TAPAS for wireless PDA

by
Eirik Lühr

"Within the TAPAS framework wireless devices are to be integrated and their mobility to be supported. The task of this project compromises Re-specification and Implementation of Plug-and-Play support functionality using J2ME for wireless PDA. The PaP functionality, so far, is supported mainly for PCs using J2SE. The task could also include a small application for testing these devices."

Department of Telematics, NTNU, Trondheim
Spring 2003

Abstract

Telematics Architecture for Plug-and-play Systems, or TAPAS, is a software architecture that facilitates for dynamic introduction of new distributed services, or upgrading of existing ones, in a communication network. The Department of Telematics, in cooperation with SINTEF, have implemented a prototype of this architecture using the Java programming language. This reports main area of focus lies in re-specifying and implementing a 'lightweight' version of TAPAS, MicroTAPAS, for use on small handheld wireless devices – Personal Digital Assistants (PDA's). In addition, suitable target platforms and operating systems are discussed, along with the introduction of a small application that will illustrate the basic features of this new architecture.

Chapter one will introduce the reader to the basic principles behind TAPAS and present some of the research previously done on the topic. Similar technologies are also briefly presented, before an introduction to TAPAS for wireless components. Chapter two moves on to discuss the concepts of the new architecture, its extended support functionality and changes made to the layered design model.

In chapter three, a brief introduction to Java 2 Micro Edition (J2ME) is given before explaining the implementation of various parts of the system. Chapter four discusses possible development environments, and various elements of hardware and test set-up.

The simple test application, MicroTester, is presented in chapter five and is then used in chapter six for testing the new architecture. Chapter six also includes some discussions dealing with the performance of MicroTAPAS on different processor architectures. The report is rounded off with an overview of faced challenges and solutions, suggested improvements and a conclusion, in chapters seven through nine.

In the relative short period that this project has lasted, there has been a significant evolution taking place in the industry that surrounds small handheld wireless devices. Important new technologies have become available that enabled the development of features which were impossible, or nearly so, to implement when the project first started. While this has made the area extremely interesting and full of surprises, it has also made the job that much harder, and at times it has been frustrating to not know when, if at all, a promised feature or platform would become available. Nevertheless, the whole experience has been truly enjoyable and a very instructive.

Eirik Lühr, July 2003

Contents

Figures and tables

1 Introduction

A brief introduction to TAPAS, along with a short description of previous work is presented in this section, before looking at a few similar technologies. Finally, the reader is introduced to the idea of TAPAS for wireless components in the last section of this introductory chapter.

1.1 Introduction to TAPAS

Telematics Architecture for Plug-and-play Systems (TAPAS) is a project running at the Department of Telematics since 1997 and supported by the Norwegian Research Council. TAPAS enables "…the hardware and software "parts", as well as complete network elements that constitute a communication system to have the ability to configure themselves when installed into a network and then to provide services according to their own capabilities, the service repertoire and the operating policies of the system." [MELH1]. Several papers have been presented in different international conferences, as well as Master thesis, dealing with the different parts of the architecture and introducing solutions and proposals on its various features.

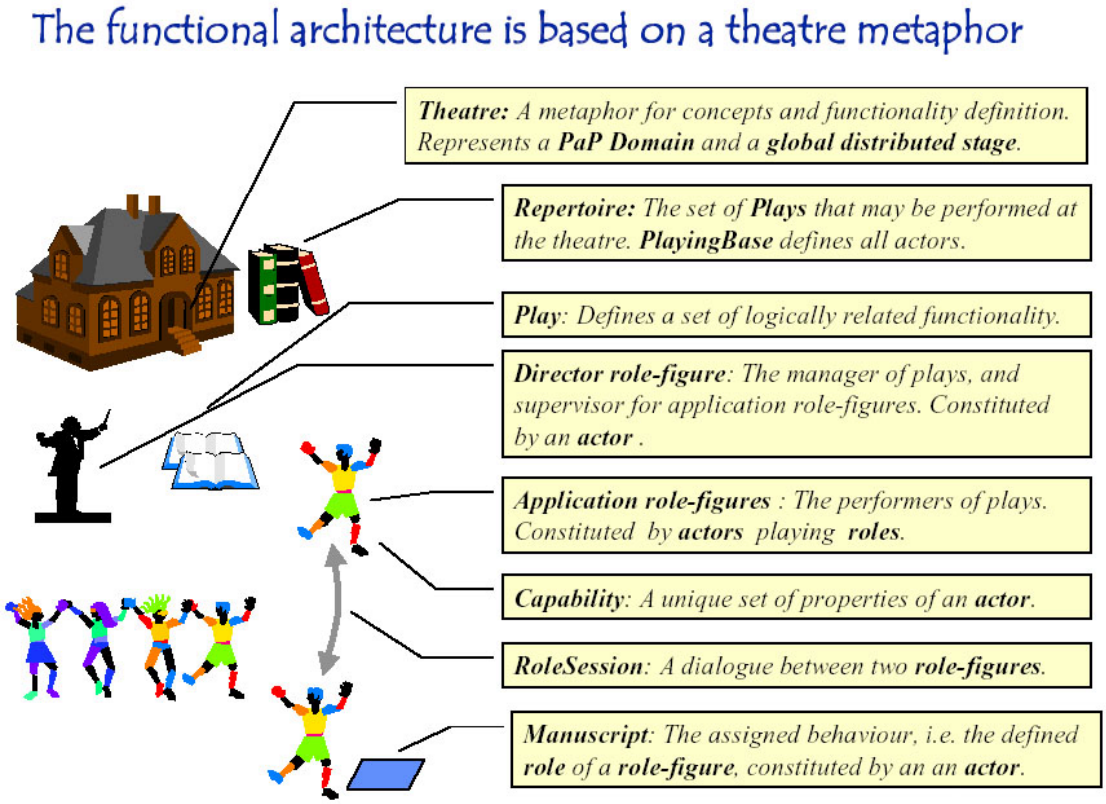


Figure 1.1: The TAPAS theatre model

TAPAS use a theatre metaphor to model the entities that constitute the architecture, and an overview is presented in Figure 1.1 [AAGF3, page 7]. Depending on its Capability, an Actor is the generic object that can be instantiated and behave according to a certain Role in a Play (a service may be implemented in one or more plays), which is described in a Manuscript. The Play has a defined autonomous, functionality. The Repertoire is the total set of plays available. A Role session is the projection of behaviour of an actor, in relation to one of its interacting actors. Finally, there is the Director, who is the repertoire and actor manager, and a guide to actors in plug-in/out phases [MELH2, page 8].

Figure 1.2 [LILL, page 3] shows the data model of TAPAS and illustrates how the different concepts in the architecture relate to each other.

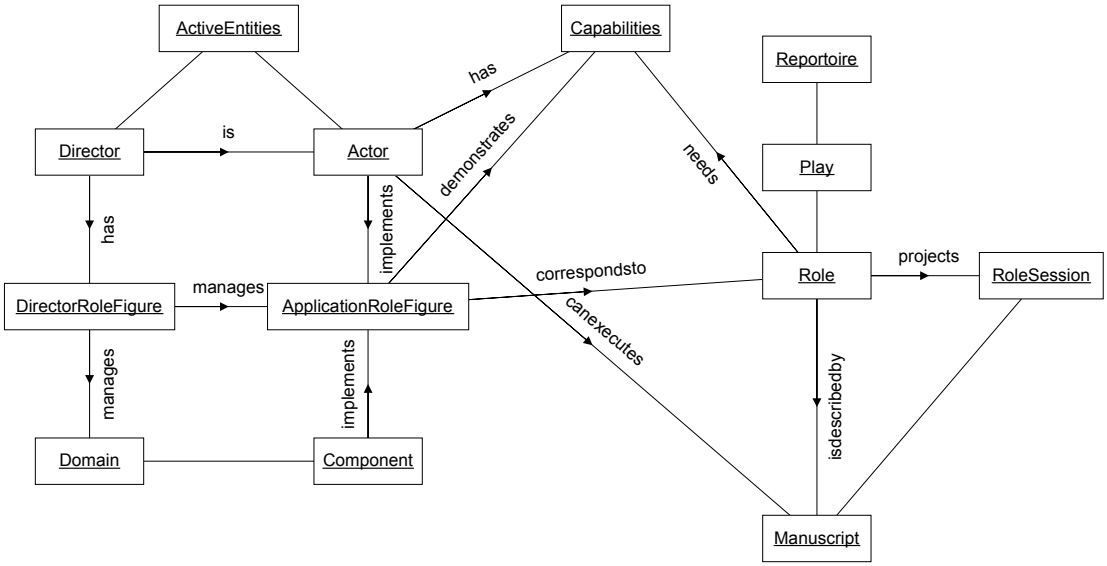


Figure 1.2: The TAPAS data model

TAPAS has been implemented in Java 2 Standard Edition (J2SE), with extensive use of Java Remote Method Invocation (RMI) for communication and the Java Abstract Windowing Toolkit (AWT) for debugging and application windows. A working version of TAPAS, as well as documentation and relevant publications, is available for download from the Internet [MELH1].

The TAPAS basic instance structure [AAGF3, page 9] is illustrated by Figure 1.3.

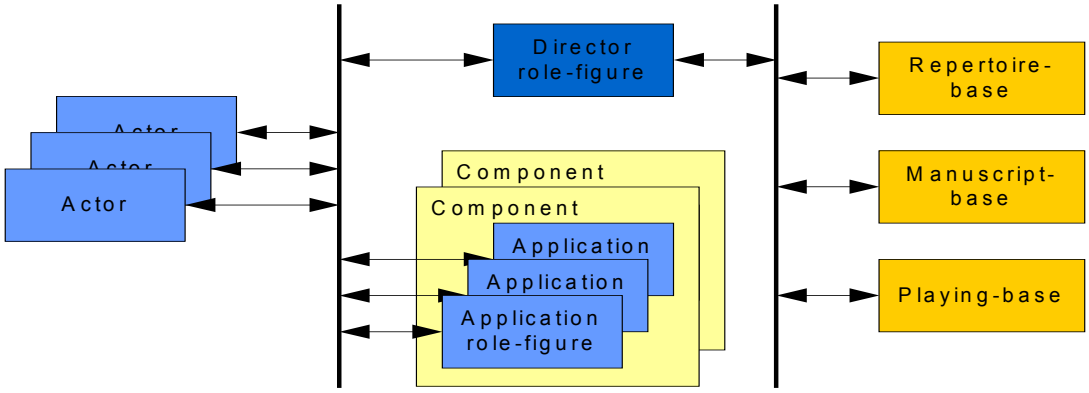


Figure 1.3: TAPAS basic instance structure

As we will see in later chapters, TAPAS, as currently implemented, is not suited for deployment to all the target platforms envisioned in the architecture, among them PDA's and other small handheld devices. Throughout the rest of this report, extensive knowledge of both TAPAS and J2SE is assumed, as it is outside the scope of this report to introduce the reader to these fundamentals. It is appropriate, however, to give a quick overview of the work previously done on TAPAS, the papers published on the subject as well as a comparison to similar popular technologies.

1.2 Previous work on TAPAS

In recent years, a number of different papers have been published about TAPAS, often as an input to various international conferences, dealing with different aspects of the TAPAS architecture, or closely related areas of research. In addition to these papers, university students have completed and presented a few Master of Science (MSc) diploma theses as well. This report draws from some of this work previously done on the subject, and a quick overview of a selected few documents are presented below.

[MALM1] presents mobility management within TAPAS, and the paper starts with a brief introduction to essential TAPAS concepts and the TAPAS layered model, before presenting different approaches for actor, terminal, user and session mobility management. For each type of mobility management, they have presented "an early set of mobility management algorithms or methods", as well as exploring "a few issues related to implementation design and propose as set of components to facilitate the deployment of this platform in the available PaP applications." The report concludes, in essence, that several issues discussed in the paper need further investigation. The authors argue that "we need to break up the overall PaP architecture into subsystems to provide selected types of mobility, which for some will be in the support and for others in the application layers" and further "we need to find efficient ways to map our methods and procedures into proper applications and communication platforms".

Another interesting paper about the subject is "Capability Specification and Selection in TAPAS" [AAGF1]. The paper's abstract gives a clear understanding about what is discussed in the paper; "A theoretical framework for support specification and selection in Plug-and-Play (PaP) architecture is proposed with a representation, computation and reasoning mechanism for semantic description and matching of support required by a particular PaP service system and support offered by a running PaP system.". In essence, this means that description and matching of required and offered capabilities between different PaP support systems is introduced. The next section of the paper introduces the PaP system and provides definitions of the capability and status concepts used in subsequent sections. Then follows different sections which each introduces and/or discusses topics such as significant functions in support management, framework for support specification and selection, before illustrating the theory with a demonstration of the concepts using an application example, and finally concluding the paper and presenting further research topics.

"User and Session mobility in a Plug-and-Play architecture" [LILL], a master thesis, revolves around user and session mobility in TAPAS. The report first presents general object and engineering models for mobility support in TAPAS, before focusing on user and session mobility. In the chapter titled "The mobility architecture", the candidate states the functional and non-functional requirements for the framework and presents a UML use-case diagram of the user and session mobility, as well as class diagrams. In order to store user and session information between sessions, a system utilising XML-files was devised. These XML-files store information such as roles, session descriptions and user profiles. The reminder of the thesis presents and discusses two sample applications, chat and file transfer, providing UML-diagrams and message sequence charts for both.

These reports and papers try to expand the overall objective of the architecture, and develop elaborated support functionality. This report is a continuation of this trend, as it extends the TAPAS support functionality to wireless handheld devices.

1.3 Similar technologies and comparisons

The purpose of this section is to compare TAPAS with a few existing technologies, be it programming languages or complete architectures, to provide the reader with an overall look at the standing of TAPAS in comparisons to these other technologies.

1.3.1 Active and controllable networks

Active Networks are classified by two approaches: active packets and active nodes. The first builds on the integration and deployment of services in the user flow, while the second is based on deploying services dynamically in nodes. Below is a short presentation of the different approaches to the field of active networks.

The U.S. Department of Defence's (DoD) Defence Advanced Research Projects Agency (DARPA) has set in motion an active networks program that aims at producing a new networking platform. The architecture "is based on a highly dynamic runtime environment that supports a finely tuned degree of control over network services. The packet itself is the basis for describing, provisioning, or tailoring resources to achieve the delivery and management requirements." [DARP]

The Distributed Computing and Communications Lab at Columbia University have developed a programming language and environment, called NetScript, for building networked systems, and is thoroughly described in [COLU]. The programs that are designed and implemented in this language are organized as mobile agents that after deployment to remote systems can be executed either under local or remote control. The purpose of the project is to simplify the development of networked systems, and their remote programming, as "networked systems are difficult to design, implement, deploy and manage." [COLU]

An Active Networks project at Massachusetts Institute of Technology (MIT), which has been funded by DARPA, has developed the Java-based Active Node Transfer System (ANTS), for experimenting with active networks [MITE].

There are a number of other projects on active networks as well, among them the SwithWare project undertaken by the University of Pennsylvania and Bellcore, which is described in [UPEN].

These are just a few of the research projects going on in the field of active networks. There are some obvious similarities between TAPAS and the technologies presented above, in particular the 'active nodes' approach. However, TAPAS is based on code-on-demand and not pre-programmed packets, as is the case for active networks. The nodes in TAPAS need only to run a fixed-sized executable and have a set of basic settings, such as initial web address and configuration files, while active networks need to include how packets be interpreted in the packets themselves.

1.3.2 Mobile IP & Cellular IP

Mobile and Cellular IP has been the target of significant international research efforts over the last decade, and continue to be a major field of study.

When a computer is connected to a specific network, it is allocated an IP address, when that computer moves to another network it is given a new IP address, i.e. by Dynamic Host Configuration Protocol (DHCP). This scheme works fine in most cases, but a problem arises if files or resources on that computer are sought by others, since they would not know that computers address. This is where mobile IP comes into the

picture, and with this transparent scheme, computing continues as normal when a host is moved from one subnet to another. When the computer is connected to its home base, packets are routed in the usual way. When it is connected elsewhere, two agent processes take over the routing, the home agent (HA) and the foreign agent (FA) running at fixed nodes on the two subnets. When the mobile host leaves its home domain, the HA is informed of this, and the FA of the visited domain relays back to the HA that the host is available in that domain. The HA then operates as a proxy, relaying all traffic to the mobile host through the visited domains FA.

According to [COUG, page 104], "The MobileIP solution is effective but hardly efficient. A solution that treats mobile hosts as first-class citizens would be preferable, allowing them to wander without giving prior notice and routing packets to them without any tunnelling or rerouting." This is clearly a drawback to the technology, but could be amended in the future to work along the lines of how cellular phones roam networks.

There is no support for terminal mobility in TAPAS, but, as explained in section 1.2, [MALM1] discuss a Mobility Management platform for TAPAS. In addition [LILL] has proposed a scheme for user and session mobility, using the principles of mobile IP, including home and foreign agents. In the future, terminal mobility could however be added to the implementation of the TAPAS architecture, as the target terminals are highly mobile and could benefit from being truly mobile devices.

1.3.3 Mobile Agents, Multi agent and Agent based systems

A mobile agent is a program, script or package that physically travels around a network, and performs operations on hosts that have agent capabilities. These agents, which operate autonomously, usually has very specific tasks, such as fetching prices of merchandise from on-line stores, or to collect weather information. Apart from interacting with all sorts of operating systems, databases or information systems, agents can also interact with other agents, meeting in agent-gathering places to exchange information. There are a number of different mobile agent architectures and languages available today, such as Knowledge Query and Manipulation Language (KQML) as presented in [UMBC], which is part of the broader ARPA Knowledge Sharing Effort [STAU], and is a "language and protocol for exchanging information and knowledge". Although agent technologies have received a lot of attention in recent years, [REID] argues that "mobile agency has failed to become a sweeping force of change, and now faces competition in the form of message passing and remote procedure call (RPC) technologies".

The very autonomous nature of mobile agents sets them wide apart from TAPAS' request/response interactions, although the resulting action might be comparatively equal, and a TAPAS node can almost be seen upon as a stationary agent.

1.3.4 Summary

After looking at the technologies discussed above, one could say that TAPAS can be viewed as a mix between the 'active networks' and 'mobile agents' technologies; although it should be clear that not any one existing technology completely overlaps with that of TAPAS.

1.4 TAPAS for wireless components

This project is motivated by the need to integrate wireless devices and the project definition states a clear aim for this, "Within the TAPAS framework, wireless devices are to be integrated and their mobility to be supported."

The TAPAS support platform has so far not taken into consideration the limitations of the Java implementation for small wireless devices. This section will justify the need for the re-specification, and implementation, of the TAPAS architecture for porting to these devices. The modified architecture with this new support will henceforth be referred to as MicroTAPAS.

The strongest argument for a complete re-specification of the TAPAS architecture is the limitations of the target terminals. These terminals are small handheld devices with limited computational power and memory, and simply do not have support for the Java 2 Standard Edition (J2SE) virtual machine, needed to run traditional Java applications. The other argument is that these highly mobile devices should use a support system, MicroTAPAS, that takes into account their mobility by extending the support functions of the original architecture.

TAPAS, as mentioned above, is implemented using J2SE, and to ease the re-specification and development of TAPAS into MicroTAPAS, for porting to small wireless handheld devices, such as Java enabled PDAs and mobile phones, Java 2 Micro Edition (J2ME) was considered the only real alternative. The modular extension and compatibility with the original platform will enable the developers to re-use proven and well-documented components from what has already been implemented in the TAPAS architecture. Using Java also simplifies the task of deploying the architecture to different operating systems, taking advantage of Java's "write once, run everywhere" characteristics.

From the inherent limitations of J2ME, as will be discussed in chapter 1, it became apparent that a number of the features of Java, as used in TAPAS, are not available or are too resource demanding in J2ME. Therefore, the support functionality of TAPAS had to be re-specified and implemented in accordance with the Application Programming Interface (API) of J2ME.

MicroTAPAS aims at taking into consideration the limitations posed by small handheld devices. Naturally, the basic design ideas and model behind TAPAS will be used for the re-specification of MicroTAPAS, or in other words, no significant changes will be applied to the basic concept of plug and play functionality

2 MicroTAPAS architecture concepts

This chapter will present the basic architecture concepts behind MicroTAPAS. Most of the features discussed below have actually been implemented in MicroTAPAS and will be discussed in further in chapter 3, MicroTAPAS implementation.

2.1 Requirements and considerations

In summary, the requirements for MicroTAPAS are almost identical to those stated for TAPAS, as described in [MELH2, page 9], with some added support functionality specifically tailored for dealing with its wireless and mobile nature.

The initial idea was to have the traditional TAPAS system as it is, i.e. running on a fixed network, and having MicroTAPAS nodes communicating with this system and using its director, without any changes to the current TAPAS system, thus allowing interoperation between the two. When we had to take into consideration the limitations of the development language and the target devices, however, it became evident that a few architectural changes of the whole system had to take place in order for our system to be feasible. This resulted in a stand-alone architecture, complete with a test application, explicitly adapted to the possibilities and limitations presented by wireless handheld devices. As such. The issue of interoperability with the basic TAPAS platform need to be targeted and addressed in future work.

These small, lightweight devices are easy to carry around an office building or campus, and do not rely on cables to communicate with offered services, be it office applications, such as elaborate collaboration software or simple chat applications, or directly to the Internet. The dynamic nature of the TAPAS architecture seems a perfect match for such devices, in that it offer its users maximum freedom, without the hassle of manually downloading, configuring and installing applications and services.

The biggest limitation, in terms of implementation, that had to be considered was the relative lightweight nature of J2ME. Some of the features of J2SE, i.e. Java Foundation Classes (JFC)/Swing[*], are absent from J2ME, and implemented support functionality and mechanisms of TAPAS had to be re-invented with the limitations of J2ME in mind.

Another consideration we had to take into account was that the target devices would have somewhat limited capabilities compared to that of a traditional desktop or laptop, both in terms of computational power, the availability of ‘hard’ and ‘soft’ memory and display capabilities.

2.2 MicroTAPAS Support Functions

The MicroTAPAS support functions are almost identical to those offered by TAPAS, and those inherited from TAPAS have by [MELH2, page 9] been summarised and grouped as follows:

- Managing the availability of application functionality (i.e. the plays)
  - PlayPlugIn(PlayId, PlayVer, PlayLoc)
  - PlayChangesPlugIn(PlayId, PlayVer, PlayLoc)

[*] JFC/Swing is a group of Java features to develop advanced Graphical User Interfaces (GUI) first announced in 1997, and is an extension of the Java Abstract Windowing Toolkit (AWT).

- PlayPlugOut(PlayId, PlayVer)

- Managing the existence of active entities in operational systems (i.e. the actors)
  - ActorPlugIn(Location, RoleName)
  - ActorPlugOut(RoleSessionId)

- Dynamic redefinition of actor behaviour (i.e. roles)
  - ActorBehaviourPlugIn(RoleName)
  - ActorChangeBehaviour(NewRoleName)
  - ActorBehaviourPlugOut()

- Interactions between actors, actors capability change, and monitoring TAPAS activities
  - RoleSessionAction(RoleSessionId, MsgType, MsgParameters)
  - ChangeActorCapabilities(ChangeType, CapabilitySet)
  - SubscribeRequest(EventType, Scope, ApplicationTypes, WhenReport)

In addition to these basic support functions, there is the added support for dynamic connections. A dynamic connection in this context is defined as a connection that may, or may not, be available at a given moment, and MicroTAPAS will have added support for dealing with this. This new functionality will be discussed further in section 3.4.

- Handling of dynamic connections
  - ActorRegister(RoleSessionId, ActorPlugInReq)
  - ActorRegisterCancel(RoleSessionId)

2.3 MicroTAPAS layered design model

The MicroTAPAS layered design model slightly modified version of the same model in TAPAS [MELH2, page 18]. The original model is included in Appendix B.1.
When comparing the layered design model of TAPAS to that of the proposed MicroTAPAS architecture, there are two notable differences.

Firstly, in the MicroTAPAS architecture, there is a limitation of only one virtual machine running on each node at a given time, as opposed to the TAPAS architecture, where several instances can run simultaneously. This limitation stems from the lack of system resources in the MicroTAPAS targeted operating environment (i.e. on a PDA), most notably main memory and processing capacity.

Secondly, it was decided that there should only be one Plug-And-Play Actor Support (PAS) instance at each node. Following this decision it became clear that it would be advantageous to combine the functionality of the PAS layer and that of the Plug-And-Play Node Execution Support (PNES) layer into one layer. This new layer is called MicroPNES. The essential working of the new MicroPNES layer is identical to those of the combined 'old' PNES and PAS layers. By combining these two layers, we removed the need for communication and management between them and the resulting implementation thus becomes more time and memory efficient and less demanding for the intended devices to handle.

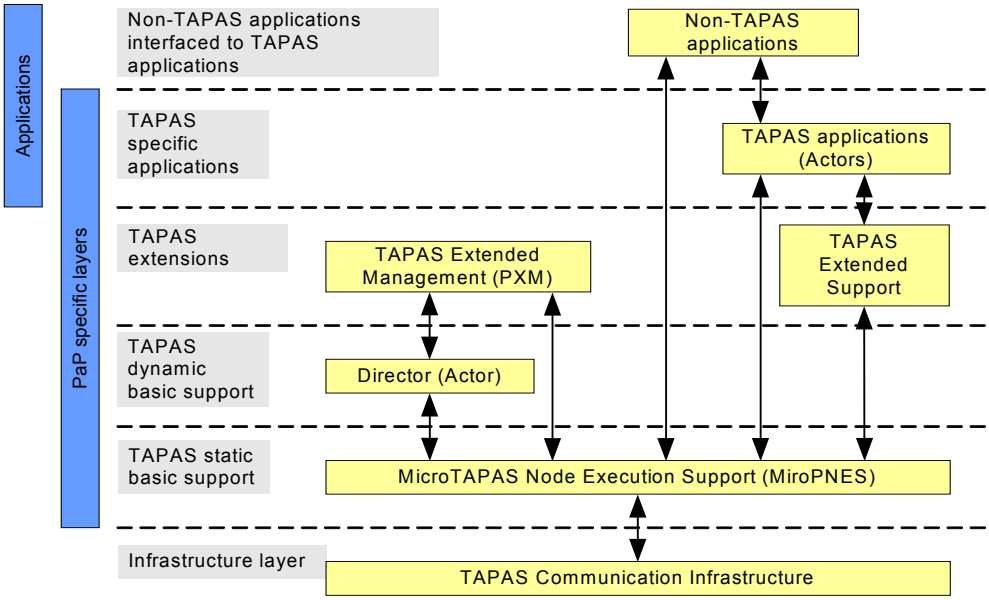


Figure 2.1: MicroTAPAS layered design model – architecture

Figure 2.1 shows the modified MicroTAPAS layered design model architecture. The PaP Actor Support (PAS) layer has been removed to make the architecture leaner for deployment on the handheld devices. Previously, it was common to instantiate several instances of a PAS within each node, where each instance could govern its own actors. In MicroTAPS however, it was, due to device constraints, decided to remove the possibility to instantiate more than one PAS on each node, thus removing the need for a separate layer to handle that functionality.

Figure 2.2 shows the nodes in a typical MicroTAPAS system, and the role of each of these nodes. A client node is a node that uses the services/resources of a server. The server is where the Director runs, and in this figure is running on a stationary computer (i.e. laptop or desktop), as it runs on a Java Virtual Machine (JVM), as opposed to the other two nodes who run the smaller J2ME VM, CVM, which would, for example, be PDA's.

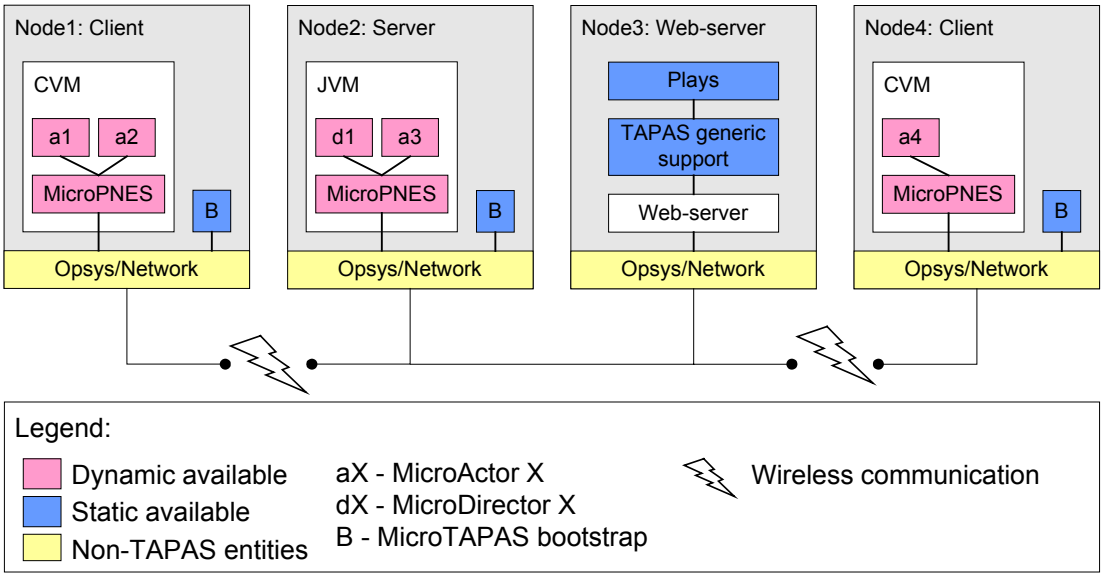


Figure 2.2: MicroTAPAS layered design model - operating MicroTAPAS system example

The communication link between the Clients and the rest of the system is here seen to be wireless, and that would almost always be the case for handheld devices. One could, however, connect the two other nodes wirelessly to the system as well. For clarity, Appendix B.1 includes a similar system example for the original TAPAS architecture.

3 MicroTAPAS implementation

This chapter deals with the actual implementation of MicroTAPAS, and starts with a brief discussion on the language in which to carry out the implementation, Java 2 Micro Edition (J2ME), and the choices made in relation to this. Subsequent sections deals with those modules of the original TAPAS architecture that have had extensive changes to them. These sections, thus, also serve to highlight the principal implementation-differences between the two architectures. The last section points out those changes that are not covered by any of the other sections.

All of the features discussed in sections 3.2 through 3.7 have to a lesser or greater degree, been implemented, and within each section is a short paragraph stating to which extent the feature has been implemented, and how extensive it has been tested. All proposed features that have had relatively little work done on them are summarized in chapter 1.

3.1 Overview of J2ME

This section will introduce the basic concepts behind the Java 2 Micro Edition (J2ME) platform API, and what limitations these pose for the MicroTAPAS architecture.

"J2ME is a family of specifications that defines various downsized versions of the standard Java 2 platform; these downsized versions can be used to program consumer electronic devices ranging from cell phones to highly capable Personal Data Assistants (PDAs), smart phones and set-top boxes." [TOPK, page3]

One of the things these devices have in common is that they lack the memory or computational resources, as found in modern desktop and laptop computers, to support the traditional requirements of either Java 2 Standard Edition (J2SE) or, naturally, Java 2 Enterprise Edition (J2EE).

J2ME consists of configurations, virtual machines (VM) and profiles. There are two types of configurations, each with their own VM; the Connected Limited Device Configuration (CLDC) using KVM, and the Connected Device Configuration (CDC) with its CVM. When to use what configuration depends on the available resources of the target device. According to [SUN1], CLDC is targeted at the low-end spectre of consumer electronics, and the devices will typically have around 512 KB memory, and run on 16 or 32-bit processors. Mobile phones and low-end PDAs would fall into this category. CDC, on the other hand, is targeted at systems that lie between those served by CLDC and J2SE. These devices will typically boast more than 2 MB of memory and run on 32-bit processors. Powerful PDAs and palmtop computers would be typical platforms for this configuration. Virtually all Java enabled mobile phones on the market today support the CLDC, while only a few hybrid phone-PDA models have support for CDC, among them are the QTEK 1010 PocketPC.

Figure 3.1 [SUN2] shows some of the configurations and profiles that are available for J2ME, along with the distinct separation between the two available configurations. The figure also illustrates where in the Java-series two configurations belong.

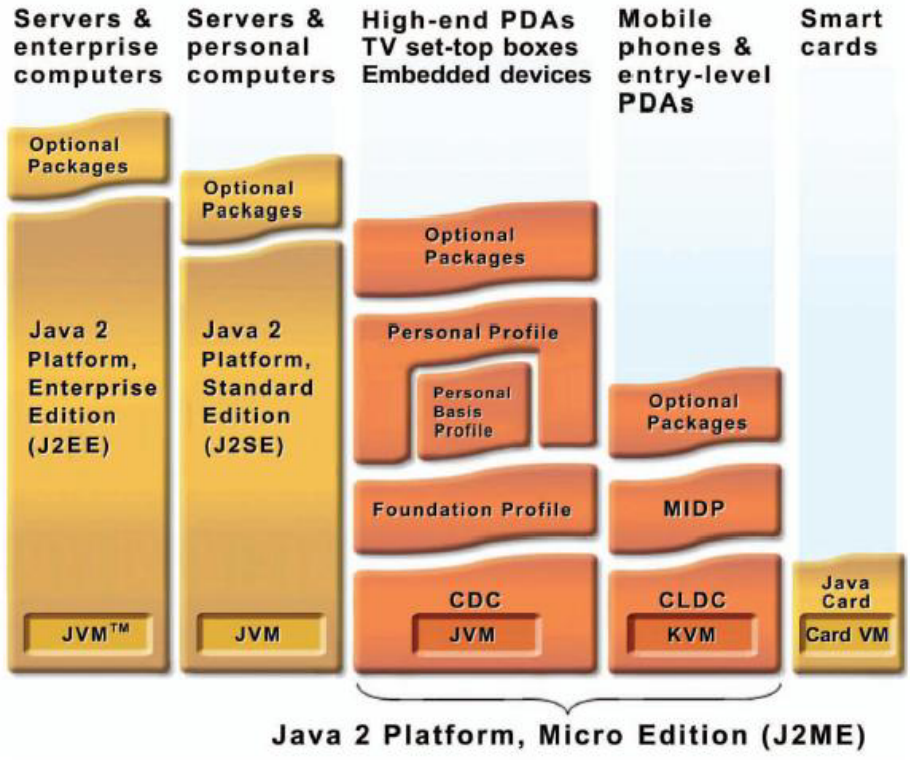


Figure 3.1: Overview of J2ME and other Java technologies

A brief overview of the different profiles in J2ME CDC has been included in Appendix A.1.

3.1.1 Choosing a configuration

Before any work with the MicroTAPAS implementation could be done, the capabilities of the two configurations (CLDC and CDC) had to be compared before choosing one of them as the platform for development. Finally, the chosen configuration and its profiles had to be investigated to establish what changes would be necessary to the original architecture for it to conform to the chosen configuration.

After scrutinising the original TAPAS source code, a few core elements were found that were of instrumental importance to the new system. In other words, those elements were considered so basic that a re-specification without them would become at least very demanding and difficult, if not impossible. The most notable such element was the J2SE package '`java.net`', and its `URLClassLoader` class. This package is used by TAPAS to dynamically download required files/classes from a web-server. Without this package, the TAPAS system would loose much of its intended flexibility and, thus, usefulness. This was to become *the* limiting factor in the decision of which J2ME configuration to use, as the J2ME CLDC specification does not include the '`java.net`' package.

In addition to the discovery above, the general limitations of CLDC compared to those of CDC was found to be too limiting for any development, and possible further expansions. Thus CDC was chosen as the configuration on which foundation the re-specification and development of MicroTAPAS would take place.

3.1.2 Limitations

Although CDC looked as to provide, basically, the same set of functionality as J2SE, it became apparent that at least one crucial component was not included. According to

[TOPK, page 240] “Since CDC devices are typically used in the role of the RMI client, only the client RMI functionality is included in this profile”. Especially the inability to run a RMI registry at each node proved to be a challenge. This obstacle clearly had to be dealt with, as all TAPAS nodes, whether ‘clients’ or ‘servers’, starts its own RMI registry, thus operating as RMI server.

Another challenge faced was the fact that the J2ME architecture is so new that all the relevant profiles have yet to be fully specified or implemented. A brief overview of the various profiles has been included in Appendix A.1.

It should at this point be stressed, once again, that the J2ME technology is relatively young, and that amendments and further enhancements to the technology occur at a monthly, or even weekly, basis. Some of the specific techniques discussed above are thus almost certain to be changed in the future. Further discussions and subsequent research papers should consider this, and investigate what changes has occurred since this report was published. The remaining part of this report will focus on the use of J2ME and its CDC configuration as the platform for re-specifying TAPAS and implementing the MicroTAPAS architecture.

3.2 Communication model

The biggest difference between TAPAS and MicroTAPAS, in terms of how the communication is carried out, is the absence of RMI. Previously we have established that because of the inability to create a RMI registry in J2ME, the communication model for MicroTAPAS had to be re-invented, as opposed to re-using components from TAPAS. It was decided to carry out all communication by using a combination of Java sockets and local method calls. It is also worth mentioning that having handhelds start their own RMI server consumes already limited resources, and even if J2ME offered this option, as it seems likely it will in the future, it is an unnecessary dispersion of limited resources.

In TAPAS, communication between nodes, between local PAS and PNES instances and, in some cases, between Actor and PAS instances, RMI was used. In MicroTAPAS however, the communication between MicroPNES and Actors are carried out using local method calls. For communication between different PNES instances, sockets[*] are used. See Appendix B for an overview of the TAPAS communication model. Figure 3.2 below shows the synchronous communication model of MicroTAPAS. The MicroComServer is the entity that is responsible for handling all inter-node communication in MicroTAPAS, and its operation is entirely transparent to the user.

[*] A socket is one endpoint of a two-way communication link between two programs running on a network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

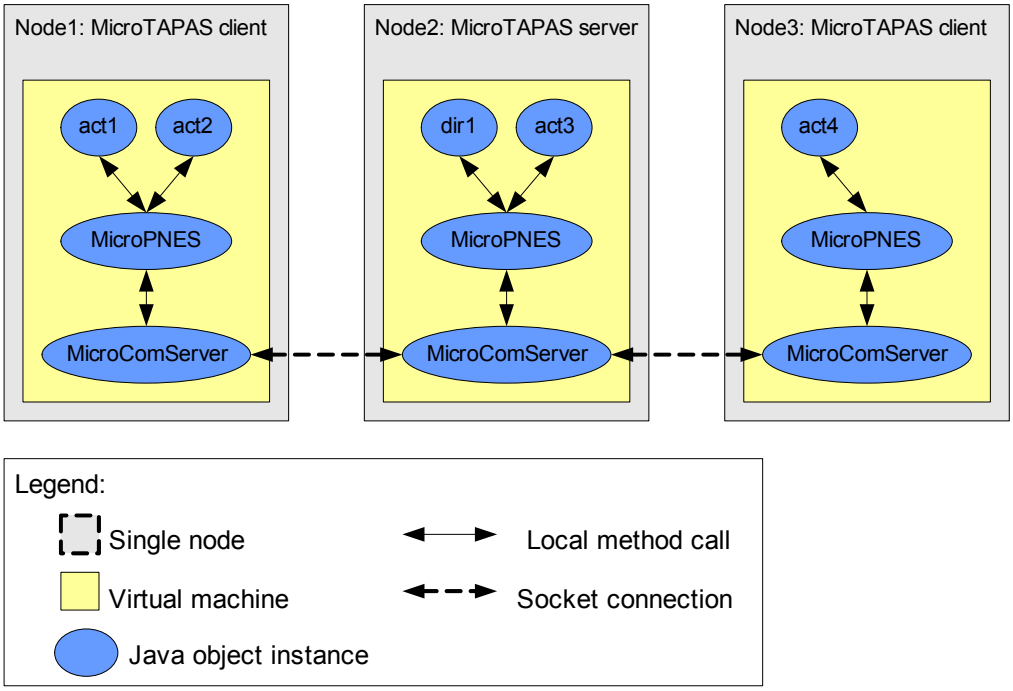


Figure 3.2: MicroTAPAS synchronous communication model

When different nodes are communicating in TAPAS, 'threading' of messages are quite common (i.e. for `actorPlugIn` and `actorPlugOut`). Threading in this case means that the originator of the initial request might receive one or several other requests, whereupon yet a new request is sent, before receiving the result of the initial request. When using RMI, this causes no problem, with sockets, on the other hand, if not handled properly, might lead to confusion as to which request a particular result belongs to, and could finally leave the system in an inconsistent state. This lead to the introduction of unique request ID's. When a request is constructed, a unique number is inserted, the request-id, and the result to this specific request will contain that id, and based on these ID's it is simple to determine which result belongs to which request.

Figure 3.3 shows the simplified flow of messages between two nodes during a request initiated by Node A to Node B. Each request contains a unique ID, which is included in the result sent back to the initiator of the request. It can be seen from the figure, that a problem might arise if the messages did not contain ID's, as Node A would not know which result belongs to which request (it can not be assumed that the first result received belongs to the last request sent).

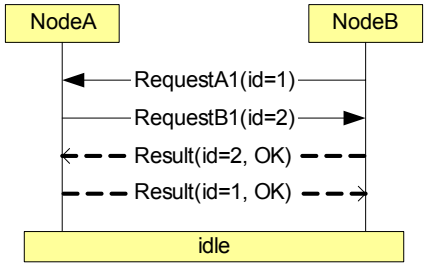


Figure 3.3: Simplified threaded communication

The rather complex Figure 3.4 illustrates how an actual `ActorPlugOut` request from Node B to the Director at Node A would look like. The reader will notice the

introduction of the `MicroComServer`, and its `MicroRPServer` and `MicroRRServer`. These new additions to the architecture greatly simplify communication between nodes.

The flow of the communication is explained in the paragraph below. Numbers in parenthesis correspond to the numbers in the figure. This is a threaded communication similar to the one shown in Figure 3.3, and shows the different mechanisms for ensuring an error-free communication link

The `MicroRRServer`, which operates at a well known port (default port is 9998) is responsible for handling results of a request sent earlier. Before a request is sent from one node to another (2), the local MicroPNES instance registers the request with its `MicroRRServer` (1). This registration contains the unique ID of the request, and the port it will use to receive this result. When a result is received by the `MicroRRServer` (13), it is easy to determine (by looking up the request ID) whether it expects this result, and, if so, which port to use when delivering it to the initiating MicroPNES instance (14). The `MicroRPServer`, which also operates at a well known port (default port is 9999), is responsible for receiving any requests (of type `RequestPars`) sent to the node (2). It forwards the request to its parent MicroPNES instance using a local method call (3). When the request is finished processing by MicroPNES and the return value (of type `RequestResult`) is received (11), this return value is sent both to its own `MicroRRServer` (12) and to the `MicroRRServer` belonging to the originator of the request (13).

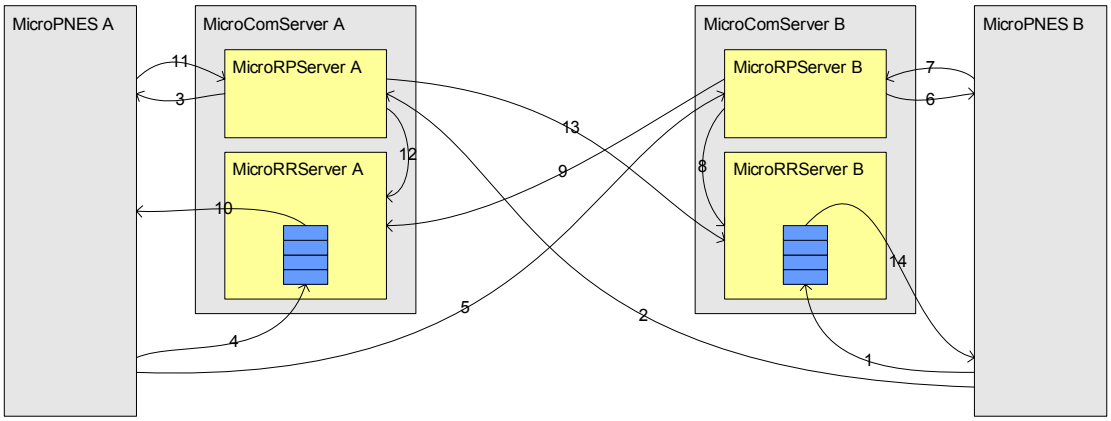


Figure 3.4: Schematic overview of actual communication flow (`ActorPlugOut`)

This new feature has been implemented and tested, and seems to be working as planned. There have been some instability issues when running this on the handheld device, due to sockets closing before the entire message was sent through, but this has been partly fixed by leaving the sockets open a bit longer than usual, before closing them down to conserve system resources.

3.3 Addressing and routing values

The addressing and routing in MicroTAPAS are based on the same principles as that of TAPAS, although the absence of the PAS layer required a slight modification. The PAS instance identifier in the original scheme has been replaced by a copy of the MicroPNES instance identifier. This is done to make future integration with TAPAS easier. Appendix B.3 contains an overview of the original TAPAS addressing scheme.

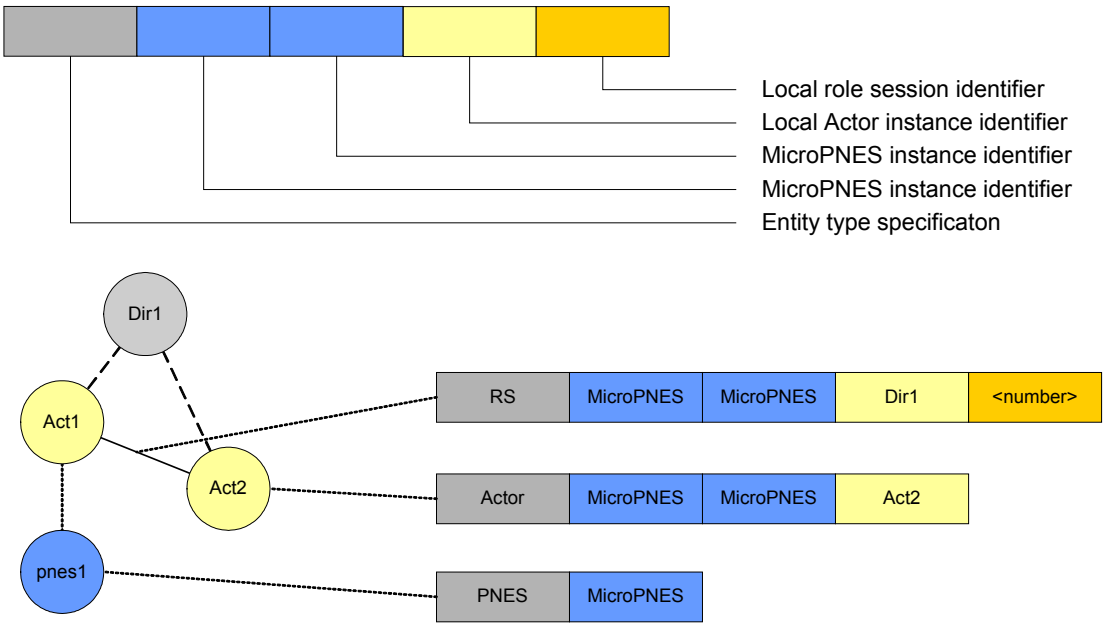


Figure 3.5: GAI for MicroTAPAS

All the necessary alternations have been made to the original code, and the new addressing scheme works fine in the prototype.

3.4 Dynamic connections and device mobility

An important addition in MicroTAPAS, is the support of dynamic connections. When using wireless handheld devices, it will not be uncommon to enter an area where the network connection suddenly is lost. The loss of a connection can be the result of physical obstructions within an area of coverage, or can simply occur if the device leaves the specific are of coverage. MicroTAPAS have been modified to handle such dynamic connections through the introduction of three relatively simple entities. These entities are described below and depicted in Figure 3.6.

- The *MicroPingServer* class listens at a pre-specified port for any incoming socket request from a MicroPingClient. The connection is accepted, whereupon it is immediately closed – signalling to the client, that the port, and thus the node itself, is online.

- The *MicroPingClient* operates in one of two modes, depending upon whether the node contains, or houses, the Director actor or not (i.e. server and client nodes).

    - If the node contains the Director, this client is responsible for retrieving a list, from the MicroConnectionManager, of all the nodes that currently have any actors plugged in with the Director, and at predefined intervals ping these nodes to establish whether they are online or not. If a node appears to be offline, an actorPlugOut request is issued to the Director, and upon completion, removes the entry from its ActorRegistry. This also makes it possible to track which actors are plugged in at which nodes, and any changes to these.
    - If the node, however, does not contain the Director, it simply pings the node at which the Director resides, at predefined intervals. If, or when,

a connection can not be established, an actorPlugOut request is issued to the local MicroPNES instance. Upon completion of this request, the client will continue to monitor the connection. If the connection to the network, and the Director node, can be re-established, the client will issue an automatic actorPlugIn request for those actors that were previously plugged-out.

- The *MicroConnectionManager* is the class that keeps track of which actors have been plugged in, their addresses and their initial RoleSession id's (used for plugging out an actor). When an actor is plugged in, it will automatically notify the local MicroPNES instance and the MicroPNES at the home interface node of its existence, this operation is called actorRegister. When an actor performs a normal plug-out, likewise, the actor notifies these instances, and is called actorRegisterCancel. The actorRegister, and actorRegisterCancel functionality have been added to the MicroApplicationActor class, which all actors inherit, thus making this functionality transparent to any developers of MicroTAPAS applications. It was decided to keep this functionality outside the director, as both client nodes and director nodes need an instance of this entity to deal with dynamic connections.

The MicroConnectionManager introduced above have characteristics similar to those of a manager dealing with terminal mobility (i.e. MobilityManager), as discussed in [MALM1]. This functionality may be extended to cover support for dealing with mobility issues, the functionality of the MicroConnectionManager should then be incorporated into that enhanced support functionality.

Figure 3.6 shows the components needed for dealing with dynamic connections in MicroTAPAS. MicroPNES A houses the director, while MicroPNES B is a normal client-node in the MicroTAPAS system. When Actor B is plugged in it notifies both the MicroConnectionManager (MCM) located at the director node (1), and its local instance of MCM (2), of its existence, i.e. it registers itself with enough information, so that an actorPlugOut request can be issued at a later stage, if necessary (i.e. loss of connection). While the system is up and running, ping requests (X and Y) are constantly (at predefined intervals) sent among all the registered nodes of the system, thus ensuring that all nodes that are registered are actually alive. If a node ceases to respond to the requests it is automatically assumed to be down or offline, and appropriate action is taken.

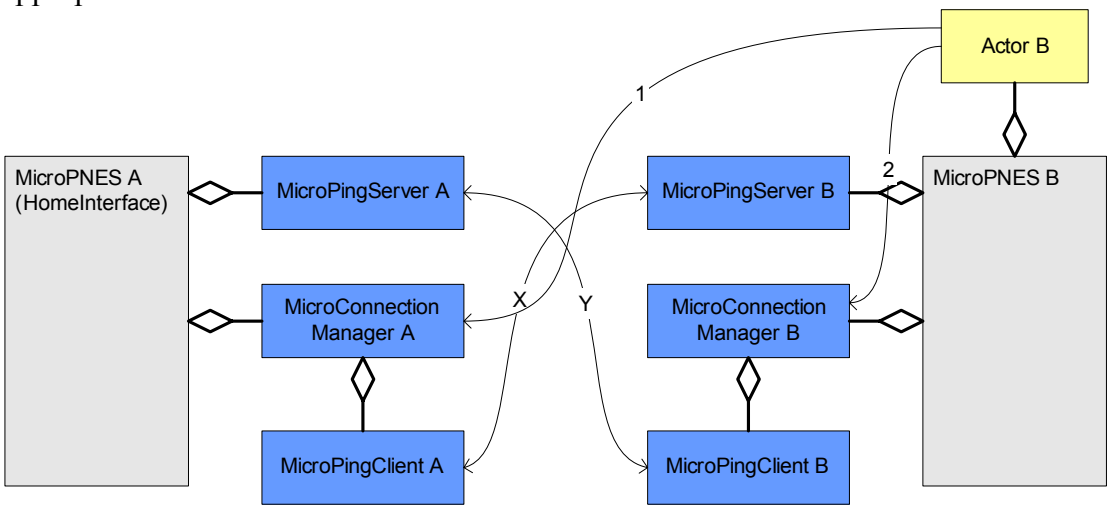


Figure 3.6: Schematic figure of the fundamental elements of the dynamic connection.

This feature has to some degree been implemented in the prototype, and limited testing has been performed with positive results. In the future, these ping requests could be extended to include information about the originator node. The information could be internal state, clock synchronising or other variables.

3.5 The configuration file

In response to the altered nature of MicroTAPAS, a number of changes to the configuration file system have taken place. This file, by default named '`tapas.cfg`', resides at each node in a MicroTAPAS network, and contains key configuration information.

Previously, the contents of this file was read prior to loading the support system itself, then the values were passed as arguments to the newly created MicroPNES instance. This has now been changed, so that the file is read directly by each MicroPNES instance upon initialisation. The main reason behind this change was that the eventual introduction of new configuration values would lead to changes in the bootstrap package at each node, something that is self-contradictory to the flexible nature of TAPAS. Comments can now be inserted into the configuration file, by prefixing each line with a hash (#), blank lines are also tolerated; both are improvements over the original implementation. As a comparison, Appendix B.4 includes an example of how the old and new configuration files may look like.

Currently, there are four required attributes in the configuration file:

- codebase: The Internet address of the binary files of the whole system.
  Example: '`codebase = http://10.0.0.1/tapasroot/`'

- homeinterface: The MicroTAPAS address (GAI) of the Director.
  Example: '`homeinterface = Actor://10.0.0.1/MicroPNES/MicroTAPAS.MicroDirector1`'

- debugserver: The IP address and port of the debugserver
  Example: '`debugserver = 10.0.0.1:9990`'

- nodeprofile : The profile of this node. This attribute should reflect the type of terminal in use. This attribute is currently not in use by MicroPNES.
  Example: '`nodeprofile = profMicro`'

3.6 Requests and Results

To support the added functionality of dynamic connections, new types of requests were introduced. These new types of requests, `ActorRegister` and `ActorRegisterCancel`, are used when registering a newly created actor locally and with the Director node, or cancel registration of a plugged out actor.

Slight modifications also had to be done on `RequestPars` and `RequestResult` when the concept of unique message ID's were introduced. The ID's themselves are automatically inserted into a `RequestPars` upon creation, and is simply the systems `currentTimeMillis` variable, for example the number of milliseconds since January 1, 1970. When creating a new result, the ID from the original request is included as a parameter in the instantiation of the result.

All necessary changes have been made, and no problems have been encountered during testing of these modifications.

3.7 Final remarks

It was by [LILL, page 72] reported that "it takes forever from you send the first PlayPlugIn request until the director is up and running and the play plugged in". Upon investigation, this problem seemed to stem from a while(true) loop [PNES.java, line 247], which consumed almost all system resources, in the creation process of new PAS instances at a node. The removal of the PAS layer naturally solved this problem in MicroTAPAS, but the original architecture would benefit from a different approach to the creation process.

Presently, there is no concrete model of how TAPAS and MicroTAPAS could interoperate, and this should be investigated in a separate report. It is clear that the overall system would benefit from such a solution, as applications could work across different platforms, although it is suspected that for a valuable solution to be presented, much work and careful considerations would have to be invested.

4 Development environment

4.1 Software packages to aid in development

Although the actual development work on MicroTAPAS was carried out using an ordinary text editor, it is believed that any individual working on such a complex system would benefit from using a more dedicated development environment.

Some of the software packages mentioned below also contain software emulators of the target devices, thus making it more convenient to test the system before final deployment. In addition, different target operating systems normally require different packaging and deployment techniques, with which the software also can assist.

Below in Table 4.1 is a list of three readily available and popular development tools on the market. They should all be well suited for developing software for wireless, handheld devices, such as MicroTAPAS. The list is not meant to be exhaustive, and given the speed at which new software arrives, producing such a list would be a daunting task anyway.

Software	J2ME CDC platforms OS/CPU type	Evaluation version
IBM WebSphere Studio Device Developer 5.0	QNX/Arm, Pocket PC/Arm, Linux/Arm, ++	✓
Metroworks CodeWarrior Wireless Studio 7 PDA Edition	Linux/Arm, Pocket PC/Arm	×
Microsoft	Pocket PC/Arm	N/A

Table 4.1: Possible development packages

4.2 Target devices and operating systems

It can be quite a challenge to get the complete picture of what types of personal digital assistants (PDA's) are available out there, and it does not stop there. Additionally you have to consider what types of processors they all have and what operating system (OS) they run. This is important at this stage, as not all versions of the different J2ME CDC virtual machines are guaranteed to run on all devices, even if they are from the same maker and run the same OS. This section will try to highlight a few of the different options one have, starting with operating systems.

4.2.1 Operating systems

There are essentially three different types of OS's for PDA's that are available today[*], Palm OS from Palm Computing Inc., Pocket PC from Microsoft (formerly Windows CE) and Linux. Due to the nature of the Palm OS, and the devices on witch in runs, it is not possible to run the J2ME CDC VM on these terminals, and those devices running Palm OS (Palm, Handspring, Sony etc.) are not included in any further discussion.

The two PDA operating systems left, Linux and Pocket PC are well suitable for deploying and running J2ME CDC applications.

[*] The EPOC operation system from Psion is not included due to their announcement in 2002 that they were pulling out of the market.

4.2.2 Processors

In addition to operating systems, there are a few different processors to consider as well. Those that are most popular by the PDA manufacturers are StrongArm and Intel's XScale, for which there are proprietary CVM's available.

4.2.3 Manufactures

There are many different manufacturers of PDA's, including Hewlett Packard, Toshiba, Sharp, Siemens, Viewsonic, Dell and Packard Bell, and the list is constantly growing. However, who manufactures a device does not really matter, but rather what OS it is running, and to a certain degree what type of processor is has. For this specific project, it is also important that a device have the ability to communicate wirelessly with a network, using the IEEE 802.11b standard, either through built-in WLAN, through WLAN cards attached to the CompactFlash expansion slot or by other means.

4.2.4 Summary

In the table shown below, a few of the possible deployment devices are listed

Currently, there are no Secure Digital (SD) WLAN expansion cards, as there are for CompactFlash (CF) slots, however, that may change in the near future. The aforementioned table does therefore not contain devices that lack CF slots, or does not have built-in WLAN, this includes devices from Toshiba and Viewsonic, among others.

Make/model	OS	CPU	RAM/ROM	Exp. slot(s)*
HP iPAQ H5450†	MS Pocket PC 2002	XScale 400 MHz	64MB/48MB	SD, MMC
SHARP Zaurus SL-5500	Linux 2.4 (Embedix)	StrongArm 206 MHz	64MB/16MB	CF, SD
Toshiba e740‡	MS Pocket PC 2002	Xscale 400MHz	64MB/32MB	CF, SD
Fujitsu-Siemens Pocket LOOX 600§	MS Pocket PC 2002	XScale 400 MHz	64MB/32MB	CF, SD, MMC
Dell Axim X5 Handheld	MS Pocket PC 2002	XScale 300 MHz	32MB/32MB	CF, SD, MMC

Table 4.2: Possible target terminals (PDA's)

4.3 Test environment proposal

For a minimum test environment, it would be advisable to have at least one WLAN running, with an attached web-server, and at lest one handheld device connected wirelessly. In addition one or more stationary (i.e. desktop or laptop) computers should also be connected. For a more realistic environment, at least two WLAN's should be running on different subnets, and a number of handheld devices that could be placed within 'reach' of both WLAN's. This would ensure that all the implemented support functionality could be tested to its full potential, and it would provide data for

* Expansion slots: CF = Compact Flash, SD = Secure Digital, MMC = Multimedia Card.
† Built in WLAN (802.11b), Bluetooth and biometric fingerprint reader.
‡ Built-in WLAN (802.11b).
§ Built-in Bluetooth.

analyzing traffic-load and performance as well – something that would be difficult with only a few terminals running.

In addition to this hardware, one has to use a software package with support for development on handhelds, and a web-server. We have seen, however, that two of the three packages mentioned above, either is free or offers a free evaluation version that should be well suited for academic development and testing. The free Apache (`http://www.apache.org`) web-server is an excellent alternative for a web-server choice.

Minimum test-environment setup

- One laptop/desktop computer with WLAN, running a web-server, and installed Java2 SDK version 1.4.1 or later. The MicroTAPAS Director will run at this node, and will in later chapters be referred to as the server.
- One laptop/desktop computer with WLAN, and installed Java2 SDK version 1.4.1 or later. This node will run as a client.
- One PDA with WLAN and an installed CDC compatible CVM, this node will also run as a client.

This setup will allow the three terminals to connect using an ad-hoc wireless network, and will allow testing between the server and the two clients. Given the relatively low complexity of the communication protocols and added support functionality of MicroTAPAS, in normal circumstances this set-up would probably be sufficient to test the implementation of the new architecture.

Realistic test-environment setup

- Two laptop/desktop computer running web-servers and WLANs at different subnets. The MicroTAPAS Director should also run at these nodes.
- Two or more PDA's with WLAN and an installed CDC compatible CMV.

This will allow the testers to test inter PDA communication, Director-PDA communication and how well the PDAs adjust when switching from one subnet to another. Provided the testing would occur within a relatively small area, and a limited number of nodes would participate, an ad-hoc network strategy would probably be sufficient, if not, two wireless access points would have to be set up in addition to the mentioned hardware. This is a more realistic set-up than the minimum set-up, as it would allow for more extensive testing, not only of functionality, but of reliability as well. Stress-tests could be designed such that any communication and computational bottlenecks in the architecture could be found and dealt with.

5 Sample application - MicroTester

In order to test the functionality of MicroTAPAS, a sample, or test, application had to be developed, and tried out on at least two hosts, a client and a server. This application is MicroTester. The MicroTester application was especially developed to allow for testing of the support functionality of the MicroTAPAS architecture. This application will be used for a thorough test of the implemented functionality in chapter 1.

5.1 Functional requirements

- Upon start-up of the actor MicroTester (client) the user will be presented with a Graphical User Interface (GUI) for executing commands, and view status information, the is the main application window.

- All available commands should be accessible by menu items on a menu toolbar in the application window.

- The application will automatically plug out when the application window is closed.

- It should be possible to PlugIn the MicroTesterServer (server) at the Director node.

- It should be possible to PlugOut the server from the Director node.

- It should be possible to send a RoleSessionAction from the client to the server, and from the server to the client.

- It should be possible to ActorChangeBehaviour the client actor to an almost identical MicroTesterNew actor, and back again to the original client.

- It should be possible to ActorRegister the client.

- It should be possible to ActorRegisterCancel the client.

- When executing a command, the application window should display information to the user that indicates that work is in progress. This will ensure that time-consuming tasks will not be perceived as a lock-up.

- It should be possible to stop the execution of a command once it has been started.

- The application should run as equally well on a desktop/laptop computer as on a handheld device.

5.2 Non-functional requirements

- The program shall be easy and intuitive to use. This includes extra thought given to the input of information on a device lacking a standard mouse and/or keyboard – i.e. a PDA having a touch-screen and a stylus[*].

- The program shall be compact and well written in order to minimise the necessary download time to, and memory footprint needed on, a handheld device.

5.3 MicroTester overview

The MicroTester program consists of three actors; MicroTester, MicroTesterNew (clients) and MicroTesterServer (server). Each plugged in actor of type MicroTester or MicroTesterNew will instantiate a copy of TesterMainWindow. This window will be used by a user to interact with the application, and to display various information back the user. The window consists of a menu bar, containing three groups of choices (File, Basic and Extended), a text-output area and a status bar at the bottom.

The File menu consists of a Stop and Exit command. When an action is selected from one of the two other menus, the Stop action will be enabled, and, if chosen, will stop the execution of the current command. By choosing the Exit command, the window will close and the client actor will be plugged out. The Basic menu consists of four commands and each performs according to their names, PlugIn (plugs in the server), PlugOut (plugs out the server), Upgrade (performs an ActorChangeBeahviour) and RoleSessionAction (sends a role session action to the server). The Extended menu consists of only two commands, ActorRegister (registers the actor for dealing with dynamic connections) and ActorRegisterCancel (cancels a previous registration).

The text-output area simply displays various messages to the user, including what action has been selected, the result of the performed action and the time (in milliseconds) it took to complete. The status bar at the bottom of the screen displays the text ‘Ready!’ when the client is idle. When the client is busy performing a task, the text ‘Working’ is displayed, along with a number of ‘#’ characters. Each 0,5 seconds another ‘#’ appears, until six in a row is displayed, whereupon the counter is reset and starts over again with one. This is done to provide the user with the information that work is in progress and that any unresponsiveness from the user interface stems from this, and not an program error.

When instructed to, the client will plug in the MicroTesterServer at the Director node, i.e. at the same node as the Director actor resides. The only purpose of the server is to respond to an incoming RoleSessionAction sent from the client. The response from the server will be shown in the application window belonging to the client, and includes a short greeting along with the current date and time on the server node. The server can at any time be plugged out executing the PlugOut-command located in the Basic menu in the application window.

5.4 Screenshots

In Figure 5.1 and Figure 5.2 several screenshots are shown of the application executing on a PDA, running the Microsoft PocketPC2000 operating system, and a laptop computer running Microsoft Windows XP, respectively.

[*] A ’stylus’ is the name given to the pen look-alike used by users to interact with the touch sensitive screen on some PDAs.

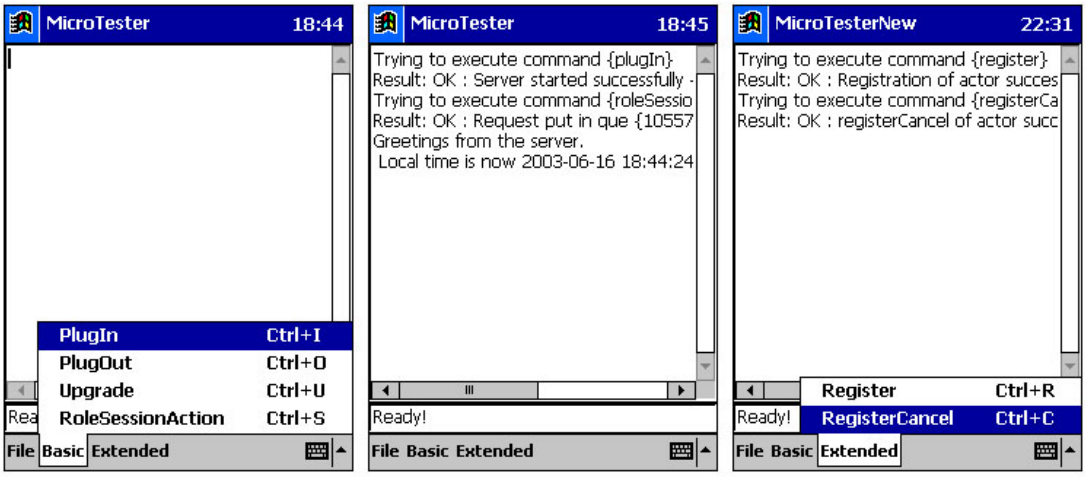


Figure 5.1: Screenshots from the MicroTester application running on a PDA

The three included screenshots from the PDA show the two menus Basic and Extended (left and right), as well as some output from a test-run. In the middle screen, both the PlugIn and PlugOut commands have successfully completed. In the screen on the right, note that it is the upgraded MicroTesterNew actor which is running, and both the Register and RegisterCancel operations have completed successfully.

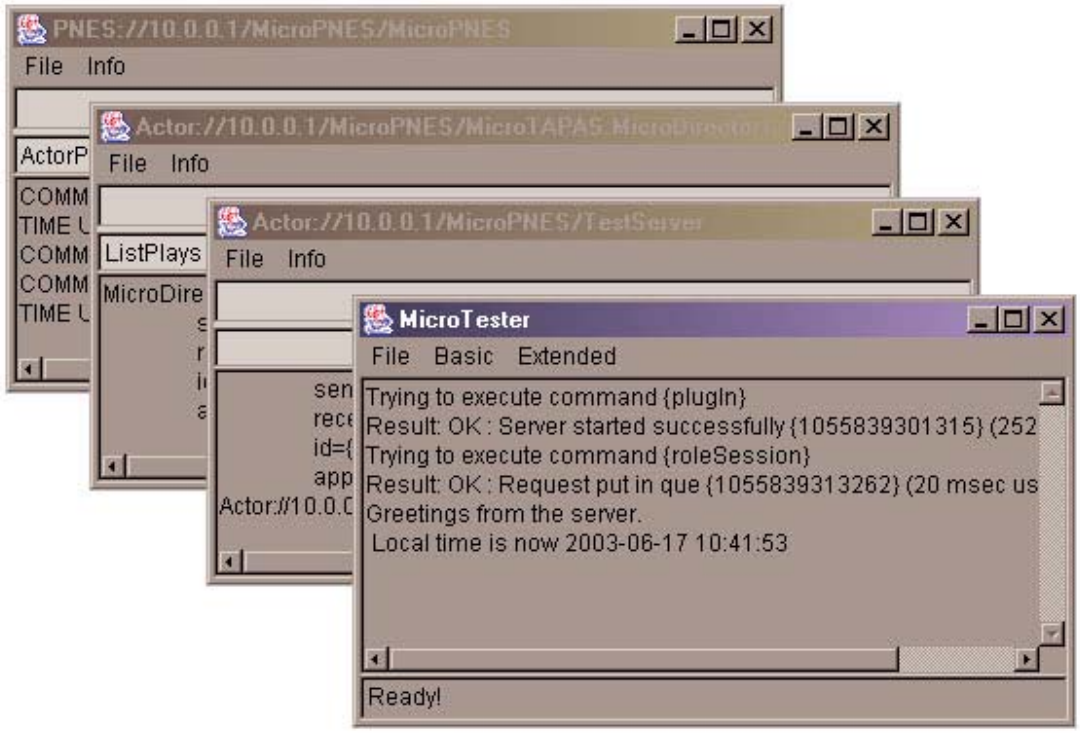


Figure 5.2: Screenshots from the application running on a laptop computer

The screenshot from the laptop computer shows the application running, and, as is shown in the text output area, the MicroTesterServer has been plugged in, and a RoleSessionAction command has been executed, with the response from the server also shown. Behind the application window, various debug windows are shown; from left to right: MicroPNES, MicroDirector1 and MicroTesterServer.

5.5 Class diagram

Figure 5.3 is the class diagram of the MicroTester application. The diagram was automatically generated using the open-source tool ESS-Model [S4].

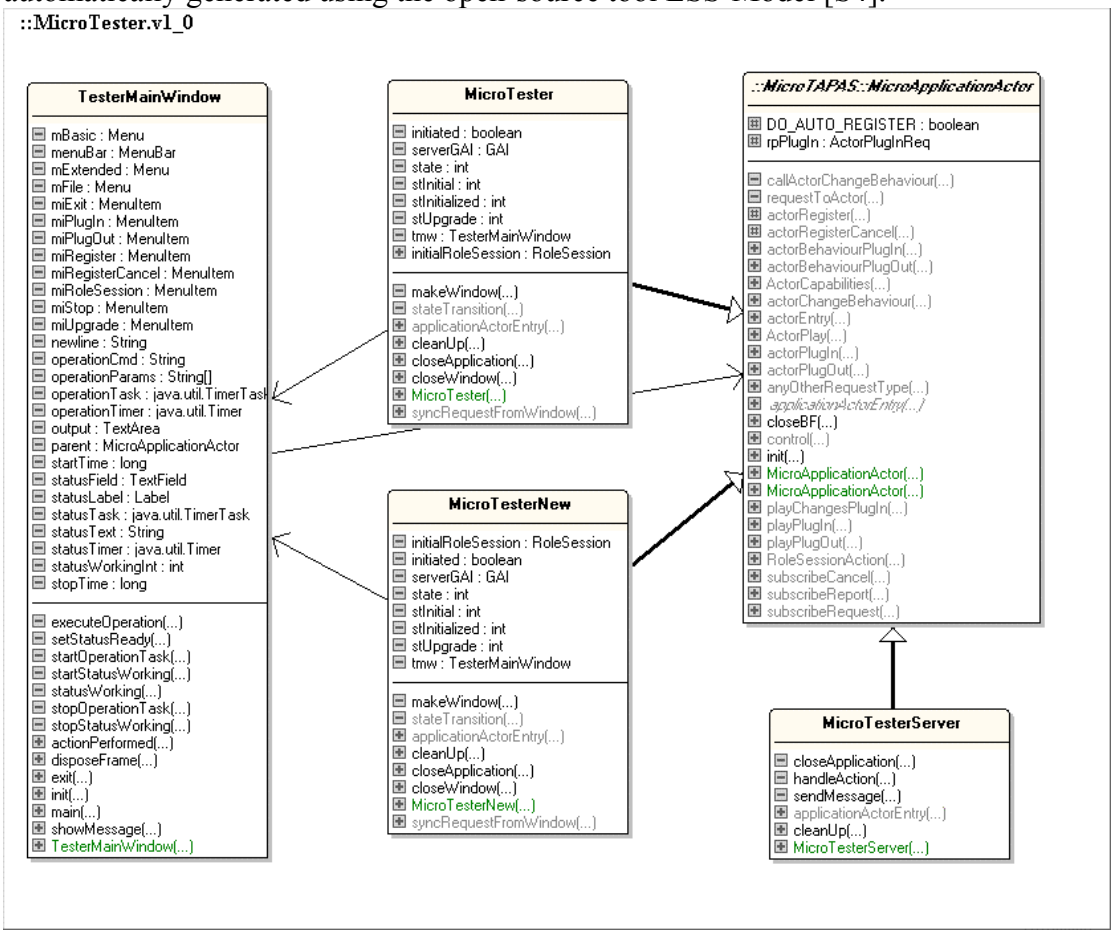


Figure 5.3: MicroChat class diagram

5.6 Message sequence charts

This section contains the Message Sequence Charts (MSC) from the MicroTester application, and show the communication between the server and client. Each heading corresponds to the choice from the menu bar in the application window.

5.6.1 PlugIn

The MSC in Figure 5.4 shows how the MicroTesterServer is plugged in at the Director node. The request sent from the client is a normal ActorPlugIn request, specifying what and where is to be plugged-in.

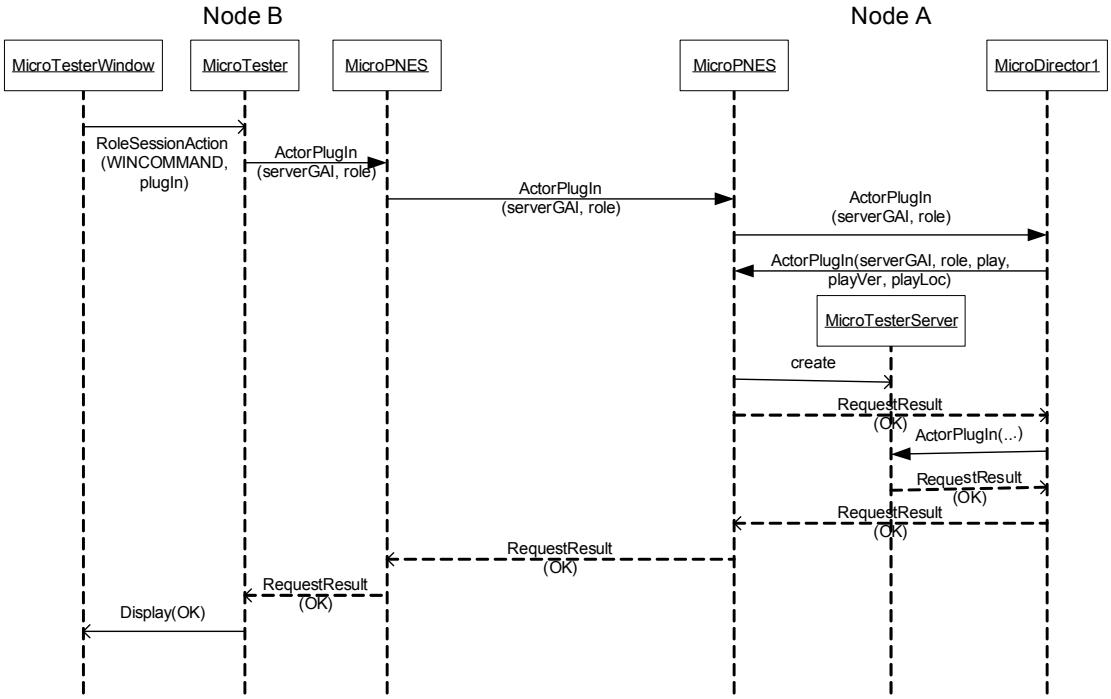


Figure 5.4: MSC for PlugIn of the MicroTesterServer

5.6.2 PlugOut

The MSC for this action, Figure 5.5, shows how the MicroTester server is plugged out and terminated. Instead of sending the plug-out request directly from the client to the Director, the client instructs the MicroTeserServer to plug-out, whereupon it sends the necessary request to the Director. Note that ApplicationMessages (contained in the RoleSessionAction request) is a partly asynchronous operation. The request is delivered to the MicroPNES instance of the node where the recipient is plugged-in, synchronously, and then the request is put in a buffer (implemented by a Vector) to be consumed by the recipient at its leisure (usually that means immediately), and this part is performed asynchronously.

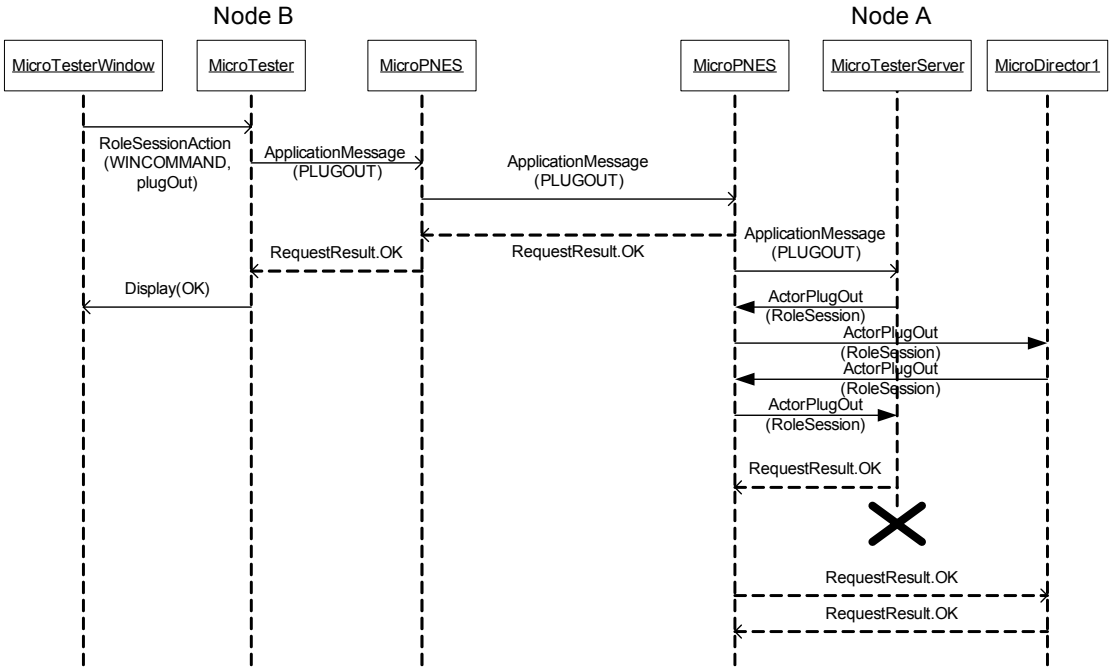


Figure 5.5: MSC for PlugOut of the MicroTesterServer

5.6.3 Upgrade

The Upgrade MSC, Figure 5.6, shows how the MicroTester issues an ActorChangeBehaviour. This request is used to replace or update the current manuscript (i.e. behaviour) of an actor with a new one. In reality, the issuing actor is terminated, and a new actor is instantiated, leaving role-sessions and addresses unaltered

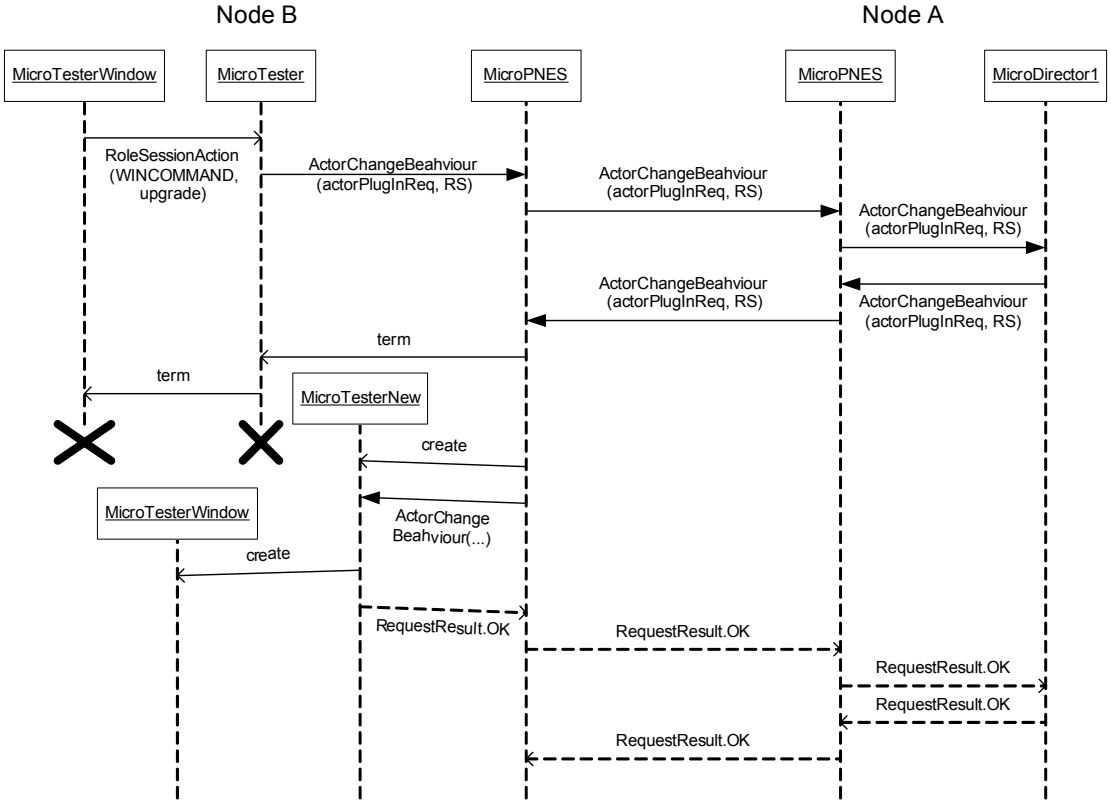


Figure 5.6: MSC for performing an ActorChangeBehaviour

5.6.4 RoleSessionAction

A RoleSessionAction, Figure 5.7, is a request designed to help different actor instances control and communicate with each other. The PlugOut MSC showed how this could be used to control another actor, this time is shows how two actors can communicate through the use of a RoleSessionAction. The client simply sends an application message to the server, asking for its status. The server, if available, replies with a simple text message. The delivery of RoleSessionActions can only be guarantied for to the receiving actor node's MicroPNES instance, as explained in the PlugOut MSC.

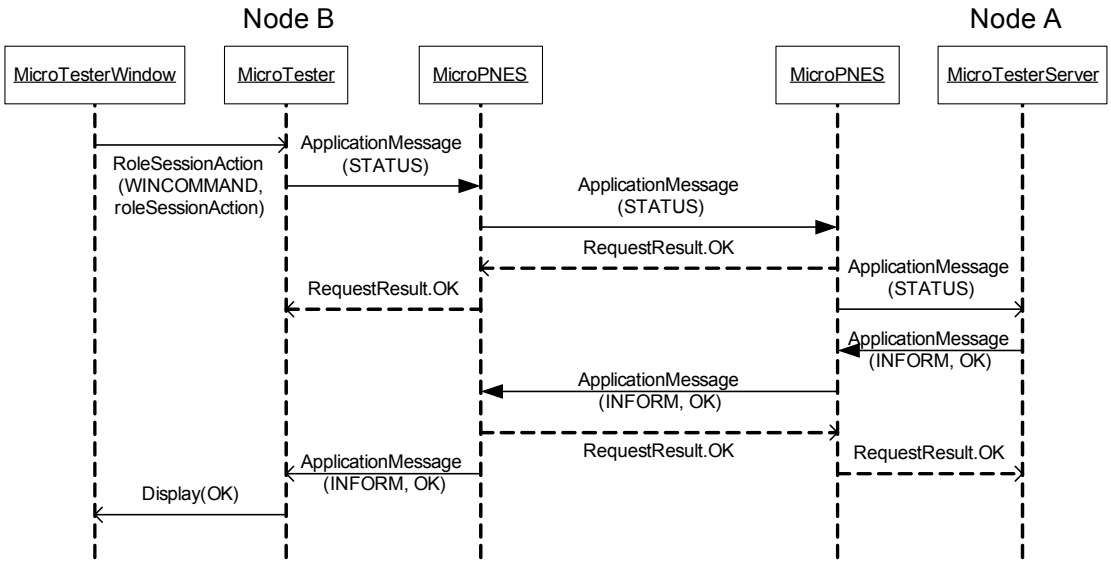


Figure 5.7: MSC for sending a RoleSessionAction to the MicroTesterServer

5.6.5 Register

The ExtendedSupportFunctinality (ESF), Figure 5.8, of MicroTAPAS includes the ability to register an actor with two instances of MicroPNES, the first local, the other located at the node of the Director. The purpose of the registering is to provide the nodes with enough information for them to plug-out an actor if the connection between the actor and the Director node is somehow lost. This ensures that applications/actors that require a connection do not consume resources while it is waiting for the connection to become available again.

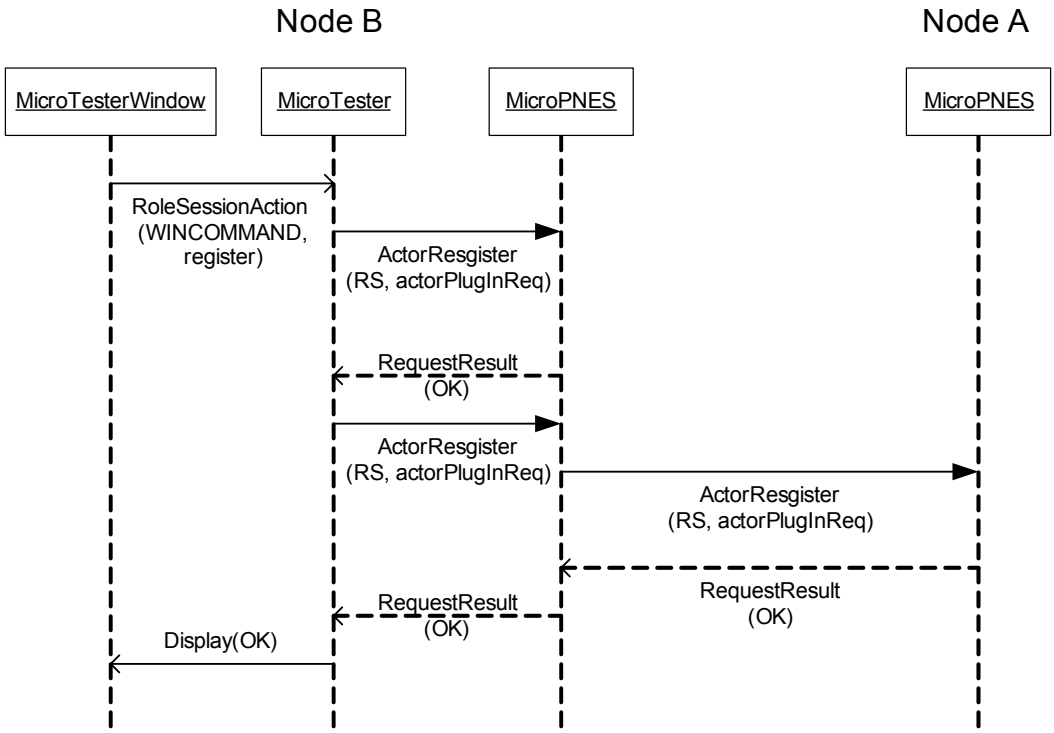


Figure 5.8: MSC for the registration of the MicroTester actor

5.6.6 RegisterCancel

The RegisterCancel request, Figure 5.9, is meant to cancel a previously registered actor. This would normally occur when an actor is explicitly plugged-out (i.e. the user chooses to terminate the current session or application).

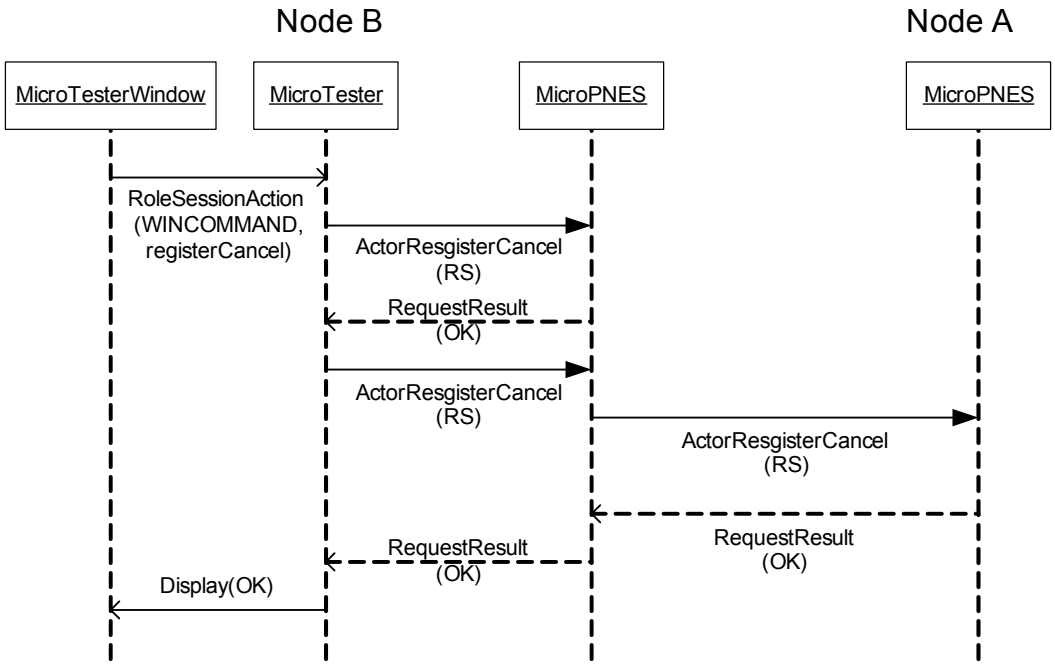


Figure 5.9: MSC for cancelling of a previous registration of the MicroTester actor

5.7 Suggested improvements

Below are listed a number of improvements that could be implemented in a further version of the MicroTester application. Some of these suggestions may overlap with suggested improvements to the MicroTAPAS architecture itself. These are presented in later in chapter 1.

- Secure communication to prevent eavesdropping.
- Include more advanced features for sending requests of different types, where the user could edit all the fields in the request.
- Add different levels of transparency, from a level similar to what is implemented in this version, to a very detailed view of all internal states, communication, requests and results.
- Implement a statistic view, which would keep track of connections, execution times, received and sent requests and results.

6 Setup and testing of MicroTAPAS

The testing of the MicroTAPAS architecture was carried out on a minimum set-up, similar to the one that was proposed in section 4.3. The three nodes are connected through an ad-hoc WLAN connection. The purpose of the test was to verify that the implemented support functionality worked in accordance with the specification presented in section 2.2, MicroTAPAS Support Functions. The development of the sample application presented in chapter 1 was specifically designed for the purpose of testing the support functionality, and will be used for the tests performed.

An overview of the node details are provided in Table 6.1. For detailed information about each node, refer to Appendix C.1. The meaning of a node in the system is identical to that of a terminal/device a user would use to interact with the support system. The 'Type' and 'Connection' headings should be clear to all readers, but the 'Virtual Machine (VM)' and 'Role' headings might need further explanation. The VM column show what type of Java VM is installed on each node, as there are several options for this choice. The Role column explains what 'role' is implicitly assigned to each node; client and director nodes, and web-server. Each node might have more than one role. This role does not have anything to do with the roles used in the theatre metaphor used by the MicroTAPAS architecture.

Node	Hardware	Connection	Virtual Machine (VM)	Role
A	Laptop (IBM)	WLAN	Sun JVM	Web-server, Director node, Client node
B	PDA	WLAN	IBM J9	Client node
C	Laptop (HP)	WLAN	Sun JVM	Client node

Table 6.1: Node configurations

The terminals may be referenced to simply as 'the laptop' or 'the PDA', in addition to the specific node assignments shown above.

6.1 Installation and start-up

The installation and start-up instructions for desktop and laptops, compared to those for handheld devices, are somewhat different, and the two are therefore handled in two separate sections below.

The one common requirement which is fundamental to any TAPAS system is the presence of a web-server on a reachable network node. All the MicroTAPAS binaries should be put in a directory on the server and that directory should be accessible by anyone. The complete URL to this directory should then be included in the configuration file, as described earlier in section 3.5.

6.1.1 Desktop and Laptop computers

The installation and start-up of MicroTAPAS is very similar to that of TAPAS, except for the absence of any PAS instance. The following three points explain what is required to set-up and run MicroTAPAS on each node. The TAPAS web-server could, in theory, be running on any node. In a one-node system, the single node could run its own web-server, thereby eliminating the need for a network connection in order to dynamically 'download' and execute the support system. This is, however, an exception and would only be used for local testing, but it illustrates how flexible the architecture is.

- Each node should have Java2 1.4.1 or later versions installed, along with the binaries from the MicroTAPASBoot directory.

- The configuration file should be checked for any errors and omissions, and be placed at the root level of the MicroTAPASBoot directory.

- To start each node, simply execute the provided '`MicroTAPASstart.bat`' file included in each MicroTAPASBoot directory, and after a short while, the MicroPNES debug/interaction window will appear on the screen. One is now ready to interact with the support system.

Figure 6.1: MicroTAPASBoot directory on a laptop computer

Figure 6.1 shows the contents of the MicroTAPASBoot directory. The MicroTAPAS directory contains two java .class files that are used during start-up. The whole bootstrap occupies a modest 5 KB of disk space on the computer.

6.1.2 Handheld computers

The installation of MicroTAPAS onto handheld devices can require some more work compared to the stationary computers, and can vary from manufacturer to manufacturer. The steps below explain how to deploy the application onto our test PDA, the iPAQ H3660 running Microsoft PocketPC2000 (PPC2000), from a Windows PC.

- Make sure a Java Virtual Machine is installed on the device, and the required libraries/profiles. A good alternative is to use IBM's J9 VM.

- Transfer the MicroTAPASboot binaries onto the device, i.e. by using Microsoft Active Sync.

- Make sure the configuration file is up to date, and place it in the root directory of the *device* (note the difference to the desktop/laptop installation).

- Edit and place a link somewhere in the directory structure of the device, i.e. in the '\Windows\Start Menu\Programs' folder. One such link-file example is included in the downloaded MicroTAPASBoot directory. See Appendix A.2 for an explanation of the PPC2000 link-file.

- Navigate to the link and execute it by tapping on it. The application will start up and download the required binaries from the web-server, and the user will shortly be ready to interact with the system.

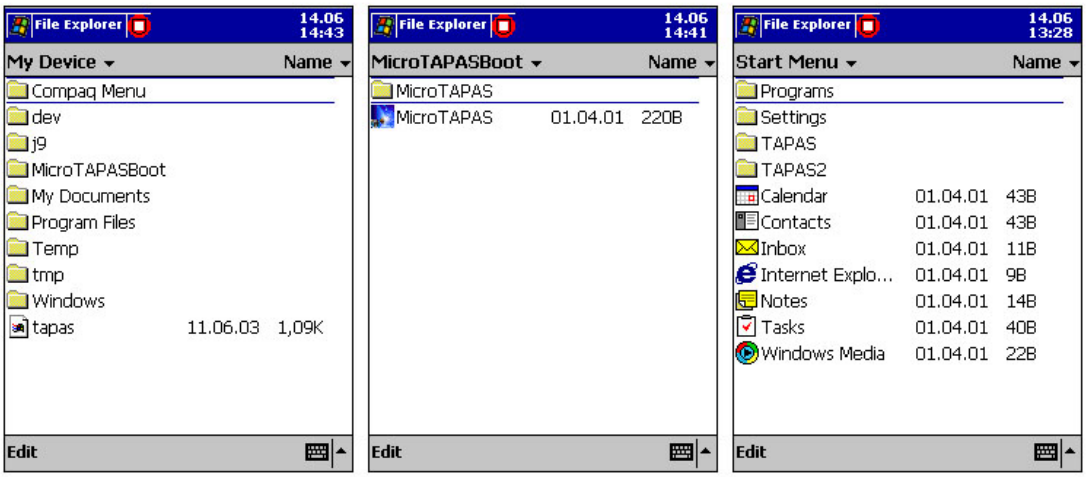


Figure 6.2: Screenshots from the iPAQ showing two directories (on left) and a menu

The leftmost screenshot in Figure 6.2 shows the root directory of the handheld device. The complete path of the configuration file is thus `\tapas.cfg`. In PPC2000 there is no notion of different discs, and therefore drive and the disc letters (i.e. 'A:' and 'C:' on Windows systems) are left out.

The middle screenshot shows the contents of the MicroTAPASBoot directory on the handheld. The file in the picture, MicroTAPAS (with the blue icon), is actually a PPC .lnk file (similar to shortcut files in Windows), and is used for starting programs. Appendix A.2 shows an example of such a file, and explains how to edit exiting ones, and where to put them. The contents of the whole boot directory are as follows:

```
\MicroTAPASBoot\MicroTAPAS\PaPurlLoader.class
\MicroTAPASBoot\MicroTAPAS\startMicroTAPAS.class
\MicroTAPASBoot\MicroTAPAS.lnk
```

The last picture shows the Start Menu of the iPAQ. This directory consists entirely of .lnk files and other directories. One can see the TAPAS directory which in turn contains a copy of the MicroTAPAS.lnk file discussed in the paragraph above. The complete path to this file would therefore be

```
\Windows\Start Menu\TAPAS\MicroTAPAS.lnk
```

6.2 Testing

This section summarizes the tests performed on the system. Each support function is handled in its own sub-section. The testing has been classified into two different categories; Basic Support Functionality (BSF) and Extended Support Functionality (ESF). BSF refers to everything that was implemented in the original version of TAPAS, and ESF is the functionality that was added with the introduction of MicroTAPAS. All the commands are executed on the given node towards the Director node. Where applicable, the various commands have thus first been executed at one of the client nodes, then on the Director node (towards itself). This was done to measure communication and platform overhead, if any, and the time column in the far right of each table states the time it took to execute each command.

Throughout this section, the sample application from chapter 1 was used for testing, and for more in-depth information about that application, refer to that chapter. For each test performed there are two tables.

The first table summarizes the test conditions and prerequisites. The first column show the test number (1 through 12) for simple reference. The second column states from where (i.e. which application or window) the command was executed, for example, 'MicroPNES' means that the command was executed from the MicroPNES window. In the third column the executed command is shown. The command might be a text string, as will be the case for MicroPNES, or a menu choice, which will be the case for the sample application. The last column indicates what other steps must have been performed prior to executing this command, i.e. one can not plug out a play before it is plugged in, and numbers in this column refers to the test numbers in column one.

The second table lists the result of the test for each node. The result can be either 'OK', or 'FAIL', or, as not all tests are applicable to all nodes involved, 'n/a' (not available). The third column is a short note about the result or test itself, and finally, the last column states the time it took to perform the execution of the command. This information will be analyzed in section 6.3. The details of each operation is described in sections 5.6.1 through 5.6.6, and is therefore left out of this section.

6.2.1 Basic functionality

6.2.1.1 PlayPlugIn

No:	From:	Command executed:	Req:
1	MicroPNES	`PlayPlugIn MicroTester v1_0 http://10.0.0.1/tapasroot/`	--

At node:	Result:	Note:	Time (sec.):
A	OK		2,45
B	OK		4,43
C	OK		1,25

6.2.1.2 PlayPlugOut

No:	From:	Command executed:	Req:
2	MicroPNES	`PlayPlugOut MicroTester`	1

At node:	Result:	Note:	Time (sec.):
A	OK		0,60
B	OK		3,53
C	OK		0,57

6.2.1.3 ActorPlugIn of MicroTester

No:	From:	Command executed:	Req:
3	MicroPNES	`ActorPlugIn Actor://<node IP>/MicroPNES/Tester1 MicroTester`	1

At node:	Result:	Note:	Time (sec.):
A	OK		2,53
B	OK		20,04
C	OK		4,23

6.2.1.4 ActorPlugIn of MicroTesterServer

No:	From:	Command executed:	Req:
4	MicroTester	Choose **PlugIn** from the **Basic** menu.	1,3

At node:	Result:	Note:	Time (sec.):
A	OK		3,03
B	OK		11,62
C	OK		1,43

6.2.1.5 ActorPlugOut of MicroTestServer

No:	From:	Command executed:	Req:
5	MicroTester	Choose **PlugOut** from the **Basic** menu.	1,3,4

At node:	Result:	Note:	Time (sec.):
A	OK	This command uses an asynchronous message, thus no result is delivered and no time can be recorded for the full execution time, just the time it takes to deliver the message to the asynchronous message buffer.	0,09
B	OK		4,16
C	OK		0,43

6.2.1.6 ActorPlugOut of MicroTest

No:	From:	Command executed:	Req:
6	MicroTester	Close the application by choosing **Exit** from the **File** menu.	1,3

At node:	Result:	Note:	Time (sec.):
A	OK	The command is executed from the window that would display its time and this window will be closed if the action is successful, therefore the time used is written to standard out instead.	0,74
B	OK		14,02
C	OK		2,04

6.2.1.7 ActorChangeBehaviour

No:	From:	Command executed:	Req:
7	MicroTester	Choose **Upgrade** from the **Basic** menu.	1,3

At node:	Result:	Note:	Time (sec.):
A	OK	The command is executed from the window that would display its time, but the start time is included as a parameter in the request sent, which is eventually forwarded to the newly plugged in actor, who den can determine the time used.	0,96
B	OK		19,16
C	OK		0,99

6.2.1.8 PlayChangesPlugin

No:	From:	Command executed:	Req:

8	MicroTester	`PlayChangesPlugIn MicroTester v1_1 http://10.0.0.1/tapasroot/`	1

At node:	Result:	Note:	Time (sec.):
A	OK		2,76
B	OK		5,17
C	OK		3,09

6.2.1.9 RoleSessionAction

No:	From:	Command executed:	Req:
9	MicroTester	Choose **RoleSessionAction** from the **Basic** menu.	1,3,4

At node:	Result:	Note:	Time (sec.):
A	OK	This command uses an asynchronous message, thus no result is delivered and no time can be recorded for the full execution time, just the time it takes to deliver the message to the asynchronous message buffer.	0,11
B	OK		9,52
C	OK		0,31

6.2.2 Extended functionality

6.2.2.1 ActorRegister

No:	From:	Command executed:	Req:
10	MicroTester	Choose **ActorRegister** from the **Extended** menu.	1,3

At node:	Result:	Note:	Time (sec.):
A	n/a	Actors located at the Director node can not Register/RegisterCancel, as the feature is provided to handle dynamic connections between the actor and the Director.	n/a
B	OK		7,41
C	OK		0,93

6.2.2.2 ActorRegisterCancel

No:	From:	Command executed:	Req:
11	MicroTester	Choose **ActorRegisterCancel** from the **Extended** menu.	1,3,10

At node:	Result:	Note:	Time (sec.):
A	n/a	Actors located at the Director node can not Register/RegisterCancel, as the feature is provided to handle dynamic connections between the actor and the Director.	n/a
B	OK		5,58
C	OK		0,42

6.2.2.3 Dynamic connection (implicit)

No:	From:	Command executed:	Req:

12	Node	Disconnect the device from the network card.	1,3,10

MicroTAPAS will detect the loss of connection according to the ping timer constant specified for that node (default is 10 seconds).

At node:	Result:	Note:	Time (sec.):
A	n/a	A client running on the Director node can not loose connection with itself.	n/a
B	OK		n/a
C	OK		n/a

All the tests that were carried out did perform as expected. One can see that execution times, where included, can be substantially higher for the PDA than for the laptop, and this is should be of no surprise, considering the computational limitations of the device. The next sections discuss some performance issues.

6.3 Performance

This section contains a brief discussion of the relative performance of MicroTAPAS, executing on handheld devices, compared with the performance on its stationary counterparts.

6.3.1 Handheld vs. stationary

In performing the tests on the MicroTAPAS architecture, only one type of handheld device has been available, and it is difficult to say anything about the general performance of MicroTAPAS running on these devices. Instead, performance testing was conducted between the laptop computers and the PDA, aimed at documenting the differences in execution times seen section 6.2.

The comparison of the performances of the architecture running on a laptop and a PDA was done fairly straight forward; HeapTest [S5], which is a small test application developed in Java, was download from the Internet, compiled and run on both terminals. The program was originally developed in February 2001 for testing the performance of a Java application running on two different Java Virtual Machines for Solaris. HeapTest is a multithreaded program and iterates through a number of loops of work. In each loop, it performs a set of memory allocation and computation tasks.

HeapTest is started, from the command line, with two input parameters; the number of threads and the number of cycles. For example, starting the program with

```
java heaptest.HeapTest 5 2
```

will start the program with a maximum of five threads and performs a maximum of two loops for either memory allocation or computation. With the number of cycles set to two, HeapTest will perform a set of tasks three times; the first time, it performs two loops of memory allocation, the second time it performs one loop of memory allocation and one loop of computation, and the third time it performs two loops of computation. Upon completing the execution, the program prints to screen the time, in milliseconds, it took to complete each task for the given thread.

The program was run twice on each terminal, and the results are summarized in Figure 6.3. Refer to Appendix C.2 for a complete listing of the data collected during the six test-runs.

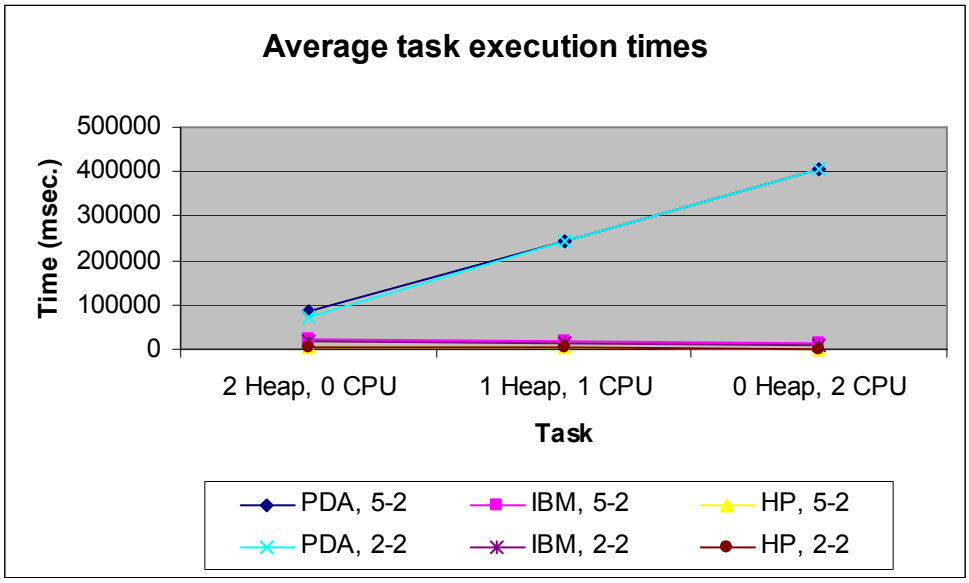


Figure 6.3: Summary of performance test-results

It should be noted that the laptops and PDA used two different Java Virtual Machines, Sun Microsystems Inc's Java HotSpot(TM) Client VM, version 1.4.1_02-b06 and IBM's J9, version 2.0, respectively, and that the results are not directly comparable, although it gives a clear indication as to the relative performance of each platform. It is interesting, though not surprising, to note that memory allocation is almost equally fast on all terminals, but that the PDA is drastically slower when it comes to computation.

This simple test can explain the findings in section 6.2, that the execution times for the PDA is higher than that of the laptops.

This test supports, to a limited degree, the claim that it is the limitations of the PDA itself that caused MicroTAPAS to execute considerably slower on this device than on a stationary computer. More testing is, however, required to confirm this claim. One way would be to test the architecture on PDA's with more resources (i.e. more memory and a faster processor), and compare those results with the ones shown in section 6.2 and 7.3.

7 Faced challenges and solutions

Several challenges were encountered while working on this project, and solutions were engineered, where possible. This section gives a brief overview of these challenges and solutions, and an explanation of why they were encountered in the first place, and what might have been done to prevent them from occurring.

Probably the biggest challenge faced was TAPAS itself and, in particular, the complexity of the system. It took several weeks to get a sufficient understanding of the system before any real work on the task could be started. Part of this problem probably stems from the sheer size of TAPAS, involving around 40 different classes (of which several lacks proper commenting) and several 1000 lines of code. The available documentation of the system is good at giving a general overview, but it is felt that it fails to provide comprehensive knowledge of the inner workings. Another reason for struggling with this part of the project was that it was conducted in another country, far away from any persons that could easily have explained the more intricate parts of the system. This form of consultancy is not suitable to conduct over telephone or email. In the future, to avoid this, it would be very helpful with a more complete documentation of TAPAS, including class diagrams and message sequence charts etc.

The biggest challenge during actual re-specification and implementation of TAPAS into MicroTAPAS was the lack of support for RMI in J2ME (to the extent used in TAPAS). This caused many changes to the original TAPAS, and would in the end prevent existing TAPAS to cooperate/communicate with the new MicroTAPAS. The communication model had to be completely re-engineered using sockets, thereby loosing some of the inherent flexibility and security of TAPAS. In the future, as PDA's continue to evolve along with the functionality of J2ME, it might be possible to conduct communication by RMI once again, until then; sockets are probably the best alternative.

Using different virtual machines and processor architectures can lead to unwelcome surprises. It was discovered, after a number of unexplainable lock-ups and inconsistent system-states, that some of the Java operations used in MicroTAPAS behaved differently on the stationary and handheld computers. For example, the `java.awt.Frame.dispose()` method (used to close an open window) made one of the laptops freeze, while it worked fine on the PDA. Then the `java.awt.Frame.hide()` method was used instead, but made the PDA unstable. The solution was to detect the processor architecture of each device (i.e. x86 for the laptops and ARM for the PDA), and select appropriate methods based on that. Further, it was discovered that use of the `System.exit(int)` method does not work on the PDA, causing it to freeze. No alternative has so far been found, but one can simply close the J9 execution window to exit the Java VM. Use of the java.util.Calendar.setTimeInMillis(long) is also prohibited on the PDA. These inconsistencies are unfortunate, and can hopefully be attributed to the fact that the J2ME technology is still in its infancy, and that these issues will be addressed in subsequent releases.

There was also some confusion that resulted from the fact that much of the technology, both software and hardware, that had to be used or considered for this project is still in its infancy, and the rate of significant improvements over previous versions are still very high. This is a common challenge facing everybody doing research within new fields of high complexity, and cannot easily be avoided. During the few months this

project were undertaken, there are already new technologies that render some of the earlier research obsolete, and this had to be replaced with up-to-date information.

8 Suggested improvements and further issues

This section presents a few suggested improvements to future versions of the MicroTAPAS architecture.

8.1 Dynamic connections

In the future, more functionality should be added to this feature. For example, the actor itself ought to be able to decide if it should automatically be plugged-in or out, following the loss or re-establishment of the connection. Nodes themselves should also be given this choice, overriding the choice of actors.

8.2 The configuration file

Although extensive changes have been made to the configuration file, and how parameters are transferred to a running node, further extensions and/or amendments could be introduced.

These are some possible future extensions:

- Addition of different debug options (text output, windows of different sizes etc.).
- Addition of different ports for communication and ping-requests (must be the same for all nodes).
- Timing options for ping-requests between nodes (can be different from node to node).
- Addition of desktop/laptop and handheld node options (should replace/extend the '`nodeprofile`' attribute).

8.3 Encryption & checksums

Different levels of encryption of all communication could be introduced to the MicroTAPAS architecture. The user, or the application, could be given the choice as to whether messages sent between MicroPNES instances should be encrypted, and to what degree, i.e. bit-length of keys and encryption algorithms. There are a lot of possibilities here that should be investigated, and could probably warrant a separate project in itself.

8.4 Mobility

In [MALM1], the authors suggest how to introduce four types of mobility management to the TAPAS architecture; actor, terminal, user and session mobility. Although this paper was not specifically written with the MicroTAPAS architecture in mind, the architecture would benefit from the implementation of the mechanisms proposed in the report.

Especially the introduction of session mobility, also described by [LILL] and terminal mobility, briefly discussed in section 1.3.2, would be advantageous to MicroTAPAS. The introduction of session mobility should be seen in association with dynamic connections, where the connection, and possibly, the application, can terminate without warning, thus loosing any unsaved data. This would be devastating for more complex applications where, for example, data is collected over time and should be kept for further analysis, i.e. a network monitoring and statistics application.

8.5 Reliability

Presently, there are no specific features built into the support system to ensure its reliability. This is one area that clearly should be addressed in the future, as the system is highly vulnerable to exceptions, failures and down-time of the web-server, the Director node or the underlying network itself.

8.6 Interoperability with TAPAS

As mentioned in Chapter **Error! Reference source not found.**, a future version if MicroTAPAS should incorporate a model as how to accomplish interoperability with TAPAS.

8.7 Applications

There are a number of applications that could be developed to take advantage of the MicroTAPAS on PDA's, and here is a short list describing a few possible contenders.

- A SNMP monitoring application. The server maintains a list of nodes and status' that can be viewed by any MicroTAPAS client.
- Mobile dynamic measurement of WLAN coverage and signal strength, sent back to server and made available to all connected clients.

9 Conclusion

The purpose of this project was to re-specify and develop a new version of ITEM's TAPAS architecture suitable for deployment to wireless personal digital assistants. Initially it was assumed that only minor changes to the already existing architecture would be necessary to accommodate this, and that the relative cheep Palm OS powered devices would be the target platform of choice. In this report, however, it has been shown that neither of these assumptions would hold, and that the architecture had to undergo greater changes, partly due to the lack of RMI support in J2ME. The intended Palm OS powered target devices were deemed unsuitable, due to lack of features and processing power, thereby concentrating the development towards more powerful devices, powered by Microsoft's Pocket PC and a lightweight version of Linux.

The report shows the steps taken in the re-specification process, and the introduction of new features to the architecture. The implementation of the re-specified architecture is thoroughly explained, ranging from the decisions taken in choosing J2ME as a development platform, to the detailed explanation of the newly introduced features. A simple example application was also introduced and developed to show the features of the new architecture.

The testing of the MicroTAPAS architecture shows that it does work well on the intended platforms; handheld wireless devices, as well as on 'regular' computers. However, there are still some issues with performance that should be addressed in the future, as the execution times are considerably higher than for stationary computers. It was shown that this probably stems from the computational limitation of the PDA's, and not from the implementation itself.

It should have been made clear to the reader that there are a number of outstanding issues and suggestions that should be addressed in subsequent reports. Most notably among them are the issues relating to performance, security, mobility and reliability, as these areas continue to receive a lot of attention in any distributed and dynamic architecture.

List of acronyms

Alphabetical list of common acronyms encountered in the text.

API	Application Programming Interface
AWT	Abstract Windowing Toolkit
CDC	Connected Device Configuration
CF	Compact Flash
CLDC	Connected Limited Device Configuration
CVM	C Virtual Machine
FA	Foreign Agent
HA	Home Agent
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JFC	Java Foundation Classes
KQML	Knowledge Query and Manipulation Language
KVM	Kilobyte Virtual Machine
MMC	Multimedia Card
MSC	Message Sequence Chart
PDA	Personal Digital Assistant
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SD	Secure Digital
TAPAS	Plug-and-Play
VM	Virtual Machine
WLAN	Wireless Local Area Network

Bibliography

[AAGF1] Finn Arve Aagesen, Chutiporn Anutariya, Mazen Malek Shiaa and Bjarne E. Helvik, "Capability Specification and Selection in TAPAS", IFIP WG6.7 Workshop and Eunice Summer School on Adaptable Networks and Teleservices, Trondheim, Norway, September 2002, Tapir, ISBN: 82-993980-5-3.

[AAGF2] Aagesen, F. A., Helvik, B. E., Johansen, U., Meling, H., "Plug and Play for telecommunication functionality – architecture and demonstration issues", http://www.item.ntnu.no/~plugandplay/publications/IConIT-2001.pdf [Accessed February 2003]

[AAGF3] Aagesen, F. A., "Plug and Play for telecommunication functionality – architecture and demonstration issues", presented at IConIT 2001, Bangkok, Thailand. http://www.item.ntnu.no/~plugandplay/publications/IConIT.pdf [Accessed March 2003]

[AAGG] Aagesen, Gustav , "Plug-and-Play Application Management System", MSc thesis, Department of Telematics, NTNU, 2001.

[COLU] Colombia University, Department of Computer Science, NetScript, http://www.cs.columbia.edu/dcc/netscript/. [Accessed March 2003]

[COUG] G. Coulouris, J. Dollimore, T. Kindberg, "Distributed systems, concepts and design", third edition, Addison-Wesley, 2001.

[DARP] U.S. Department of Defence, Advanced Technology Office, http://www.darpa.mil/ato/programs/activenetworks/actnet.htm. [Accessed March 2003]

[LILL] Lars Erik Liljebäck, "User and Session mobility on a Plug-and-Play architecture", MSc thesis, Department of Telematics, NTNU, 2002.

[MALM1] Mazen Malek and Finn Arve Aagesen, "Mobility management in a Plug and Play Architecture", IFIP TC6 Seventh International Conference on Intelligence in Networks, Saariselka, Finland, April 2002. Published by Kluwer Academic Publishers.

[MELH1] H.Meling, TAPAS, http://www.item.ntnu.no/~plugandplay/ [Accessed September 2002]

[MELH2] H.Meling, http://www.item.ntnu.no/~plugandplay/documentation/SystemDoc/Main/Main.pdf.[Accessed September 2002]

[MITE] Massachusetts Institute of Technology, Active Networks, http://www.sds.lcs.mit.edu/activeware/. [Accessed April 2003]

[REID] Reilly, David, "Mobile Agents - Process migration and its implications", http://www.davidreilly.com/topics/software_agents/mobile_agents/. [Accessed March 2003]

[STAU] Stanford University, Department of Computer Science, Knowledge Sharing Effort, http://www-ksl.stanford.edu/knowledge-sharing/. [Accessed March 2003]

[SUN1] Sun Microsystems, Java 2 Platform Micro Edition, http://java.sun.com/j2me/j2me-ds-0201.pdf, page 1-2. [Accessed July 2003]

[SUN2] Sun Microsystems, J2ME Connected Device Configuration (CDC) http://java.sun.com/j2me/docs/j2me_cdc.pdf [Accessed June 2003]

[TOPK] Topley, K., "J2ME in a nutshell: A desktop quick reference", O'Reilly, 2002.

[UMBC] University of Maryland, Baltimore County, Lab for Advanced Information Technology, KQML Web, http://www.cs.umbc.edu/kqml/. [Accessed March 2003]

[UPEN] University of Pennsylvania, Department of Computer and Information Science, Bellcore, http://www.cis.upenn.edu/~switchware/. [Accessed April 2003]

General online references and resources

Software		
No.	**Internet address**	**Description**
[S1]	http://www.ibm.com/	Home of the Web Sphere Device Developer (WSDD) developers. Information on their products and evaluation download of development tools.
[S2]	http://www.codewarrior.com/	Information about the Codewarrior software suite for developing java applications wireless devices.
[S3]	http://www.pocketpc.com/	Official site with information and downloads for the popular PDA operating system, Microsoft PocketPC.
[S4]	http://www.sourceforge.net/projects/essmodel	ESS-Model is a simple, reverse engine, UML-tool for Delphi/Kylix and Java-files. Its an open-source project.
[S5]	http://developer.java.sun.com/developer/technicalArticles/Programming/JVMPerf/	An article about 'Java Virtual Machine Performance' and download of the HeapTest application.

Hardware		
No.	**Internet address**	**Description**
[H1]	http://www.hp.no/	Information about the iPAQ series of PDA's.
[H2]	http://www.myzaurus.com/	Zaurus is Sharp Electronic's contribution to the growing market of PDA's. This official site is dedicated to their products.
[H3]	http://www.fujitsu-siemens.no/	Information about handheld computers from Fujitsju-Siemens.
[H4]	http://www.dell.no/	Product information and details for the Dell Axim A-5 handheld device.
[H5]	http://qtek.dk/	The site of the Qtek phone/PDA hybrid.

Appendix A – J2ME and PocketPC

A.1 Overview of J2ME CDC profiles

This is a brief overview of the different profiles that are available for the J2ME CDC configuration. Currently J2ME CDC supports three different profiles; Foundation profile, personal basis profile and personal profile [SUN2].

- **Foundation profile:**
  - J2SE technology-based class library
  - No GUI support
  - CLDC 1.0 compatibility library
  - javax.microedition.io

- **Personal basis:**
  - Lightweight component support
  - xlet support
  - Foundation Profile APIs
  - java.awt
  - java.beans
  - java.rmi
  - javax.microedition.xlet

- Personal profile:
  - Full Abstract Windowing Toolkit (AWT) support
  - Applet support
  - Migration path for the PersonalJava application environment
  - Personal Basis Profile APIs
  - java.applet
  - java.awt
  - java.awt.datatransfer

In addition to these profiles, a number of additional packages have been proposed and specified, including the RMI Optional Package and the JDBC Optional Package.

A.2 PocketPC2000 link file

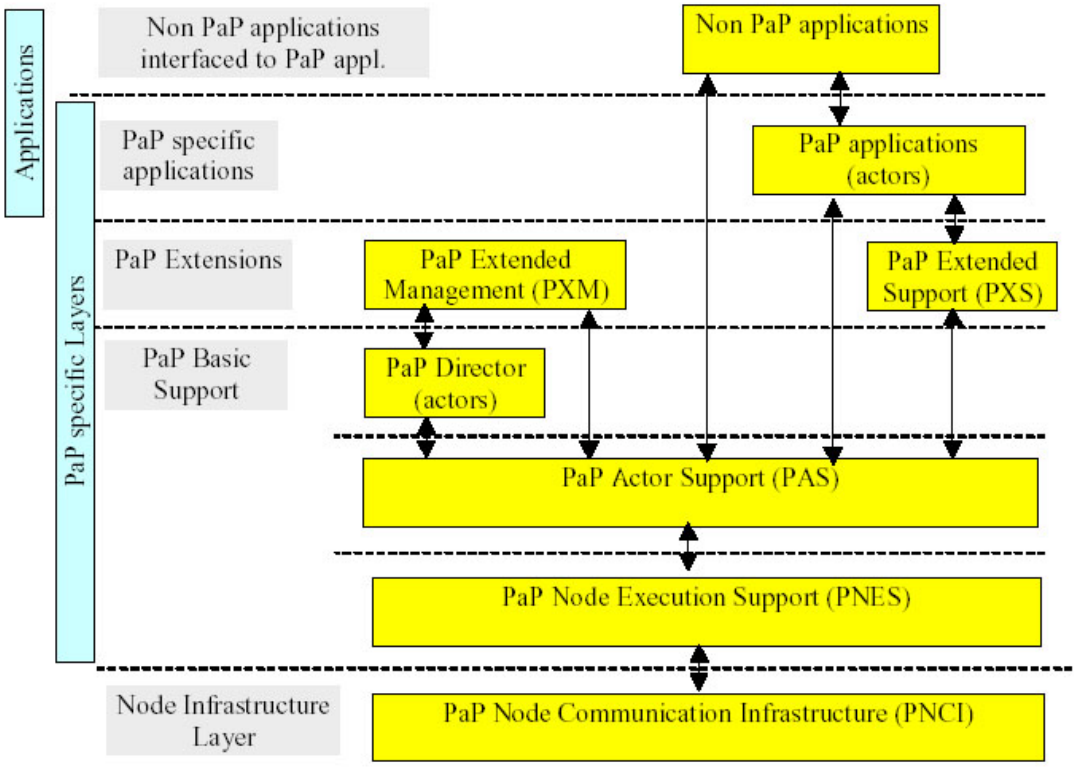
This is an example link-file for the PocketPC, and is very similar to a .bat file in DOS. Normally all the text would appear in one line, but has been wrapped here. The first argument of the file; '300#' is meant to declare to the OS how many characters there are in the line, however, one can have fewer characters than you declare (during testing, no effect was seen when having more characters than declared). There is a limit of 255 characters, and by setting the value to 300 no error will occur, and one do not have to count the number of characters each time one edits a file. The rest of the arguments are specific for the J9 program. The first of these starts the VM, then follows a long argument which specifies what Java libraries to include, then the starting of the actual MicroTAPAS application.

```
300#"\j9\bin\j9.exe" "-
Xbootclasspath:\j9\lib\jclFoundation\classes.zip;\j9\lib\jclFoundation\locale.zi
p;\j9\lib\jclPPro\ppro-ui-win.zip;\j9\lib\jclCdc\classes.zip;\dev"
"MicroTAPAS.startMicroTAPAS" "MicroTAPAS.MicroPNES"
```

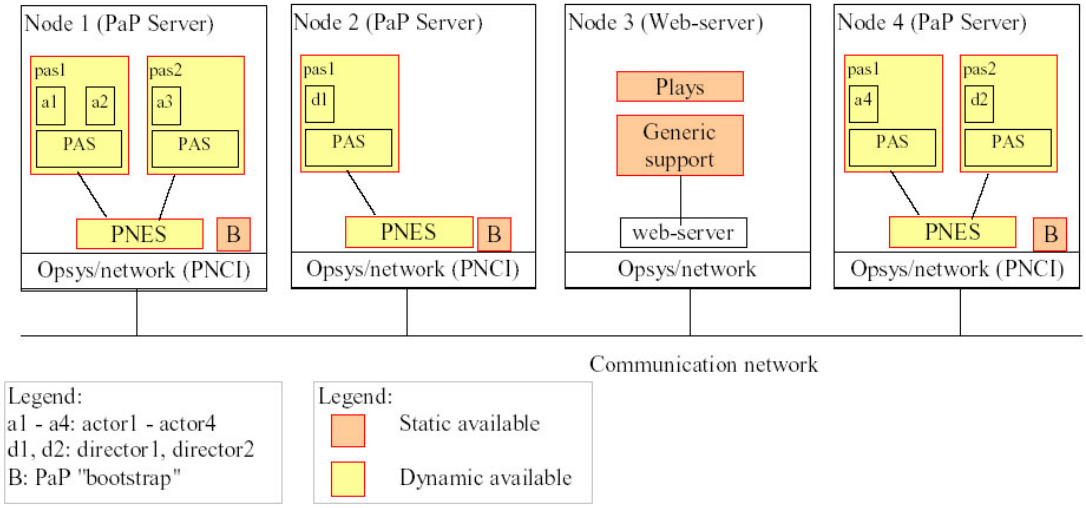
Appendix B - TAPAS

B.1 TAPAS architecture

The layered design model – Architecture [MELH2, page 15]:

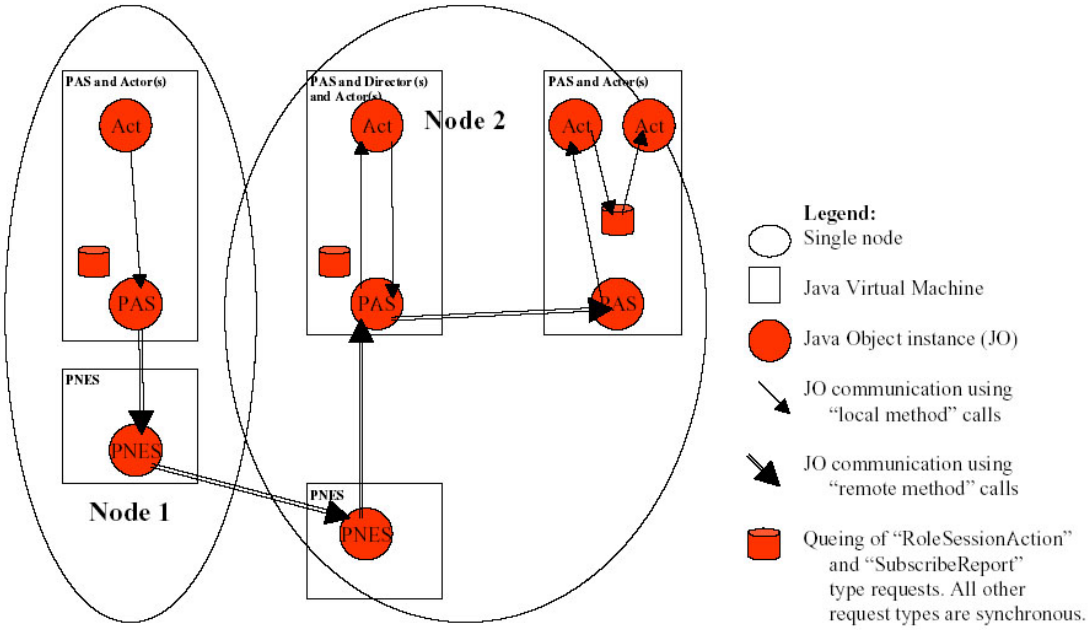


The layered design model – Operating TAPAS system example [MELH2, page 18]:



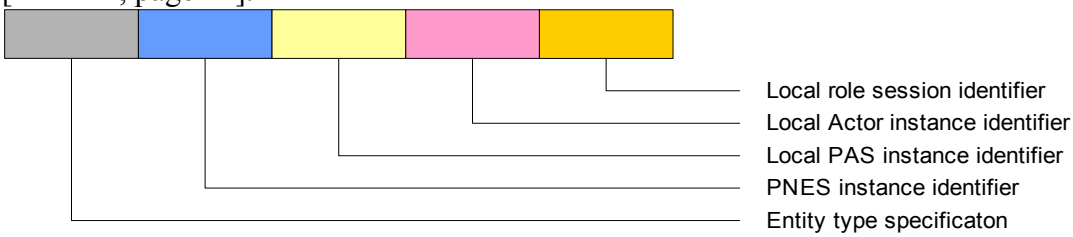
B.2 TAPAS communication model

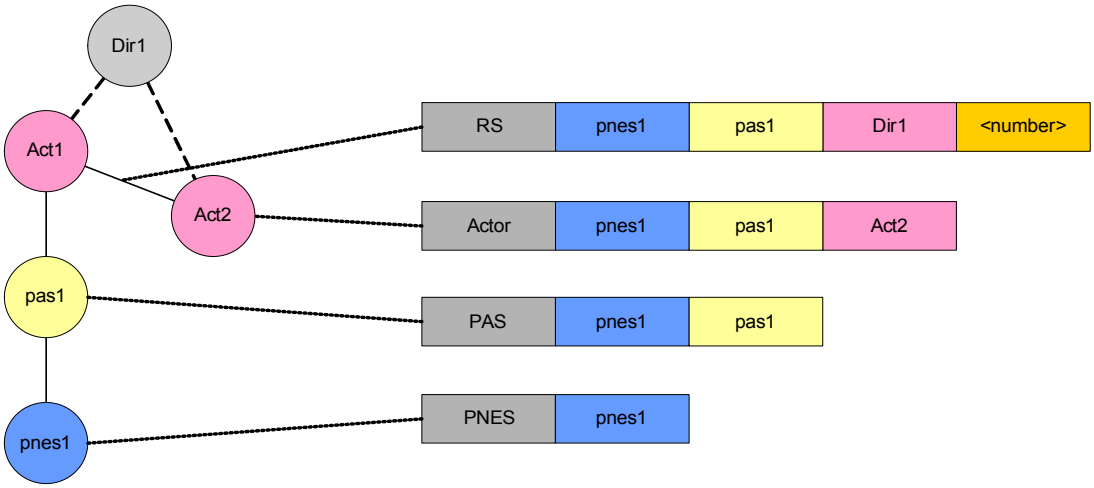
This is the old TAPAS communication model [MELH2, page 20].



B.3 TAPAS addressing

In the original version of TAPAS, the Global Actor Identifier (GAI) was built up like this figure shows, and the difference to MicroTAPAS is only slight, as the PAS instance identifier has been replaced with a copy of the PNES instance identifier [MELH2, page 21].





B.4 Configuration files

The old configuration file

This is an example of how and old version of the configuration file may have looked like. No empty lines or comments were allowed here.

```
codebase = http://127.0.0.1/tapasroot/
policy = http://127.0.0.1/tapasroot/policy
homeinterface = Actor://127.0.0.1/MicroPNES/MicroTAPAS.MicroDirector1
debugserver = localhost
nodeprofile = profMicro
```

The new configuration file

This file is an example of how the configuration file may look like using the updated functionality of MicroTAPAS, and as can be seen from the file, readability is greatly improved over the previous format.

```
# Now it is safe to add comments, prefixing each line with a hash '#'.
# Empty lines are also ok.

# The Internet address of the binary files of the whole system.
# The address should be on the form:
#  http://<ip.addr or url>/<tapas directory>/
codebase = http://10.0.0.1/tapasroot/

# Not in use anymore - will be deleted
policy = http://10.0.0.1/tapasroot/policy

# The GAI (address) of the Director
homeinterface = Actor://10.0.0.1/MicroPNES/MicroTAPAS.MicroDirector1

# The location of the debugserver
debugserver = localhost

# What profile this client is running, could be profMicroPDA or
# profMicroDSK could also be automatically detected by MicroPNES
nodeprofile = profMicro

# Ping-server port: must be the same for all clients
# Interval between pings, in milliseconds
# Port for incomming requests: must be the same for all clients
# Port for outgoing results: must be the same for all clients
# Port range for use by incomming results
# Is debug info to be printed to screen?
debug = true
```

Appendix C – Setup and Testing

C.1 Hardware used

The following table show in detail the hardware used for testing the architecture.

	Node A	**Node B**	**Node C**
Classification	LAPTOP	PDA	LAPTOP
Model	IBM Thinkpad 390E	iPAQ H.3660	HP Pavillion ze5244
Processor	Intel Pentium II, 266 MHz	Intel StrongARM, 206 Mhz	Intel Pentium 4 M, 2,53 GHz
Memory (RAM/ROM)	192 MB/x	64 MB/16 MB	512 MB/x
Operating System (OS)	Microsoft Windows XP, Professional Edition	Microsoft PocketPC 2000	Microsoft Windows XP, Home Edition
Network	SMC	ZyAIR B-100, IEEE 802.11b compliance at 2.4 GHz, PCMCIA card	LAN-Express IEEE 802.11 PCI Adapter
Virtual Machine (VM)	Sun Microsystems Inc's Java HotSpot(TM) Client VM, version 1.4.1_02-b06	IBM's J9, version 2.0	Sun Microsystems Inc's Java HotSpot(TM) Client VM, version 1.4.1_02-b06

C.2 Test data from HeapTest

Screen output from program execution

The following data was printed to screen during one of the test runs, and, as can be seen on the top, it was started with five threads and two cycles. All the times quoted are in milliseconds. The first column states the number of threads used to execute the tasks, the other three columns show the different tasks, as explained in section 6.3.1.

```
C:\dev\MicroTAPAS>java heaptest.HeapTest 5 2
Unable to open log file. Printing to System.out...

Max # threads =      5
Total (heap + CPU) cycles = 2


# Threads       2 Heap, 0 CPU   1 Heap, 1 CPU   0 Heap, 2 CPU

1       21981   20409   15161

2       20420   14901   15863

3       31085   17125   10254

4       20480   12858   11396

5       19718   14441   12718
```

Data from performance testing of laptop vs. PDA

This is the data values from the performance testing performed in section 6.3.1. The top-left heading in each table states which terminal and start parameters were used. For example, IBM, 5-2 states that it was run on the IBM laptop (node A) with a maximum of five threads and two cycles. Figure 6.3 was drawn by computing the average execution time for each task (i.e. the first value for 'IBM, 5-2' in the figure was found by summing the column '2 Heap, 0 CPU' and dividing by the number of threads; five).

IBM, 5-2	Threads	2 Heap, 0 CPU	1 Heap, 1 CPU	1 Heap, 1 CPU
	1	21981	20409	15161
	2	20420	14901	15863
	3	31085	17125	10254
	4	20480	12858	11396
	5	19718	14441	12718

HP, 5-2	Threads	2 Heap, 0 CPU	1 Heap, 1 CPU	1 Heap, 1 CPU
	1	4046	2844	1422
	2	3706	2613	1422
	3	3325	2364	1412
	4	3264	2434	1422
	5	3044	2173	1422

PDA, 5-2	Threads	2 Heap, 0 CPU	1 Heap, 1 CPU	1 Heap, 1 CPU
	1	64285	258435	419323
	2	80407	239302	403596
	3	87770	238447	403535
	4	93045	237556	404077
	5	93945	238113	404354

IBM, 2-2	Threads	2 Heap, 0 CPU	1 Heap, 1 CPU	1 Heap, 1 CPU
	1	19197	13920	9193
	2	17705	11467	9193

HP, 2-2	Threads	2 Heap, 0 CPU	1 Heap, 1 CPU	1 Heap, 1 CPU
	1	4096	2894	1432
	2	3605	2574	1412

PDA, 2-2	Threads	2 Heap, 0 CPU	1 Heap, 1 CPU	1 Heap, 1 CPU
	1	62779	246339	406098
	2	81344	241102	406206