agents and Active Networks. It is out of scope of this report to explain all these solutions, but TAPAS have been based on similar ideas used by some of these.

2.2.1 Mobile code

TAPAS is based on the principle of *mobile code.* Mobile code is software that is transmitted across a network from a remote source to a local system and is then executed on that local system. The local system is often a personal computer, but can also be a smart device (such as a PDA, mobile phone, Internet appliance, etc.). Mobile code architectures can be designed using one of the following paradigms; *remote-evaluation* (REV), *code-on-demand* (COD) or *mobile agent* (MA). REV means that a node can send its code to a resource, where the code can cooperate with other pieces of code, and a result of this cooperation can be sent back. With COD a node can request code from resources and use the code segments locally. In the MA paradigm the code physically migrates between nodes in the network along with the necessary state/recourses required to perform a task.

Mobile agent architectures have program instances that are able to autonomously migrate in a network within their own control and perform tasks on machines that provide agent-hosting capability. State (i.e. data state and execution state) is transported within the mobile agent. There are a number of different mobile agent architectures available today. IBM Aglets is one example that provides a software development kit (SDK) for programming mobile Internet agents in Java [AGLETS].

Active networks are another approach using mobile code. An active network is a network where the transporting components (i.e. routers) are able to execute arbitrary code. This code is provided in some systems by special network packets (active packets) that can be injected by normal users. In comparison to mobile agents, which operate on the application layer of the OSI model, the active networks are targeted at the lower layers. A DARPA[1] funded research project has encouraged several academic institutions to work with active networks [DARPA]. One result of this is Active Node Transfer System (ANTS), developed at MIT, which has been used experimenting with the concepts of the active network solution [ANTS].

In TAPAS the COD paradigm is used, which means that code is downloaded when needed. This code can be part of the generic support system that enables nodes to run TAPAS compliant software, or describes behaviour (*manuscript*) that may be part of an application (*play*) running in several nodes. This can be compared to active networks, where new functionality and support can be plugged in network components. However, the active network approach are targeted specially at transporting components, and do not support the same flexibility that TAPAS aims at providing. Compared to mobile agents, there are not that many similarities, but the TAPAS concept also supports that autonomous entities travels between nodes. In TAPAS this is defined as *actor mobility*, and will be explained in more detail in the next chapter. However, as argued in [MALE02], moving actors between nodes is just movement of executing functionality. Example of use of this mechanism is re-instantiation of functionality when experiencing QoS deterioration.

2.2.2 Autonomic computing

The solutions mentioned above focus on flexibility in execution of services and efficiency introducing new services. According to [AAGE03], research in the area is

[1] U.S. Department of Defence's (DoD) Defence Advance Research Projects Agency

changing its focus to adaptability and evolution of such services. These issues are seen as crucial to be able to cope with the development trends in the industry. Due to the increased networking complexity and heterogeneity there is a need for having services and systems that automatically adapts to its environment with less human intervention. An example of research in this area is IBM's autonomic computing project [AUTONO]. This is an approach of defining "computer systems that can manage themselves given high-level objectives from administrators" [KEPH03]. The term "autonomic computing" derives from the body's autonomic nervous system, which regulates the body's key functions, thus freeing our conscious from being occupied with controlling these functions. In comparison, the IBM solution aims at building computer systems that regulate themselves much in the same way as the autonomic nervous system. The systems should be self-managing, in the sense of these four key characteristics; self-configuration, self-optimization, self-healing and self-protection. Self-configuration means that the system dynamically should adapt to changing environments. Self-optimization means tuning of resources and balancing of workloads for efficient use of resources. Self-healing systems should detect and act on error situations for recovery. Self-protecting means that the system should foresee, protect and detect attacks and identify a possible attacker. Similar projects are running at Hewlett-Packard Labs, which is referred to as planetary computing.

TAPAS and the autonomic computing project have obvious similarities. Even if the autonomic computing approach focuses mainly at the computer domain, both approaches aims at developing an adaptive and self-configuring architecture or system that are able to cope with increasing complexity and heterogeneity in networking. The methods for solving these issues are slightly different, and separate these projects from each other.

2.2.3 Zero configuration networking

A working group within the Internet Engineering Task Force (IETF) called Zeroconf aims at proposing a vendor independent network protocol that enable small IP-based networks to automatically configure themselves [ZEROCO]. When components are put together in a network, they should be able to address each other and detect available services (e.g. printer). Proprietary network protocols for this purpose are available, but are not designed for cross-platform support. The proposal is based on IP because it is the most popular platform independent network protocol. However, the purpose is to enable such support without having available DHCP, DNS and Directory servers in the network. The simplest example of such a network could be two laptop computers directly connected together with an Ethernet cable that are automatically configured. Apple Rendezvous is an open protocol that has been submitted to the Zeroconf working group as part of the standardization progress [APPLE]. With this protocol, devices are able to automatically detect each other, and locate and advertise services among each other.

The TAPAS architecture concept is not limited to a specific network protocol. TAPAS should be able to apply for several types of networks and protocols. The Zeroconf approach has a more specific goal, which is support for IP-based networks. However, there are some similarities in which both enable service functionality over heterogeneous platforms.

2.2.4 Mobile IP

Personal mobility is an important mobility feature that mobile telecommunication systems should have elaborated schemes for. Mobile systems available today (GSM) and in the near future (UMTS) provide mobility for user and their terminal to seamlessly roam in and between enterprise-domains. While this feature is well established, the integration of application and service platforms introduce new challenges. Here, personal mobility is centered at the user and its personal content. In order to provide integration of these platforms, concepts from technologies like Mobile IP can be utilized.

Mobile IP provides solutions for mobility in IP-based networks. The interest in this area has increased in resent years, resulting in many research and development projects. Mobile IP (RFC2002) is a standard proposed by a working group within the Internet Engineering Task Force (IETF). The standard allows a mobile node to be associated with two separate IP addresses: one home-address and one care-of-address. The care-of-address is used to address the current location of a mobile node and changes as the node roams in the network. The home-address is a well-known and static address that remains the same at all time. In order to manage the routing of packets, the Mobile IP standard introduces a home agent (HA) and a foreign agent (FA). The home-address is associated with HA, and this entity is situated at the home network of the mobile node. As the mobile node leaves the home network, the mobile node should be given a care-of-address by a FA situated at the foreign network where the mobile node moves. The mobile node will inform the HA about its new care-of-address, so packets can be tunnelled[2] to the new location. This will allow sending nodes to use the static home-address associated with the node, even if the location of the receiving node is changed.

In TAPAS, the mobility handling schemes for personal mobility is similar to the Mobile IP solution. Here, user with its personal content and its session move along the user as it changes its access point as long it is accepted by the visiting domain. The mobility is handled by agents in the home and visiting domain, analogous with the home and foreign agents in Mobile IP.

2.3 Wireless technologies and devices

In resent years, significant technological advances are taking place in the areas of wireless devices and wireless communications. Wireless technologies are becoming an affordable, convenient alternative to fixed network covering telephone conversations to Internet connectivity and service access, adding mobility and ease of access unavailable in the fixed networks. The introduced complexity added by mobility and the dynamic operating environment for any accompanying services, must be well handled by an architecture that should be able to cope with future wireless technologies.

2.3.1 Comparison of wireless technologies

Two of the fastest growing wireless technologies today are wireless local area network (WLAN) technology and wireless personal area network (WPAN)

[2] Tunnelling means that a packet is encapsulated in another packet. The destination address of the encapsulating packet will be used in routing of the packet through the network.

technology. These are not in direct competition with each other, because their standards are designed for different purposes.

WLAN provides network communication over short distances using high frequency radio signals or infrared light instead of traditional network cabling. This allows for a new degree of freedom for their users within rooms, buildings and production halls. Also, WLAN technology will give complementary access to cellular networks.

WPANs are small, unwired ad-hoc networks that provide communications within a few meters. It can be used for connecting mobile devices (e.g. mobile phones, PDAs, wireless printers, pagers, etc.) carried by users to other mobile and stationary devices. WPANs are typically made possible by use of radio technology and have the capability to enable devices to autonomously detect and acquire one another. The WPAN communication involves small mobile devices, which has to be considered in terms of power-consumption, connectivity awareness and resource utilization.

There are a number of WLAN standards specified (see Appendix A), but the currently most widespread and dominating one is IEEE[3] 802.11b. It operates in the 2.6 GHz frequency band and supports data rates up to 11 Mbit/s. In this project assignment, equipment supporting this standard has been used when investigating and experimenting with PaP support functionality. The most recently approved WLAN standard, IEEE 802.11g (July 2003), is a high rate extension of 802.11b. The 802.11g standard allows data rates up to 54 Mbit/s in the 2.4 GHz frequency band. With an extension of 802.11b, 802.11g products will be backward compatible with the 11 Mbit/s 802.11b standard.

The Bluetooth technology, which is the most common WPAN standard, operates in the 2.4 GHz band and offers speeds up to 1 Mbit/s [BLUETO]. It comprises hardware, software and interoperability requirements and is designed for low power consumption, with a short range (approximately 10 meters) and low-cost transceiver microchip in each device. The latest adopted specification, Bluetooth v1.2 (November 2003), has enhanced radio interface[4] to avoid interference with other technologies sharing in the unlicensed 2.4 GHz band (e.g. IEEE 802.11b, IEEE 802.11g and cordless phones). It has backward compatibility with the previous Bluetooth v.1.1 specification.

To summarize, WPAN technology are principally designed for small range, ad-hoc connections, supporting devices with critical energy demands. WLAN is used for wider range and more robust connections (compared to WPAN), with no special requirements for power consumption. WLAN supports higher data rates, thus consuming more energy.

2.3.2 Mobile and wireless devices

In order to utilize the enhancements in the wireless technologies, mobile and wireless devices need to follow the evolution in supporting higher transmission speeds, and being capable of processing new and demanding real-time multimedia applications

[3] Institute for Electrical and Electronics Engineers (IEEE)
[4] Using adaptive frequency hopping (AFH) that takes advantage of available frequencies.

(e.g. video and voice). The future devices will be more powerful, less heavy, and compromise new interfaces to the user and the network. There are already a number of mobile and wireless devices available, but most of these are limited in order to meet the prospects of the future.

The types of mobile and wireless devices differ in size, shape, weight, display, memory, user interfaces, and computing power. Examples of devices that already exists or will be available in the future are: *sensors* (simple transmitters), *embedded controllers* (controllers in appliances), *pagers* (receivers with tiny display), *mobile phones* (voice, text, multimedia support with small display), *smart-phones* (a mobile phone and PDA combination), *PDAs* (runs simple office software), *palmtop/pocket computers* (handheld, small computers), *notebooks/laptops* (battery driven, normal computers).

Some of the problems that have to be dealt with in order to meet the requirements for the future are the issues of energy supply and user interfaces. Mobile devices are battery powered, and support for higher bandwidths and increasing features in the devices, consume more energy. Also, handheld mobile devices have small user interfaces, which make user interaction less convenient. New ways of interacting with the user will probably emerge (e.g. touch sensitive screen and voice recognition).

2.4 Wireless development with J2ME

In order to support small and wireless devices in the TAPAS platform, an appropriate Application Programming Interface (API) suitable for such devices have to be used in the development. Mobile phones and PDAs are now able to run mobile applications and support programming environments like Java. This subchapter will describe J2ME, the Java Platform for consumer and embedded devices (e.g. mobile phones, PDAs, TV set-top boxes, etc.), and its latest developments, which is used in the TAPAS platform suitable for such devices.

2.4.1 The J2ME Architecture

The J2ME architecture consists of configurations, profiles and optional packets for building device specific Java Virtual Machines (VMs). With this structure, J2ME address the diversity in consumer devices such as form, features and functions, by modularity and customizability. A configuration tries to customize a Java runtime environment for a group of devices sharing the same capabilities. The configurations describe the details of which libraries that are kept from the J2SE platform, or classes that are newly created. The purpose of the profiles is to add domain-specific classes to a configuration to fill in missing functionality and support specific uses of a device. The illustration in Figure 2.1 [SUN1] shows how configurations and profiles together constitute a high-level API.

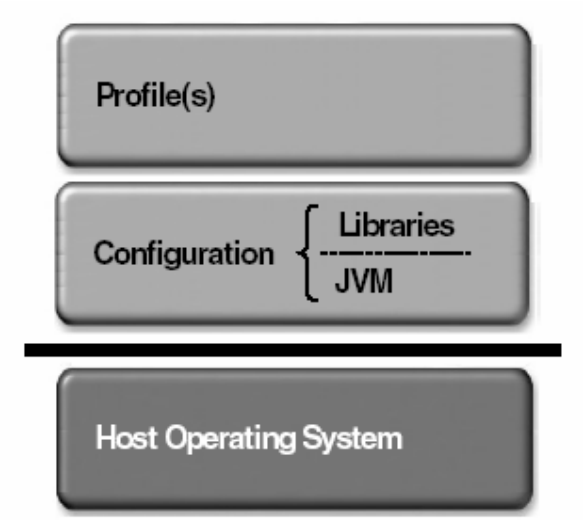

Figure 2.1 - The high-level architecture of a typical J2ME device

Figure 2.2 [SUN2] shows an overview of the J2ME platform, and its relation to other Java platforms.

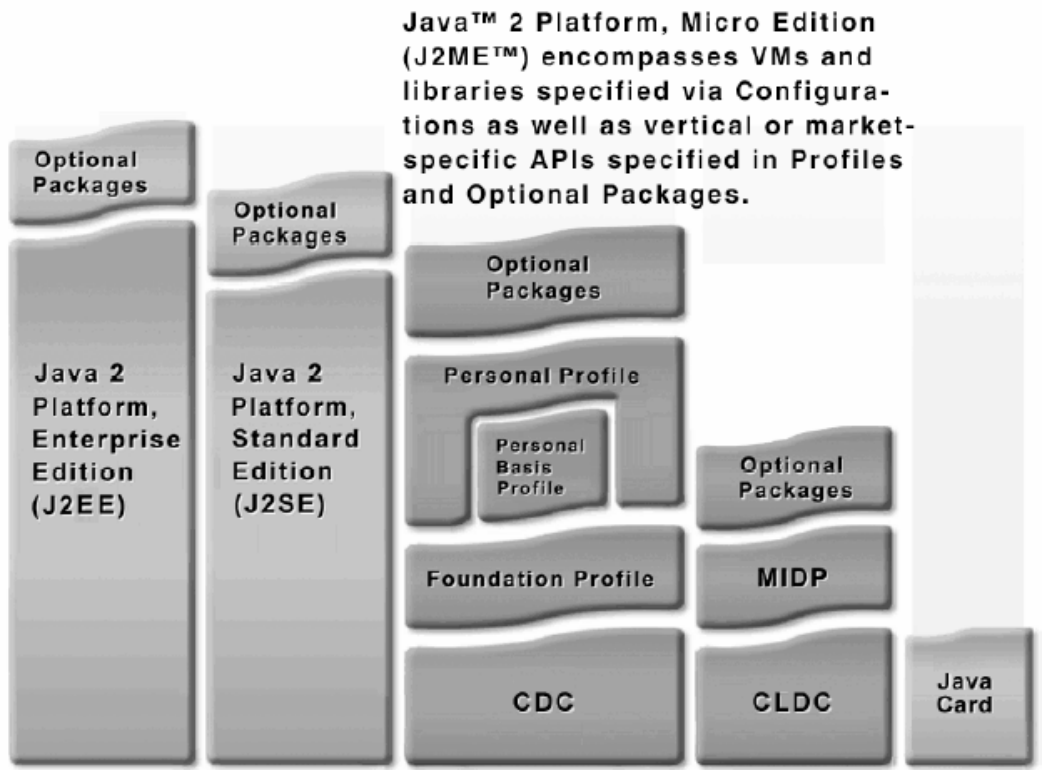

Figure 2.2 - The Java 2 Micro Edition and its relation to other Java technologies

2.4.2 Configurations

J2ME are currently targeted at two categories of products, each having a defined configuration by the Java Community Process[5] (JCP). The two J2ME configurations available are the Connected Device Configuration (CDC) and the Connected Limited Device Configuration (CLCD). Both configurations are designed to work in limited memory environments, but CDC was designed to achieve as much J2SE compatibility as possible within these environments [SUN2].

The CLCD configuration is targeted at a wide variety of small personal and mobile devices that have resource constraints. Because of the diversity of such devices, the configuration only has minimum requirements to memory. Thus, CLCD target devices may have considerably more memory than required. According to [SUN1], the devices targeted by the CLCD configuration have the following general characteristics:

- at least 192 kilobyte of total memory budget available for the Java platform,
- a 16-bit or 32-bit processor,
- low power consumption, often operating on battery power,
- connectivity to some kind of network, often with a wireless, intermittent connection and with limited bandwidth.

These characteristics apply for mobile phones and similar devices.

Some of the limitations of this configuration compared to J2SE are that it has no custom class loaders, no thread groups and daemon threads, and no standard `java.net` package is included. As we will see in chapter 3, these limitations were of vital importance when choosing a configuration for TAPAS for small devices.

There are several VM's available that implements CLCD. One of them is Sun Microsystems' KVM (Kilobyte VM), which is designed to work on devices that only have kilobytes of memory available for the Java runtime environment.

The CDC configuration is targeted at devices that have more memory available for the Java platform, more processing power, and greater network bandwidth. This will apply for emerging higher-end next-generation communication devices, like smart phones, PDAs, communicators and home appliances. In order to be as J2ME compatible as possible, several J2SE class libraries have been adopted and optimized for resource limited environments. According to [SUN2] the configuration fit comfortably within a memory budget of 2 MB of RAM and 2 MB of ROM. Also, the device should run a 32-bit processor.

Sun Microsystems provide a CDC Hotspot Implementation (formerly CVM), which is a J2SE-compliant VM that is optimized for resource limited environments. This implementation is available for various devices and operating systems under commercial license. Other J2ME CDC compatible implementations are also available,

[5] The Java Community Process (JCP) is an open organization of Java developers and licensees whose charter is to develop and revise Java technology specification, reference implementations, and technology compatibility kits (http:// jcp.org).

for instance the IBM's J9 VM, which can be found in IBM's Websphere Studio Device Developer[6].

2.4.3 Profiles

The profiles enable flexibility in order to support different types of mobile device categories. In combining profiles with the configurations, complete frameworks for specific device groups are obtained. This section will describe some of the J2ME profiles.

The Mobile Information Device Profile (MIDP) is designed to work on top of the CLCD configuration. The latest version of this profile is MIDP 2.0 (November 2002) and is backwards compatible with MIDP 1.0 (September 2000). MIDP are targeted at devices that should fulfill these minimum requirements:

- small display (min. 96x54),
- one or more of the following user-input mechanisms: one-handed keyboard, to-handed keyboard, or touch-screen,
- two-way, wireless, possibly intermittent, limited bandwidth,
- ability to play tones.

MIDP provides the core application functionality, including user interface, network connectivity via sockets and datagrams, local storage, a small XML parser, and applications lifecycle management for these devices. Simple multimedia and game support is also included. In MIDP 2.0 the Over-the-Air (OTA) Provision Model has been formally included, which enables update and deployment of applications over-the-air. The MIDP specification defines how applications can be discovered, installed, updated and removed. MIDP applications can be browsed at a distributed web server, downloaded and run a mobile device. MIDP applications can also be installed and run locally. These functionalities resemble the dynamic code availability in TAPAS (COD).

Currently, there are three profiles built for the CDC configuration. The Foundation Profile is the most basic. This profile does not include any graphics or GUI support, only basic application-support classes such as network support and I/O support. The Foundation Profile provides, together with CDC, a full application environment for resource limited devices. Examples of target devices are routers, network printers and residential gateways, which is not typical mobile devices.

The Personal Basis Profile adds lightweight standard GUI framework to the CDC configuration. This is support for lightweight components and some Java 2D graphics. It also includes all of the application support APIs in the Foundation Profile. Together with the CDC configuration, it provides a full application environment for consumer products and embedded devices. It is suitable for devices that only require simple graphics support.

The Personal Profile is the most comprehensive of the three profiles suited for CDC. It adds full AWT[7] graphics support, applet support and limited bean support. In addition, all APIs provided by the Personal Basis Profile is adopted. Combined with

[6] http://www-3.ibm.com/software/wireless/wsdd/, evaluation version is free to download.
[7] Abstract Window Toolkit supports basic graphical user interface programming.

CDC, Personal Profile provides a full application environment for devices that require full AWT compatibility and applet support. This applies for devices as high-end PDAs and embedded Web browsers.

The JCP are continuously working on updated versions of the profiles in order to keep up with the J2SE development and the mobile device evolution. A comparison of the packages in CDC profiles and J2ME is shown in Appendix B.

2.4.4 Optional packages

A further extension of the J2ME platform is possible combining optional packages with CDLC, CDC and their corresponding profiles. The packages are modular, which enables device manufactures to include them as needed to address specific market requirements. Optional packets can be technology-specific APIs that extend the basic J2ME application environment. Some of the existing and coming optional packets will be presented above.

The RMI Optional Package (June 2002) provides Java-to-Java remote method invocation for Java devices and interoperates with J2SE RMI. It supports a complete implementation of the Foundation Profile and CDC. It requires minimum 2.5 MB of available ROM[8], minimum 1.0 MB of available RAM[8] and TCP/IP connectivity to the network. This package is a subset of the J2SE Remote Method Invocation (J2SE RMI) package.

Other packages for the CDC/Foundation profile are JDBC Optional Package (November 2003) for Java database connectivity support and the in-progress Advanced Graphics and User Interface Optional Package, which will migrate the core APIs for advanced graphics and user interface facilities (e.g. Swing[9]) from J2SE to J2ME.

There are several optional packets for CLCD/MIDP implementing devices. The Mobile Media API (June 2002) allows easy and simple access and control of basic audio and multimedia resources. The Wireless Messaging API (August 2002) provides standard access to wireless communication resources (e.g. Simple Messaging Service, Cell Broadcast Service).

The Java APIs for Bluetooth (March 2002) enables CLCD Java enabled devices to integrate a Bluetooth environment. As discussed in section 2.3.1, the Bluetooth technology is one of the most exiting offerings to the wireless industry today. The Java APIs for Bluetooth are targeted at devices with 512-kilobyte minimum of total available memory[8] with Bluetooth network connectivity. The APIs are specified in a way that allows layering for more capable Java platforms such as CDC and J2SE.
This optional package could be utilized in order to add Bluetooth support in the TAPAS platform.

[8] Application and localization memory requirements are additional.
[9] APIs that extend the AWT to provide rich, extensible GUI components with look-and-feel that can adapt to its runtime environment.

Optional packages that provide standard access from J2ME to Web Services are also being specified, which will support both CDC and CLCD based profiles. For a complete list of APIs developed through JCP for J2ME, please refer to [JCP].

2.5 Next-generation mobile applications

With the introduction of new wireless technologies and development environments, it is possible to develop new and innovative mobile applications. New technologies make more capacity available for services that require higher speed communications. Also, better wireless coverage will increase the possibility of use of mobile applications. One key characteristic of wireless systems developed now are that most include access to the Internet. It is hard to predict the future, and only the imagination can limit what we can expect from next-generation mobile applications. Even so, some applications environments seem to be predestined for their use and will be described in the following sections.

2.5.1 Communications

Basic forms of communication applications, like voice and simple text messaging (e.g. SMS), are already available. Person-to-person communication is one the driving forces of mobile communications today. Some of the traditional communication applications for computers connected to Internet, such as E-mail services, are beginning to make its way to mobile systems. Simple E-mail applications, for sending en receiving messages, are already available on many mobile devices. Another popular communications application available on the Internet is Instant Messaging (IM). On desktop computers, users are able to communicate in real-time, sending messages, voice and video. With increasing device capabilities and bandwidth in mobile and wireless systems, these sorts of applications could evolve to mobile devices as well. Examples of voice-based IM applications for mobile devices are push-to-talk capability, which enables mobile phones to operate in a walkie-talkie mode offering real-time voice communication over IP compatible mobile systems (e.g. GSM/GPRS, 3G). This technology can also enable video-based communication, since embedded digital cameras are becoming common feature in mobile phones. With this service, private mobile communications infrastructures can be set up. This can be used in industrial and emergencies areas, where instant, ad-hoc communications often are necessary.

In general, the possibilities of multimedia-based communications are increasing, and seem to be one of the most rapidly evolving application services. In Japan, more advanced mobile multimedia messaging is already available, and will probably become available in Europe and US with the introduction of new wireless technologies.

2.5.2 Business and financial

Innovative mobile applications are often designed for the business and financial market. Today, traveling business employees need instant access to e-mail, calendars, company's Intranets and other company information systems. With the globalization and internationalization of many of today's business industries, this also requires access cross national borders. With new wireless technologies, business users should be able to utilize applications that can interact with company resources, transfer large files, set up video-conferences and access company databases.

Technical personnel and engineers could also have advantage of new applications, accessing and modifying database information using mobile handheld devices. Examples of use can be registering of performed services or gathering information about available products in the company's product database.

Mobile communication applications can be used in several business areas in order to solve problems and increase effectively in production and distribution. An example is a pilot project at VTT Information Technology in Finland, which aims at using mobile communications in newspaper distribution and transport [ERCIM, page 49].

Consumer services as Internet banking are already available for desktop computers through web interfaces. These kinds of services could also be interesting next-generation mobile applications. With improved security in coming mobile systems, mobile users should be able to access banking information from their mobile equipment. These applications should enable the user to view their account balances and perform transactions.

2.5.3 Information

Information services can include text alerts of news headlines, sporting results, or delivery of entire news stories. SMS have been used to provide this kind of services with today's mobile technology. With the multimedia capabilities in the next-generation mobile communication technologies, the information services can become more advanced provided by new mobile applications. Wireless networks can provide up-to-date information at any appropriate location. Mobile users can subscribe to services, and be able to receive location-aware information. Example of such can be tourist information, shopping guides, traffic alerts, timetables, weather reports and information of local services such as restaurant and cinema listings.

The possibilities for mobile applications providing information services are almost unlimited. Thus, a great deal of the next-generation applications will fall into this category.

2.5.4 Entertainment

Entertainment and games seems to be growing types of wireless network applications, e.g., ad-hoc gaming networks as soon as people meet to play together. Today, simple Java based (CLCD/MIDP) games have grown very popular, and can be downloaded from network operators onto mobile devices. Next-generation games will probably be based on more interactivity between devices, because of larger bandwidths, and increase in graphics complexity, due to screen enhancements and increased processing power in mobile devices. Manufactures have already started producing special mobile devices targeted at the gaming market.

With better multimedia capabilities other kind of entertainment applications will evolve. Music services have become very popular for desktop computers connected to the Internet. Next-generation mobile applications should enable users to download music and play it on their mobile devices. One application example is pay-per-listen services, where users are charged for the each song that is downloaded. Similar applications could be available for video services as well. Users could access movie

archives, providing both movie trailers and full-length movies. This requires that the next-generation wireless technologies are able to provide mobile users with the necessary bandwidth and devices.

2.5.5 Government

Wireless technology and next-generation mobile applications can be used to increase accessibility of governmental information services, like public documents, and public notices. With increased security mechanisms and better coverage, governmental sectors could use wireless applications in distributing information to their workers. Public service departments, as the police, could by use of wireless applications effectively distribute information, like missing and wanted persons, to officers carrying mobile devices. Public voting services can be used in order to enable users to carry out voting remotely. Governmental funded applications related to educational and medical areas are discussed in the following sections.

2.5.6 Education

In educational environments mobile applications can be used to extend fixed Internet services, such as online e-learning. Access to educational resources, such as lecture-notes, course information, and prescribed texts, will be of great value for students using mobile devices. Also, mobile applications that enable students to hand-in work are of interest. A number of university campuses already provide wireless access to Internet and the campus Intranet by using WLAN technology.

In school environments mobile applications can be used in similar way. Also, applications could enable parents to check their children progress. By use of location-based technology, the children could be tracked down at any time when carrying mobile devices.

2.5.7 Medical

Next-generation mobile applications for medical use are likely to be important. Mobile applications could, with use of location technology, enable users to find the nearest available doctor or pharmacy while traveling. Also, advanced monitoring applications for patients could be used, where information is transmitted to a local control centre. With the possibilities of real-time multimedia, applications for video consultations could also be a possibility.

Medical personnel could have great advantage of new mobile applications. For instance, mobile medical personnel could be provided with handheld devices, which can be used to register patient information for updating medical records. In [ERCIM, page 44] there is an example of an organization called ‘Care4U’ that have used a similar application in medical home-care. Also, applications for ambulance personnel could be useful in giving advance information for arriving patients. With this information the hospital’s emergency staff could better prepare for incoming patients.

It should be noted that when dealing with private and sensitive patient information, it is critical that security and data integrity is ensured. This requires use of robust and reliable wireless technologies and mobile applications.

The TAPAS concept tries to meet the requirements set by the trends and development in wireless technologies, in order to provide an architecture for next-generation services and applications. The suitability of the concept will be discussed in the subsequent chapters, based on the TAPAS platform using J2ME technology.

3 TAPAS and wireless support

In this chapter the TAPAS concept and architecture will be described in more detail. Also, the platform developed for handheld and wireless devices using J2ME will be presented. This platform has been used in the investigation and experimentation with TAPAS support functionality, discussed in the next chapter.

3.1 The TAPAS concept

The TAPAS concept is founded on a theatre metaphor, where generic *actors* are able to perform *roles* defined in corresponding *manuscripts*. A *director* manages their performance. This metaphor models the entities used in the TAPAS architecture. The illustration in Figure 3.1 [AAGE03, slides page 10] shows an overview of the theatre metaphor used in TAPAS.

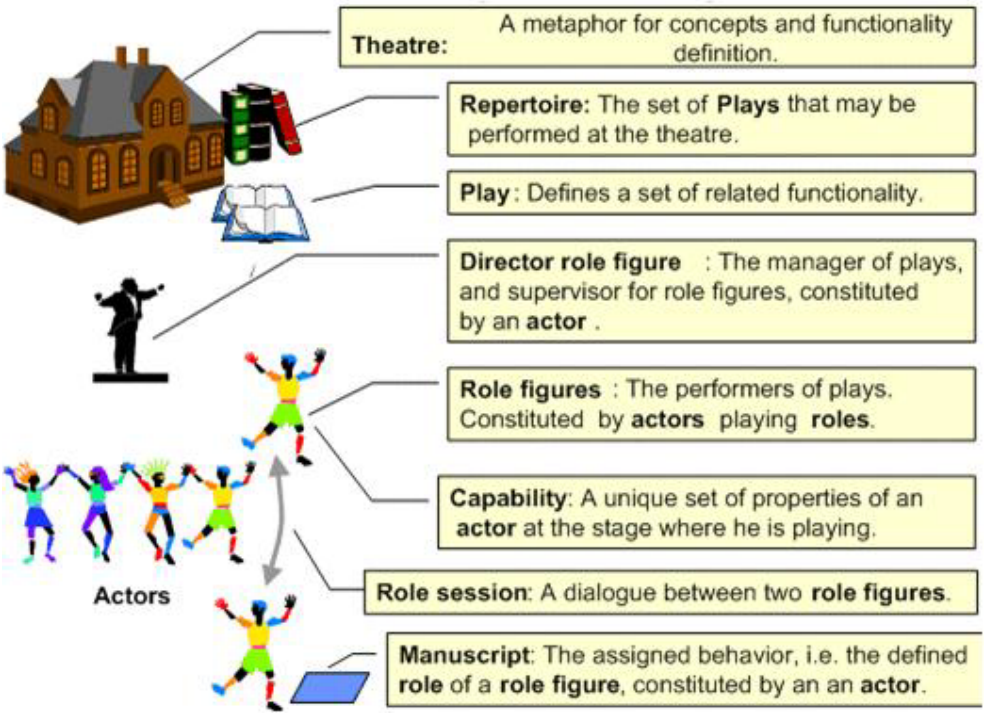

Figure 3.1 - The theatre metaphor used in TAPAS

The theatre is a metaphor for the concepts and functionality definitions used in TAPAS. In the theatre we find a *repertoire* that is a collection of *plays* that can be performed in the theatre. A *play* is a definition of a set of functionality that can be performed by *role figures*. Generic *actors* performing *roles* constitute the *role figures*. The *director* is a special *role figure* that manages available *plays* and supervises the active *role figures*. An *actor* has a unique set of properties associated with it when performing at a certain stage. These properties are called *capabilities* and are important when deciding if an *actor* is able to perform a certain *role* or not. The dialogue between two *roles* is called a *role session*. *Manuscripts* are used to define the behavior of a specific *role*, performed by an *actor*.

Based on this metaphor, a TAPAS service system will be composed by a play, where a set of predefined manuscripts defines its functionality and behavior. The service components in the network, which actually executes the service system, are generic actors able to perform assigned roles in the play. With this approach, the TAPAS project aims at developing an architecture that meets the requirements introduced in chapter 1. The illustration in Figure 3.2 [AAGE03, slides page 5] shows the TAPAS property requirements that form the platform basis for user applications development.

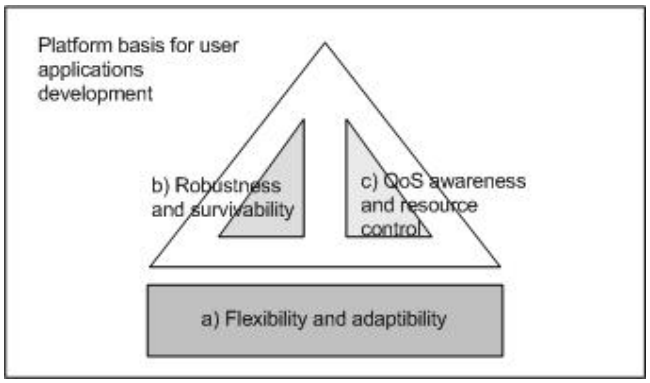

Figure 3.2 - TAPAS property requirements

The flexibility and adaptability properties (a) are the most *basic* functionalities that should be supported by the system architecture. This means that the system should have dynamic structure and functionality. According to the needs and offered system resources, components should be added, moved and removed. Continuous adaptation to changes in the environment should be well handled by addressing the changes by appropriate actions. The system should automatically detect introduction of new components with their inherent capabilities.

The other properties (b and c) are *extended* functionality that should be well handled by the system architecture.

Robustness and survivability (b) means that the system should be dependable. Enhancement of dependency can be achieved by replication of system resources and functionality, and by preventing access of harmful and unauthorized components. The system should also be able to handle fault situations by reconfiguring itself. Fault situations can be handled by re-instantiation of components and preventing error propagation. Also, the system should provide continuous operation even if fault occurs.

By having QoS awareness and resource control properties (c), the system should support negotiation of QoS parameters and resource allocation. Properties as transmission capacity, processing capacity and storage capacity are examples of negotiation issues. If a component is to behave in a certain manner, the available resources should meet the requirements. By monitoring the resource utilization, the system should take actions if the available resources deteriorate, which may cause rearranging of workload or relocation of functionality.

In this project assignment, focus has been on reliability, flexibility and performance issues, which are strongly related to these property requirements. Functionalities,

which have been investigated and experimented with, try to meet parts of these requirements.

3.2 The TAPAS architecture

In the TAPAS project, four main architectures have been developed - the basic architecture, the mobility handling architecture, the dynamic configuration architecture and the adaptive service architecture. Each of these architectures provides specific functionality, that put together constitute the complete TAPAS architecture. Here is a short description of each of the architectures:

- **The basic architecture**
  This is the primary architecture, which provides the basis for all dynamic behaviour functionality. The architecture is based on the theatre metaphor where generic actors in the nodes have the possibility to play different roles specified in corresponding manuscripts. Nodes can be network components and terminals. Actors are software units that can be executed on the nodes in the network. The roles are modelled as Extended Finite State Machines. Directors are specialised actors that manage actors in a domain and have a base of installed manuscripts and repertoires.
- **The mobility handling architecture**
  This architecture is an extension of the basic architecture and adds a new layer of functionality handling mobility functions. Mobility is an important aspect of dynamic and adaptive networking, and is needed for flexible service execution. The architecture is the basis of all functionality related to flexibility in personal, terminal and actor movement.
- **The dynamic configuration architecture**
  The dynamic configuration architecture provides functionality for configuration management. In a dynamic network environment the availability of nodes and changes in their capabilities and status needs to be handled dynamically. Configuration and reconfiguration of the nodes must be done on the fly and not in a predefined manner.
- **The adaptive service architecture**
  Assuming that there never will be only one and only service architecture, the adaptive service architecture is a solution that tries to solve interoperability issues between different architectures. Because of today's highly distributed, heterogeneous and fragmented computing systems, service providers develop composite higher-level services that are composed of lower-level services. In order to provide interoperability between system services, a well-established infrastructure is needed.

Reading this report, a detailed understanding of all architectures is not needed. The main focus of this report is based on the basic architecture and the mobility handling architecture. More detailed information about the architectures can be found in [AAGE03, MALE02, MALE03, JIAN03]. The following sections will describe the basic architecture and platform, and the mobility handling architecture in more detail.

3.2.1 The basic architecture

The basic architecture provides the fundamental TAPAS functionality. The object model of the basic architecture is illustrated in Figure 3.3.

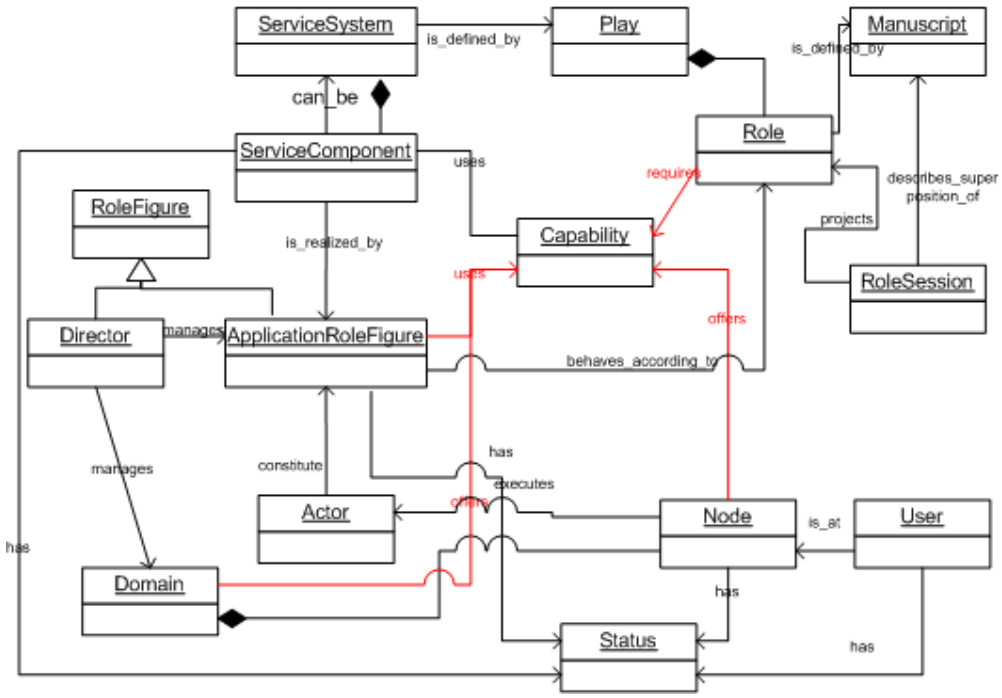

Figure 3.3 - Object model of the TAPAS basic architecture

Actors constitute *role figures* that behave according to a *role*. This behaviour is defined by the *role's* corresponding *manuscript*. The *roles* have different requirements on *capabilities* and *status* of the *node* that the *actor* executes. A *role figure* is realised in an executing environment and utilises the *capabilities* offered on a *node*. *Role sessions* are projections of an *actor's* behaviour when interacting with other *actors*. *Nodes* are part of a *domain*, which is managed by a *director*. The complete *service system* is defined by a *play* and consists of *service components*. A *service component* is realised by a *role figure*, which is constituted by *actors* playing certain *roles*.

An actor's possibility to play different roles depends on the capabilities that the role requires and the capabilities offered by the node where the actor executes. Capabilities are the ability or power to do something. The actor will utilise these capabilities when executing on the node. Capabilities can be resources like processing, storage, display and transmission resources (e.g. CPU, hard disk, screen resolution, bandwidth), extra equipment (e.g. printers), data (e.g. user identification and authentication), and functions (e.g. pure software or combined software/hardware components).

Status is dynamic and reflects the current state of the system at a certain time instant. It reflects the situation of the system in respect to the number of nodes, available plays, running actors, traffic situation, etc. The information can consist of observable counts and QoS measures or calculated predicated of these counts and calculated measures.

The TAPAS platform, which implements the basic architecture, provides a set of basic support functions. The support functions are realised by a set of procedures that

perform specific operational tasks in the TAPAS platform. [JOHA01, Appendix B, page 7] have grouped these procedures as follows:

- Managing the availability of application functionality (i.e. the plays)
  - *PlayPlugIn(PlayId, PlayVer, PlayLoc)*
    Installs a play according to the specified parameters at a Director.
  - *PlayChangesPlugIn(PlayId, PlayVer, PlayLoc)*
    Replaces an existing play version with a new version.
  - *PlayPlugOut(PlayId, PlayVer)*
    Removes an already installed play from a Directors repertoire.
- Managing the existence of active entities in operational systems (i.e. the actors)
  - *ActorPlugIn(Location, RoleName)*
    Requests establishment of a role session at another actor.
  - *ActorPlugOut(RoleSessionId)*
    Requests an existing role session to be removed.
- Dynamic redefinition of actor behaviour (i.e. roles)
  - *ActorBehaviourPlugIn(RoleName)*
    Assigns behaviour to an already existing actor, which only has generic behaviour.
  - *ActorChangesBehaviourPlugIn(NewRoleName)*
    Requests change in behaviour of an already existing actor.
  - *ActorBehaviourPlugOut()*
    Removes the assigned behaviour of an existing actor.
- Interactions between actors, actors capability change and monitoring PaP activities
  - *RoleSessionAction(RoleSessionId, MsgType, MsgParameters)*
    Communicate between actors in a role session.
  - *ChangeActorCapabilities(ChangeType, CapabilitySet)*
    Specifies change (set, add, remove) in a capability set of an actor.
  - *SubscribeRequest(EventType, Scope, ApplicationTypes, WhenReport)*
    Subscribes for a specified event type to occur.

In the TAPAS implementation for small and wireless devices, there is added mobility support and support for dynamic connections. This functionality will be described in relevance to this platform later in this chapter. The following section will introduce the general concepts of the mobility handling architecture in TAPAS.

3.2.2 The mobility handling architecture

With the focus on handheld and wireless devices, it is evident that the need for mobility handling functions is necessary to cope with the highly dynamic environments where such devices operate. [MALE03] argues that "mobility is regarded as *the* most important feature needed to achieve adaptability and flexibility in the execution of service components". This means that in order to meet the property requirements set to the architecture, its important that the architecture have functionality that handles dynamic changes in availability of resources and position of users. Users, software components and terminals should be able to move free in the

distributed system architecture. In particular, access to services that are user-oriented and personalized should be available for users independent of location and equipment.

In the TAPAS project a mobility handling architecture has been developed, which introduce mobility management functionality as an extension of the TAPAS basic architecture. This architecture adds a new layer of functionality, handling mobility functions. The architecture is the basis of all functionality related to flexibility in personal, terminal and actor movement.

Four mobility features are supported by the architecture: *User*, *user session*, *terminal* and *actor mobility* [MALE02]. With user and user session mobility, basic *personal mobility* is provided for any system or application based on TAPAS. This means that users are able to access subscribed services and suspend and resume user sessions independently of the used terminal. Terminal mobility means that terminals should be able to move in the network and still be able to access services and applications. End-users use terminals to get access to the network. Actors are instantiated functionality at a node that should be able to move along with its associated role sessions, state and variables. Actor mobility can be used e.g. to reallocate functionality, overcome resource deterioration problems and handle configuration changes.

The illustration in Figure 2.1 [MALE03, page 4] shows the mobility concepts in TAPAS, and its relations.

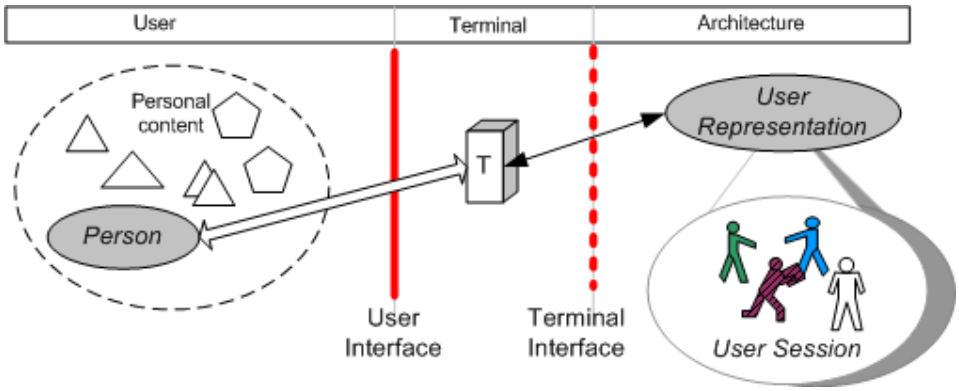

Figure 3.4 - Basic mobility concept in TAPAS

A *user* is represented by its *personal content* and can be related to a *terminal* through a *user interface*. The *user* can be identified by a username. The *terminal interface* can be identified by a network address and relates to a *user representation* that identifies the *user* in the system. A *user* may interact with the system, or services, within a defined *user session*. The double interface between the *user* and the system, with the *terminal* in the middle, enables a flexible way of representing *users* and *terminals* independently of each other.

The mobility handling architecture extends the TAPAS basic architecture by introducing new objects that enables mobility management. The extended object model for the mobility handling architecture is showed in Figure 3.5 [MALE03, page 5].

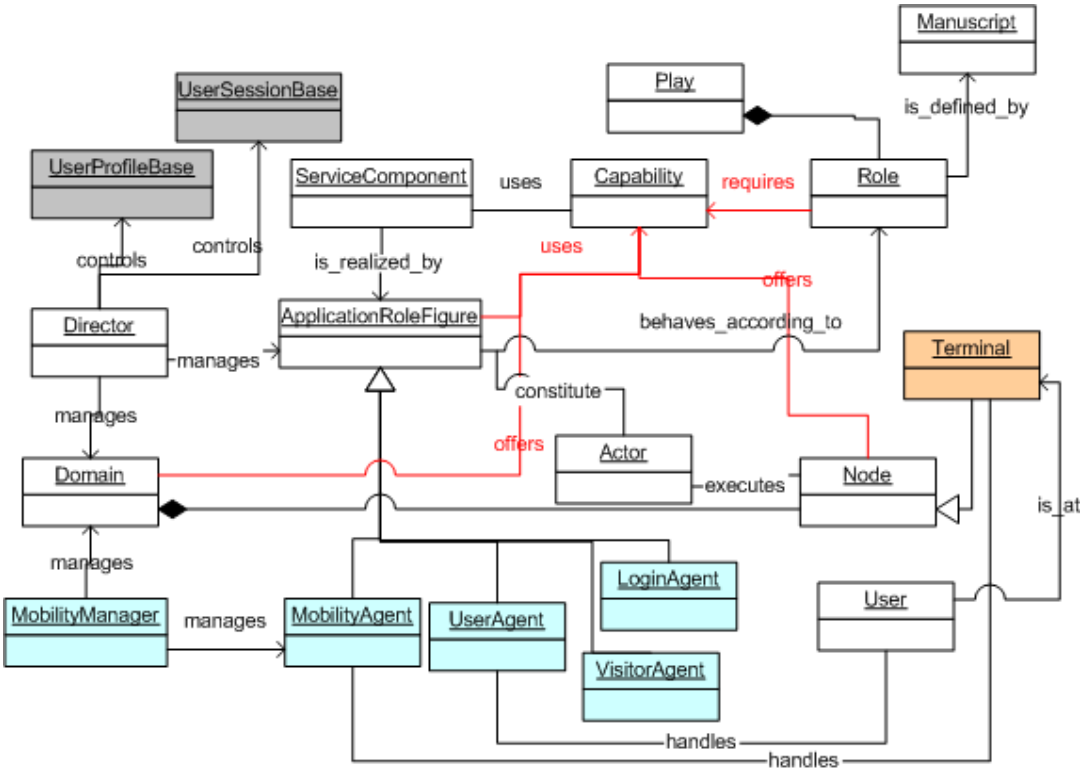

Figure 3.5 -Object model of the TAPAS mobility handling architecture

In *user session base*, all user session information used by actors is stored. In *user profile base*, the information about users is stored. This information contains the users configurations and service subscriptions. The director of a domain controls both user *profile base* and *user session base*. In a domain, there is also one *mobility manager*. This object is responsible of managing actors and terminals mobility. *Mobility agents* that run in terminals aid this management and update the location-related information. A *user agent* manages user interactions with its home domain, and *visitor agent* manages user interactions with its visitor domain. The *visitor agent* is necessary because the user profile for a user is present in the user's home domain, but not in the visitor domain. A *login agent* enables controlled user access to services in the system. [MALE03] gives supplementary information on terms and introduced objects in the mobility architecture.

Actor and terminal mobility is explained in more detail in relevance to TAPAS for small and wireless devices in the following subchapter.

3.3 Platform for wireless devices

The limitations that characterize small and wireless devices have been accounted for when integrating the TAPAS platform to such devices. In order to meet the requirements set by such devices a complete re-specification of the original TAPAS platform has been done. The complete argumentation for this choice can be found in [LÜHR03]. The following sections will present this platform and its properties.

3.3.1 Mobility handling functionality

The optimized TAPAS platform for small and wireless devices (also known as MicroTAPAS) is based on the original TAPAS platform, thus no significant changes has been applied to the basic support functionality. However, new support functionality has been added in order to support the mobility of the target devices.

This TAPAS platform has adopted concepts from the mobility handling architecture. The *mobility manager* and *mobility agent* objects are added in this platform to provide actor and terminal mobility support. With the introduction of these objects new support functionality procedures have been introduced. These procedures are:

- *ActorMove (NewLocation)*
  Re-instantiates an actor with their parts at a new location. This procedure is equivalent with a sequence of procedures part of the basic architecture.
- *LocationUpdate (Actor/Node, NewLocation)*
  Updates the *MobilityManager's* register with the new location of an Actor or a Node.
- *ActorDiscovery (Actor)*
  Gets the current location of an actor from the *MobilityManager's* register of actors.
- *NodeDiscovery(Node)*
  Gets the current location of a node from the *MobilityManager's* register of nodes.

Actor mobility is the movement of an instantiated actor object along with its parts, such as interfaces, behavior, capabilities, queue of incoming requests, and methods accessible by other actors at specified interfaces. The mobility of actors is achieved by re-instantiating their parts at the new location. An example of an ActorMove procedure is illustrated in Figure 3.6 [MALE03].

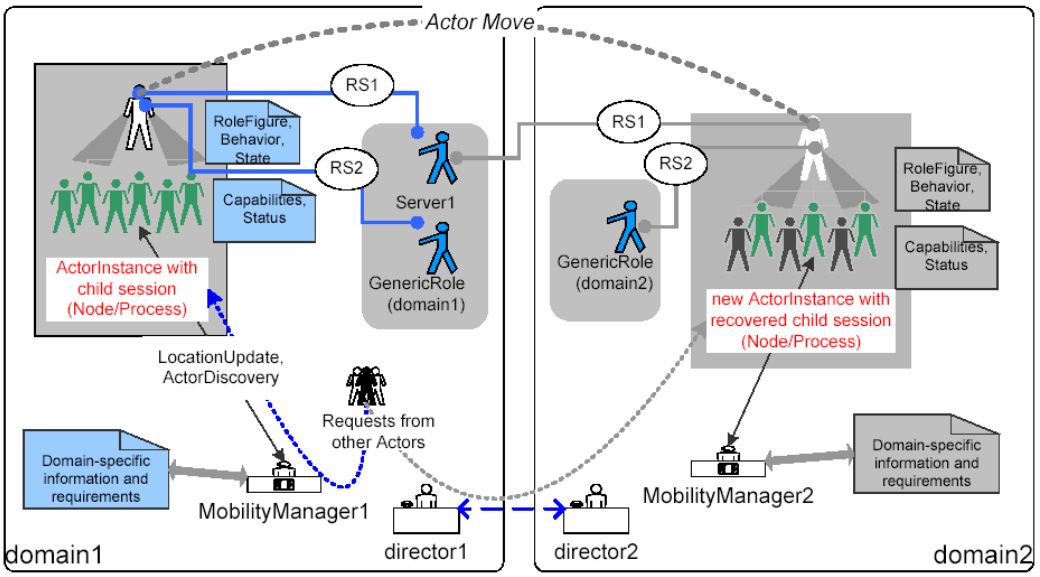

Figure 3.6 - An example of actor mobility in TAPAS

In this scenario the actor moves between two domains, managed by different directors, and having different mobility managers. When the ActorMove procedure is initiated at the actor, its parts are saved and a *LocationUpdate* is sent to its responsible *mobility manager*. At the new location, its parts should be recovered. The capabilities at the moved-to node must sufficient for all parts to be recovered properly. This is not always the case, and should be handled in an appropriate manner. Requesting actors will send an *ActorDiscovery* to the *mobility manager* to get its new location.

Terminal mobility is achieved by executing a *mobility agent*, which tracks the location of the node at any time, and updates the *mobility manager* when the terminal moves to a new location. The *mobility manager* should operate in a fixed location, so that all other nodes know its address. When the terminal is out-of-coverage, it will be seen as offline, but still registered in at the *mobility manager*. An example of Terminal mobility is shown in Figure 3.7 [MALE03].

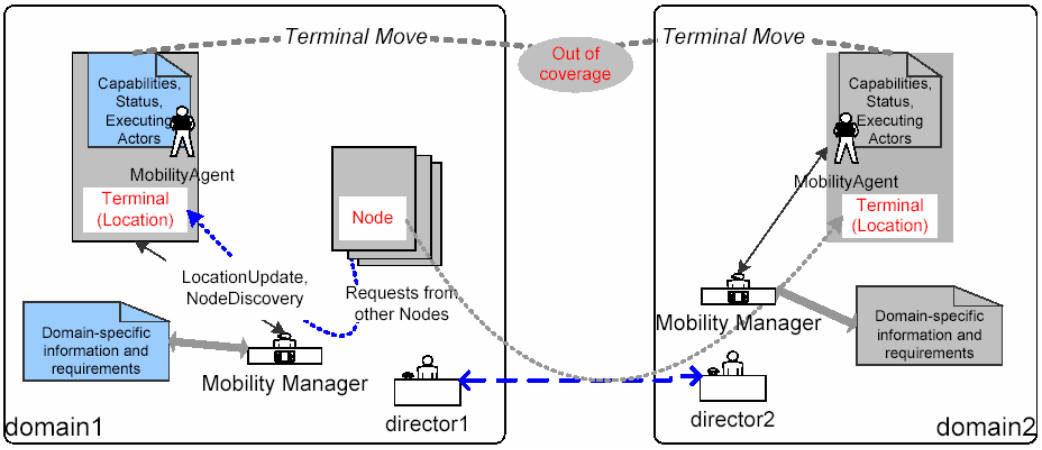

Figure 3.7 - An example of terminal mobility in TAPAS

The illustration shows a terminal that moves from one domain to another. The *mobility agent* executing at the terminal will issue a *LocationUpdate* procedure, when the terminal changes its location. Other nodes will perform *NodeDiscovery* procedures in order to send request to the moved terminal.

3.3.2 Handling dynamic connections

In this platform, there is also added functionality that handles the highly dynamic connections introduced in wireless environments. Dynamic connections are in this context defined as connections that may, or may not, be available at a given moment. This functionality is incorporated in the *mobility manager* and *mobility agent* objects. *Ping messages* are used to determine if a sender has connectivity to the receiver. Figure 3.8 shows how ping messages are sent and received between *mobility agents* and the *mobility manager* in a domain.

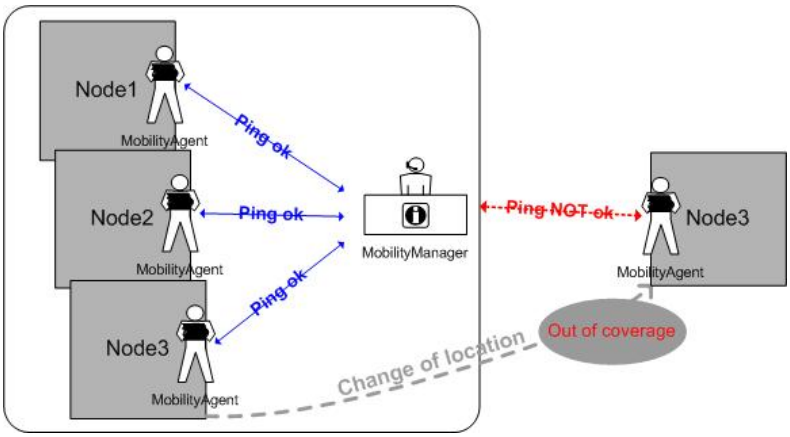

Figure 3.8 - Ping messages determine the connectivity status of nodes in TAPAS

The *mobility manager* sends *ping messages* to all registered nodes, and the *mobility agent* at each node sends ping messages to the *mobility manager*. This ensures that connectivity status is determined in both ways. In the scenario presented in the figure, *Node3* moves out-of-coverage. The ping messages from *mobility manager* to *mobility agent* at *Node3* will fail, and *mobility manager* registers that the node is unavailable. The *mobility agent* at *Node3* will send ping messages to *mobility manager* that fails, and the node will know that it is out-of-coverage.

3.3.3 Layered design model

This TAPAS platform has a slightly different layered design model than the original TAPAS platform. The original model can be found in Appendix C. The new model has been modified to adapt to the limitations of the target devices. In the original model, several VMs should be able to run simultaneously. Because of the lack of system resources in small and wireless devices, only one VM should be able to run. Also, functionality of the PaP Actor Support (PAS) layer and PaP Node Execution Support (PNES) layer are combined in one layer called MicroPNES. This is done in order to avoid communication and management operations between these layers, which would be resource demanding on the target devices. MicroPNES is responsible of handling the Actor instances on the node, and route requests and results between Actors on the same node and to/from other nodes.

The modified layered design model is shown in Figure 3.9 [LÜHR03, page 9].

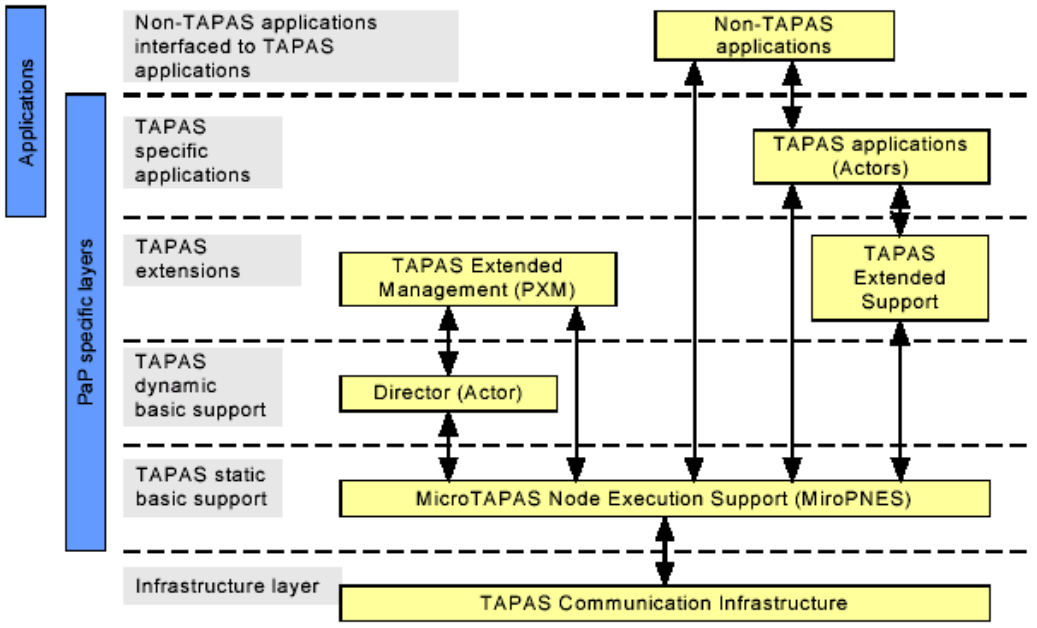

Figure 3.9 -The layered design model of TAPAS for small and wireless devices

The illustration in Figure 3.10 [LÜHR03, page 9] shows an operating system example of TAPAS for small and wireless devices. The client nodes have wireless connections to the network, and utilize resources in the server node. The client nodes are devices with limited resources (e.g. PDAs), thus running a CVM (CDC compliant VM, see chapter 2.4). The MicroPNES instance running on the nodes is able to handle several actor instances. The server node is a stationary computer, which runs the normal Java VM (JVM). The director, which is responsible managing the domain, runs on this server node. Also, there is a web-server that holds TAPAS generic support and plays, which can be requested for download when needed. All nodes able to run TAPAS applications should have static available bootstrap code that initializes needed MicroPNES functionality.

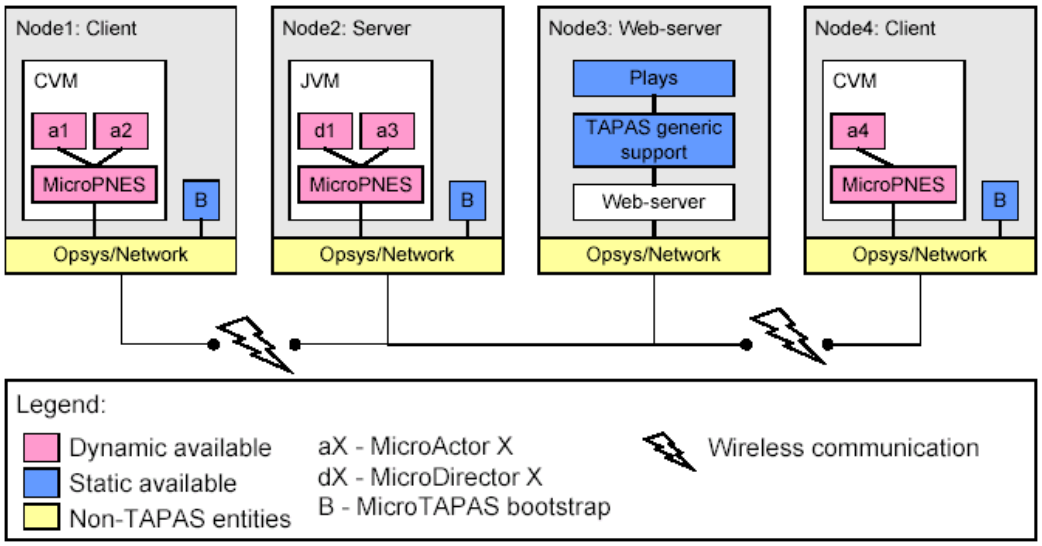

Figure 3.10 - An operating system example of TAPAS for small and wireless devices

With this modified model, only one combined PAS/PNES (MicroPNES) is initiated on each TAPAS node. In the original model, several PASs could be initiated at one node, each controlling a set of actor instances. In Appendix C a system example of the original layered design model is shown.

3.3.4 Implementation

The implementation of TAPAS for small and wireless devices is done with the use of J2ME. The primary reason for this is that the original TAPAS platform is implemented using J2SE. The basic idea was to enable re-use of proven and well-documented components from the already implemented platform. However, because of absence of different J2SE functionalities in J2ME, some implemented support functionality and mechanisms had to be re-invented in order to adapt to J2ME.

CDC was chosen as the appropriate J2ME configuration, primarily because the lack of the `java.net` package, with its `URLClassLoader` class, in the CLCD configuration. This class has to be present in order to support dynamic download of code when required, which is an important feature in TAPAS.

The communication model of this implementation is based on sockets[10], opposed to the J2SE implementation that uses RMI. This is partly because of limitations in the RMI Optional Package. Sockets are used in communication between nodes (MicroPNESs) and local method calls are used in communication between local actors. Also, the use of sockets and local method calls are less resource consuming.

The communication model used in this implementation consists of request and result pairs. Requests are given a unique identification that determines which result that corresponds to the given request. By use of threads, a node can send and receive requests, even if it is waiting for a result of an earlier request.

The most interesting with this implementation is the additional support for mobility and dynamic connections. Objects from the mobility handling architecture have been introduced in order to handle actor and terminal mobility. Incorporated in these, is support for handling of dynamic connections. The provided functionality by these objects is highly important for the platform to be able to handle wireless devices in a properly manner.

[10] A communication socket is one end of a bi-directional communication channel between two programs on a network. The socket is bound to a given port, which allows the transport layer protocol to send data to the correct application.

4 Investigation of TAPAS for wireless environments

In order to investigate and experiment with plug-and-play functionality and services for wireless devices, the available TAPAS platform for such devices (MicroTAPAS) has been used as reference. The work has been done with the emphasis on reliability, flexibility and performance. This chapter will describe the work that has been done, such as study of the available functionality and improvements. The next chapter will present some test cases with results, which will be used discussing the work presented in this chapter.

4.1 Introduction

The work has been focusing on performance of the mobility handling mechanisms, and the handling of wireless connections. If the TAPAS concept is going to be suitable for next-generation wireless systems, these are key functionalities that have to be supported. Operating in wireless environments, the platform and services are continuously exposed to dynamical changes, as mobility of users, user sessions, actors and terminals, and unstable and slow connections. Furthermore, the wireless devices in such systems typically have limited capabilities. With these characteristics, the functionalities should address reliability, flexibility and performance issues suitable for such environments. In the effort to achieve such goals, it seems important to focus on modular solutions, which on the one hand extend the support functionality, while on the other hand allows for flexible mapping to different operating conditions and execution environments.

The following subchapters will give a detailed description of the tasks and problems encountered throughout the fulfilment of this project assignment, all will refer to the existing TAPAS/MicroTAPAS functionality, in some cases completely altered.

4.2 Actor mobility functionality

The work was started out producing a demonstration of queue handling in the actor mobility functionality. The purpose of this was twofold; to obtain a better insight of the available source code for further investigation and experimentation, and to verify this functionality in the available implementation. Furthermore, the work with this example revealed some limitations and problems in the available implementation. These obstacles had to be dealt with in order to continue the investigation and experimentation with the platform and its functionality.

The actor move procedure is used handling problems like deterioration in offered capabilities and network connectivity. When an actor moves, its parts should be saved and moved along with the actor. At the new location the parts should be recovered, according to the available capabilities in the destination and the certain conditions specified to control this procedure. In this TAPAS implementation, the strategy for the moving procedure is determined in a configuration file corresponding to the actor. The strategies have different requirements of parts that should be transferred and recovered. The simplest strategy will recover capability and interface parts and dismiss queued request and method parts, while the most elaborated strategy will try

to recover such existing and queued requests. This strategy will be highly configurable, and will, in a possible follow-up work, be dealt with.

The demonstration should show how queued requests were handled when an actor is moving or changing its location. This functionality was already supported in the platform, but never comprehensively demonstrated. The actor mobility procedure, if executed with the strategy of moving the queued requests to the new location of the actor, should be successful in providing a working example of an actor that carries out with the functioning at its new location.

In order to demonstrate this functionality, a scenario including two actors where developed. One actor sends a number of requests to another actor, which receives and queue them. The receiving actor should process every incoming request by applying some time consuming or delaying computation. As incoming requests are queued, a move request is instructed so that it is possible to suspend the execution of an actor and move to another location. The procedure is successful if no requests are lost and the actor in the new location handles the requests in the correct order.

An extension of the available MicroTester application, introduced and described in [LÜHR03], was made for demonstrating this functionality. The extended application provides actors with an additional GUI-dialog for setting up a list of RoleSessionAction requests to another actor, and a dialog for dynamic setting of move strategy of an actor. Also, request logging at sending and receiving actors is supported. In Figure 4.1, two screenshots from the application is shown. In the left screenshot (from desktop computer) the dialog for sending several requests to a special actor is shown. In the right screenshot (from PDA) the display of a log of incoming processed requests are shown.

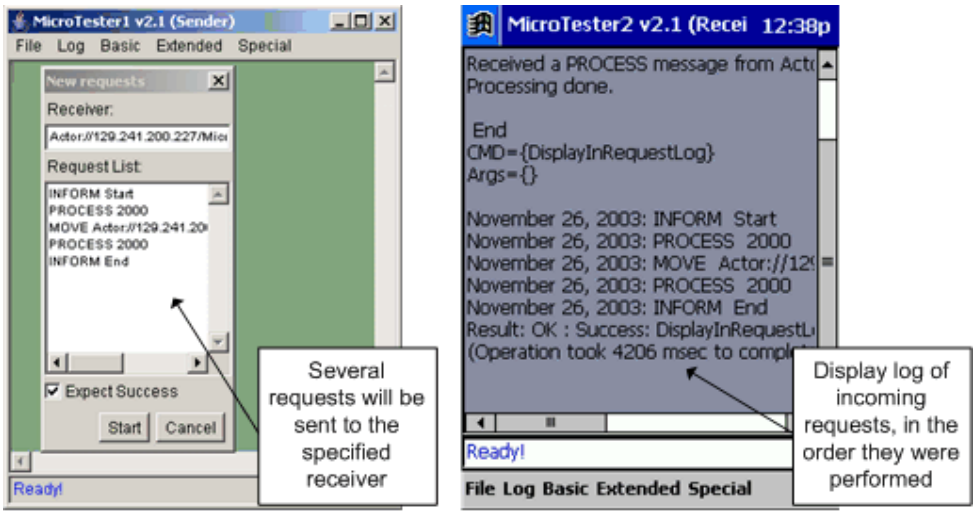

Figure 4.1 - Screenshots from the extended MicroTester application

In the demonstration, three different RoleSessionAction requests were used, all specific for the MicroTester application:

- INFORM <message>
  Displays the *message* at the receivers output display.

- *PROCESS <time>*
  Delays the receiver with an occupying computation for a specific *time* in milliseconds.
- *MOVE <new location>*
  Causes the receiving actor to issue an actor move request to a *new location*.

The inform application messages were simply used for displaying trace information at the receiving actor when performing the demonstration. The process messages were used to delay the receiving actor so that the following requests were queued. The move messages were used to activate the move procedure at the receiving actor. In the left screenshot in Figure 4.1 a sequence of such requests is shown that was used in the demonstration.

When preparing the demonstration, it was found that queued requests at the receiving actor were handled out-of-order, even before the move request was issued. Therefore, the work was focused on solving this problem in order to produce a correct working example of the actor move procedure.

The queued requests should be treated in a first-in-first-out (FIFO) manner. In order to localize the problem, the source code of both the platform and the MicroTester application was scrutinized. The reason for the problem was located in MicroPNES, that routes requests to and from the actors it is responsible for. A single thread running in MicroPNES handles queuing of incoming RoleSessionAction requests. This enables the MicroPNES to accept requests independently of other tasks. Each incoming request should be sent to the correct actor instance, which MicroPNES is responsible for. Upon receiving several requests, MicroPNES will queue them, and forward them one by one to the receiving actor instance. The problem was that the wrong element in the queue was forwarded, which caused the actor instances to receive requests out-of-order. When first located, the error was easy to fix just by altering some lines of code. It is equally important to notice that due to the asynchronous manner how the whole architecture tends to operate, there will be no requirement on maintaining the order of these requests at the actor instance where they are meant to arrive.

After this change, the demonstration was completed by showing that the actor mobility mechanism in the platform worked as expected. All the existing queued requests of an actor moving or changing its location were processed in the correct order at the new location. However, when working with this demonstration, a problem with inefficient handling of requests were discovered and had to be dealt with. This is described in the following subchapter.

4.3 Avoiding blocking of incoming requests

Inefficiencies in the handling of incoming requests were found when investigating the platform and its functionalities. Upon sending several requests to different actor instances at the same node, it was found that the actors were unable to receive requests independently of each other. With this solution, only one actor could be busy at a time and no concurrent processing is possible.

The separate thread that handles incoming requests in MicroPNES, forwards only one request at a time to the correct actor instance. Depending on the type of application message, the request can take different times to process and complete. In the platform, the MicroPNES request handling thread passes control over to the actor instance, and will not be able to continue before the control is returned. With this solution, the following requests in the queue will be blocked even if destined for another actor instance that is idle. However, immediate return of control to the MicroPNES thread does not completely solve the problem. If the next request in the queue is destined for a busy actor, this request will block the following requests. The first request in the queue will not be removed until the receiver of this request is ready (idle). This is similar to a problem often referred to as head-of-line (HOL) blocking, in relation to packet routers and switches. Figure 4.2 illustrates the problem where a request to an idle actor is blocked in MicroPNES' input queue by a request to a busy actor.

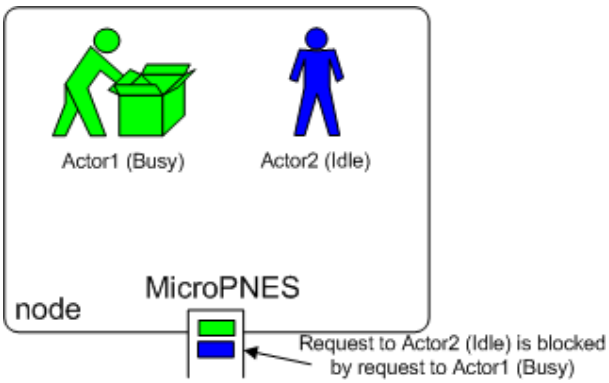

Figure 4.2 - A request to an idle actor is blocked by a request to a busy actor

If several application actor instances are able to run at a node, it should be possible to send requests to all of these and expect that they are able to receive requests independently of each other. The actors running at a node could be part of separate applications, which should be able to operate independently of each other. If requests to idle actors are blocked, the response times to these actors increase, even if they are idle and ready to perform the requests. This is clearly a problem affecting performance, and should be handled properly. Actually, this problem emerged from the decision to remove the actor support functionality (PAS) from MicroTAPAS, with regard to TAPAS, that was taken to ease and shrink the support and code requirements for the small-capability handheld devices. In TAPAS multiple PAS instances were used to gather related actors in one virtual machine.

The actor instances should be able to return control immediately to the MicroPNES instance, and hold several requests at a time. In order to handle these issues, a separate queue object was associated with an actor. Similar to MicroPNES, the queue handling for an actor runs in a single thread, which enables the actor to operate independently of queuing incoming requests. In this way, MicroPNES can put requests in a receiving actors input queue and return immediately, serving the next request in the line. If several requests are destined for the same actor, the actor is able to hold these requests in its input queue, and serve them one-by-one as it progresses.

Figure 4.3 shows that by introducing a separate queue for each actor running in the node, the blocking of requests to idle actors is avoided.

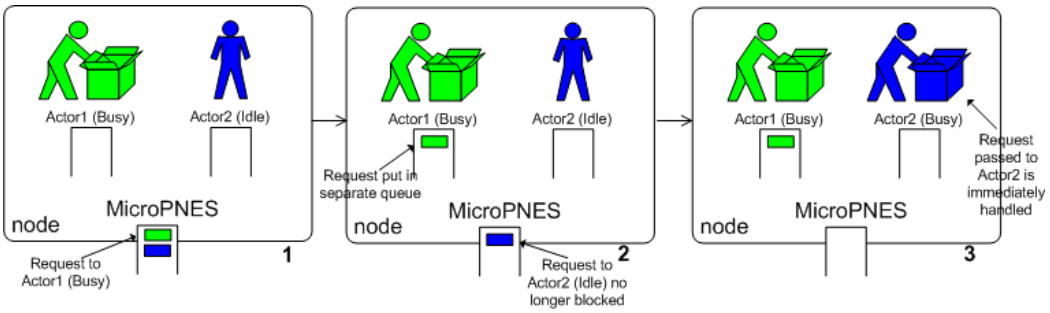

Figure 4.3 - Request to idle actor is not blocked by use of separate input queues

The figure shows a sequence of illustrations describing the procedure of the queue handling. There are two actors, *Actor1* and *Actor2*, running at the same *node*. The routing of requests between actors is managed by *MicroPNES*. Two requests are received and queued by *MicroPNES*. The first request in the queue is destined for *Actor1*, and the second request is destined for *Actor2* (1). Even if *Actor1* is currently occupied and not ready to process a new request, the request destined for this actor can be removed from *MicroPNES*' queue and put in the actor's separate queue for later processing (2). This enables *MicroPNES* to handle the next request in the queue, which is destined for *Actor2*. This request is then forwarded to the input queue for *Actor2*. However, *Actor2* is currently unoccupied and ready to receive a request. Therefore, the request is immediately consumed, and removed from the actor's input queue (3). Both actors at the node are able to process requests independently of each other.

The described approach was implemented in the platform, and the functionality was tested by sending requests to several actors located at the same node, thus handled by the same MicroPNES. The extended MicroTester application was used in order to test this proposed solution. A simple scenario including three actors was set up. Two of the actors ran at the same node and received requests generated by a third actor situated at another node. In order to keep the actors busy, process application messages were sent to the actors. Also, inform messages were used for tracing the progress. First, a sequence of requests was sent to the one of the two actors at the same node. This sequence included inform and process messages, which caused the receiving actor to be occupied. Second, a similar sequence was sent to the other actor at the node that was idle. The scenario showed that the idle actor was able to receive a sequence of requests independently of the other busy actor at the node.

4.4 Improving efficiency of wireless connectivity

In the TAPAS platform for wireless devices, the dynamic connections introduced by wireless environments are handled by the use of communication sockets and pinging. Some work has been done investigating the possibilities of other solutions and improving the efficiency of the existing scheme.

4.4.1 Alternative methods

The variable signal strength and coverage in wireless environments implies that the network connectivity of wireless nodes is highly dynamic. The platform should be able to detect these conditional changes in order to provide flexible and reliable

services. When a node moves out of coverage, both the system and the node itself should be able to adjust to the changes that have occurred in an appropriate way. The hardware in wireless devices has the ability to continuously detect the available signal strength. This information could be used in detecting deterioration of network connectivity, both by adding extra reliability to the existing scheme or by replacing parts of the scheme in removing ping overhead messages from the devices to director/mobility manager node. However, a standard API for accessing such hardware specific information is not provided by J2ME. Currently, this information is only reachable through native and proprietary API's, which do not provide the flexibility needed for the TAPAS platform. This might require some workaround; by using and accessing the drivers associated with such wireless connections, and using some operating system variables and functions. Signal information could also be requested using Simple Network Management Protocol[11] (SNMP) queries. The director/mobility manager could request relevant information when updating connectivity status for attached nodes. Some wireless equipment supports this, or a SNMP agent could monitor the device. However, this solution also depends on the hardware and technology used, and is not applicable in this case. Therefore, the existing scheme using simple network routines is the currently the best approach for detecting network connectivity in the TAPAS platform. The approach has been designed and tested on WLAN connections, but other radio signal based connections may apply similar approach. Two other major radio connections seem to be dominant, Bluetooth that operates in the lower range of the WLAN, and 3G that operates in the upper range in terms of distance, signal strength and user capacity. Anyway, the mobility management schemes introduced here are applied at the higher-level, as they are part of a middleware that exists on top of a network technology and uses the API of a given operating system. Unless so-called native methods and/or SNMP are used, mobility/connectivity detection and improvement is only possible using the mentioned network routines or dedicated interface routines specific to either equipment or technology.

4.4.2 Improvement of connectivity scheme

The implemented scheme for network connectivity awareness was investigated in order to do it more efficient. When investigating and experimenting with the available platform, inefficiency in the scheme was found. Also, new functionality has been introduced in order enable flexibility and dynamic adjustment of the scheme according to changes in the environment.

The pinging, or connection detection, is implemented using sockets. A ping is simply done by establishing a socket connection to another node at a dedicated port. The port number used for pinging is determined in the configuration of the system. Each node will listen and accept socket connections at this port number. If the socket connection is established, this indicates that the node is reachable and the socket is closed. If the socket fails to establish, this can be interpreted as the node is unreachable, and deterioration of network connectivity has occurred.

[11] Simple Network Management Protocol (SNMP) can be used to monitor network attached devices and the conditions for it. Information is requested through SNMP queries.

The mobility manager sends pings to all nodes that it has in its registry at a fixed interval. When any loss in connections is detected, the mobility manager registers this and updates the status of the relevant nodes. When this functionality was investigated, it was found that the current implementation could suffer from severe delays when establishing sockets to nodes with lost network connections. This is because the `java.net.Socket` class has a relatively long connecting timeout. If the node it is trying to reach is down, or if there is no route to the node, the socket timeout will freeze the sequential pinging procedure and the status of one node will affect the update of others. The call to the socket constructor will not return until the timeout has occurred, and there is no way of changing this timeout value in Java. This is problematic because this timeout will introduce delays when updating the status of each node. If several nodes registered at a mobility manager are down or out of reach, the traversal of the pinging sequence will take relatively long time and affect the reliability of the system. The time between updates for a specific node will be the defined and fixed interval between pings in addition to the introduced delays. Figure 4.4 shows the principle of how timeouts affect the update of nodes. Here, there are two nodes that are being pinged. It is shown that the update of *node1* will be delayed because *node2* is not responding.

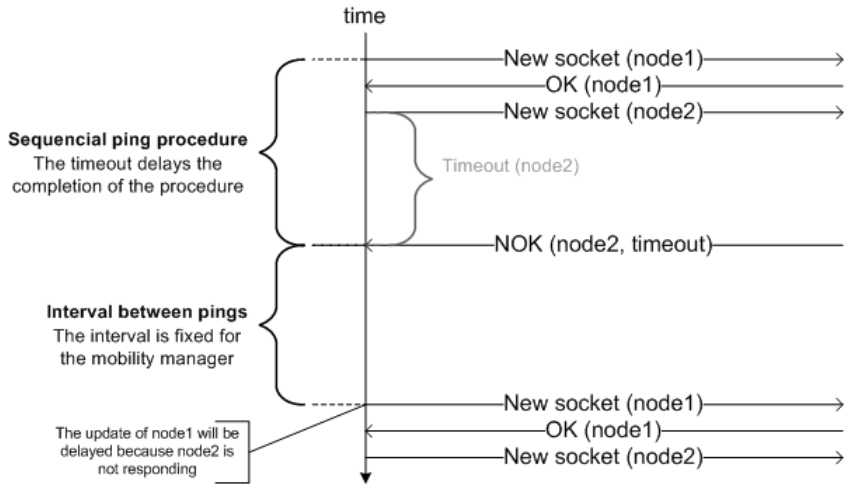

Figure 4.4 - An example of the sequential ping procedure

Two alternative solutions to the current scheme were proposed and implemented. The first solution to this problem was the introduction of separate pinging threads for each node registered at the mobility manager. Instead of pinging the nodes in a sequential order, the nodes should be pinged in parallel. With this procedure, nodes causing socket connection timeouts do not affect the pinging of others. When the mobility manager pings its registered nodes, it starts a separate thread for each node, which is responsible of creating a socket connection to the correct address and port. The thread will update the status of each node according to the results of the ping and die. In order to avoid that several threads are trying to ping the same node, the mobility manager holds a list of active threads that it checks at each interval. If a node still has an active ping thread, this thread will continue until the socket connection timeout occurs or that a connection is detected. The mobility manager only starts new ping threads to nodes that do not have active threads. In other words, there will be maximum one active thread per node at any time. With this procedure, the delay introduced by some nodes will only affect the update of these nodes. The time

between updates for each node will be the defined and fixed interval between pings, or the time that it takes for a socket timeout or the connection to be detected. Figure 4.5 shows the principle of this scheme with the same example used in Figure 4.4. Each ping is started in a separate thread that runs in quasi-parallel. Here, the update of *node1* will not be delayed by the timeout introduced by *node2*.

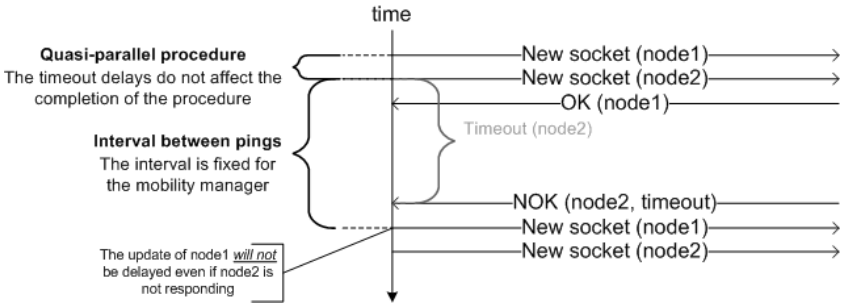

Figure 4.5 - An example of the parallel ping procedure

A second solution to the problem with socket connect timeouts, lead to the implementation of a new class called `TimedSocket`. This class is based on [JAVAWO] and generates socket connections without stalling for long time periods. It is possible to specify a self-defined timeout value, which provides more flexibility. The class uses a multithreaded approach, where an instance of the standard `java.net.Socket` is created in a single thread, and a primary thread polls this at certain instances in order to determine if the connection is established. If the socket is established, the socket connection will be returned. However, if the socket connection is not established within the timeout period, the control will be returned and the ping is interpreted as unsuccessful. By using this class, rapid changes in the connectivity can easily be detected. However, when using the standard `java.net.Socket` class directly, nodes that become reachable again before the timeout expires may result in establishment of a connection. When investigating this, different results were found. If a node got network connectivity again in a relative short period after the socket creation was issued at the mobility manager, a connection could be established. When the connectivity was gained later, the sockets were almost never established before the timeout expired. When testing the socket creation time in a WLAN environment, the sockets were established after a relatively short period when the nodes were reachable. Therefore, the new `TimedSocket` class was chosen in order to enable more rapid and effective detection of the network connectivity.

The result of the investigation was that both solutions were combined in the mobility manager. The timeout period for socket creation was set to be the fixed interval between pings. With this approach, the network connection of each node is updated in parallel and at a fixed and defined interval.

When investigating the behavior of the mobility agents running in the mobile nodes, similar problems with socket connect delays were found. The mobility agent only pings its responsible mobility manager, but long delays was experienced when the connection to the mobility manager node was lost. Therefore, the `TimedSocket` class was used in the mobility agents as well. As for the mobility manager, changes in the

connectivity can be more rapidly detected and the delays introduced by direct use of the `java.net.Socket` class are avoided.

Furthermore, some additional functionality was introduced in the mobility agent's ping procedure. While the mobility manager pings its registered nodes at a fixed interval, the mobility agent should be able to dynamically adjust its interval between pings according to environment changes, such as deterioration in network connectivity, and activity intensity and connectivity sensitivity for running applications. Because the diversity of applications and the dynamical changes in wireless environments, is should be possible to apply different degrees of connection awareness to the node based on these factors. If a node detects loss in connectivity, this could mean that the node is entering more unstable coverage area, and the connectivity awareness should be increased. In good coverage areas, connectivity loss will be less frequent, and the connectivity awareness can be decreased according to when the last connectivity loss was detected. Applications running on the nodes will have different requirements to the offered network connectivity. An application that is dependent on having network connectivity at all times will require that the connectivity awareness is high in order to be able to take appropriate actions when deterioration is detected. Other applications may not have the same requirements, and will not affect the overall connectivity awareness on the nodes it is running. The impact of one applications connectivity sensitivity will depend on its activity level. An application with high sensitivity and high activity will require that the connectivity status is updated rapidly, so deterioration can be detected at an early stage and appropriate actions can be taken. With this approach, changes were done in order to develop an appropriate scheme that will take these considerations into account. The illustration in Figure 4.6 shows an overview of the different environment changes that the connectivity continuously should adjust to.

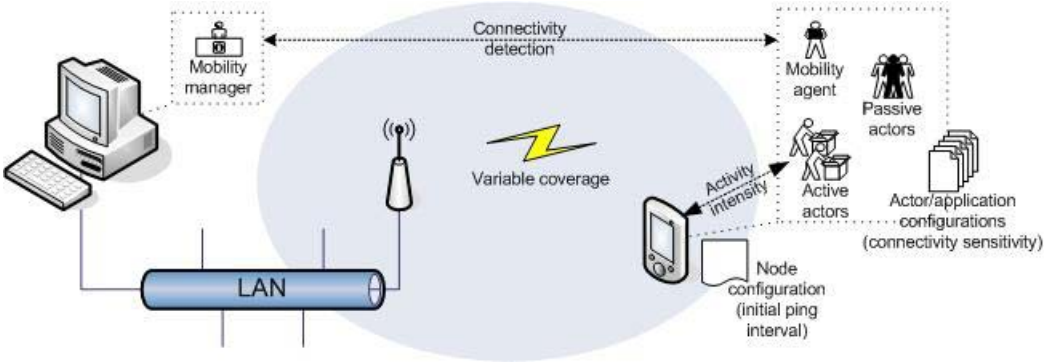

Figure 4.6 - Environment changes that affect the connectivity scheme

Dynamic adjustment of the interval between pings according to detection of connection loss was already implemented in the platform. The initial value of this interval is possible to set in the corresponding configuration file of the node. In this solution, the interval between pings is determined dynamically, where loss of connection will cut the current interval in half, and an online connection of more than ten times the initial interval will increase the current interval by the initial interval. This will enable the node to increase the degree of connectivity awareness if connection loss is detected, which may be caused by deterioration in the node's network connectivity. The interval will increase according to the last detected loss,

which may be interpreted as the node is operating in a more stable environment. However, the interval has limit values, with both a maximum and minimum value that will not be exceeded.

The different behavior of applications and their requirement of network connectivity have been taken into account by enabling classification of actors. These classifications group actors by their connectivity sensitivity. An actor can be classified using three levels of connectivity sensitivities: a) low, b) medium, and c) high. In order to set the connectivity sensitivity for each actor, a parameter has been added in the configuration facilities for actors. The overall connectivity awareness of a node will be constituted by the connectivity sensitivities of the actors running at the node. Three levels of connectivity sensitivity for actors have been chosen to experiment with, but in reality higher differentiation may be needed, thus having more classification levels.

In the proposed solution an actor's impact of the node's overall connection sensitivity will be adjusted according the actors activity intensity. Therefore, continuously activity registration of all running application actors is implemented. Activity registration is done by registration of RoleSessionAction requests that originate from other nodes, or are destined to other nodes. It is only the network traffic that should have any influence on the overall network connectivity awareness. The activity intensity is calculated for each interval between pings. Before a new ping is issued, each actor's activity intensity over the last interval is calculated and compared to the previous value. If an actor has increasing activity, its impact on the node's interval between pings will be determined by its configured connectivity sensitivity: a) no impact, b) small decrease of ping interval, and c) medium decrease of ping interval. An actor with high connectivity sensitivity will require higher connectivity awareness when it is active. Therefore, such actors will affect the node's interval between pings by decreasing this value. The impact on the node's ping interval of each actor is relevant to the total number of application actors running on the node. If there is only one application actor running on the node, the impact of this actor will be high.

In addition to this, another functionality that copes with deterioration in the network connectivity parameter has been proposed. The proposed functionality introduces some steps of precaution by using actor mobility to reinstatiate actors that are most vulnerable for deterioration. The proposed classification of actors is applicable for this solution as well. Actors classified with high connectivity sensitivity should be the first to move to a more suitable location when deterioration is detected. The suitable move location should be given in the corresponding configuration file of the actor. The determination of when to move actors is done by the use of a deterioration parameter that increases and decreases according to the continuously detected connection status. The parameter is based on registration of uptime and downtime, and changes according to these times. If the node detects a connection loss, the last uptime will affect the deterioration parameter by decreasing the value according to what is registered. When the node detects connection again, the downtime will affect the parameter by increasing the value according to what is registered. The deterioration parameter should not exceed absolute minimum and maximum limit values. When this parameter reaches a threshold value, *ActorMove* procedures will be issued for some application actors. Each classification level has a corresponding threshold value. The lowest threshold applies for actors configured with c), and the highest threshold

applies for actors configured with a). Hence, application actors configured with c) will have the lowest tolerance of deterioration of network connectivity and should be the first to re-instantiated when deterioration is detected. Application actors configured with a) may have an infinite threshold value, which means that the actor never will move.

4.5 Measurement of performance metrics

A performance-monitoring solution has been developed in order to measure and display certain performance metrics in the running system. Some of the concepts of the performance monitor are based on a debug server that also is part of the existing TAPAS platform for small and wireless devices. The main reason for this implementation was the lack of performance measuring capabilities in the existing framework.

The performance metrics that should be monitored are response times, delays and overload traffic. *Response time* is the round-trip time of a transaction that is generated by an actual end-user to a networked application located somewhere in an enterprise network. In TAPAS, the definition can be broadened to include transactions or RoleSessionAction requests that are generated to any actor or director that can acknowledge receipt of the transaction and return a response. *Delay* is the time that it takes to transfer a message from one actor to any other actor or director. *Overload traffic* is traffic introduced by overall mobility management schemes applied to overall traffic.

In order to monitor these metrics, the nodes should be able to capture and measure the correct values and send them to the corresponding performance monitor for display. This support had to be implemented in the platform itself, because overload traffic is not visible for running applications on top of the platform. However, the response times are often more application specific because the times are measured from end-user interaction. In order to implement this support in the platform, some simplifications have been done. The response times are measured from the time that the general actor instance receives a RoleSessionAction request, to the time that a result is received. Thus, response time measurement is supported by the platform, but it should be noted that the times are not measured from the end-user application. It is believed that this method gives results comparable with an application specific solution.

Accurate measurement of the delay between two nodes implies that the internal clock in each node has to be synchronized. If this is the case, the time is registered at the sending node when the message is transferred, and the time in the receiver is registered when the message is received. The delay is simply measured by subtracting the receiving time from the sending time. However, the system is distributed on different nodes with different internal clocks. Network synchronization schemes could be implemented, but these involve complexity and computations that are unsuitable for this approach. Therefore, another solution had to be implemented in order to measure the delay times. In the platform, messages are sent using communication sockets. Before a message can be transferred from one node to the other, a socket connection between the nodes has to be established. Therefore, the delay of transferring the message can be measured at the receiver. The receiver registers the

time when the socket connection is established and the time the message is completely received. The delay is easily calculated by subtracting the end time with the start time.

Overload traffic consists of traffic from mobility management and connectivity handling. The measurement of these is simply done by counting the number of messages sent from the different nodes. The number of mobility management requests from one node is incremented each time a request is issued. Also, the number of pings that the node has issued is also registered and incremented continuously. These numbers are kept separate from each other in order to differentiate between the two types of overload traffic introduced in the system.

The metrics that are measured in the nodes are sent to the performance monitor where it is displayed. The performance monitor should be running at a fixed address, known to all other nodes that should be able to send metric information for display. The data carried in the messages gives additional information when the results are interpreted. In all the messages, the node where the metric is measured and the metric value is given. For response times and delay times, the relevant sender (from) and receiver (to) are given. Also, an additional field can hold different information like relevant requests (e.g. mobility management requests) or other information (e.g. current interval between pings at a node). With this approach, it is easier to monitor the performance of the overall system because all metrics are collected and displayed in one place. Figure 4.7 shows the basic concepts of the performance monitor. Nodes that run the TAPAS platform is able to send measured and calculated performance metrics to a fixed address, which runs a performance monitor that displays the received data.

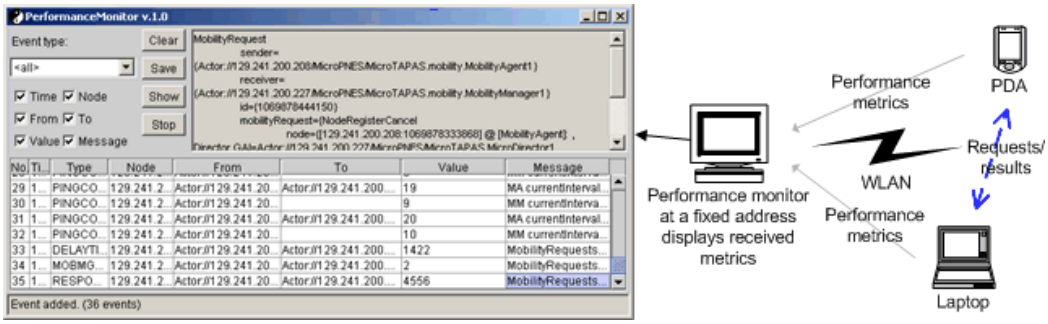

Figure 4.7 - Performance monitoring of nodes running the TAPAS platform

4.6 Challenges and problems

When working with this project assignment, several problems and challenges where encountered. These where related to TAPAS and the existing platform for wireless devices, and physical equipment. Previous subchapters have already discussed some of problems that where dealt with related to the functionalities and implementation of the TAPAS platform for wireless devices. This section will describe some of the other challenges and problems that in some way affected the work with this project assignment.

4.6.1 TAPAS concept and platform implementation

The basics of the TAPAS concept are in principle quite simple. However, the complete architecture has many aspects, and is rather comprehensive. One challenge was therefore to grasp the context of TAPAS, and relate this understanding to real feature implementations of the architecture concept.

The task was performed with the focus on the existing platform for wireless devices. In order to be able to investigate and experiment with this platform, it was necessary to understand the implementation and how it worked. This turned out to be a challenge, because of the complexity of the system, and resent changes and adjustments that only were documented with inline comments in the code. Therefore, some effort was done in order to get the complete picture of some of these new and adjusted features. However, the basic features of the system are well documented in earlier reports and articles and were easier to understand.

4.6.2 Equipment

In order to perform the task of the project assignment, wireless equipment was used for testing. The equipment has been used in earlier work and was therefore configured to function with the existing platform for handheld and wireless devices. However, some problems were encountered with the PDA and its WLAN support. The PDA used in this project does not have built-in WLAN support, but adds this by using an expansion pack that enables insertion of a WLAN PCMCIA- card. The functioning of the expansion pack and the WLAN card was quite unreliable. The specific reason for these problems was not found, but the problem seemed to be caused by the actual PDA device. Therefore, the same expansion pack and WLAN card was used, and the PDA were changed with another device of the same model. After some configuration with this new device, the equipment functioned properly and could be used in the work. This problem introduced some unnecessary delays in performing the task of the project.

5 Testing of functionality

In this chapter some simple test cases that are related to the work described in the previous chapter are presented and discussed. The tests were performed in order to show how the solutions affected performance issues in the system.

5.1 Performance issues

It turned out to be more difficult than expected to show that the improvements in the connectivity scheme directly affected performance in the system. Even with the developed performance monitor utility, described in the previous chapter, it was quite problematic. The main reason for this is that even with an improved connectivity scheme, the platform lacks of functionality that use the information gathered from this scheme. This is the case for mobile nodes, running mobility agents that pings the mobility manager/director. The information about the current status of the node, should affect the behaviour of the overall system, e.g. MicroPNES should not try to setup socket connections to other nodes if the detected status is offline. This could lead to increased delays and response times caused by socket connection timeouts, as described in the previous chapter. The connectivity scheme is developed in such way that the awareness of the node's current status is increased if connection loss is detected, thus MicroPNES should for instance wait for connectivity detected again before trying to setup new socket connections. It is believed that without some additional functionality, that the connectivity scheme alone does not affect system performance metrics as delay and response times. However, dynamic adjustment of ping intensity will affect overhead traffic.

With the mobility manager and its detection of the connectivity of registered nodes, the case is different. The currently registered information about the nodes is used e.g. when an actor is trying to send a request to another actor (located at another node). An ActorDiscovery request is issued before a RoleSessionAction request is sent. The RoleSessionAction request will be dismissed by the sender if the node where the receiving actor is currently registered as offline.

Some tests were carried out that show the dynamical behaviour of the ping intensity and its effect on overhead traffic. The tests were done with both dynamic and static schemes in order to compare and discuss the results in both cases.

5.2 Test of the connectivity scheme

The tests that are presented in this subchapter show how the general behaviour of the connectivity scheme affects overhead traffic introduced by ping messages. It also shows how the dynamic scheme performs compared to static configured intervals under different conditions.

5.2.1 Setup and configuration

In order to show the dynamic behaviour of the connectivity scheme, two different tests scenarios were used. Both scenarios use the same setup and configurations, but the node involved in the tests are exposed to different changes in connectivity status. Also, for each scenario, the dynamic scheme is compared to the performance of two other methods using static ping intervals.

Both scenarios involve two nodes; one running a mobility manager and another running a mobility agent. The node running the mobility manager is a desktop computer connected to a LAN. The node running the mobility agent is a PDA with a WLAN card, which uses an access point that is connected to the LAN. The scenario is showed in Figure 5.1.

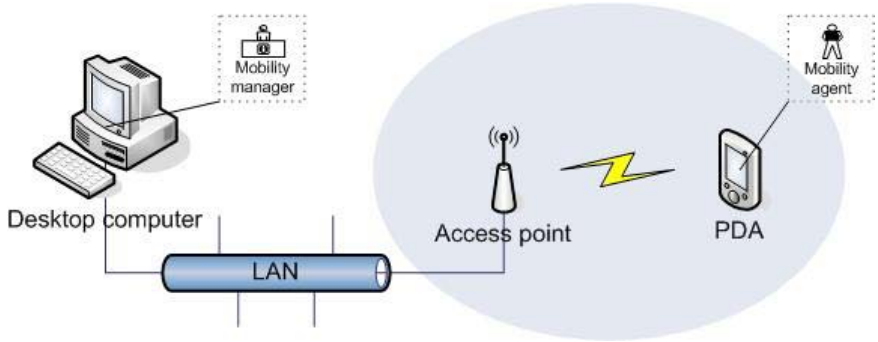

Figure 5.1 - The setup used in the performed tests

The details of the equipment and Java software running in the nodes can be found in Appendix D.

The mobile and wireless node (PDA) will send ping messages to the desktop computer that runs its associated mobility manager, in order to determine its connectivity status. With the dynamic scheme, the ping intensity will adjust itself according environment changes; the continuously detected connectivity and running actor's activity intensity and connectivity sensitivity. The tests scenarios are performed within a fixed time interval with simulation of connectivity loss at fixed times. Therefore, these tests only expose the ping scheme to changes in connectivity. It is sufficient that only one factor affect the ping scheme in order to show its dynamical behaviour and its effect on overhead traffic.

The first test scenario is used to show how the scheme is performing in relative stable environments where connectivity losses are less frequent. In the second scenario a more dynamic environment is simulated with more frequent connectivity losses. Both scenarios span over a time period of 5 minutes (300 seconds). The time is started when the mobility agent is up and running at the PDA, and runs until the period expires. Within this period, changes in the wireless connection are simulated by removing and inserting the WLAN card in the PDA. This is done at fixed times, and how quick the node will detect the changes depends on its active connectivity scheme.

In the first scenario the sequence in Table 5.1 is simulated.

Time (sec)		Connected	Not connected
0	210	X	
210	240		X
240	300	X	

Table 5.1 - The simulated sequence in the first test scenario

In the second scenario the sequence in Table 5.2 is simulated.

Time (sec)		Connected	Not connected
0	80	X	
80	120		X
120	180	X	
180	240		X
240	290	X	
290	300		X

Table 5.2 - The simulated sequence in the second test scenario

For each test scenario three different connectivity schemes were used. The reason for this is to demonstrate how the different schemes perform compared to each other. In addition to the improved dynamic connectivity scheme, the scenarios were carried out using two schemes with static intervals between pings. The configurations for both the dynamic and the static schemes are showed in Table 5.3.

#	Scheme	Initial ping interval (sec)
1	Dynamic	3
2	Static	3
3	Static	15

Table 5.3 - The configured ping intervals in the test scenarios

The results of using each scheme are compared to each other in order to demonstrate the advantage of dynamical adjustment of the ping intensity. The two static schemes represent upper and lower intervals, for demonstrating differences in efficiency in different operating environments (e.g. stable and unstable). In the following sections, the schemes will sometimes be referred to by using the numbers in the left column in Table 5.3.

5.2.2 Results from the first test scenario

The illustration in Figure 5.2 shows the actual interval between pings related to time in the first scenario. The static schemes use the same interval the entire period, and the dynamic scheme adjusts the interval according to changes in detected connectivity.

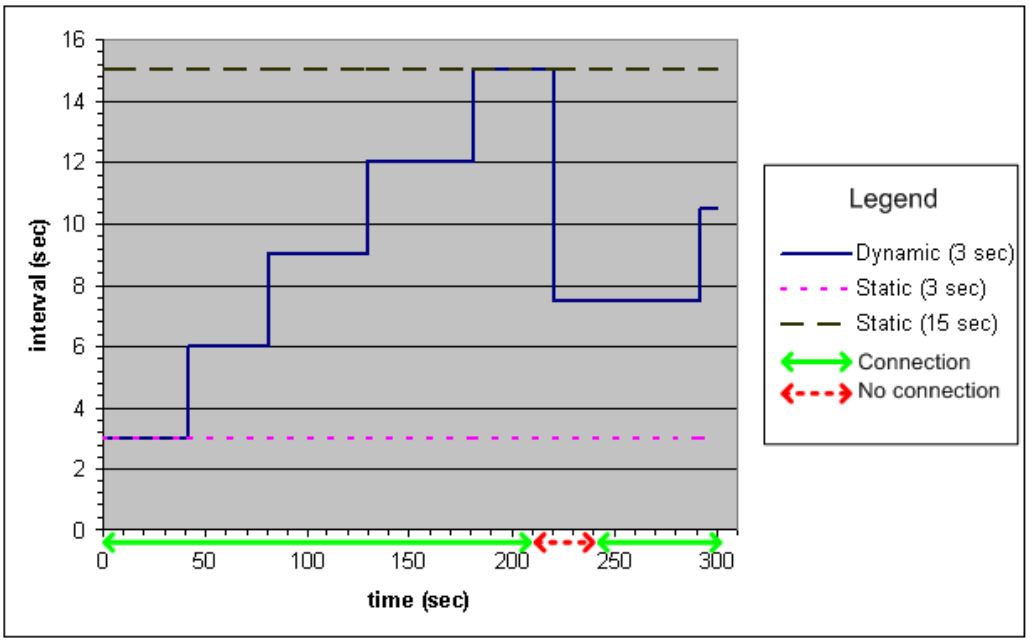

Figure 5.2 - A graph of the ping intervals in the first test scenario

In the dynamic scheme, the interval increases as long as the connection seems stable. When loss is detected, the interval is cut in half to increase the connectivity awareness.

Table 5.4 shows how the different schemes perform according to detection of connectivity events. The delay from the actual event occurred to the detection time is shown for each scheme. The times are rounded to whole seconds, because the values can not be given with a higher accuracy. The reason for this is a small uncertainty relevant to the actual time which the node loose and obtain connectivity (this is simulated removing and inserting the WLAN card). However, it is believed that the values given are representative and is used in the discussion. The times will variate within the current interval; the worst case is when the node loose connection right after a successful ping, then the loss will not be detected before a new ping is issued after the interval period.

Type	**Actual time (sec)**	**Detected after (sec)**		
		Scheme 1	**Scheme 2**	**Scheme 3**
No connection	210	2 (212)	3 (213)	4 (214)
Connection	240	11 (251)	4 (244)	20 (260)
Total		*13*	*7*	*24*

Table 5.4 - The detection delays of the different schemes in the first test scenario

Also, it is interesting to se how the different schemes affect the overhead traffic by issuing ping messages. The number of messages that each scheme has produced is shown in Table 5.5. Also, the numbers is grouped according to if messages were successful or not, thereby affecting overhead traffic or not.

Ping messages	Scheme 1	Scheme 2	Scheme 3
Successful	34	87	17
Failed	5	10	3
Total	*39*	*97*	*20*

Table 5.5 - The number of pings in the first test scenario

5.2.3 Results from the second test scenario

The interval between pings in the second scenario is showed in Figure 5.3. In this scenario connectivity loss is more frequent, which is indicated on the x-axis in the diagram.

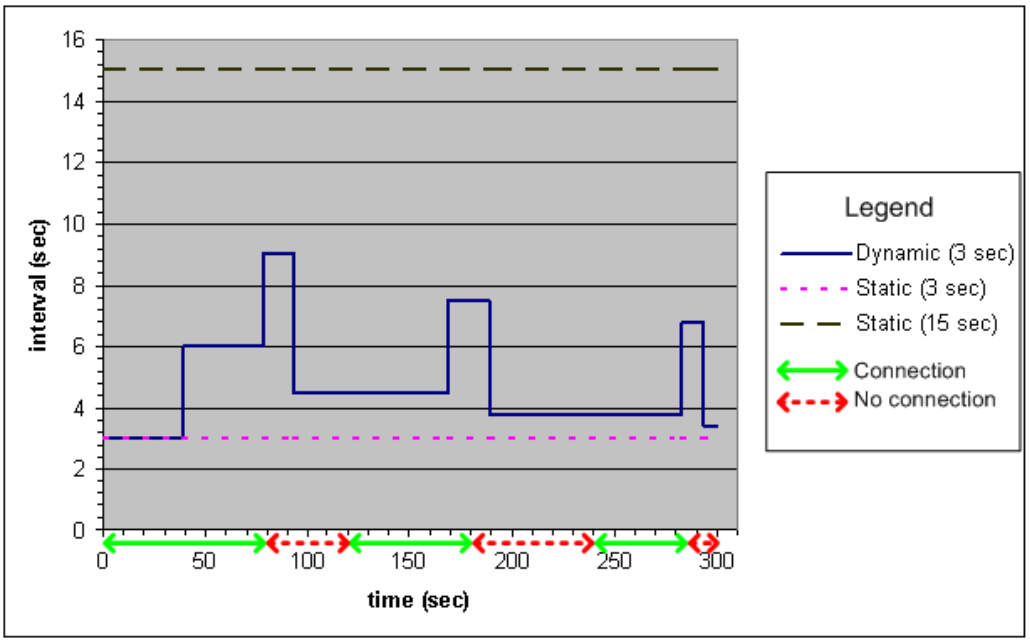

Figure 5.3 - A graph of the ping intervals in the second test scenario

The delay times for detection of changes in connectivity are showed in Table 5.6.

Type	Actual time (sec)	Detected after (sec)		
		Scheme 1	Scheme 2	Scheme 3
No connection	80	9 (89)	4 (84)	16 (96)
Connection	120	10 (130)	13 (133)	22 (142)
No connection	180	5 (185)	2 (182)	7 (187)
Connection	240	6 (246)	12 (252)	23 (263)
No connection	290	1 (291)	2 (292)	4 (294)
Total		*22*	*33*	*72*

Table 5.6 - The detection delays of the different schemes in the second test scenario

In Table 5.7 the number of ping messages sent by each scheme is given.

Ping messages	Scheme 1	Scheme 2	Scheme 3
Successful	38	55	12
Failed	28	42	8
Total	*66*	*97*	*20*

Table 5.7 - The number of pings in the second test scenario

For additional information about the results, please refer to the test logs which are included in Appendix D.

5.2.4 Discussion of the results

The results from the first scenario shows that when using the dynamic scheme, the ping interval is increased from 3 seconds to 15 seconds before the first loss is detected (at 212 sec). The loss causes the interval to be changed to 7.5 seconds, thus increasing the connectivity awareness. The connection remains stable for the rest of the period, causing the interval to be increased to 10.5 seconds in the end of the period (at 291 sec). According to these results, the ping interval reaches a maximum value of 15 seconds, and a minimum value of 3 seconds. The interval of the static schemes is, of course, unchanged for the whole period. This demonstrates that the behavior of the dynamic connectivity scheme is adapting to the changes in connection status.

In Table 5.4, the time each scheme uses in detecting changes in the status is showed. When comparing these values, the static scheme with the highest ping intensity (3 sec) seems to have the best performance. This scheme introduces a total delay of 7 seconds, while the corresponding values for the dynamic scheme is 13 seconds, and 24 seconds for the static scheme with the lowest ping intensity (15 seconds). As one would expect, the scheme with the highest intensity have the best performance (or awareness) when detecting changes in the connection. The dynamic scheme has relatively good results compared to the others. These results show that a low intensity means lower awareness.

The static scheme with the highest ping intensity performs well according to detection of changes. However, this scheme issues many more ping messages than the other schemes. The ping messages are part of the overhead traffic, and should therefore be kept at a minimum when not necessary. Here, the dynamic scheme has the advantage of increasing the ping interval, and therefore issuing less ping messages, when the connection is relatively stable. The dynamic scheme issues a total of 39 messages. The static schemes, with fixed intervals of 3 seconds and 15 seconds, issue a total of 97 and 20 ping messages, respectively. If only successful messages are counted, the numbers show similar relations to each other (see Table 5.5). In this view, the static scheme with the lowest intensity has the best performance. Low ping intensity means that fewer messages are issued.

The results from the first scenario shows that a high intensity leads to better performance when detecting changes, but introduce much overhead traffic. The static scheme with low ping intensity has better performance according to overhead traffic, but has longer delays when detecting changes.

In the second scenario, the dynamic interval is kept lower because of more frequent changes in the connectivity of the node. Each time a loss is detected, the interval is

decreased, causing the node to send ping messages more frequently. According to the results, the ping interval reaches a maximum value of 9 seconds, and a minimum value of 3 seconds. As for the first scenario, the results demonstrate that the dynamic scheme adapts well to the connectivity changes.

The results from the second scenario show that the dynamic scheme has the smallest total delay time in detecting changes (22 seconds). For the static schemes, the high intensity scheme has much smaller total delay (33 seconds) than the low intensity scheme (72 seconds). This scenario strengthens the results from the first scenario, which showed that higher ping intensity gives increased awareness and smaller detection delays. One important difference from the first scenario is that the dynamic scheme now shows the best performance.

The number of issued ping messages for the dynamic scheme is higher for the second scenario. This is because the ping intensity is increased by the frequent changes in connectivity. For the static schemes that same number of ping messages are issued, but the number of successful and failed ones are of course different. Anyway, the results show the same as for the first scenario; increased ping intensity means additional overhead traffic.

5.2.5 Test conclusion

The two test scenarios have showed how the ping interval affects certain issues as overhead traffic and connectivity awareness. The schemes using static ping interval can only satisfy one of the issues at a time. The static low intensity scheme used in these scenarios shows good results in regard to overhead traffic, but has poor connectivity awareness compared to the other schemes. This scheme is best suited for operating environments with relative stable connections. The static high intensity scheme shows good results in regard to connectivity awareness, but introduces more overhead traffic. This scheme is best suited for operating environments that have very unstable connections all the time. However, a better solution is to have a dynamic scheme which adapts to the operating environment in order to adjust to the most suitable ping intensity at any time. The results showed that the dynamic scheme performed quite well in both test scenarios. Therefore, one could conclude from the results given in these scenarios, that the dynamic scheme provides the best overall solution.

However, it should be noted that the test is quite simple; only two scenarios and three different configurations are used in the test, and the system is observed in a relatively short time period (5 minutes). In order to have more reliable results and a wider basis for discussion, several more test scenarios should be done with different configurations and using longer time periods. Also, one should include actors with different configurations (e.g. connectivity sensitivity) and stimulate these in order to see how the activity intensity affects the dynamic ping intensity.

6 Conclusion

In this project report, the results and achievements from the investigation and experimentation with Plug-and-Play support functionality for wireless devices are described. The TAPAS concept and its support platform developed for such devices have been used as a basis for this work.

The recent advances in the areas of wireless communications and devices have lead to widespread use of wireless technologies. IEEE 802.11g and Bluetooth are promising network technologies that probably will dominate in the future. The TAPAS concept should be able handle the mobility and highly dynamic environment introduced by such technologies.

The existing TAPAS platform based on J2ME has been used in the investigation and experimentation with Plug-and-Play support functionality. With the emphasis on reliability, flexibility, and performance issues, some of the key functionalities needed to handle nowadays and future challenges posed by wireless environments have been studied. The achievements from this work are described in detail in this report. Demonstrations of existing functionality related to actor mobility have revealed flaws and inefficiencies. Solutions for these problems have been given and are implemented to improve efficiency in the platform. A proposal for more efficient handling of dynamic wireless connections has also been given. This proposal aims at providing a scheme for connectivity awareness that adapts to environmental changes (e.g. connectivity deterioration, changes in application activity, etc.).

Some test scenarios have been carried out to show how the implemented connectivity scheme for handling wireless connections performs compared to schemes with static behavior. The results from these test scenarios showed that this scheme performs well, and is the best solution for dynamic connections compared to the static schemes. Its adaptive behavior should make it suitable for many different types of operating environments in terms of changes in connectivity.

However, tests that show how these improvements affect the overall performance in the platform have not been carried out. This is because the platform is missing some functionality that takes advantage of the information from the proposed connectivity scheme. Preparations for such a test have been done by developing a performance monitor application, which is able to display performance metrics calculated and measured by the nodes. This application has, however, not been used in the testing in this project, but should be possible to use in tests of further developments in the platform.

With elaborated schemes for mobility handling and dynamic connections, the TAPAS concept should be suitable for wireless environments. However, further development and experimentation with the TAPAS platform for small and wireless devices should be performed in order to reveal limitations and enhance proposed solutions. The TAPAS platform can be further developed to support Bluetooth and Java-enabled mobile phones. The connectivity in Bluetooth network is highly dynamic, and performance issues of the TAPAS concept for such technology can be studied.

References

AAGE03 Finn Arve Aagesen, Bjarne E. Helvik, Chutiporn Anutariya, and Mazen Malek Shiaa: ***On Adaptable Networking***. The 2003 International Conference on Information and Communication Technologies (ICT 2003), Bangkok- Thailand, April 2003. Slides: http://tapas.item.ntnu.no/presentations/ICT2003.pdf, [Accessed November 2003]

AAGE99 Finn Arve Aagesen, Bjarne Helvik, Vilas Wuwongse, Hein Meling, Rolv Bræk and Ulrik Johansen: **Towards a Plug and Play Architecture for Telecommunications**, IFIP Fifth International Conference on Intelligence in Networks (SmartNet99), Bangkok - Thailand, November 1999

AGLETS IBM Research, Aglets, http://www.trl.ibm.com/aglets/, [Accessed November 2003]

DARPA DARPA, Active Networks page, http://www.darpa.mil/ato/programs/activenetworks/actnet.htm, [Accessed November 2003]

ANTS ANTS, Active Node Transfer System, http://www.cs.washington.edu/research/networking/ants/, [Accessed November 2003]

MALE02 Mazen Malek and Finn Arve Aagesen: ***Mobility management in a Plug and Play Architecture***, IFIP TC6 Seventh International Conference on Intelligence in Networks (SmartNet2002), Saariselka - Finland, April 2002. Published by Kluwer Academic Publishers

AUTONO IBM Research, Autonomic computing, http://www.research.ibm.com/autonomic/, [Accessed October 2003]

KEPH03 Jeffrey O. Kephart, Davic M. Chess: ***The Vision of Autonomic Computing***, Published by the IEEE Computer Society, January 2003

ZEROCO Zero Configuring Networking (Zeroconf), http://www.zeroconf.org/, [Accessed December 2003]

APPLE Apple Rendezvous, http://developer.apple.com/macosx/rendezvous/, [Accessed December 2003]

BLUETO Bluetooth v1.2 core specification, November 2003, https://www.bluetooth.org/foundry/adopters/document/Bluetooth_Core_Specification_v1.2, [Accessed November 2003]

SUN1 Connected Limited Device Configuration, Specification Version 1.1, Java 2 Platform, Micro Edition, Sun Microsystems Inc, March 2003

SUN2 White Paper: ***CDC: An application Framework for Personal Mobile Devices, Java 2 Platform, Micro Edition (J2ME)***, Sun Microsystems Inc, June 2003. http://java.sun.com/products/cdc/wp/cdc-whitepaper.pdf, [Accessed October 2003]

JCP Java Specification Requests for J2ME, http://jcp.org/en/jsr/tech?listBy=1&listByType=platform, [Accessed November 2003]

ERCIM European Research Consortium and Mathematics (ERCIM) News, Number 54, July 2003. Special issue: ***Applications and Service Platforms for the Mobile User***,

http://www.ercim.org/publication/Ercim_News/enw54/index.html [Accessed November 2003]

MALE03 Mazen Malek Shiaa, ***Mobility Support Framework in Adaptable Service Architecture***, Network Control and Engineering for QoS, Security and Mobility 2003 IFIP/IEEE Conference (NetCon'2003), Muscat-Oman, October 2003

JIAN03 Shanshan Jiang and Finn Arve Aagesen: *XML-based Dynamic Service Behaviour Representation*. NIK'2003. Oslo, Norway, November 2003

JOHA01 Ulrik Johansen, ***Plug-and-play - Software design, implementation and use***. Plug-and-Play Technical Report, Department of Telematics, NTNU, 2001-02-10, ISSN 1500-3868

LÜHR03 Eirik Lühr, ***TAPAS for Wireless PDA***, Project Report, Department of Telematics, NTNU, 2003

JAVAWO David Reilly, ***Simple handling of network timeouts***, JavaWorld article, September 1999, http://www.javaworld.com/javaworld/jw-09-1999/jw-09-timeout.html, [Accessed November 2003]

Appendix A – Wireless technologies

A.1 WLAN standards

Table A.1 summarizes some of the existing WLAN standards. Currently, the most widespread standards operate in the 2.4GHz band. However, standards operating in the 5GHz band have the advantage of having less interference risks. In the US, the 802.11a standard may have a role for fixed wireless in the corporate market. In Europe, the HIPERLAN/2 may succeed in other markets, such as healthcare, where security, QoS and response time predictability are particular critical. HIPERLAN/2 has the advantage of being more than a WLAN solution, having classes of services defined in the specification that guarantee consistent performances in time critical applications such as multimedia.

Standard	Description	Approved
IEEE 802.11	Standard for WLAN operations at data rates up to 2 Mbit/s in the 2.4 GHz band.	July 1997.
IEEE 802.11a	Standard for WLAN operations at data rates up to 54 Mbit/s in the 5 Ghz band.	Sept 1999. (US)
IEEE 802.11b	Standard for WLAN operations at data rates up to 11 Mbit/s in the 2.4 GHz band.	Sept 1999.
IEEE 802.11g	High-rate extension to 802.11b allowing for data rates up to 54 Mbit/s in the 2.4 GHz band.	July 2003.
HIPERLAN/1	Standard for WLAN operations at data rates up to 20 Mbit/s in the 5 GHz band.	Oct 1996. (Europe)
HIPERLAN/2	Standard for WLAN operations and wireless access to a variety of networks (3G, ATM, IP) allowing data rates up to 54 Mbit/s in the 5 GHz band.	Feb 2000. (Europe)

Table A.1 - Summary of WLAN compatible standards and their characteristics

Appendix B – Java 2, Micro Edition Platform

B.1 Comparison of CDC profiles and Java 2, Standard Edition

Table B.1 [SUN2] shows a comparison of available class archives in CDC profiles and the J2SE 1.3.1 Platform.

Package	J2SE 1.3.1	FP 1.0	PBP 1.0	PP 1.0
`java.applet`	●	-	-	P
`java.awt.*`	●	-	P	P
`java.beans.*`	●	-	P	P
`java.io`	●	●	●	●
`java.lang.*`	●	●	●	●
`java.math`	●	P	P	●
`java.net`	●	●	●	●
`java.rmi.*`	●	OP	OP	OP
`java.security.*`	●	●	●	●
`java.sql`	●	OP	OP	OP
`java.text`	●	●	●	●
`java.util.*`	●	●	●	●
`javax.accessibility`	●	-	-	-
`javax.naming.*`	●	-	-	-
`javax.rmi.*`	●	-	-	-
`javax.sound.*`	●	-	-	-
`javax.swing.*`	●	-	-	-
`javax.transactions`	●	-	-	-
`javax.omg.*`	●	-	-	-
`javax.microedition.io.*`	-	●	●	●
`javax.microedition.xlet.*`	-	-	●	●
●: full support, P: partial support, -:not supported, OP: replaced by optional package				

Table B.1 - Comparison of the CDC profiles and J2SE 1.3.1

Appendix C – TAPAS architecture

C.1 Original layered design model

The original layered design model [JOHA01, Appendix B, page 14]:

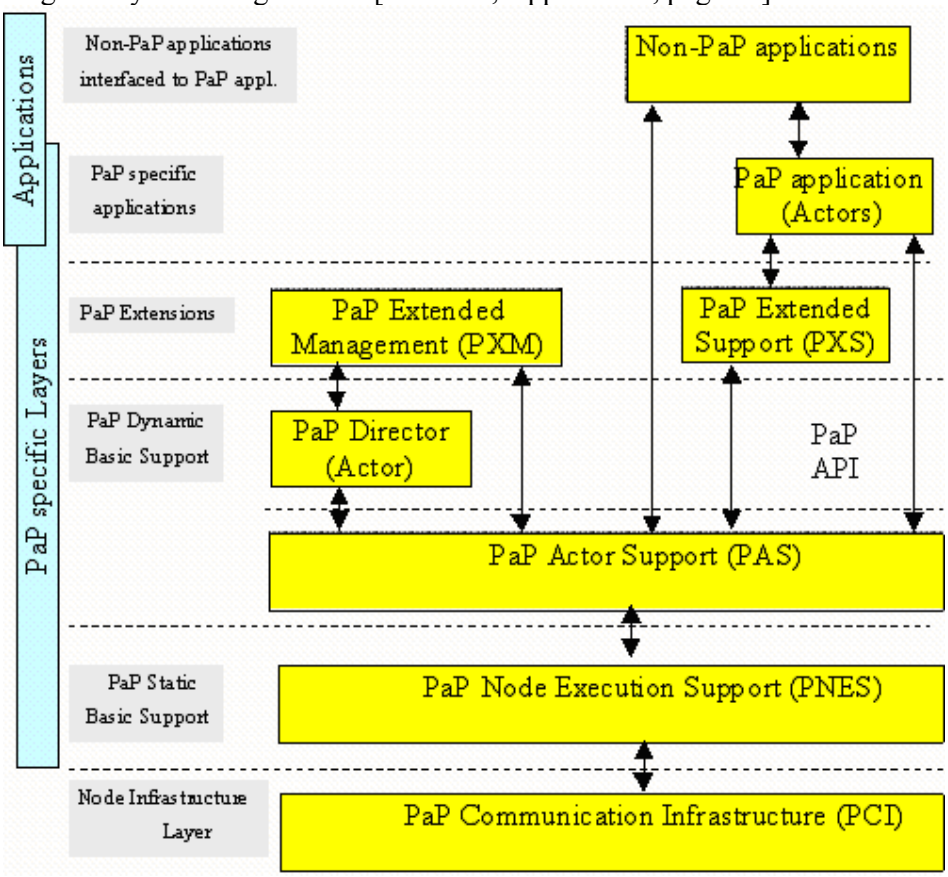

Figure C.1 - The original layered design model of TAPAS

C.2 Example view of execution environment

Example view of TAPAS platform for software execution [JOHA01, Appendix B, page 18]:

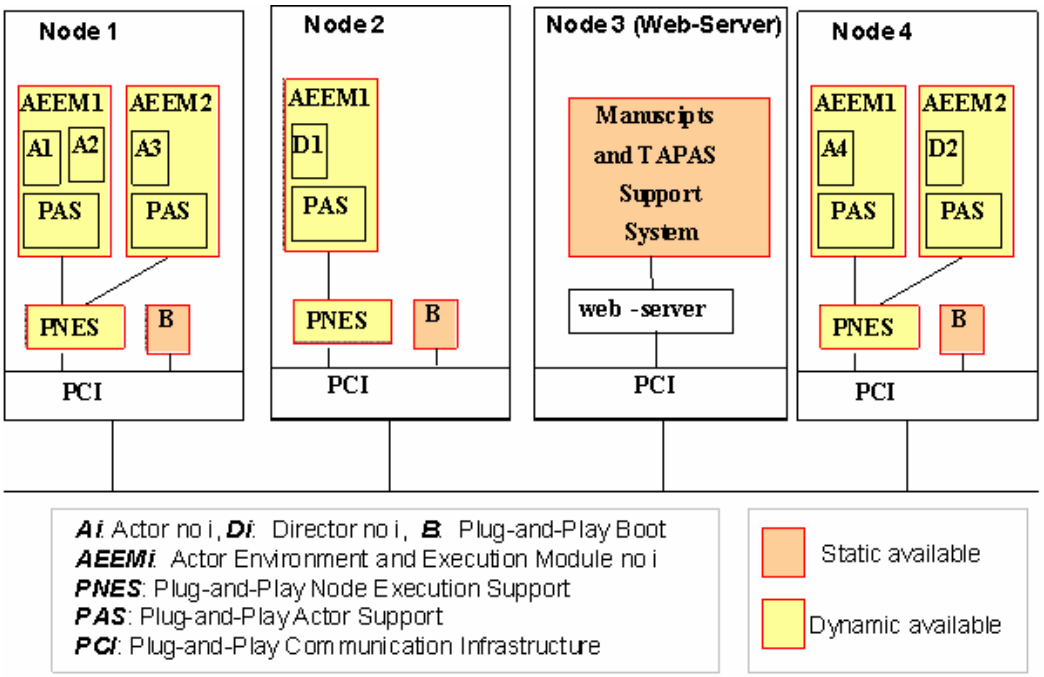

Figure C.2 - Example view of TAPAS platform for software execution

Appendix D – Test data

D.1 Test equipment specifications

Specification	Desktop computer	PDA
Model	N/A	Compaq iPAQ 3660
Processor	AMD Athlon XP 1700+, 1467 MHz	StrongARM SA1110, 206 MHz
Memory	256 MB RAM	64 MB RAM, 16 MB ROM
Operating system	Windows 2000, 5.00.2195	Windows CE, 3.0.9348
Network support	3COM Etherlink 10/100	Expansion slot with ZyAIR B-100 802.11b PCMCIA
Java support	Java Hotspot™ Client VM, 1.4.2_03-b02 by Sun Microsystems Inc.	IMB J9™, 2.0 (Included in IBM's WebSphere Studio Device Developer)

Table D.1 - Specifications of the test equipment used in the project

D.2 Logs from test scenarios

The complete logs for each test scenario are shown in the following sections. For each test scenario, there are three corresponding logs for each connectivity scheme used (as described in section 5.2.1). These logs also show how the indicator for deterioration in connectivity changes (indicated by *connDet*).

D.2.1 First test scenario

Dynamic scheme (start interval 3 sec)

```
***Ping # 1, currentInterval=3000msec, uptime=5658msec, status=offline
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=6761msec)
***Ping # 2, currentInterval=3000msec, uptime=9022msec, status=online
***Ping # 3, currentInterval=3000msec, uptime=12024msec, status=online
***Ping # 4, currentInterval=3000msec, uptime=15027msec, status=online
***Ping # 5, currentInterval=3000msec, uptime=18030msec, status=online
***Ping # 6, currentInterval=3000msec, uptime=21033msec, status=online
***Ping # 7, currentInterval=3000msec, uptime=24036msec, status=online
***Ping # 8, currentInterval=3000msec, uptime=27039msec, status=online
***Ping # 9, currentInterval=3000msec, uptime=30042msec, status=online
***Ping # 10, currentInterval=3000msec, uptime=33045msec, status=online
***Ping # 11, currentInterval=3000msec, uptime=36048msec, status=online
***Ping # 12, currentInterval=6000msec, uptime=42051msec, status=online
***Ping # 13, currentInterval=6000msec, uptime=48054msec, status=online
***Ping # 14, currentInterval=6000msec, uptime=54057msec, status=online
***Ping # 15, currentInterval=6000msec, uptime=60060msec, status=online
***Ping # 16, currentInterval=6000msec, uptime=66064msec, status=online
***Ping # 17, currentInterval=6000msec, uptime=72067msec, status=online
***Ping # 18, currentInterval=9000msec, uptime=81070msec, status=online
***Ping # 19, currentInterval=9000msec, uptime=90073msec, status=online
***Ping # 20, currentInterval=9000msec, uptime=99076msec, status=online
***Ping # 21, currentInterval=9000msec, uptime=108079msec, status=online
***Ping # 22, currentInterval=9000msec, uptime=117082msec, status=online
***Ping # 23, currentInterval=12000msec, uptime=129085msec, status=online
***Ping # 24, currentInterval=12000msec, uptime=141088msec, status=online
***Ping # 25, currentInterval=12000msec, uptime=154131msec, status=online
***Ping # 26, currentInterval=12000msec, uptime=166134msec, status=online
***Ping # 27, currentInterval=15000msec, uptime=181137msec, status=online
***Ping # 28, currentInterval=15000msec, uptime=196140msec, status=online
***Ping # 29, currentInterval=15000msec, uptime=211143msec, status=online
***connDet:0 (-13)
```

```
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 1250.
(uptime=212393msec)
***Ping # 30, currentInterval=7500msec, uptime=219643msec, status=offline
***Ping # 31, currentInterval=7500msec, uptime=227147msec, status=offline
***Ping # 32, currentInterval=7500msec, uptime=234650msec, status=offline
***Ping # 33, currentInterval=7500msec, uptime=242153msec, status=offline
***Ping # 34, currentInterval=7500msec, uptime=249657msec, status=offline
***connDet:5 (+5)
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=250663msec)
***Ping # 35, currentInterval=7500msec, uptime=258035msec, status=online
***Ping # 36, currentInterval=7500msec, uptime=266131msec, status=online
***Ping # 37, currentInterval=7500msec, uptime=273634msec, status=online
***Ping # 38, currentInterval=7500msec, uptime=281137msec, status=online
***Ping # 39, currentInterval=10500msec, uptime=291640msec, status=online
```

Static scheme (interval 3 sec)

```
***Ping # 1, currentInterval=3000msec, uptime=5608msec, status=offline
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=8825msec)
***Ping # 2, currentInterval=3000msec, uptime=9018msec, status=online
***Ping # 3, currentInterval=3000msec, uptime=12029msec, status=online
***Ping # 4, currentInterval=3000msec, uptime=15095msec, status=online
***Ping # 5, currentInterval=3000msec, uptime=18142msec, status=online
***Ping # 6, currentInterval=3000msec, uptime=21196msec, status=online
***Ping # 7, currentInterval=3000msec, uptime=24273msec, status=online
***Ping # 8, currentInterval=3000msec, uptime=27331msec, status=online
***Ping # 9, currentInterval=3000msec, uptime=30347msec, status=online
***Ping # 10, currentInterval=3000msec, uptime=33376msec, status=online
***Ping # 11, currentInterval=3000msec, uptime=36428msec, status=online
***Ping # 12, currentInterval=3000msec, uptime=39488msec, status=online
***Ping # 13, currentInterval=3000msec, uptime=42563msec, status=online
***Ping # 14, currentInterval=3000msec, uptime=45603msec, status=online
***Ping # 15, currentInterval=3000msec, uptime=48677msec, status=online
***Ping # 16, currentInterval=3000msec, uptime=51743msec, status=online
***Ping # 17, currentInterval=3000msec, uptime=54813msec, status=online
***Ping # 18, currentInterval=3000msec, uptime=57875msec, status=online
***Ping # 19, currentInterval=3000msec, uptime=60934msec, status=online
***Ping # 20, currentInterval=3000msec, uptime=63938msec, status=online
***Ping # 21, currentInterval=3000msec, uptime=66987msec, status=online
***Ping # 22, currentInterval=3000msec, uptime=70035msec, status=online
***Ping # 23, currentInterval=3000msec, uptime=73066msec, status=online
***Ping # 24, currentInterval=3000msec, uptime=76068msec, status=online
***Ping # 25, currentInterval=3000msec, uptime=79071msec, status=online
***Ping # 26, currentInterval=3000msec, uptime=82074msec, status=online
***Ping # 27, currentInterval=3000msec, uptime=85077msec, status=online
***Ping # 28, currentInterval=3000msec, uptime=88080msec, status=online
***Ping # 29, currentInterval=3000msec, uptime=91083msec, status=online
***Ping # 30, currentInterval=3000msec, uptime=94086msec, status=online
***Ping # 31, currentInterval=3000msec, uptime=97089msec, status=online
***Ping # 32, currentInterval=3000msec, uptime=100092msec, status=online
***Ping # 33, currentInterval=3000msec, uptime=103095msec, status=online
***Ping # 34, currentInterval=3000msec, uptime=106098msec, status=online
***Ping # 35, currentInterval=3000msec, uptime=109101msec, status=online
***Ping # 36, currentInterval=3000msec, uptime=112104msec, status=online
***Ping # 37, currentInterval=3000msec, uptime=115107msec, status=online
***Ping # 38, currentInterval=3000msec, uptime=118110msec, status=online
***Ping # 39, currentInterval=3000msec, uptime=121113msec, status=online
***Ping # 40, currentInterval=3000msec, uptime=124116msec, status=online
***Ping # 41, currentInterval=3000msec, uptime=127119msec, status=online
***Ping # 42, currentInterval=3000msec, uptime=130122msec, status=online
***Ping # 43, currentInterval=3000msec, uptime=133125msec, status=online
***Ping # 44, currentInterval=3000msec, uptime=136128msec, status=online
***Ping # 45, currentInterval=3000msec, uptime=139131msec, status=online
***Ping # 46, currentInterval=3000msec, uptime=142134msec, status=online
***Ping # 47, currentInterval=3000msec, uptime=145137msec, status=online
***Ping # 48, currentInterval=3000msec, uptime=148140msec, status=online
***Ping # 49, currentInterval=3000msec, uptime=151144msec, status=online
***Ping # 50, currentInterval=3000msec, uptime=154147msec, status=online
***Ping # 51, currentInterval=3000msec, uptime=157150msec, status=online
***Ping # 52, currentInterval=3000msec, uptime=160153msec, status=online
***Ping # 53, currentInterval=3000msec, uptime=163157msec, status=online
***Ping # 54, currentInterval=3000msec, uptime=166946msec, status=online
```

```
***Ping # 55, currentInterval=3000msec, uptime=169949msec, status=online
***Ping # 56, currentInterval=3000msec, uptime=172952msec, status=online
***Ping # 57, currentInterval=3000msec, uptime=175956msec, status=online
***Ping # 58, currentInterval=3000msec, uptime=178959msec, status=online
***Ping # 59, currentInterval=3000msec, uptime=181966msec, status=online
***Ping # 60, currentInterval=3000msec, uptime=184969msec, status=online
***Ping # 61, currentInterval=3000msec, uptime=187972msec, status=online
***Ping # 62, currentInterval=3000msec, uptime=190975msec, status=online
***Ping # 63, currentInterval=3000msec, uptime=193978msec, status=online
***Ping # 64, currentInterval=3000msec, uptime=196981msec, status=online
***Ping # 65, currentInterval=3000msec, uptime=199984msec, status=online
***Ping # 66, currentInterval=3000msec, uptime=202987msec, status=online
***Ping # 67, currentInterval=3000msec, uptime=205990msec, status=online
***Ping # 68, currentInterval=3000msec, uptime=208993msec, status=online
***Ping # 69, currentInterval=3000msec, uptime=211996msec, status=online
***connDet:0 (-67)
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 1004.
(uptime=212999msec)
***Ping # 70, currentInterval=3000msec, uptime=215905msec, status=offline
***Ping # 71, currentInterval=3000msec, uptime=218908msec, status=offline
***Ping # 72, currentInterval=3000msec, uptime=221911msec, status=offline
***Ping # 73, currentInterval=3000msec, uptime=224914msec, status=offline
***Ping # 74, currentInterval=3000msec, uptime=227917msec, status=offline
***Ping # 75, currentInterval=3000msec, uptime=230920msec, status=offline
***Ping # 76, currentInterval=3000msec, uptime=233923msec, status=offline
***Ping # 77, currentInterval=3000msec, uptime=236926msec, status=offline
***Ping # 78, currentInterval=3000msec, uptime=239929msec, status=offline
***Ping # 79, currentInterval=3000msec, uptime=242932msec, status=offline
***connDet:10 (+10)
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=243957msec)
***Ping # 80, currentInterval=3000msec, uptime=246742msec, status=online
***Ping # 81, currentInterval=3000msec, uptime=249745msec, status=online
***Ping # 82, currentInterval=3000msec, uptime=252748msec, status=online
***Ping # 83, currentInterval=3000msec, uptime=255753msec, status=online
***Ping # 84, currentInterval=3000msec, uptime=258755msec, status=online
***Ping # 85, currentInterval=3000msec, uptime=261757msec, status=online
***Ping # 86, currentInterval=3000msec, uptime=264760msec, status=online
***Ping # 87, currentInterval=3000msec, uptime=267763msec, status=online
***Ping # 88, currentInterval=3000msec, uptime=270766msec, status=online
***Ping # 89, currentInterval=3000msec, uptime=273769msec, status=online
***Ping # 90, currentInterval=3000msec, uptime=276772msec, status=online
***Ping # 91, currentInterval=3000msec, uptime=279775msec, status=online
***Ping # 92, currentInterval=3000msec, uptime=282778msec, status=online
***Ping # 93, currentInterval=3000msec, uptime=285781msec, status=online
***Ping # 94, currentInterval=3000msec, uptime=288784msec, status=online
***Ping # 95, currentInterval=3000msec, uptime=291787msec, status=online
***Ping # 96, currentInterval=3000msec, uptime=294790msec, status=online
***Ping # 97, currentInterval=3000msec, uptime=297793msec, status=online
```

Static (interval 15 sec)

```
***Ping # 1, currentInterval=15000msec, uptime=1559msec, status=offline
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=6474msec)
***Ping # 2, currentInterval=15000msec, uptime=17339msec, status=online
***Ping # 3, currentInterval=15000msec, uptime=32374msec, status=online
***Ping # 4, currentInterval=15000msec, uptime=47432msec, status=online
***Ping # 5, currentInterval=15000msec, uptime=62493msec, status=online
***Ping # 6, currentInterval=15000msec, uptime=77496msec, status=online
***Ping # 7, currentInterval=15000msec, uptime=92499msec, status=online
***Ping # 8, currentInterval=15000msec, uptime=107502msec, status=online
***Ping # 9, currentInterval=15000msec, uptime=122505msec, status=online
***Ping # 10, currentInterval=15000msec, uptime=137508msec, status=online
***Ping # 11, currentInterval=15000msec, uptime=152511msec, status=online
***Ping # 12, currentInterval=15000msec, uptime=168153msec, status=online
***Ping # 13, currentInterval=15000msec, uptime=183156msec, status=online
***Ping # 14, currentInterval=15000msec, uptime=198159msec, status=online
***Ping # 15, currentInterval=15000msec, uptime=213162msec, status=online
***connDet:0 (-13)
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 1014.
(uptime=214175msec)
***Ping # 16, currentInterval=15000msec, uptime=229008msec, status=offline
***Ping # 17, currentInterval=15000msec, uptime=244011msec, status=offline
```

```
***Ping # 18, currentInterval=15000msec, uptime=259014msec, status=offline
***connDet:3 (+3)
MicroTAPAS.mobility.MicroPingClient::Connection    to    director    (re-)established.
(uptime=260006msec)
***Ping # 19, currentInterval=15000msec, uptime=274891msec, status=online
***Ping # 20, currentInterval=15000msec, uptime=289894msec, status=online
```

D.2.2 Second test scenario

Dynamic scheme (start interval 3 sec)

```
***Ping # 1, currentInterval=3000msec, uptime=875msec, status=offline
MicroTAPAS.mobility.MicroPingClient::Connection    to    director    (re-)established.
(uptime=5773msec)
***Ping # 2, currentInterval=3000msec, uptime=6404msec, status=online
***Ping # 3, currentInterval=3000msec, uptime=9411msec, status=online
***Ping # 4, currentInterval=3000msec, uptime=12481msec, status=online
***Ping # 5, currentInterval=3000msec, uptime=15486msec, status=online
***Ping # 6, currentInterval=3000msec, uptime=18488msec, status=online
***Ping # 7, currentInterval=3000msec, uptime=21491msec, status=online
***Ping # 8, currentInterval=3000msec, uptime=24494msec, status=online
***Ping # 9, currentInterval=3000msec, uptime=27497msec, status=online
***Ping # 10, currentInterval=3000msec, uptime=30500msec, status=online
***Ping # 11, currentInterval=3000msec, uptime=33503msec, status=online
***Ping # 12, currentInterval=6000msec, uptime=39506msec, status=online
***Ping # 13, currentInterval=6000msec, uptime=45509msec, status=online
***Ping # 14, currentInterval=6000msec, uptime=51512msec, status=online
***Ping # 15, currentInterval=6000msec, uptime=57515msec, status=online
***Ping # 16, currentInterval=6000msec, uptime=63518msec, status=online
***Ping # 17, currentInterval=6000msec, uptime=69521msec, status=online
***Ping # 18, currentInterval=9000msec, uptime=78524msec, status=online
***Ping # 19, currentInterval=9000msec, uptime=87527msec, status=online
***connDet:0 (-9)
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 1027.
(uptime=88554msec)
***Ping # 20, currentInterval=4500msec, uptime=92842msec, status=offline
***Ping # 21, currentInterval=4500msec, uptime=97345msec, status=offline
***Ping # 22, currentInterval=4500msec, uptime=101848msec, status=offline
***Ping # 23, currentInterval=4500msec, uptime=106351msec, status=offline
***Ping # 24, currentInterval=4500msec, uptime=110854msec, status=offline
***Ping # 25, currentInterval=4500msec, uptime=115357msec, status=offline
***Ping # 26, currentInterval=4500msec, uptime=119860msec, status=offline
***Ping # 27, currentInterval=4500msec, uptime=124363msec, status=offline
***Ping # 28, currentInterval=4500msec, uptime=128866msec, status=offline
***connDet:9 (+9)
MicroTAPAS.mobility.MicroPingClient::Connection    to    director    (re-)established.
(uptime=129883msec)
***Ping # 29, currentInterval=4500msec, uptime=134235msec, status=online
***Ping # 30, currentInterval=4500msec, uptime=138740msec, status=online
***Ping # 31, currentInterval=4500msec, uptime=143243msec, status=online
***Ping # 32, currentInterval=4500msec, uptime=147746msec, status=online
***Ping # 33, currentInterval=4500msec, uptime=152249msec, status=online
***Ping # 34, currentInterval=4500msec, uptime=156752msec, status=online
***Ping # 35, currentInterval=4500msec, uptime=161255msec, status=online
***Ping # 36, currentInterval=7500msec, uptime=168758msec, status=online
***Ping # 37, currentInterval=7500msec, uptime=176792msec, status=online
***Ping # 38, currentInterval=7500msec, uptime=184295msec, status=online
***connDet:2 (-7)
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 1007.
(uptime=185302msec)
***Ping # 39, currentInterval=3750msec, uptime=188930msec, status=offline
***Ping # 40, currentInterval=3750msec, uptime=192683msec, status=offline
***Ping # 41, currentInterval=3750msec, uptime=196436msec, status=offline
***Ping # 42, currentInterval=3750msec, uptime=200189msec, status=offline
***Ping # 43, currentInterval=3750msec, uptime=203942msec, status=offline
***Ping # 44, currentInterval=3750msec, uptime=207695msec, status=offline
***Ping # 45, currentInterval=3750msec, uptime=211448msec, status=offline
***Ping # 46, currentInterval=3750msec, uptime=215201msec, status=offline
***Ping # 47, currentInterval=3750msec, uptime=218954msec, status=offline
***Ping # 48, currentInterval=3750msec, uptime=222707msec, status=offline
***Ping # 49, currentInterval=3750msec, uptime=226460msec, status=offline
***Ping # 50, currentInterval=3750msec, uptime=230213msec, status=offline
***Ping # 51, currentInterval=3750msec, uptime=233966msec, status=offline
```

```
***Ping # 52, currentInterval=3750msec, uptime=237719msec, status=offline
***Ping # 53, currentInterval=3750msec, uptime=241472msec, status=offline
***Ping # 54, currentInterval=3750msec, uptime=245226msec, status=offline
***connDet:18 (+16)
MicroTAPAS.mobility.MicroPingClient::Connection    to    director    (re-)established.
(uptime=246270msec)
***Ping # 55, currentInterval=3750msec, uptime=249870msec, status=online
***Ping # 56, currentInterval=3750msec, uptime=253623msec, status=online
***Ping # 57, currentInterval=3750msec, uptime=257376msec, status=online
***Ping # 58, currentInterval=3750msec, uptime=261129msec, status=online
***Ping # 59, currentInterval=3750msec, uptime=264882msec, status=online
***Ping # 60, currentInterval=3750msec, uptime=268635msec, status=online
***Ping # 61, currentInterval=3750msec, uptime=272388msec, status=online
***Ping # 62, currentInterval=3750msec, uptime=276142msec, status=online
***Ping # 63, currentInterval=6750msec, uptime=282895msec, status=online
***Ping # 64, currentInterval=6750msec, uptime=289648msec, status=online
***connDet:12 (-6)
MicroTAPAS.mobility.MicroPingClient::No  connection  to  director.  Timeout  after  1289.
(uptime=290936msec)
***Ping # 65, currentInterval=3375msec, uptime=293892msec, status=offline
***Ping # 66, currentInterval=3375msec, uptime=297792msec, status=offline
```

Static scheme (interval 3 sec)

```
***Ping # 1, currentInterval=3000msec, uptime=3960msec, status=offline
MicroTAPAS.mobility.MicroPingClient::Connection    to    director    (re-)established.
(uptime=6976msec)
***Ping # 2, currentInterval=3000msec, uptime=8005msec, status=online
***Ping # 3, currentInterval=3000msec, uptime=11108msec, status=online
***Ping # 4, currentInterval=3000msec, uptime=14036msec, status=online
***Ping # 5, currentInterval=3000msec, uptime=17039msec, status=online
***Ping # 6, currentInterval=3000msec, uptime=20042msec, status=online
***Ping # 7, currentInterval=3000msec, uptime=23045msec, status=online
***Ping # 8, currentInterval=3000msec, uptime=26048msec, status=online
***Ping # 9, currentInterval=3000msec, uptime=29051msec, status=online
***Ping # 10, currentInterval=3000msec, uptime=32054msec, status=online
***Ping # 11, currentInterval=3000msec, uptime=35057msec, status=online
***Ping # 12, currentInterval=3000msec, uptime=38060msec, status=online
***Ping # 13, currentInterval=3000msec, uptime=41063msec, status=online
***Ping # 14, currentInterval=3000msec, uptime=44066msec, status=online
***Ping # 15, currentInterval=3000msec, uptime=47069msec, status=online
***Ping # 16, currentInterval=3000msec, uptime=50072msec, status=online
***Ping # 17, currentInterval=3000msec, uptime=53075msec, status=online
***Ping # 18, currentInterval=3000msec, uptime=56078msec, status=online
***Ping # 19, currentInterval=3000msec, uptime=59081msec, status=online
***Ping # 20, currentInterval=3000msec, uptime=62084msec, status=online
***Ping # 21, currentInterval=3000msec, uptime=65087msec, status=online
***Ping # 22, currentInterval=3000msec, uptime=68090msec, status=online
***Ping # 23, currentInterval=3000msec, uptime=71093msec, status=online
***Ping # 24, currentInterval=3000msec, uptime=74096msec, status=online
***Ping # 25, currentInterval=3000msec, uptime=77099msec, status=online
***Ping # 26, currentInterval=3000msec, uptime=80102msec, status=online
***Ping # 27, currentInterval=3000msec, uptime=83105msec, status=online
***connDet:0 (-25)
MicroTAPAS.mobility.MicroPingClient::No  connection  to  director.  Timeout  after  981.
(uptime=84086msec)
***Ping # 28, currentInterval=3000msec, uptime=86978msec, status=offline
***Ping # 29, currentInterval=3000msec, uptime=89981msec, status=offline
***Ping # 30, currentInterval=3000msec, uptime=92984msec, status=offline
***Ping # 31, currentInterval=3000msec, uptime=95987msec, status=offline
***Ping # 32, currentInterval=3000msec, uptime=98990msec, status=offline
***Ping # 33, currentInterval=3000msec, uptime=101993msec, status=offline
***Ping # 34, currentInterval=3000msec, uptime=104996msec, status=offline
***Ping # 35, currentInterval=3000msec, uptime=107999msec, status=offline
***Ping # 36, currentInterval=3000msec, uptime=111002msec, status=offline
***Ping # 37, currentInterval=3000msec, uptime=114005msec, status=offline
***Ping # 38, currentInterval=3000msec, uptime=117008msec, status=offline
***Ping # 39, currentInterval=3000msec, uptime=120011msec, status=offline
***Ping # 40, currentInterval=3000msec, uptime=123014msec, status=offline
***Ping # 41, currentInterval=3000msec, uptime=126017msec, status=offline
***Ping # 42, currentInterval=3000msec, uptime=129020msec, status=offline
***Ping # 43, currentInterval=3000msec, uptime=132023msec, status=offline
***connDet:16 (+16)
```

```
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=133069msec)
***Ping # 44, currentInterval=3000msec, uptime=135892msec, status=online
***Ping # 45, currentInterval=3000msec, uptime=138895msec, status=online
***Ping # 46, currentInterval=3000msec, uptime=141898msec, status=online
***Ping # 47, currentInterval=3000msec, uptime=144901msec, status=online
***Ping # 48, currentInterval=3000msec, uptime=147904msec, status=online
***Ping # 49, currentInterval=3000msec, uptime=150907msec, status=online
***Ping # 50, currentInterval=3000msec, uptime=153910msec, status=online
***Ping # 51, currentInterval=3000msec, uptime=156913msec, status=online
***Ping # 52, currentInterval=3000msec, uptime=159916msec, status=online
***Ping # 53, currentInterval=3000msec, uptime=162919msec, status=online
***Ping # 54, currentInterval=3000msec, uptime=165922msec, status=online
***Ping # 55, currentInterval=3000msec, uptime=168925msec, status=online
***Ping # 56, currentInterval=3000msec, uptime=171928msec, status=online
***Ping # 57, currentInterval=3000msec, uptime=174931msec, status=online
***Ping # 58, currentInterval=3000msec, uptime=177934msec, status=online
***Ping # 59, currentInterval=3000msec, uptime=180937msec, status=online
***connDet:0 (-16)
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 1059.
(uptime=181996msec)
***Ping # 60, currentInterval=3000msec, uptime=184799msec, status=offline
***Ping # 61, currentInterval=3000msec, uptime=187802msec, status=offline
***Ping # 62, currentInterval=3000msec, uptime=190805msec, status=offline
***Ping # 63, currentInterval=3000msec, uptime=193808msec, status=offline
***Ping # 64, currentInterval=3000msec, uptime=196811msec, status=offline
***Ping # 65, currentInterval=3000msec, uptime=199814msec, status=offline
***Ping # 66, currentInterval=3000msec, uptime=202817msec, status=offline
***Ping # 67, currentInterval=3000msec, uptime=205820msec, status=offline
***Ping # 68, currentInterval=3000msec, uptime=208823msec, status=offline
***Ping # 69, currentInterval=3000msec, uptime=211826msec, status=offline
***Ping # 70, currentInterval=3000msec, uptime=214829msec, status=offline
***Ping # 71, currentInterval=3000msec, uptime=217832msec, status=offline
***Ping # 72, currentInterval=3000msec, uptime=220835msec, status=offline
***Ping # 73, currentInterval=3000msec, uptime=223838msec, status=offline
***Ping # 74, currentInterval=3000msec, uptime=226841msec, status=offline
***Ping # 75, currentInterval=3000msec, uptime=229844msec, status=offline
***Ping # 76, currentInterval=3000msec, uptime=232847msec, status=offline
***Ping # 77, currentInterval=3000msec, uptime=235850msec, status=offline
***Ping # 78, currentInterval=3000msec, uptime=238853msec, status=offline
***Ping # 79, currentInterval=3000msec, uptime=241856msec, status=offline
***Ping # 80, currentInterval=3000msec, uptime=244859msec, status=offline
***Ping # 81, currentInterval=3000msec, uptime=247862msec, status=offline
***Ping # 82, currentInterval=3000msec, uptime=250865msec, status=offline
***connDet:23 (+23)
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=251883msec)
***Ping # 83, currentInterval=3000msec, uptime=254685msec, status=online
***Ping # 84, currentInterval=3000msec, uptime=257688msec, status=online
***Ping # 85, currentInterval=3000msec, uptime=260690msec, status=online
***Ping # 86, currentInterval=3000msec, uptime=263693msec, status=online
***Ping # 87, currentInterval=3000msec, uptime=266696msec, status=online
***Ping # 88, currentInterval=3000msec, uptime=269700msec, status=online
***Ping # 89, currentInterval=3000msec, uptime=272703msec, status=online
***Ping # 90, currentInterval=3000msec, uptime=275705msec, status=online
***Ping # 91, currentInterval=3000msec, uptime=278709msec, status=online
***Ping # 92, currentInterval=3000msec, uptime=281711msec, status=online
***Ping # 93, currentInterval=3000msec, uptime=284715msec, status=online
***Ping # 94, currentInterval=3000msec, uptime=287718msec, status=online
***Ping # 95, currentInterval=3000msec, uptime=290720msec, status=online
***connDet:10 (-13)
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 1034.
(uptime=291753msec)
***Ping # 96, currentInterval=3000msec, uptime=294560msec, status=offline
***Ping # 97, currentInterval=3000msec, uptime=297563msec, status=offline
```

Static scheme (interval 15 sec)

```
***Ping # 1, currentInterval=15000msec, uptime=3785msec, status=offline
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=6570msec)
***Ping # 2, currentInterval=15000msec, uptime=19546msec, status=online
***Ping # 3, currentInterval=15000msec, uptime=34551msec, status=online
***Ping # 4, currentInterval=15000msec, uptime=49554msec, status=online
```

```
***Ping # 5, currentInterval=15000msec, uptime=64557msec, status=online
***Ping # 6, currentInterval=15000msec, uptime=79560msec, status=online
***Ping # 7, currentInterval=15000msec, uptime=94563msec, status=online
***connDet:0 (-5)
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 957.
(uptime=95520msec)
***Ping # 8, currentInterval=15000msec, uptime=110502msec, status=offline
***Ping # 9, currentInterval=15000msec, uptime=125514msec, status=offline
***Ping # 10, currentInterval=15000msec, uptime=140517msec, status=offline
***connDet:3 (+3)
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=141530msec)
***Ping # 11, currentInterval=15000msec, uptime=156338msec, status=online
***Ping # 12, currentInterval=15000msec, uptime=171341msec, status=online
***Ping # 13, currentInterval=15000msec, uptime=186344msec, status=online
***connDet:0 (-3)
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 966.
(uptime=187310msec)
***Ping # 14, currentInterval=15000msec, uptime=202163msec, status=offline
***Ping # 15, currentInterval=15000msec, uptime=217167msec, status=offline
***Ping # 16, currentInterval=15000msec, uptime=232171msec, status=offline
***Ping # 17, currentInterval=15000msec, uptime=247175msec, status=offline
***Ping # 18, currentInterval=15000msec, uptime=262178msec, status=offline
***connDet:5 (+5)
MicroTAPAS.mobility.MicroPingClient::Connection to director (re-)established.
(uptime=263199msec)
***Ping # 19, currentInterval=15000msec, uptime=278003msec, status=online
***Ping # 20, currentInterval=15000msec, uptime=293011msec, status=online
***connDet:3 (-2)
MicroTAPAS.mobility.MicroPingClient::No connection to director. Timeout after 992.
(uptime=294002msec)
```