Wireless telephony applications : a study to the possibilities of WTA

de Vries, P.M.

Award date:
2001

TECHNISCHE UNIVERSITEIT EINDHOVEN

Department of Mathematics and Computing Science

MASTER'S THESIS

Wireless Telephony Applications

A study to the possibilities of WTA

by

P.M. de Vries

June 2001

Supervisor: prof. dr. P.M.E. De Bra (Technische Universiteit Eindhoven)

Advisor(s): P.W.J. Essers BSc (Ericsson)
dr. M. Voorhoeve (Technische Universiteit Eindhoven)

Prepared (also subject responsible if other)		No.		
ETM/BL/IB Peter de Vries		ETM/BL/IB-01:0117 Uen		
Approved	Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers		12/06/01	A	H:\etmptvr

Prepared (also subject responsible if other)	No.		
ETM/BL/IB Peter de Vries	ETM/BL/IB-01:0117 Uen		
Approved / Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers	12/06/01	A	H:\etmptvr

Abstract

This graduation report is the result of a study into the possibilities of the Wireless Telephony Application (WTA) specification, a part of the WAP specification that makes it possible to use in-device telephony related functions in a WAP context. To find these possibilities first the WTA specification, the current situation, the future and alternative technologies are studied. With these results a number of potentially useful applications is discussed and finally a prototype is designed and implemented.

Prepared (also subject responsible if other)		No.		
ETM/BL/IB Peter de Vries		ETM/BL/IB-01:0117 Uen		
Approved	Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers		12/06/01	A	H:\etmptvr

Table of Contents

Foreword

This report is written for my graduation at the Eindhoven University of Technology (official name: Technische Universiteit Eindhoven). I study Computing Science at the department of Mathematics & Computing Science and graduate at the group of Information Systems. During my study I always wanted to graduate externally at a company. Because of my interest for mobile telephony, I contacted Ericsson Telecommunicatie b.v, which offered me the opportunity to do this study.

During the last nine months that I spent at Ericsson, I learnt a lot about the telecom world. When I started at Ericsson it felt like I knew nothing. All the abbreviations that are common in the telecomworld did not say me anything at all. It felt like I was drowning in all the information that was new for me. Now, at the end of this period, I know much more about telecommunication, but I can still say that there are so much interesting things to read. Besides my personal grow with respect to content I have also learnt a lot about Ericsson as a company, as an employer. I have for example experienced the creation of a new organization, but also the uncertainty as a result of cost reductions. I can use all these experiences in my further career.

I would like to thank my coach at Ericsson, Patrick Essers and my coach at the university, Paul De Bra for their support during my graduation. Furthermore I would like to thank my colleagues at Ericsson for their help about telecom issues. And last but not least I would like to thank my parents for their mental and financial support that made my study possible.

Rijen, June 2001

Peter de Vries

Prepared (also subject responsible if other)		No.		
ETM/BL/IB Peter de Vries		ETM/BL/IB-01:0117 Uen		
Approved	Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers		12/06/01	A	H:\etmptvr

Prepared (also subject responsible if other)
ETM/BL/IB Peter de Vries
Approved
ETM/BL/IB Patrick W.J. Essers
Checked

No.
ETM/BL/IB-01:0117 Uen
Date
12/06/01
Rev
A
Reference
H:\etmptvr

Executive Summary

This report is the result of a graduation study to the Wireless Telephony Application specification. The objective of this study is to investigate the possibilities of WTA and to build a prototype application that uses this framework. To reach this goal, the following subgoals are defined:

- What is WTA?
- What are the alternative techniques for WTA?
- What is the current situation with respect to support of WTA?
- What can be expected from the future with respect to WTA?
- Which applications are possible with WTA?

WTA is a telephony extension to the WAP specifications that can provide in-device telephony related control mechanisms. With WTA it is possible to develop services that manipulate mobile unit features and trap various events. To make this possible, WTA extends the Wireless Application Environment with:

- An interface from WTA-WML and WMLScript to a specific set of local, telephony related functions in the client.
- Network event handling, such that events from the mobile network can be detected and actions in response to these events can be defined.
- A repository, which is a storage container that persistently stores content that executes WTA-services.

Because of the nature of the applications possible with WTA, WTA defines also a model for the user-agent state, WTA context management and a mandatory security model.

There are no direct replacement alternatives for WTA. However, two techniques can be marked as related: the SIM Application Toolkit (SAT) and the Mobile Execution Environment (MExE). The SAT defines the interface between applications on the SIM card in the mobile phone and the mobile phone itself. The SAT is, because of a more limited execution environment and less functionalities, more limited that WTA. MExE on the other hand offers a much richer set of functions that WTA and allows full application programming using Java.

At this moment the latest approved WTA specification is part of the WAP June 2000 conformance release. But the WTA specification is an optional part of the WAP specification. As a result of this none of the currently available terminals support WTA. The WAP Gateway and WTA server that are necessary to constitute the WTA architecture are currently available.

In the future the mobile networks will evolve to the so-called third generation networks. The most important changes are the step from circuit-switched to packet-switched networks and the increased bandwidth. These new networks offer the possibility to have a data and voice connection simultaneously. This makes new applications possible. The WAP specification is still evolving and the introduction of WAP 2.0, probably at the end of June 2001, introduces some improvements. With regards to the availability of terminals that support WTA the future does not look very bright.

During this study a number of potentially useful applications, that use the WTA technique, is investigated. At first sight there are some very interesting applications. However most of these applications can not be implemented as a result of one or more limitations. These limitations can be limitations of the network used, like for example the limitation of using only one connection at the time and the limitation that some functions are not implemented. Other limitations are limitations of the WAP architecture, like the limited user interface and the lack of some kind of file storage mechanism. Finally there are limitations that are limitations of the WTA specification itself, like the lack of an authentication mechanism between the gateway and the server.

One of the objectives of this study is the implementation of a prototype. For this study the 'I See You' (ICU) application is chosen as prototype. The idea behind this application is that a picture of the calling person is displayed on an incoming call. Therefore users can create a contactlist on the ICU server and store pictures in this contactlist. As a result of the lack of WTA terminals, this prototype does not work as a whole, but gives a good impression of the possibilities.

The conclusion that can be drawn from this study is that WTA offers some interesting opportunities, but unfortunately there are a lot of limitations. At the moment of writing these limitations make the development of WTA services impossible. Some of these limitations will probably be removed in future versions, but for the time being it does not seem that WTA will be successful.

1 Introduction

This report is the result of a graduation study that has been carried out at Ericsson. This section gives an introduction to this study. First an introduction to Ericsson is given. Then the objective of this study is described followed by the approach. Finally the structure of this report is described.

1.1 Ericsson

Ericsson is the world's leading supplier in telecommunications with the largest customer base, including the world's top 10 operators. Ericsson provides total solutions covering everything from systems and applications to mobile phones and other communications tools. Since 1876, Ericsson has been active worldwide and today operates in more than 140 countries.

This study has been done at the unit Business Line Internet Applications and Solutions. This unit has the primary goal to support customers (these can be operators, service providers and corporate markets) with technical knowledge about Internet applications and solutions. The focus is not only on the Dutch market, but also on the region like Belgium and the United Kingdom.

1.2 Objective of this study

A growing number of mobile telephones are nowadays equipped with the option to access the Internet by using the Wireless Application Protocol. But up till the time of writing the mobile Internet functionality and the ordinary speech functionality are living as neighbors side by side in one device. A part of the WAP specification foresees in the option to combine these functionalities: the Wireless Telephony Application specification.

This report will focus on the WTA part of the WAP specification. The WTA framework supports wireless telephony applications that interface with the in-device telephony related functions and the network telephony infrastructure. A simple example of a such a wireless telephony application is the initiation of a call by clicking on a hyperlink in a WML page.

The objective of this study is to investigate the possibilities of WTA and to build a prototype application that uses this framework.

1.3 Approach

To reach the goal as described above this study is divided into three parts with for each part a subgoal. A visualization of the approach is given in Figure 1. The goal of the first part is to find out what the characterizations of WTA are. To achieve this goal the WAP and especially the WTA specification are studied. The results of the first part form the basis of the second part.

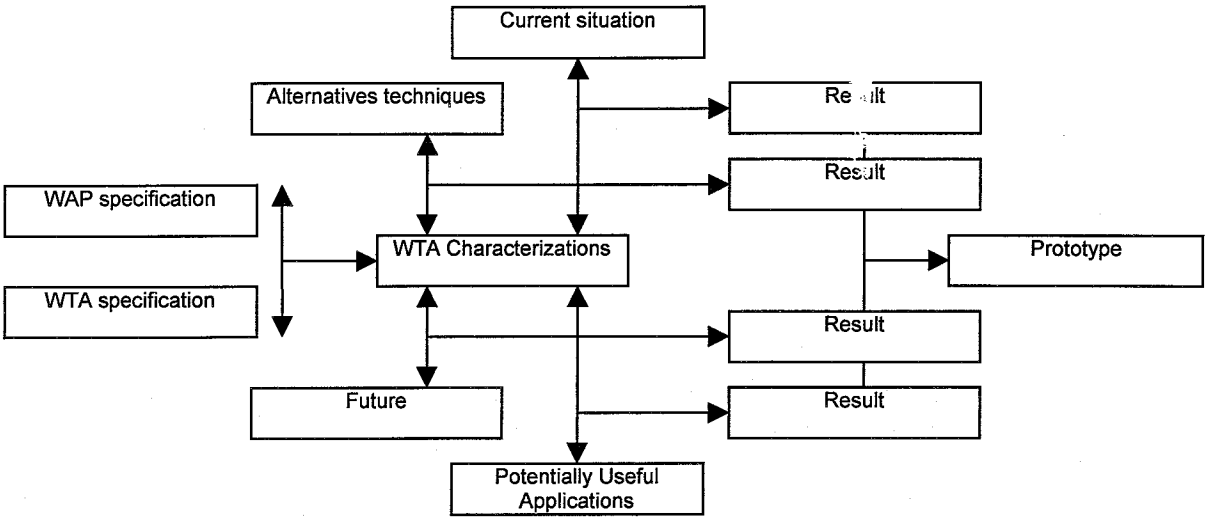


Figure 1. Visualization of the approach of the study

The goal of the second part is to investigate the applicability of WTA. To achieve this goal, four subgoals are defined:

- What are the alternative techniques for WTA?
- What is the current situation with respect to support of WTA?
- What can be expected from the future with respect to WTA?
- Which applications are possible with WTA?

The results of these four subgoals form both the answers on the first part of the main goal and the basis of the third part, the prototype.

The goal of the third part of this study is to build a prototype application that uses the WTA framework and demonstrates some of the possibilities of WTA. Besides this goal, the implementation of the prototype is used to find potential problems when actually implementing a WTA application.

1.4 Structure of this report

The structure of this report is based on the subgoals as described above. First of all section 2 introduces the reader to WTA. Then in section 3 the alternative techniques are discussed. Sections 4 and 1 respectively discuss the current situation and what can be expected from the future. In section 1 a survey of potentially useful applications using the WTA framework is given. Section 7 describes the prototype that is built during this study and finally the conclusion can be found in section 8.

2 Wireless Telephony Application

This section introduces WTA. In the first subsection a very short introduction to WAP is given. The following subsections describe the WTA specification. A more detailed description can be found in appendix A.

2.1 The Wireless Application Protocol

The Wireless Application Protocol (WAP) is a protocol developed by the WAP Forum, an industry association comprising over 500 members that has the primary goal to bring together companies from all segments of the wireless industry value chain to ensure product interoperability and growth of wireless market. WAP specifies an application framework and network protocols for wireless devices such as mobile telephones, pagers and personal digital assistants.

The WAP architecture provides a scaleable and extensible environment for application development for mobile communication devices. This is achieved through a layered design of the entire protocol stack. The upper layer, the Wireless Application Environment (WAE) is a general-purpose application environment based on a combination of World Wide Web (WWW) and mobile telephony technologies. The primary objective of the WAE is to establish an environment that allows operators and service providers to build applications and services that can reach a variety of different wireless platforms in an efficient and useful manner. The WAE includes a micro-browser environment containing the following functionality:

- Wireless Markup Language (WML) – a lightweight markup language, similar to HTML, but optimized for use in hand-held mobile terminals.
- WMLScript – a lightweight scripting language, similar to JavaScript.
- Content Formats – a set of well-defined data format, including images, phone book records and calendar information.
- Wireless Telephony Application (WTA) – telephony services and programming interfaces.

2.2 What is WTA?

Wireless Telephony Application (WTA) is a telephony extension to the WAP specifications that can provide in-device telephony related control mechanisms to advanced mobile network services. With WTA it is possible to develop services that manipulate mobile unit features (e.g. call control and the phonebook), and trap various events (e.g. incoming call).

The WTA framework extends the Wireless Application Environment by adding [18]:

- An interface from WTA-WML and WMLScript to a specific set of local, telephony related, functions in the client. This interface is called the Wireless Telephony Applications Interface [16].
- Network event handling. This means that events originating from the mobile network could be detected by the WTA user-agent and actions in response to the events could be defined.
- A repository, which is a storage container, used by the WTA user-agent, that persistently stores content that executes WTA-services. The purpose of the repository is to fulfill the real-time requirements that are placed on the execution of WTA services.
- A model for WTA user-agent state and WTA context management.
- A mandatory security model.

2.3 Architectural Overview

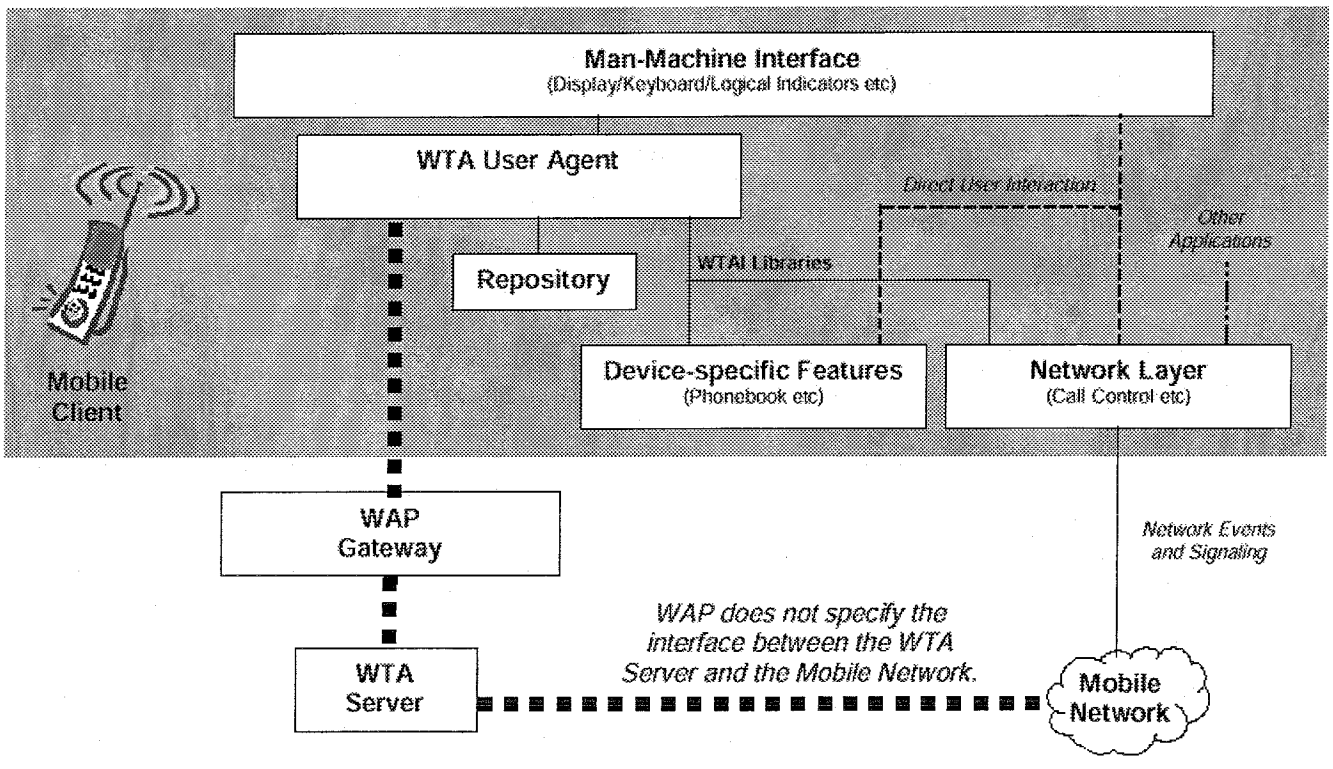


Figure 2. Overview of the WTA architecture [18]

The WTA architecture primarily defines the components required in a client device to make telephony services available to WAP content. It also defines the role played by certain network elements. The components that constitute the WTA framework are:

- WTA user-agent.
- WTA server.
- WTA Interface libraries.
- Repository.

Figure 2 above illustrates how the WTA user-agent, the repository and WTAI can interact with each other and other entities in a WTA-capable mobile client device. The WTA user-agent is able to retrieve content from the repository and WTAI ensures that the WTA user-agent can interact with mobile network functions (e.g. setting up calls) and device specific features (e.g. manipulating the phonebook). The WTA user-agent receives network events that can be bound to content, thus enabling dynamic telephony applications.

2.3.1 WTA User-Agent

The WTA user-agent is a micro-browser, quite similar to a WAE user-agent in that it has the ability to execute and present WML and WMLScript content to a user. In fact, it is a WAE user-agent in almost every respect including executing content within the boundary of a well-defined context. The primary difference is that a WAE user-agent only retrieves its content via the WAP gateway and only has access to the WTAI Public Library functions. These functions expose simple functionality such as the ability to place a call, but do not allow fully featured telephony control. Only a WTA user-agent is able to fully control the telephony features of the device.

2.3.2 WTA Server

A WTA server can be thought of as a web server where WTA content and services are hosted. The primary distinction from any other origin server is that the mobile operator regards it as a 'Trusted Content Server'. The reasons a WTA server must be a trusted server will become clear later in section 2.5.

Like an Internet web browser, a WTA user-agent uses a URL (Uniform Resource Locator) to reference content on the WTA server. A URL can also be used to reference an application on a web server that is executed when it is referenced. A WTA server may also make use of this concept. By referencing applications on a WTA server it is possible to create services that use URLs to interact with the mobile network and other entities (e.g. a voice mail system). This is possible because, as can be seen in Figure 2, the WTA server can have a direct connection to the mobile network. This mechanism provides a powerful model to seamlessly integrate services in e.g. the mobile network with services executing locally in the WAP client.

2.3.3 WTA Interface Libraries

WTAI is a set of function libraries that are accessible either via a WML-deck or WMLScript. As can be seen in Figure 2, services that execute in the WTA user-agent have access to WTAI functions. The WTAI is partitioned in three

function libraries: Public WTAI, Network Common WTAI and Network Specific WTAI.

The public WTAI library contains simple functions that are available to applications executed in both the standard WAE user-agent and the WTA user-agent. These functions always require explicit user confirmation. The public WTAI library contains functions to initiate a voice call, add an entry to the local phonebook and send DTMF (Dual Tone Multi Frequency) tones.

The network common WTAI contains functions that cover the common features that are available in all networks. This interface contains functions to handle calls, handle messages, edit the phonebook and view call logs.

Finally, network specific functions are only available in certain types of networks. For GSM devices for example, the GSM specific library is mandatory. The GSM specific library contains functions to put a call on hold, transfer a call, handle multiparty calls and get GSM location information.

2.3.4 Repository

The repository is a persistent storage module within the mobile terminal that may be used to eliminate the need for network access when loading and executing frequently used WTA services. The repository also addresses the issue of how a WTA service developer ensures that time-critical WTA events are handled in a timely manner.

The repository consists of channels and resources. A channel contains a set of links to resources. Resources are data that have been downloaded to the client. This can be for example a WML-deck or an image.

Only WTA applications (that is, content loaded or otherwise received from the WTA server) may access the repository.

2.4 WTA Services

WTA services are what the end-user ultimately experiences from using the WTA framework. A WTA service appears to the client in the form of various content formats, e.g. WTA-WML, WMLScript etc. The WTA user-agent executes content that is persistently stored in the client's repository or content retrieved from a WTA server. Figure 3 shows the possible ways of initiating a WTA service in the WTA user-agent.

A Service Indication contains a URL for a service that the user can choose to start immediately, or postpone for later handling. A typical example would be an end-user that is notified about an incoming e-mail with an URL that he can choose to open directly or later.

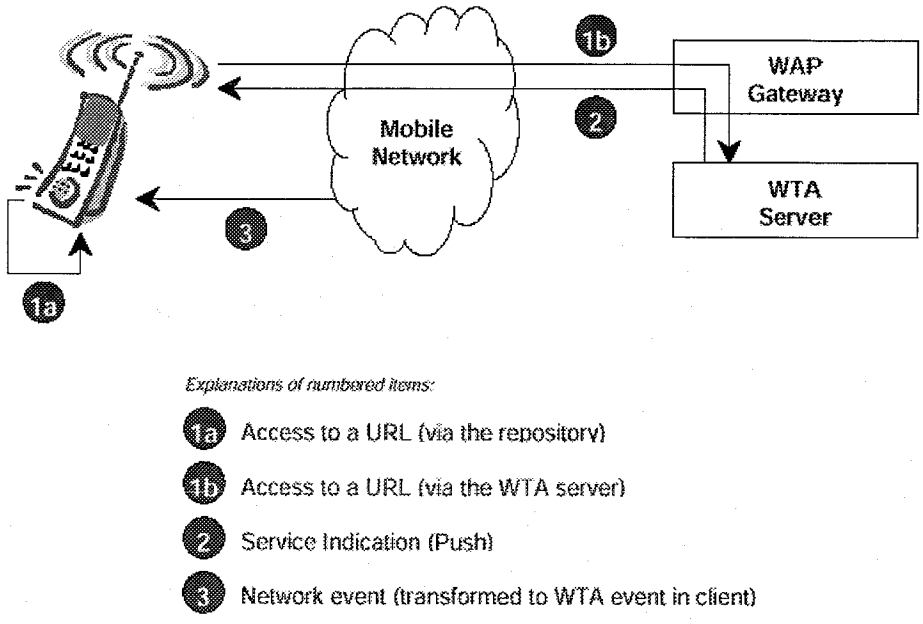


Figure 3. Initiation of WTA services [18]

2.5 WTA Security

Telephony services have a stronger security requirement than other types of WAP services. Because of the nature of the operations possible when executing WTA content it must be ensured that only authorized WTA services are allowed to execute. Therefore the following architectural decisions to ensure security are taken:

- The WTA user-agent uses a secure port on the WAP gateway for the session it needs. Any content received outside a session established over a non-secure port will be discarded.
- The WAP gateway has to ensure a secure link between itself and the WTA server, either using a private network or by means of strong authentication mechanisms.
- The user configures the mobile phone to specify user permissions for the execution of different WTAI library functions. The user can give for example blanket permission or permission to use a function one time.

The duty of providing an acceptable level of security lies with the mobile network operator. It is possible to do this only if:

- The mobile network operator has administrative control over the WTA server or at least has a trust relationship with the provider of the WTA server.
- The mobile network operator controls or at least has a trust relationship with the provider of a WAP gateway that is used by its subscribers.

2.6 WTA Events

Network events are events that originate from the mobile network. Network events can be transformed from their abstract form into a predefined format called WTA events.

The WTA user-agent can be programmed to take certain actions when receiving a specific WTA event. It is for example possible to display a page on an incoming call event. WTA events can be associated with any content in the repository (global binding) or associated with actions inside a page (temporary binding). The network common library contains events for call control, like an event for an incoming call, incoming messages and network status changes. The GSM specific library contains some events for call waiting and receipt of an unstructured supplementary service data (USSD) message.

2.7 Characteristics

In the preceding subsections the WTA architecture has been discussed. This discussion can be summed up in the following WTA characteristics, which will be used in the following section.

The WTA architecture offers the following characteristics:

- **Call control** – the WTA user-agent can fully control calls. This includes the basic functions like making and receiving voice calls and advanced, GSM specific functions like holding, transferring and controlling multiparty calls. The network information functions are also part of this characteristic.
- **Messaging control** – the WTA user-agent can handle messages. The WTA user-agent is capable of sending and receiving short messages (SMS) and USSD.
- **Feature control** – the WTA user-agent can use device-related functions like editing the phonebook, editing the call log list.
- **Event handling** – the WTA user-agent can react on events originated from the mobile network like an incoming call alert.

3 Alternative Techniques

There are no direct replacement technology alternatives for the WAP standard, especially the WTA standard. There are however two related technologies: the SIM Application Toolkit (SAT) and the Mobile Execution Environment (MExE). Both techniques are discussed shortly here and in more detail in appendices B and C.

3.1 SIM Application Toolkit

3.1.1 Overview

As its name suggests, the SIM Application Toolkit (SAT) is based around the use of a smart card, the Subscriber Identity Module (SIM) card in a GSM phone. The SAT is a protocol that enables an application running on a SIM to interact with a mobile device. The SAT enables operators to send applications over the air as SMS or broadcast messages to update SIM cards with adapted or new services. Because with SAT the application is implemented in a SIM, this provides an even more limited environment than the mobile device.

The SAT is part of the GSM standard and specifies the interface between the SIM and the mobile phone. This means that the SAT is independent of phone manufacturers and designs and that the design of the SIM itself is not mandated either, only the interfaces. The SAT defines a set of fairly simple operations for 'non-smart' mobile phones. It provides mechanisms that allow SIM card-based applications to interact with any mobile phone that supports the mechanisms. The SAT specification defines the mechanisms described below:

- **Profile Download** - the profile download mechanism allows the mobile phone to tell the SIM what SAT facilities it is capable of supporting, so that the SIM can limit its instruction range accordingly.

- **Proactive SIM** - the SIM can inform the mobile phone that it has some information or commands for the mobile phone to carry out, which the mobile phone can then fetch. Such actions include displaying text from the SIM to the mobile phone, sending a short message, setting up a voice or data call to a number held in the SIM and so on.

- **Menu Selection** - the menu selection command allows a set of possible menu options to be supplied by the SIM using a proactive command. If the user subsequently uses the mobile phone keypad to select a menu option, the mobile phone informs the SIM using this mechanism.

- **Call Control** - when this service is activated by the mobile phone, all call set-up attempts will result in the telephone numbers, supplementary services and USSD strings being sent first to the SIM. The SIM can then decide whether to allow those actions to be carried out or can selectively bar them.

Support of the SAT is optional for mobile phones. Any mobile phone claiming to support SAT needs not to support all functions but must support a specified subset.

3.1.2 SAT versus WTA

It is not easy to compare SAT with WTA because of the different techniques. It also depends on the application which technique is the preferred one. In Appendix B the SAT and WTA are compared in more detail. Below some advantages and disadvantages are described to position SAT versus WTA.

A disadvantage of the SAT is the closed system issue. Because there are many different classes of the SAT protocol not all mobile phones allow all applications. And because of the vendor specific implementations of the SIM with proprietary solutions, applications have to be built according to each type of card. The WAP environment provides a web-centric/thin-client environment with a standardized framework, which is easier to manage and maintain. There are moves to address this interoperability issue.

The SAT offers an advantage in the security area. But WAP is making up ground. The current WAP specifications include enhanced security features for the Wireless Transport Layer Section (WTLS). As this is outside the scope of WTA this item will not be discussed here.

At this moment there are more handsets that support SAT than handsets that support WAP. This means that SAT has a bigger installed base, which is an advantage. But this advantage depends on how long it will take WAP-enabled mobile devices to penetrate the market. More and more vendors start selling WAP-enabled mobile devices. Ericsson for example has announced that from 2001 all new mobile devices will be WAP-enabled.

The last disadvantage of SAT is the limited bearer. The SAT only supports SMS, while WTA is independent of the bearer.

As said in the beginning of this subsection, the SAT is not a real alternative for WTA. It depends on the application, which of the two techniques is more suitable. Services that remain stable and that need lot of security could be run on a card. But given the limitations of the bandwidth, the limited run time environment, the closed system item and the smaller functionality, WAP with WTA offers more functionality.

3.2 Mobile Execution Environment

3.2.1 Overview

The Mobile Execution Environment (MExE) is a 3GPP standard [1] that specifies the application environment in mobile phones, allowing applications to run in a browser environment or as a stand-alone application in the mobile terminal.

The MExE standard makes it possible to execute operator or service provider specific applications. It offers the ability to deliver such content from a

standardized application platform giving access to a range of common services across a variety of MExE compatible handsets. These MExE handsets have a fully embedded application execution environment with varying levels of functionality that are defined using a system of classmarks. At the moment of writing there are three classmarks specified. Capability negotiation is used to determine the classmark of the mobile station. For classmark 1 or WAP MExE the MExE standard references to the WAP specifications. So a MExE classmark 1 compliant terminal is basically a WAP enabled terminal.

MExE classmark 2 supports PersonalJava and the JavaPhone API. Personal Java [12] is a subset of the Java 2 Platform, Standard Edition (J2SE) specification and is optimized for use with consumer electronic devices. It allows aspects such as the user interface to be personalized. JavaPhone contains a range of APIs that could be built into a Personal Java device including telephony parameters, call control, calendaring, address book management and messaging.

MExE Classmark 3 supports the Java Virtual Machine, which is part of the Java 2 Platform Micro Edition (J2ME) architecture. It also includes support for the Connected Limited Device Configuration (CLDC) and the Mobile Information Device Profile (MIDP) Java. Classmark 3 is capable of supporting small untrusted applications being delivered to the terminal [9].

Within the specifications there are guidelines on the design of applications and it is strongly recommended that applications are designed to be backward compatible to earlier versions of classmarks.

3.2.2 MExE versus WTA

In appendix C a more detailed comparison is made between MExE and WTA. Here the outcome of this comparison will do.

MExE offers a much richer set of functions for application programming than WTA. Both classmarks 1 and 2 cover all the WTA functionality. MExE classmark 3 devices do not support telephony functions according to the standard. Whereas WAP incorporates some scripting, graphics, animation and text, MExE allows full application programming which requires significant processing resources on the mobile client. That's why MExE is primarily aimed at the next generation of powerful smart phones.

According to [9], the first implementations of the MExE standard have taken place in Korea where the global manufacturers LG have already delivered a J2ME compliant handset for CDMA phones and are currently working on GSM devices. The CDMA device can download Java applications over the air and is working commercially in Korea today. Other Java capable devices are expected from Motorola shortly and the new Nokia 9210 Communicator will support the Java Virtual Machine.

Prepared (also subject responsible if other)		No.		
ETM/BL/IB Peter de Vries		ETM/BL/IB-01:0117 Uen		
Approved	Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers		12/06/01	A	H:\etmptvr

4 Current Situation

In this section the situation with respect to WTA at the time of writing is discussed. As the information provided below is constantly subjected to changes, this section only provides a snapshot.

4.1 Specification

The WTA specification as described in section 2 is part of the WAP June 2000 Conformance release, also denoted as WAP 1.2.1. Prior to the June 2000 release three other WAP Specifications Suites have been released: WAP 1.0, 1.1 and 1.2. The WAP 1.1 release is the first release for which a certification program has been defined. Although the WTA specification exists in the WAP 1.0 and 1.1 release, the WAP Forum explicitly recommended in their Specification Policy Statements that the WTA 1.1 specification should not be used for commercial deployments.

The WAP 1.2 specification suite is the first release that supports WTA functionality. However, this release has been updated to the WAP June 2000 release before the WAP 1.2 Certification Documents have been written. The WAP Forum has decided not to make these documents for the WAP 1.2 release but to make these directly for the WAP June 2000 release.

The WAP June 2000 specification suite is the latest approved specification at the moment of writing. This suite contains the WTA specifications as described in section 2 and will most likely be supported in future products. This specification is not backward compatible with the prior WTA specifications, but as it is explicitly recommended by the WAP Forum not to use the prior specifications for commercial deployment, this should not lead to big problems.

According to the WAP Class Conformance Requirements [15], three device class profiles have been defined on the basis of cross level functionalities: A data profile class (or Class C) client or server provides all mandatory data features of the WAP specification. A data and telephony profile class (or Class B) client or server device provides all mandatory data and telephony features of the WAP specifications. A complete data and telephony profile class (or Class A) client or server device provides all mandatory and all optional data and telephony features of the WAP specifications.

From a client or server device customer perspective, these profiles provide the high-level functionality indication that "data" or "data and telephony" based applications are supported or not. According to the WAP June 2000 specification WTA functionality is only mandatory for the Class A and Class B devices, and optional for Class C devices.

4.2 Mobile Client

At this moment all commercial available mobile telephones are WAP 1.1 compliant. As a result of the recommendation not to use the WTA 1.1 specification, the current commercial available mobile phones do not support

WTA functionality. Even phone emulators that support WTA do not exist yet. Some vendors however have implemented the public WTAI functions. For example the Ericsson R320 supports the public `makeCall` function. However, strictly following the specification, WTA is not supported.

At this moment the only known Ericsson mobile device that is WAP June 2000 compliant is the Ericsson R520 GPRS telephone. But unfortunately this is a class C device and therefore it does not support WTA. At the moment of writing Ericsson does not have any class B or A devices.

Competing mobile terminal vendors also do not sell mobile devices supporting WTA functionality yet.

4.3 WAP Gateway

The current available Ericsson WAP Gateway/Proxy (WGP 3.0) complies with the WAP 2000 June specification and supports WTA applications. Competing vendors of WAP gateways, like Nokia, CMG and Openwave do not support WTA yet.

4.4 WTA Server

The current generally available Ericsson WAP Application Server (2.0) supports WTA functionality. The WAP Application Server provides the possibility to support development of WTA applications. The user can download the WTA applications to the terminal from the WAP Application Server menus. It is also possible to derive from the WAP Application Server which application a user has downloaded. When new versions have been deployed on the system the system administrator can choose to send an SMS to users indicating that there are newer versions available on the server. The user can then connect to the WAP Application Server and retrieve the new version. The WAP Application Server also includes some WTA applications for demonstration purposes [4].

5 The Future

It is always difficult to talk about the future, but nevertheless in this section some expected important changes are described. Probably the most important change that's ahead is the change from circuit- to packet-switched mobile networks. This year, mobile operators will start using the General Packet Radio System (GPRS) and within two years the much-discussed UMTS systems will be operational. These new systems make new applications possible by offering a higher bandwidth. In this section these new technologies are discussed in short.

Besides these new technologies also the future of the WAP and WTA specification and the future of the terminals are discussed.

5.1 Networks

As written above, the most important change from the so-called 2G to 3G networks, is the change from circuit- to packet-switched networks. At this moment, telephone networks are circuit-switched. This means that for every connection a complete path is set up from the calling party to the called party independent of the amount of data transferred. The caller is charged for the amount of access time. With packet-switched networks, data is transferred through the network in packets and the sender of the packets is charged depending on the number of packets sent. The result of this is a more efficient network usage and the possibility for users to be logged into the network for long periods and only to be charged when they make data transfer. So users will be always online. Because no connection has to be set up, users will be able to enjoy fast access and immediate availability of services.

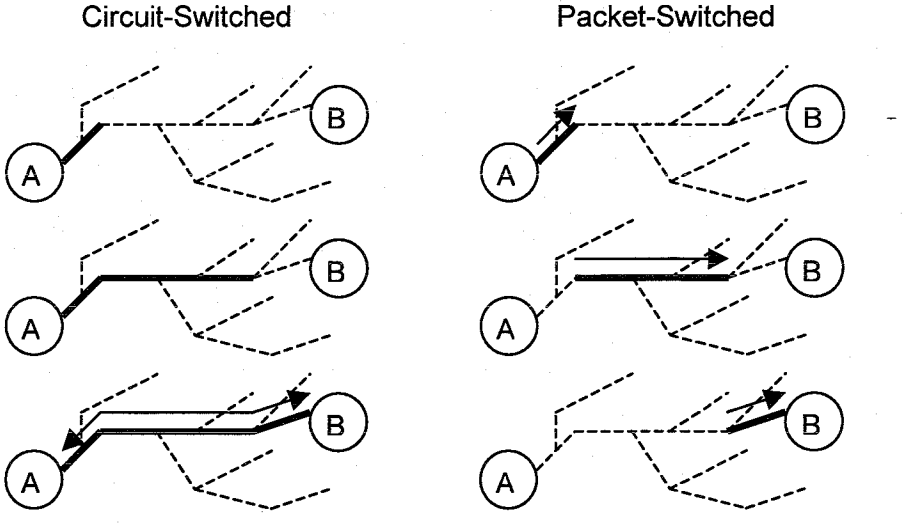


Figure 4. A Circuit-Switched versus a Packet-Switched connection

5.1.1 GPRS

General Packet Radio Services (GPRS) is a standardized packet-switched dataservice for GSM enabling mobile use of Internet. The GPRS system

provides a basic solution for Internet Protocol (IP) communication between mobile stations and Internet Service Providers (ISP) or a Local Area Network (LAN). GPRS establishes an end-to-end IP connection from the mobile network terminal to the servers at the ISP. The packet data transmission is thus carried out on an end-to-end basis, including the air interface.

GPRS users can remain online without continuously occupying a specific radio channel. GPRS will use the common pool of physical resources across the radio network in co-existence with the existing circuit switched GSM. The same physical channels will be used but in a more efficient way since several GPRS users will be able to share one channel, thus giving a better channel utilization. In addition, GPRS channels are allocated only when data is sent or received.

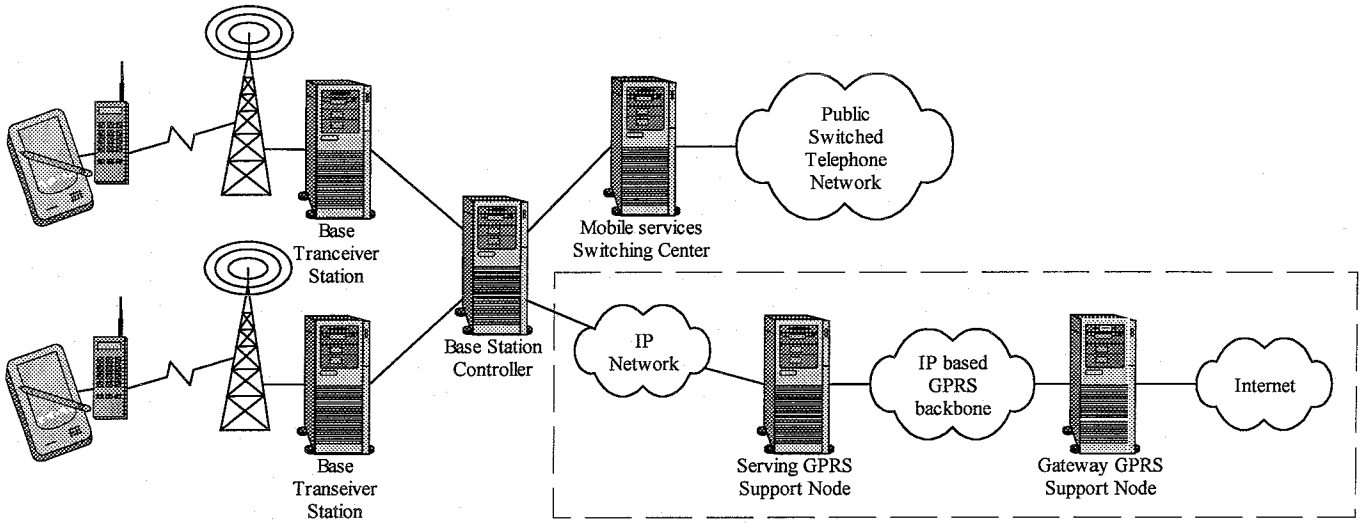


Figure 5. Overview of the GPRS network.

GPRS is an extension to the GSM architecture. The GPRS system adds a backbone IP network that is separate from the existing GSM core network that is used for circuit-switched traffic. This addition is drawn in the dotted box in Figure 5. Two new nodes form the cornerstones of the GPRS backbone: the Serving GPRS Support Node (SGSN) and the Gateway GPRS Support Node (GGSN). The SGSN handles packets of users in a geographical area. The GGSN connects to outside data networks.

GPRS uses the existing GSM radio network and a number of components are reused which reduces the overall costs for the migration from GSM to GPRS. There is no need to build a complete new radio network, which will be necessary for migrating to UMTS.

GPRS mobile stations can operate in three different modes (not to be confused with the WAP classes):

- Class A mode of operation allows a mobile station to have a circuit-switched connection at the same time that it is involved in a packet transfer

- Class B mode of operation allows a mobile station to be attached to both circuit-switched and GPRS but it can not use both services at the same time. However, a mobile station that is involved in a packet transfer can receive a page for circuit-switched traffic. The mobile station can then suspend the packet transfer for the duration of the circuit-switched connection and afterwards resume the packet transfer.

- Class C mode of operation allows a mobile station only to be attached to one service at the time. A mobile station that only supports GPRS and not circuit-switched traffic will always work in class C mode of operation.

The class A devices offer new opportunities to WTA applications: whereas with GSM and class B and C GPRS devices, the WAP connection is lost when initiating a voice connection, with class A devices it is now possible to combine data and voice at the same time. An application that could make use of this is for example a voicemail interface. A user can use WAP to navigate through the voicemail menu and use speech connections to listen to spoken messages.

5.1.2 UMTS

UMTS (Universal Mobile Telecommunications System) is a so-called "third-generation (3G) standard". UMTS offers packet-based transmission capabilities of text, digitized voice, video, and multimedia at data rates up to and possibly higher than 2 megabits per second and a consistent set of services to mobile computer and phone users no matter where they are located in the world. Based on the GSM communication standard, UMTS, endorsed by major standards bodies and manufacturers, is the planned standard for mobile users around the world by 2002. Once UMTS is fully implemented, computer and phone users can be constantly attached to the Internet as they travel and, when they roam, have the same set of capabilities no matter where they travel to. Users have access through a combination of terrestrial, wireless and satellite transmissions. Until UMTS is fully implemented, users can have multi-mode devices that switch to the currently available technology (such as GSM 900 and 1800) where UMTS is not yet available.

The higher bandwidth of UMTS promises new services, such as video conferencing. UMTS promises to realize the Virtual Home Environment in which a roaming user can have the same services to which the user is accustomed when at home or in the office, through a combination of transparent terrestrial and satellite connections.

As the WAP specification is independent of the bearer, WTA and WAP can be used with UMTS as well. The higher data rates make the transmission of larger files in an acceptable time possible. This makes more advanced applications possible. WTA can still be used as a standard to combine data and telephony services. Probably new functions will be introduced in the WTA standard to make use of new functionalities in the UMTS network, like for example a function to set up a videocall.

5.2 Specification

The WAP specification will probably follow the evolution of the network from 2G to 3G. The evolution from 2G to 3G offers, as can be read in the preceding section, new service opportunities. Next generations of WAP will offer new functionality like multimedia support, billing, external function interface, end-to-end security, large data transfer and a persistent memory.

Probably at the end of June 2001, the next release of WAP, WAP 2.0 will be made available by the WAP Forum. This new release has some architectural changes. The most important change is the introduction of WAE version 2. WAE 2 uses WML 2, which is built on top of XHTML Basic with additional wireless extension modules that provide wireless specific features for the WAP 2 WAE platform. XHTML, defined by W3C, is a reformulation of HTML in XML. A basic set of modules has been defined as XHTML Basic, and can be used as the foundation for an extension to the XHTML family of mark-up languages. This convergence makes it easier for contentproviders to support webbrowsers and wapbrowsers at the same time.

In the new WAP specification it is not necessary anymore to use the WAP gateway. The WAP gateway can however be used for performance enhancing.

With regard to the WTA specification the specification will be refined and new networks functionality will be added. In the WAP 2.0 release, the WTA specification is not changed dramatically. An important change is the usage of WML 2 instead of WTA-WML. The architecture is the same as the June 2000 specification. This means that for WTA a WAP Gateway is still needed.

At this moment the WTA specification only specifies requirements for the device client. In future releases also server side APIs might be introduced. Other items for the WTA future are multiple concurrent contexts, improving and expanding the security framework, roaming scenarios and improved event handling.

5.3 Terminals

At this moment there's not much information available about the future terminal support. Partly this is the result of the fact that manufacturers are careful with providing information about new models.

New terminals will support the WAP 1.2.1 specification. But as for class C devices the WTA support is optional, as can be read in 4.1, it will not yet be implemented with all terminals.

In the future mobile phones and personal digital assistants will grow more and more together. The first examples are already available, for example the Ericsson R380. These terminals will have more CPU and memory power and thus make it possible to run more complex applications on the mobile client.

6 Survey of Potentially Useful Applications

In this section a number of potentially useful applications are discussed. All these applications are investigated to find a good prototype. These applications are discussed here to give the reader not only an idea of what kind of applications are possible, but also what kind of applications are not possible when using WTA. All these applications are examples and the list of applications is definitely not exhausting. A more detailed description of the applications can be found in Appendix D.

6.1 Applications

6.1.1 Incoming Call Selection

With the Incoming Call Selection service users can personalize their mobile phone interface. When an incoming call reaches the mobile phone, a WML deck is loaded from the repository. This WML deck represents a menu with various call handling options like answer, reject or forward. This menu can be adapted to the user's needs. An example of this menu is shown in Figure 6.

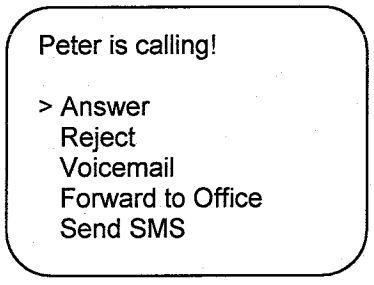


Figure 6. Example of an Incoming Call Selection Menu

According to the WTA specification, it should be possible to implement the service as described above by using WTA functions. However these functions use corresponding functions in the lower bearer layer. In case of the usage of GSM as the bearer, the forward functions make use of the GSM function `deflect` which is optional in the GSM specification and is therefore not available in all GSM networks. This means that for call handling only the answer and reject function will work, and therefore the added value will be small.

The other functionality is to send a message. Because it takes a long time on most telephones to enter a message, it would be easier if the user could select from a number of predefined messages. Predefined messages could for example be strings like "I am in a meeting now", "I'll call you back later" or "Please try again within an hour". This could be implemented by using a WML deck containing these strings. This implementation has the disadvantage that it is not easy for the user to change these strings. To change one or more of the predefined strings, the WML deck in the repository that contains the string has to be replaced. This means that the user has to access the WTA server to generate a new WML card and then load this card in the repository. It would

be a better option to store these strings in a separate locally stored file such that only this file needs to be updated. Unfortunately this is not possible.

6.1.2 Call Screening Service

Nowadays, users of mobile phones only have the choice between receiving or diverting all calls. The Call Screening Service makes it possible to make this choice per number. For example a manager can choose to forward all calls except calls from his secretary to his voicemail during a meeting.

Just like with the Incoming Call Service, the use of the forward functions is only possible when this function is implemented in the underlying network. The application will be even more interesting when the user can define a list of numbers that are allowed to interrupt and forward number to different numbers like for example a private and a business voicemail. It is not possible to store these numbers as a table in some kind of memory such that the user can easily add or remove numbers. It is also not possible to use the WTA phonebook functions to create additional fields to store which numbers are allowed to interrupt. To implement this function, the filterrules must be implemented hard-coded in the script that is installed in the repository. This means that for every update, the server has to generate and install a new script, even when the user wants to turn of this service.

When using a GPRS class A device it is possible to use a voice and a data connection simultaneously. In that case the information could be stored on the server such that on every incoming call the server is requested what to do.

The service as described above will not work when the mobile phone is out of range. That's why it might be a better solution to implement this service in the network.

6.1.3 Automatic Scheduled Divert

The Automatic Scheduled Divert extends the previous applications by adding calendar functionality. The service is intended to remove the need for manual phone diversion before attending scheduled events. Using this service the user does not have to bother about setting and resetting diverts during meetings. To set and reset diversions, service indications could be used to install a channel containing a script to divert the calls (like the incoming call screening script). But for security reasons service indications require an explicit user confirmation before installing a channel, which means that the user has to confirm every set and reset of the diversion.

As discussed in the call screening service it is not possible to temporarily store information in the mobile client like a list of forwarded calls. If a call log is available in the mobile phone, the WTA call logs functions could be used for this purpose.

Just like the call screening service, this service will not work when the phone is out of coverage. Therefore an implementation in the network might be preferable.

6.1.4 Location Information Services

With a mobile phone users can communicate everywhere. After all that is where mobile phones are intended for. The Location Information Services provides the user of local telephone information, based on the user's current position. The user can define his own profile, like selecting a number of categories, and the content will be generated based on the user's position. An example is shown in Figure 7.

Figure 7. Example of the Location Information Service. The user selects the link indicated with ">".

The WTA functionality used in this application is the function that provides the network information of the terminal. When this information is passed to the server, the server can calculate the user's position, based on the position information of the operator's base stations. The information about the cells is thus operator dependent, because every operator uses its own physical network with its own base stations, which are usually not placed at the same locations as other operator's base stations. Thus to implement an operator independent solution, you need to know the geographical locations of the base stations of all operators. But operators are not generous in distributing this information. Other systems for providing positioning information are network based systems. These systems have the advantage that the position information stays within the operator domain and they do not require new terminals supporting this option.

An alternative to this application is the Local Number Download. With this service a number of predefined numbers is automatically updated based on the user's position. For example the entry "Hospital" in the user's phonebook always represents the number of the closest hospital. When the user approaches another hospital, the entry will be automatically updated. This service requires a lot of data exchange, because, for example on every hand over, the new location information must be sent to the server.

6.1.5 Friends Locator

The Friends Locator service is designed to allow users to be notified when friends enter or leave the same area. Every user can keep up with a list of friends. When one or more friends enter the current user's area, a notification is sent to the user. Every time that the network status changes, an update should be sent to the server to update the user's current position. After this update all friends', who have the current user in their table, positions should be checked and if necessary the friends should be notified.

Like with the Location Information Services, the position information is based on the operator's network. Comparing the positions of two friends using different networks is not possible without extra information about the locations of the base stations. Just using the cell ids is not enough.

In the WTA specification no "movement" event exists. The only event that can give some information about movements of the user is the NetworkStatus event, but this event only indicates that the device has changed networks or has been handed over to another cell. Such a "movement" event can prevent the application from constantly polling.

When using the Friends Locator with GSM or GPRS class B or C, it is not possible to update the friend's location during a call, due to the fact that only one communication channel is available at the time.

6.1.6 Automatic Retry

The Automatic Retry application can be used when the called party's phone number is busy. This application keeps trying to set up a call until the called party is not busy anymore. The automatic retry application can be implemented by a simple WMLScript containing a loop.

This simple application has some disadvantages. First of all the application keeps the calling party's line busy by constantly trying to set up a call. Secondly it is not possible to warn the user when a call succeeds by using for example a sound signal. Now the user should constantly check the display of his mobile phone.

6.1.7 Camp On Callback

Instead constantly trying to call the busy number, the Camp On Callback service automatically provides a ring notification when the busy party becomes free. When a user tries to set up a call to a busy party, the service will ask the user to active Camp On Call back. When the user activates the service, the server will send a notification to the user when the called party is not busy anymore. When a service indication is used for the notification, the user can immediately set up the desired call using WTA functionality.

This service requires a tight integration between the WTA server and the telephony network to monitor the availability of the called party.

6.1.8 Corporate Directory

The Corporate Directory is an electronic phonebook stored on the server. The user can access the database and search for a number. When the user finds a match, the user can initiate a call from the result page, or add this number to the local phonebook using WTA functions. The service as described here should be possible with according to the WTA specification. But it might be interesting to extend the service with the following functionalities:

When the phonebook is also available via a web interface, the user can initiate a call from this web session. When the user chooses to initiate a call, a

service indication is sent to the user's mobile terminal, and after a confirmation a call can be initiated.

Because all numbers are now stored centrally, it is easier to update these numbers. It can be useful to synchronize the local phonebook with the numbers on the server. Using the WTA phonebook functions it should be possible to synchronize the phonebook. But as the mobile client does not maintain a last edited time stamp it is not always possible to resolve conflicts based on time. This also implies that the entire contents of the phonebook must be sent to the server to find out what modifications are made. This is however not an attractive option.

Another extension could be to combine the phonebook on the server and the local phonebook. A new phonebook can be built on top of both phonebooks, such that when the user searches for a particular number, first the local phonebook is consulted, and if no matches are found the phonebook on the server is consulted. However the local phonebook in the mobile client usually has a much better user interface to find numbers that WML decks and it is not possible to generate WML deck locally in the terminal to generate for example search results.

6.1.9 Top Ten Numbers

The Top Ten Numbers application provides the subscriber with a list of numbers that the subscriber calls most often. The most called number appears first on the list. When a number is called, the counter is updated and if necessary also the list is updated. It is not possible to use this service without a WTA server, because it is not possible to store the necessary counters locally.

When using GSM or GPRS class B or C, it is not possible to update the counters on the server during a call. This means that the update should be postponed until the call is completed.

6.1.10 Incoming Call Information

The Incoming Call Information service is designed to extend the information presented when an incoming call is detected. Most mobile phones show the calling party number or, when the name and number are stored in the local phonebook, the name of the calling party. With the Incoming Call Information service is used, the WTA server is contacted on an incoming call and additional information can be downloaded. Because the WTA server has much bigger storage capabilities, all kind of information can be provided like address information or a picture. The prototype discussed in the next chapter is based on this service.

This service requires two connections at the same time: one speech connection for the incoming call and one data connection for the WAP connection. This is possible when using GPRS class A terminals. It is also important that the response occurs quickly to prevent the calling party from hanging up.

6.1.11 Visit Card Exchange

The Visit Card Exchange makes it possible to exchange phonebook entries between two telephones. Only one of the two users has to enter the other's number. When user A entered user B's number, the Visit Card Exchange service encoded user A's number and name and sends it to user B by using a text message. When user B has received this message, the message is decoded and stored in the mobile terminal. After this user B sends his name and number to user A using the same procedure. This implementation requires that both users have the service installed on their mobile terminal.

Another implementation options is to send the name and number via the WTA server, using service indications. This option has the advantage that the users do not have to install the service. It will be sent to them using a service indication.

6.1.12 Context Aware Dialing

The Context Aware Dialing service is based on the way people start a conversation in situations where they meet physically. In these situations the initiator can see the situation and he can decide, depending on the situation to start or to postpone the conversation. When using the Context Aware Dialing service the calling party can first check the status of the called party on the WTA server after which the user can decide to continue the call. It is not possible to implement the service such that for every outgoing call the service is activated before the actual call setup starts.

6.1.13 Menu Selection

Instead of listening to the menu options in so-called interactive voice response systems, these options can be shown in a WML deck. The user can now browse through the various menu options and if necessary a voice call can be initiated to the appropriate person using WTA functionality. Simultaneous usage of data and voice capabilities requires GPRS class A.

6.1.14 Message Manager

The Message Manager Service is designed to allow subscribers to receive notifications and manage their email, voice mail and fax messages. The Message Manager will be integrated with a unified messaging system that provides access to all of the subscriber's messages. The message manager makes use of the WTA especially during the management of the messages. The subscriber can forward emails, faxes and voice mails to other subscribers selected from the phonebook. Or a subscriber can play a voice mail by setting up a call to the unified message system. In fact the use of WTA makes it possible to offer a new interface for accessing the subscriber's mailbox.

6.1.15 Call Information Service

The Call Information Service provides the user with information about recent calls. Instead of obtaining this information from the mobile terminal, this information is obtained from the network. An advantage of obtaining this

information from the network is that calls missed due to lack of coverage can be stored too. The WTA functionality in this service is limited to calling or storing a number from the call history.

6.2 Conclusion

The application discussed above give a number of possible applications. These applications are the result of doing some brainstorming. In fact the process of thinking up a number of applications from a technology is the wrong direction. A more logical direction to design an application is to think what the application should do, and after this is clear, find and use a technology to implement this application. So it might be very well possible that there are a number of much better use cases for using WTA.

After discussing a number of examples it is not possible to define real limitations of WTA. One could define three limitations: limitations as result of networklimitations, limitations as result of the WAP specification and limitations as result of the WTA specification. But these limitations are somewhat overlapping.

A networklimitation found in the discussion above are the restriction that at this moment at most one communication channel at the time is available. When GPRS class A terminals become available, this limitation will be removed. Another networklimitation is the implementation of the deflect and forward functions. These functions are available in the WTA libraries, but the corresponding functions are not implemented in the Dutch GSM networks.

A limitation of the WAP specification is the limited user interface. On the one hand the WAP specification is made for limited devices but on the other hand this does limit the possibilities for applications. Examples of limitations of the user interface are the difficulty of entering text and the absence the option to play a sound. Another limitation is that it is not possible to store information in some kind of file.

The WTA specification has the limitation that it is not easily possible to authenticate the user on the WTA server. It is possible to use authentication on the WAP gateway, but not on the WTA server. The most obvious solution is using the number of the mobile terminal, but this number is not forwarded according to the WAP specification. Another limitation is the fact that for every service indication a confirmation from the user is needed. A service load instead of a service indication is not possible. On the other hand this is a logical consequence from security point of view.

Prepared (also subject responsible if other)		No.		
ETM/BL/IB Peter de Vries		ETM/BL/IB-01:0117 Uen		
Approved	Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers		12/06/01	A	H:\etmptvr

7 Prototype: I See You

One of the objectives of this study is to build a prototype application that demonstrates the use of the WTA framework. In the previous chapter a survey of potentially useful applications is given. The prototype is based on the Incoming Call service described in 6.1.10. In this section the prototype is discussed in a high level overview. A more detailed description of the implementation can be found in Appendix E.

7.1 Introduction

Nowadays, mobile phones are able to display the number of the calling party, the caller id, on an incoming call. When this number is stored in the local phonebook, most phones can display the name instead of the number. "I See You", or abbreviated to ICU, goes one step further by making it possible to display the name and a picture of the calling party.

Every user of the ICU service has an own account on the ICU server containing the contact details and a picture of the user. Every user also has a contactlist on the server. The contactlist is used to find a name and number on an incoming call. The user can also use the contactlist to find contactdetails and, when using the WAP interface, to initiate a call by using WTA functionality.

Users can add two kinds of contacts to this contactlist: buddies and personal contacts. Buddies are other registered users on the ICU server. When a user adds a buddy to the contactlist, the contactdetails as entered by the buddy himself are used. So the user does not have to look for a picture and when the buddy updates his contact details, these details are automatically updated in the contactlist too. Buddies can be found by searching in a public database. For privacy reasons users can choose whether or not their contactdetails are publicly available to other users. A user can also choose whether other users must request authorization to add the user to their contactlist.

Personal contacts, on the other hand, are contacts created by the user himself. These contacts can for example be used when the contact is not a registered contact. The user has to enter the contactdetails and upload a picture himself.

To use the service, a channel must be installed on the user's mobile phone. This channel contacts the ICU service on the server on every incoming call, and sends the number of the calling party to the server. When this number is found in the contactlist, the name and picture of the calling party are displayed together with the options to accept, reject or forward the call. If the number is not found on the contact list, the service queries the public database. When still no entry can be found, only the number of the incoming call and the options to accept or reject the call are displayed. In the latter case, the user is asked whether he wants to add this number to the contactlist after the call has been ended.

7.2 Motivation

There are a number of reasons why this application is chosen as prototype. First of all this application uses most features of the WTA framework: the repository, the event handling mechanism, the WTAI libraries and the WTA server. This gives a good overview of the possibilities of the WTA framework.

Secondly the application can be largely demonstrated without the use of a WTA user agent. During this study it became clear that no mobile phone or mobile phone emulator with WTA support would be available before the end of the study. Though a phone with WTA support is still necessary to make this service really work, the actions performed by the WTA user agent can be easily replaced by a manual action such that the complete service still works. Applications like the Automatic Retry (6.1.6) are very client centric such that the absence of the client has a much bigger influence on the demonstration.

Thirdly the ICU service is a completely new and very suitable application to demonstrate. The service has a high "fun" content and it is easy to explain what to use the application for.

7.3 Assumptions

During the design of the prototype two assumptions are made. First of all the ICU service needs at least a dual channel connection. The ICU service uses a data connection during a voice call. This is possible when using a GPRS class A terminal. At the moment of writing these terminal are not yet available.

Secondly the connection to the server must be fast enough to make the service useful. Fast enough means in this context that the time between the detection of the incoming call and displaying the picture to the user should be as small as possible, say less than three seconds. GSM has the disadvantage that the time needed to establish a connection could be long. GPRS should be fast enough to download a page within a short time. And because of the "always online" connection no extra time is needed to establish the connection.

7.4 Architecture

7.4.1 Overview

Figure 8 gives an overview of the architecture of the I See You service. This figure shows the components that are used in the ICU service. The four shaded components are the components especially designed for the ICU application. These components will be introduced here, but discussed in more detail below. The ICU service can be divided in two parts: the client part and the server part. The client part is the WTA supporting mobile phone or an ordinary web browser. The mobile phone contains the ICU channel in the repository. On every incoming call, the WTA user agent processes this channel and requests the appropriate page from the server.

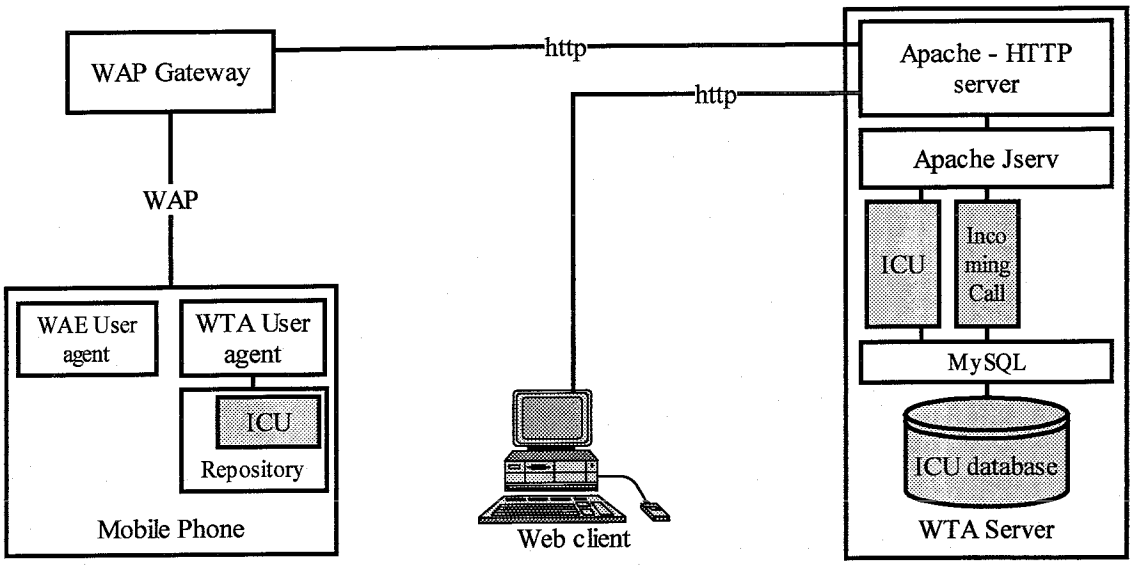


Figure 8. Overview of the I See You architecture.

The server part is based on the architecture of the Ericsson WAP Application Server. By using this architecture it should be relatively easy to use the application on the WAP Application Server such that the application can be used for demonstrations purposes on the WAP Application Server too. The WAP Application Server consists of an Apache HTTP-server with an Apache Jserv servlet engine, a database and some APIs for fast development of WAP applications. The Common Gateway Interface (CGI) makes it possible to connect a URL on a webserver to an external program. The Ericsson WAP Application Server uses Java servlets to implement these external programs. Java servlets offer the some functionalities for easy application development, like session tracking.

The main ICU application and the application that generates the incoming call pages are implemented as Java servlets. All the contactdetails are stored in a database, using MySQL.

7.4.2 Database

A database is used to store all the contactdetails for the ICU service. The database is described below using an entity-relationship diagram.

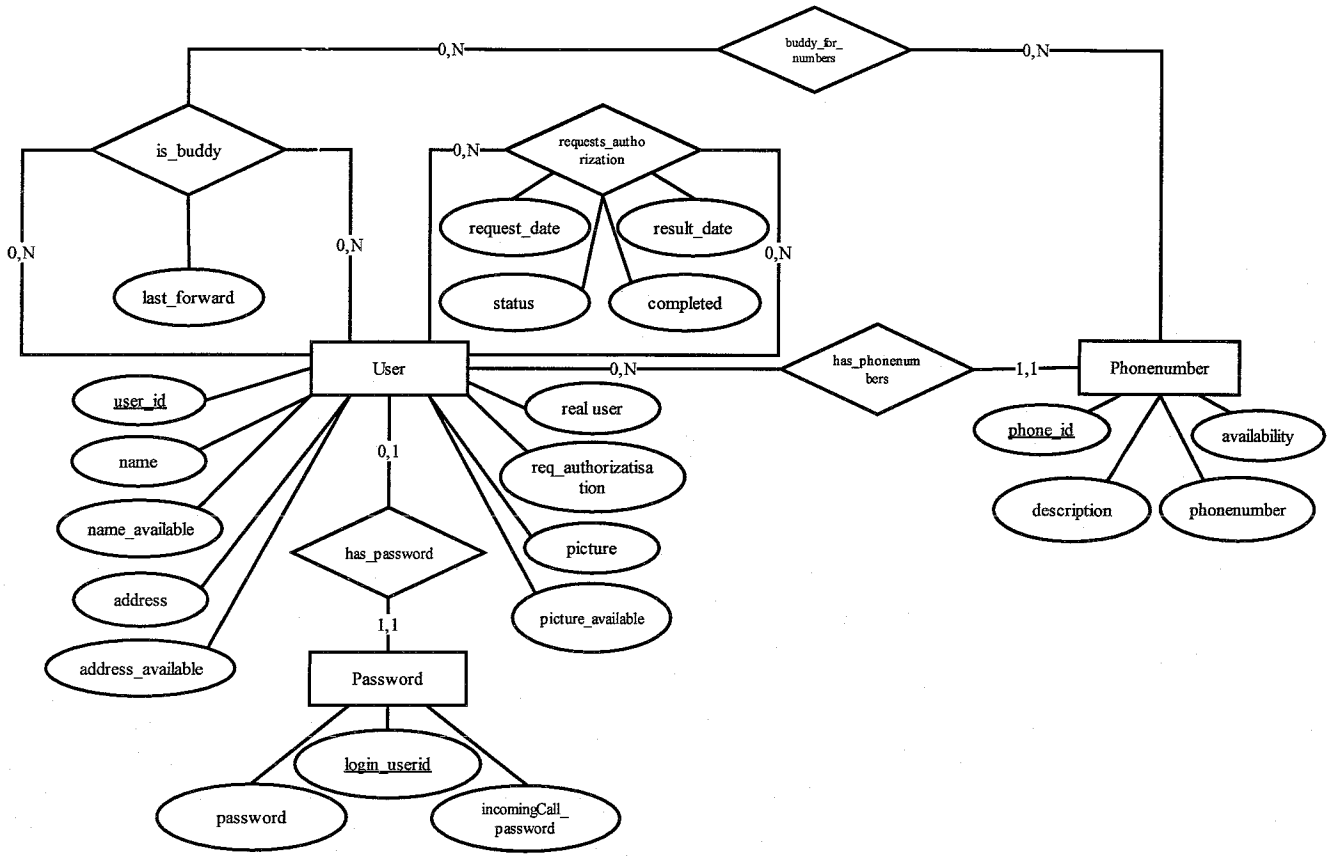


Figure 9. E-R diagram of the ICU database.

The database contains three entities: `user`, `phonenumber` and `password`. The `user` entity is the central entity. Every `user` has a unique `user_id`, the key. Every `user` also has a `name` and `address` attribute, which are so-called composite attributes. This means these attributes describe a group of simple attributes like for example `name` might be composed of `firstname`, `middlename` and `lastname`. For simplicity only these composite attributes are shown in the diagram. The `realuser` attribute describes whether the user is a real user and the `req_authorization` attribute describes whether other users must request authorization before they can add the user as buddy to their contactlist. The `picture` attribute contains the filename of the picture. Finally, a `user` also has some *`attribute`*`_available` attributes. These attributes describe whether these attributes are publicly available, that is available to other users without an `is_buddy` relation.

Every `user` has zero or more `phonenumbers`. A `phonenumber` belongs to exactly one `user`. A `phonenumber` has a `phone_id` as key. Furthermore a `phonenumber` contains a `phonenumber`, a `description` describing this `phonenumber` and an `availability` attribute. The `availability` attribute describes, like the `available` attributes of a `user`, whether this `phonenumber` is publicly available to users that do not have an `is_buddy` relation for this particular `phonenumber`.

The password entity contains information to login to the service. The password entity contains a unique login_userid as key. The value of this attribute is used to login to the service. The password attribute contains the password to login. The incomingcall_password contains the login password that is used in the incomingCall servlet. The reason why these servlets use different passwords is discussed in 7.4.5. Only real users have a password and a password belongs to exactly one user.

Every user of the service can maintain a contactlist, which is implemented by relations. The contactdetails of user appear in another user's contactlist if and only if an is_buddy relation between these users exists. Every user can have zero or more buddies and every user can be the buddy of zero or more users. The buddy_for_numbers relation defines which phonenumbers appear on the contactlist. Only the phonenumbers that have a buddy_for_numbers relation with an is_buddy relation appear on the user's contactlist. By using this relation a buddy can decide to hide some numbers for a particular user. The last_forward attribute of the is_buddy relation contains the phone numbers the buddy has been recently diverted to.

Finally the request_authorization relation defines requests for authorization. A request_authorization entry represents a request for authorization to add a user as buddy to another user's contactlist or to see more phonenumbers of a buddy. The request_date and result_date respectively represent the date when the request for authorization is created and completed. The completed attribute describes whether the buddy has processed the authorization_request and the status attribute describes the result of the processing, for example "rejected" or "accepted".

The database contains a database constraint that cannot be derived from the diagram. A buddy_for_numbers relation is valid if and only if the user to whom the phonenumber belongs is the same user as the buddy in the is_buddy relation.

In this database model both buddies and personal contacts are stored as users. Personal contacts are represented by users with the following properties:

- The realuser attribute has the value 'no'.
- All the _available attributes have the value 'no'.

And:

- The availability attributes of all phonenumbers belonging to this user have the value 'no'.
- The user belongs to exactly one is_buddy relation such that the user is the buddy of the (real) user who created the personal contact.

A translation from the E-R diagram to tables can be found in Appendix E.

7.4.3 ICU servlet

The ICU servlet is the server side application to maintain the contactlist, process authorization requests and to configure the service. Figure 10 shows a high level flowdiagram for a user of the servlet. First of all a user must enter his user id and password to login or, if the user is not registered yet, the user can register. When the user is logged in to the service, the main menu is shown. The user can choose to display his contactlist, to add a contact to his contactlist, to change settings of the service, to process authorization requests or to log out. A detailed flowdiagram of these subprocesses can be found in the appendix. A short description of these subprocesses will do here.

The subprocess contactlist displays the user's contactlist. The contactlist contains all contacts that have an `is_buddy` relation with the user. The user can search for a particular name in this list or display all contacts. When the user selects a contact, the contactdetails, like name, address, phone numbers and a picture of the selected contact are shown. When the user is connected to the servlet using a WAP browser, the user can also choose to call the contact using public WTA functions. Furthermore the user can remove the contact and edit the contact, if it is a personal contact. In case the contact is a buddy the user can add possibly hidden phone numbers. This option adds all the buddy's public phone numbers that are not yet in the `is_buddy` to the `is_buddy` relation and creates an authorization request for possible private numbers. By using this method, the user will never know whether the contact has more hidden private phone numbers, but he can always ask for them.

When the user enters the add contact subprocess, a search form is displayed to the user. The user can search the public database for a particular contact. When the right contact is found, this contact can be added to the contactlist. If authorization is required, a new authorization request is created. When no authorization is required, the contact is immediately added to the contactlist by creating an `is_buddy` relation, and connecting all public numbers to this `is_buddy` relation. In case the contact has some private numbers, also an authorization request is created to add these numbers. When the contact can not be found in the public database, the user can choose to create a personal contact. This personal contact is a special user as described in 7.4.2.

In the settings subprocess the user can change his contactdetails, upload a picture or change his password. The user can also choose to install the ICU channel necessary to use the service. When the user chooses to install the ICU channel, the channel and the pages necessary to install the channel are created. During the channel installation a new password for the incomingCall servlet is generated. A successful installation results in an update of the user's `incomingCall_password` attribute. When the installation fails, nothing happens. The reason for introducing this password is discussed in the next section about the incomingCall servlet.

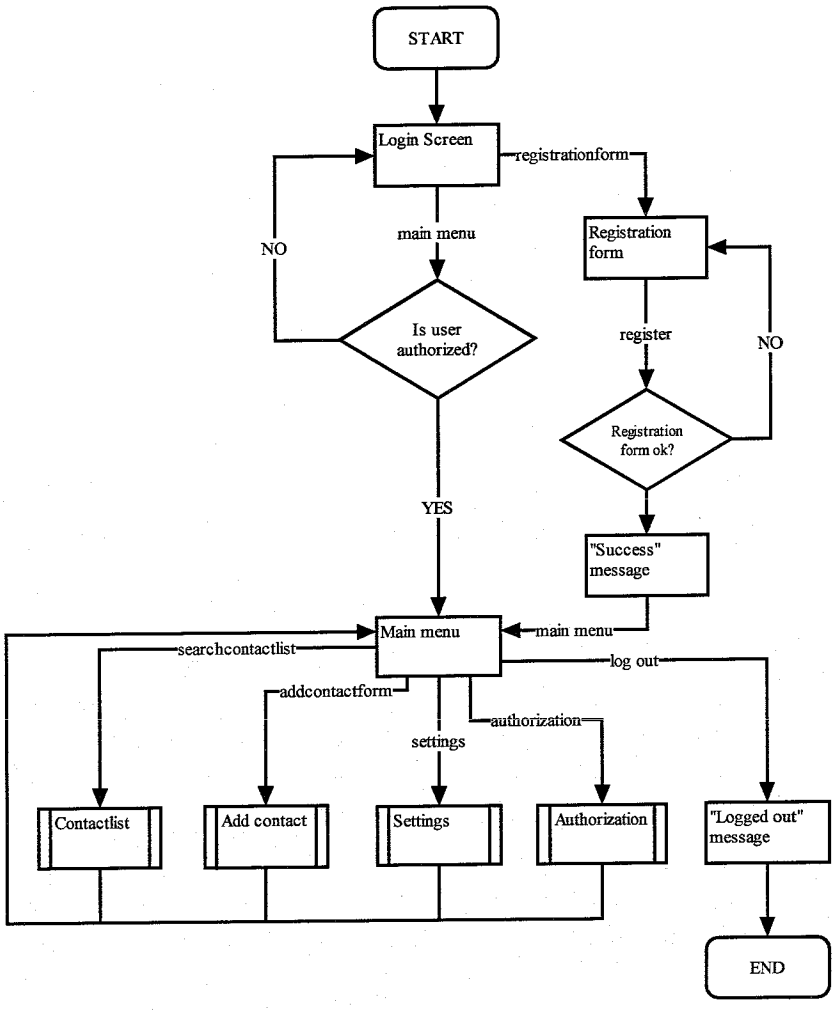


Figure 10. High level flowdiagram for a user of the ICU servlet.

7.4.4 ICU channel

The purpose of the ICU channel is to take care that the appropriate picture is downloaded from the server. In fact this is the component that makes this service possible. This channel contains a WTA-WML deck, the incomingcall deck and a WMLScript file. When an incoming call reaches the mobile phone, the WTA incoming call event is generated. This channel is bound to the WTA incoming call event and therefore the incomingcall deck specified in the channel will be loaded in the WTA browser. The incomingcall deck immediately redirects the browser to the incomingCall servlet. Between this request and the response to this request some time elapses. The user should always be able to answer the call during this delay using the normal phone interface. Therefore the incomingcall deck contains cards that correspond with the call state. When the user chooses to answer the call, a corresponding WTA event is thrown. In that case the request is aborted and the browser navigates to the card that corresponds to the current state and the browser sends a new request to the incomingCall servlet.

7.4.5 IncomingCall servlet

When the mobile phone receives an incoming call, the incomingCall servlet will generate the WML deck containing the caller's picture. To generate this deck, first of all the servlet needs to know the caller id of the calling party and the user id of the called party, the user. This information is passed to the servlet by the script in the ICU channel.

The incomingCall servlet first checks the user id and password. It would be a better option to use the user's phone number for authentication, because in that case the user id and password do not have to be stored in the mobile phone. This is however not simply possible, because according to the WAP specification, the phone number is not passed from the WAP gateway to the servlet. The Ericsson WAP Application Server is able to obtain the phone number from the Ericsson WAP gateway, but only by using a proprietary solution. Therefore in the prototype, the user id and password are stored in the ICU channel in the mobile phone. For security reasons this password differs from the password used to log in to the ICU servlet. So this password is only valid for the incomingCall servlet.

When the user is authorized, the incomingCall servlet first searches for the phone numbers in the contactlist that match the caller id. If one match is found, a WML deck containing the name and picture of the matching contact is displayed on the user's browser. When multiple matches are found, only the possible names are displayed. When no match is found, the incomingCall servlet will search the public phone numbers list. When one match is found in this list and some contactdetails are publicly available, these contactdetails will be shown to the user. In case more than one match is found, only the names will be shown to the user. Finally when no match is found, only the number of the calling party will be shown.

The generated WML decks, do not only contain the name and picture, but also WTA functions to handle the call. Functions that can be used are for example accepting, rejecting and ending the call. When a call is ended, the user can add the contact to his contactlist, if the user is not in the contactlist yet. To use these call handling functions the decks contain multiple cards. These cards correspond with the possible call states. The following call states are possible: pending, active, held and end. The navigation between these cards is triggered by WTA events. By using these different states, only the useful options for a particular state are shown to the user. For example during the pending state of a call, the user can accept, forward and reject the call, while during the active state the options hold and end are shown. To display the current state the state must be passed from the ICU channel to the incomingCall script too.

7.5 Problems

During the implementation of the ICU service a number of problems are found. First of all it was not possible to test the application using a WTA browser. At the beginning of the study it looked like a mobile phone emulator with WTA support would be available within a short time, but until the time of writing this emulator is still not available. The reason given for this delay is the

lack of interest from operators. As a result of this, the application is built using the WAP specifications only and the most critical functionality, the channel, could not be tested in practice.

Secondly the WAP architecture is a very limited architecture. Besides the limitations already mentioned in the previous section, some other limitations are found during the implementation of the prototype. For example: despite the fact that WML should be independent of the mobile phone, a WML page is presented differently on different mobile phones. To be sure that a WML page is presented correctly, a developer must therefore construct a device specific WML page. Another limitation is the phone number forwarding between the WAP gateway and the WTA server. The WAP standard does not provide a standard mechanism for this. Finally the WAP standard does not support cookies. However, these will be supported in future versions.

Prepared (also subject responsible if other)		No.		
ETM/BL/IB Peter de Vries		ETM/BL/IB-01:0117 Uen		
Approved	Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers		12/06/01	A	H:\etmptvr

8 Conclusions and recommendations

In the first section the goal of this study is defined: 'to investigate the possibilities of WTA and to build a prototype application that uses this framework'. Below are the conclusions of the subgoals and finally the total conclusion of this study, followed by some recommendations.

8.1 Conclusion

Wireless Telephony Application (WTA) is a telephony extension to the WAP specifications that can provide in-device telephony related control mechanisms to advanced mobile network services. With WTA it is possible to develop services that manipulate mobile unit features (e.g. call control and the phonebook), and trap various events (e.g. incoming call).

There are no alternative techniques that can substitute WTA. However, two techniques are somehow related to WTA: the SIM Application Toolkit (SAT) and the Mobile Execution Environment (MExE). The SAT is specified in the GSM specification and specifies the interface between applications running in the SIM and the mobile phone. The SAT offers less functionalities for application developers than WTA. The MExE is specified by the 3rd Generation Partnership Project and provides a standardized approach for the delivery of services, applications, applets and content to smart mobile phones. MExE handsets have a fully embedded application execution environment, which offers more functionalities for application developers than WTA.

At the time of writing the June 2000 release or also denoted as WAP 1.2.1 is the first and most recent WAP specification that includes the WTA specification. The WTA specification is an optional part of the WAP specification. This means that WAP terminals do not have to support the WTA specification. At the time of writing neither terminals nor terminal emulators that support WTA are available. Some terminals however do support some public WTA functions like initiating a call from a WAP page.

In the future there will be a shift from circuit-switched to packet-switched networks. This shift makes it possible to combine data and voice service, something WTA can be used for. However at the time of writing there are no signals from handset manufacturers that phones that support WTA will be available within a short time. The WAP and WTA specifications are still evolving. New functions for 3G networks and support for XHTML will be introduced in future versions.

The combination of data and telephony functions makes a lot of new applications possible, but there are a number of limitations. These limitations are not only limitation to WTA, but also to the WAP specification and the implementation of the networks. As a result of this many of these applications can not be implemented completely.

The implementation of the prototype application has partially succeeded. The biggest problem during the implementation was the lack of terminals that support WTA. Therefore the prototype can not really be used.

The conclusion that can be drawn from this study is that WTA offers some interesting opportunities, but unfortunately there are a lot of limitations. At the moment of writing these limitations make the development of WTA services impossible. But the WAP forum is still working on the specifications and the new specifications seem to offer some improvements like for example the introduction of a persistent memory. But until then there is still a long way to go.

8.2 Recommendations

During this study it has become clear that WTA is not ready for use yet. And in this form, WTA will probably never be a success.

The public library of the WTA specification however contains functions that should be available in every mobile phone that has a WAP browser. Especially the function to set up a call by selecting a hyperlink in a WAP page (compare to the mailto hyperlink in a normal web page) should be a standard function. The new Ericsson models support this function.

The idea of implementing applications that use telephony functions is a very good idea, and makes many new applications possible. But instead of using the WAP and WTA technique this can better be done using a real application environment with more possibilities. The JavaPhone API and J2ME techniques, discussed in section 3.2 about MExE offer such an environment. These techniques seem to have a brighter future. The fact that the first emulators and the first terminals are already available proves this. But also with these application environments, security is an important issue. The nature of the functions, especially functions that initiate calls or manipulate the mobile phone still require a good security model which can restrict the possibilities, just as with the WTA framework. Nevertheless Ericsson should focus more on the development of Java enabled mobile phones.

Abbreviations

2G, 3G	Second Generation, Third Generation
3GPP	3rd Generation Partnership Project
ANSI 136	TDMA Cellular/PCS – Radio Interface – Mobile Station – Base Station Compatibility Standard
API	Application Programming Interface
CDMA	Code Division Multiple Access
CGI	Common Gateway Interface
CLDC	Connected Limited Device Configuration
DTMF	Dual Tone Multi Frequency
ETSI	European Telecommunication Standardisation Institute
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio System
GPRS	General Packet Radio System
GSM	Global System for Mobile communications
HTML	HyperText Mark-up Language
HTTP	HyperText Transfer Protocol
ICU	I See You
IN	Intelligent Network
IP	Internet Protocol
ISP	Internet Service Provider
J2ME	Java 2 Micro Edition
JTAPI	JavaPhone Telephony API
LAN	Local Area Network
MExE	Mobile Execution Environment
MIDP	Mobile Information Device Profile
MMI	Man Machine Interface
PDC	Personal Digital Cellular (GSM in Japan)
SAT	SIM Application Toolkit
SGSN	Serving GPRS Support Node
SI	Service Indication
SIM	Subscriber Identity Module
SMS	Short Message Service
SQL	Structured Query Language
SS	Supplementary Service
UMTS	Universal Mobile Telecommunication System
URL	Uniform Resource Locator
USSD	Unstructured Supplementary Service Data
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WDP	Wireless Datagram Protocol
WML	Wireless Mark-up Language
WMLScript	Wireless Mark-up Language Script
WSP	Wireless Session Protocol
WTA	Wireless Telephony Application
WTAI	Wireless Telephony Application Interface
WTLS	Wireless Transport Layer Security
WWW	World Wide Web
XHTML	eXtended Hypertext Mark-up Language

Prepared (also subject responsible if other)		No.		
ETM/BL/IB Peter de Vries		ETM/BL/IB-01:0117 Uen		
Approved	Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers		12/06/01	A	H:\etmptvr

Glossary

API	The external interface of a software platform, such as an operating system, that is used by system built on top of it.
Called party	The party that receives, or has received the call.
Calling party	The party that initiates, or has initiated the call.
Card	A single WML or WTA-WML unit of navigation and user interface. May contain information to present on the screen, instruction for gathering user input, etc.
CGI	Common Gateway Interface. An interface for external programs to "talk" to the HTTP server. Programs that are written to use CGI are called CGI programs or CGI scripts. CGI programs handle forms or perform output parsing not normally done by the server.
Channel	A named collection of resources with a well-known entry point. Channels are used in the repository.
Client	A device that initiates a request for a connection to a server.
Content	Subject matter (data) stored or generated at an server. Content is typically displayed or interpreted by a user-agent in response to a user request.
Context	An execution space where variables, state and content are handled within a well defined boundary.
Deck	Collection of WML or WTA-WML cards. A WML or WTA-WML deck is also an XML document.
Gateway	A gateway is a piece of hardware or software that passes data between networks (that often use different protocols).
Global Binding	An association between a WTA event and a dedicated resource; e.g. a channel in the repository.
HTTP	Hypertext Transfer Protocol. The method for exchanging information between HTTP servers and clients.
MMI	Man-Machine-Interface. The user interface of a device.
Network event	An event or activity related to the mobile network, for example Incoming Call, which occurs outside of any WTA context, but which is conveyed to the WTA user-agent in the form of a WTA event and subsequently processed within a WTA context.
Push	When the server sends content without the client requesting it. I.e. a server initiated content delivery.
Repository	A persistent storage container containing resources collected in channels.
Resource	A network data object or service that can be identified by a URL. Resources may be available in multiple representations (e.g. multiple languages, data formats, size and resolutions) or vary in other ways.
Server	A device (or service) that passively waits for connection requests from one or more clients. A serve may accept or reject a connection request from a client. A server may initiate a connection to a client as part of a service (using

	push technology).
Service Indication	A content type that is used to cause a User Agent to indicate to the user that an external event has occurred in an application. A Contact Format that allows a WTA server to deploy dynamically new services to a WTA device.
SIM	Subscriber Identity Module. A little smartcard that contains the subscriber's information and can be plugged into any GSM mobile terminal.
SIM Application Toolkit	A standard that defines the interface between applications running on a SIM and the mobile phone.
Temporary Binding	A WTA event binding that is specified using WTA-WML content.
URL	Uniform Resource Locator. The addressing system used by the server and the client to request documents. It is often called a location. The format of a URL is [protocol]://[machine:port]/[document]. A sample URL is http://www.ericsson.se/index.html
User	A user is a person who interacts with a User Agent to view, hear or otherwise use a rendered content.
User Agent	Software executing in the Mobile Unit that acts on the End User's behalf. This may include textual browsers, voice browsers, search engines etc.
Web Server	A network host that acts as an HTTP server.
WML	The Wireless Markup Language is a hypertext mark-up language used to represent information for delivery to a narrow-band device, e.g. a mobile phone.
WMLScript	A scripting language used to program the mobile device. WMLScript is an extended subset of the JavaScript scripting language.
WTA Client	A client supporting WTA services
WTA Event	A notification, in the form of an abstract WTA event that conveys a change of state in the mobile network, e.g. an Incoming Call
WTA Framework	A framework based on the WAE framework, which contains functions in order to support WTA services.
WTA Server	An origin server that provides WTA services.
WTA Service	A WSP session created by the WTA user-agent over one of the WDP ports dedicated for WTA.
WTA User-Agent	An extension of the WAE user-agent, that uses the WTA framework.
WTA-WML	The WTA Wireless Mark-up Language has the same capabilities as WML but extends WML with WTA specific features for handling event parameters.
XHTML	eXtended Hypertext Mark-up Language. XHTML is a reformulation of HTML 4.0 as an application of the Extensible Markup Language (XML).
XML	The Extensible Mark-up Language is a World Wide Web Consortium (W3C) recommended standard for Internet mark-up languages, of which WML and WTA-WML are such languages. XML is a restricted subset of SGML.

References

1 3rd Generation Partnership Project, Technical Specification Group Terminal; Mobile Execution Environment (MExE); Functional description; Stage 2 (release 4)(3GPP TS 23.057 V4.1.0), Mar-2001. URL: http://www.3gpp.org/

2 Arehart, C. et al, "Professional WAP", 2000. Wrox Press Ltd, Birmingham, UK.

3 Berendt, A., STK vs. WAP: Industry Battle or Peaceful Coexistence?, jan-2000. Telecommunications online. URL: http://www.telecoms-mag.com/issues/200001/tci/stk.html

4 Ericsson, "WAP Application Server 2.0 Product Description", 05-Jul-2000. Document ID: ERA/AV/P 00:091 Rev B.

5 Ericsson, GSM System Survey Student Text, 1998. Document ID: EN/LZT 123 3321 R3A.

6 ETSI, GSM 11.14 V8.3.0 Digital cellular telecommunications system (Phase 2+); Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface Release 1999, Jul-2000. URL: http://www.etsi.org/

7 Kaarle, P., Access of SIM Application Toolkit from Wireless Application Environment, 10-Jun-1999. Au-system. Document ID: R109906050.

8 Kennedy, C. et al, "Ericsson WTA Pre-study", 1999. Ericsson AS. Document ID: 1/360-FCPR 103 098.

9 Mobile Lifestreams, MExE. URL: http://www.mobileMExE.com/

10 Sun, Connected Limited Device Configuration, J2ME version 1.0. URL: http://java.sun.com/aboutJava/communityprocess/final/jsr030/index.html

11 Sun, Java Servlet Technology White Paper. URL: http://java.sun.com/products/servlet/whitepaper.html

12 Sun, JavaPhone API specification 1.0, 22-Mar-2000. URL: http://java.sun.com/products/javaphone/

13 Sun, Mobile Information Device Profile, J2ME version 1.0. URL: http://java.sun.com/aboutJava/communityprocess/final/jsr037/index.html

14 Sun, PersonalJava Application Environment Specification v1.1.1, 7-jan-1999. URL: http://java.sun.com/products/personaljava/spec-1-1-1/

15 WAP Forum, "WAP Class Conformance Requirements", 13-Dec-2000. URL: http://www.wapforum.org/

16 WAP Forum, "Wireless Telephony Application Interface Specification", 07-Jul-2000. URL: http://www.wapforum.org/

17 WAP Forum, "Wireless Telephony Application Interface Specification GSM Specific Addendum", 07-Jul-2000. URL: http://www.wapforum.org/

18 WAP Forum, "Wireless Telephony Application Specification", 07-Jul-2000. URL: http://www.wapforum.org/

Prepared (also subject responsible if other)		No.		
ETM/BL/IB Peter de Vries		ETM/BL/IB-01:0117 Uen		
Approved	Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers		12/06/01	A	H:\etmptvr

A Wireless Telephony Application

A.1 Wireless Application Protocol

The Wireless Application Protocol (WAP) is a protocol developed by the WAP Forum, an industry association comprising over 500 members that has the primary goal to bring together companies from all segments of the wireless industry value chain to ensure product interoperability and growth of wireless market. WAP specifies an application framework and network protocols for wireless devices such as mobile telephones, pagers and personal digital assistants.

WAP is designed for devices that have the following characteristics:

- limited CPU and memory
- small displays
- limited input devices
- limited bandwidth

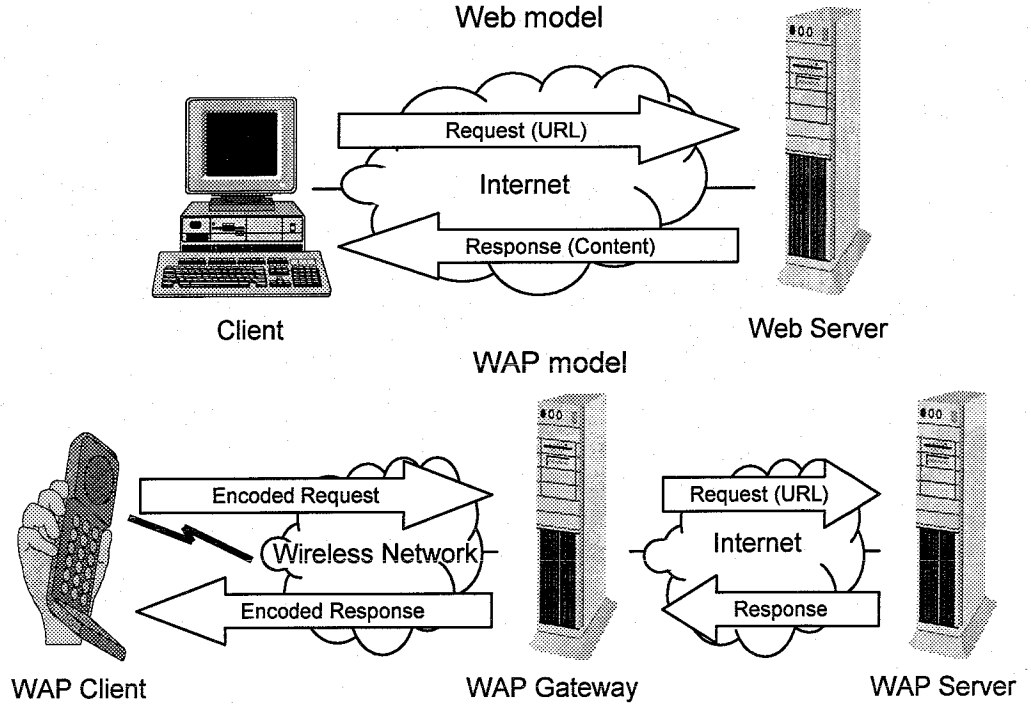


Figure 11. The web versus the WAP model.

The goal of WAP is to use the underlying web structure, but to render communication between content providers and mobile devices more efficient and less time consuming than if the web protocols themselves are used. Since the WAP architecture is designed to closely follow that of the web, WAP

inherits the client server model used by the Internet. The main difference however is the presence of the WAP gateway for translating between HTTP and WAP. Figure 1 gives a schematic overview of this difference.

The WAP architecture provides a scaleable and extensible environment for application development for mobile communication devices. This is achieved through a layered design of the entire protocol stack. The upper layer of this stack, the Wireless Application Environment (WAE) is a general-purpose application environment based on a combination of World Wide Web (WWW) and mobile telephony technologies. The primary objective of the WAE is to establish an environment that allows operators and service providers to build applications and services that can reach a variety of different wireless platforms in an efficient and useful manner. WAE includes a micro-browser environment containing the following functionality:

- Wireless Markup Language (WML) – a lightweight markup language, similar to HTML, but optimized for use in hand-held mobile terminals.
- WMLScript – a lightweight scripting language, similar to JavaScript.
- Content Formats – a set of well-defined data format, including images, phone book records and calendar information.
- Wireless Telephony Application (WTA, WTAI) – telephony services and programming interfaces.

A.2 What is WTA?

Wireless Telephony Application (WTA) is a telephony extension to the WAP specifications that can provide in-device telephony related control mechanisms to advanced mobile network services. An operator's service provider can now develop services that manipulate mobile unit features (e.g. call control and the phonebook), and also trap various events (e.g. incoming call). WTA functionality allows operators to provide personalized services to a large subscriber base.

The WTA framework extends the Wireless Application Environment (WAE) by adding [18]:

- An interface from WTA-WML and WMLScript to a specific set of local, telephony related, functions in the client. This interface is called the "Wireless Telephony Applications Interface" [16].
- Network event handling. This means that events originating from the mobile network could be detected by the WTA user-agent and actions in response to the events could be defined.
- A repository, which is a storage container, used by the WTA user-agent, that persistently stores content that executes WTA-services. The purpose of the repository is to fulfil the real-time requirements that are placed on the execution of WTA services.

- A model for WTA user-agent state and WTA context management.
- A mandatory security model.

A.3 Architectural Overview

The WTA architecture primarily defines the components required in a client device to make telephony services available to WAP content. It also defines the role played by certain network elements. The components that constitute the WTA framework are:

- WTA user-agent
- WTA server
- WTA Interface libraries
- Repository

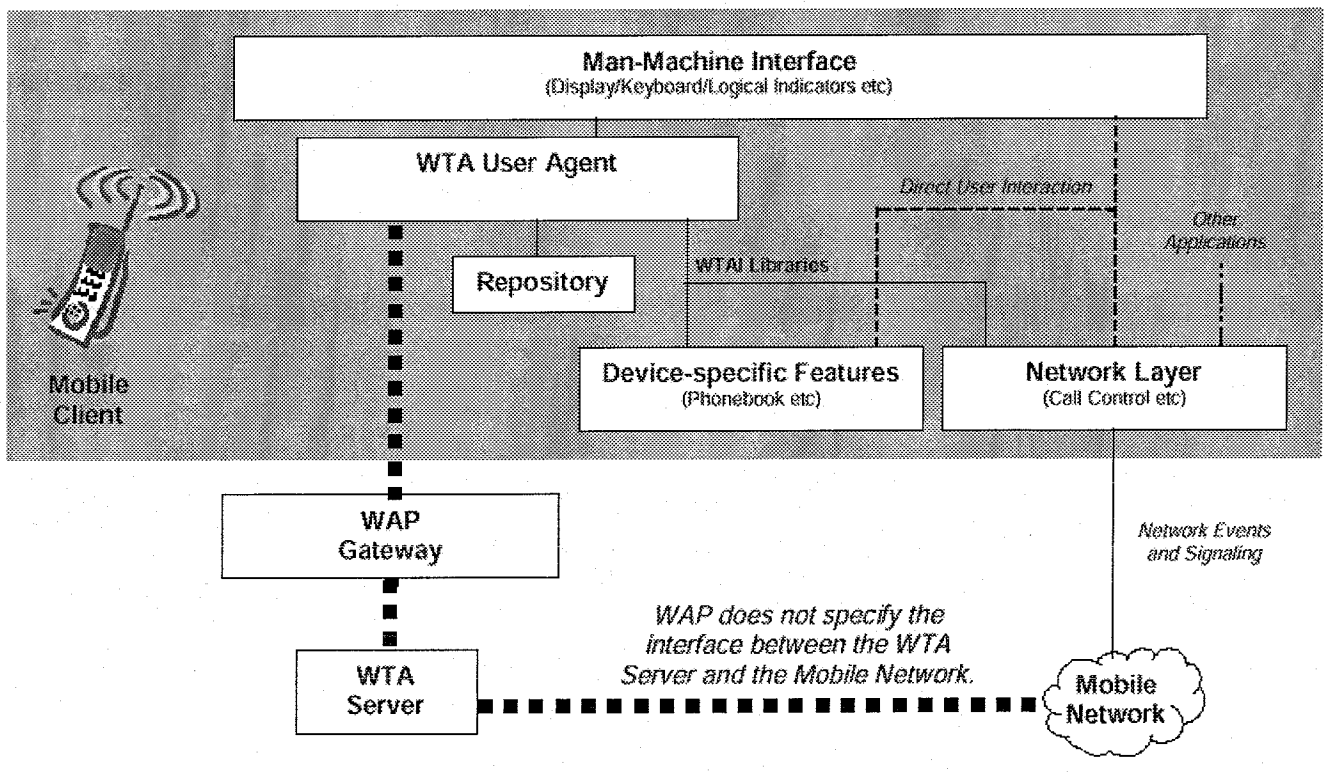


Figure 12. Overview of the WTA architecture [18]

The figure above (Figure 12) describes one possible configuration of the WTA framework. The components are briefly described in following sections.

A.3.1 WTA User-Agent

Figure 12 above illustrates how the WTA user-agent, the repository and WTAI interact with each other and other entities in a WTA-capable mobile client

device. The WTA user-agent is able to retrieve content from the repository and WTAI ensures that the WTA user-agent can interact with mobile network functions (e.g. setting up calls) and device specific features (e.g. manipulating the phonebook). The WTA user-agent receives network events that can be bound to content, thus enabling dynamic telephony applications.

Network events that will be available to the WTA user-agent are those that are the result of actions taken by services running in the WTA user-agent itself. Telephony events initiated from outside the device are also passed to the WTA user-agent, as are network text message events that are not explicitly directed towards another user-agent (e.g. events intended for a SIM). This means, for instance, that network events caused by the WML user-agents will not affect the WTA user-agent.

The WTA user-agent is quite similar to a WAE user-agent in that it has the ability to execute and present WML and WMLScript content to a user. In fact, it is a WAE user-agent in almost every respect including executing content within the boundary of a well-defined context. The primary difference is that a WAE user-agent only retrieves its content via the WAP gateway and only has access to the WTAI Public Library functions. The functions expose simple functionality such as the ability to place a call, but does not allow fully featured telephony control. Only a WTA user-agent is able to fully control the telephony features of the device. In particular, the WAE user-agent is not able to receive and react to telephony and network events.

A.3.2 WTA Server

A WTA server can be thought of as a web server where WTA content and services are hosted. The primary distinction from any other origin server is that the mobile operator regards it as a 'Trusted Content Server'. The reasons a WTA server must be a trusted server will become clear later in section A.5.

Like an Internet web browser, a WTA user-agent uses a uniform resource locator (URL) to reference content on the WTA server. A URL can also be used to reference an application on a web server that is executed when it is referenced. A WTA server may also make use of this concept. By referencing applications on a WTA server it is possible to create services that use URLs to interact with the mobile network (e.g. an IN-node) and other entities (e.g. a voice mail system). This is possible because, as can be seen in Figure 12, the WTA server has a direct connection to the mobile network. This mechanism provides a powerful model to seamlessly integrate services in e.g. the mobile network with services executing locally in the WAP client.

A.3.3 WTA Interface Libraries

WTAI is a set of function libraries that are accessible either via a WML-deck or WMLScript. As can be seen in Figure 12, services that execute in the WTA user-agent have access to WTAI functions.

WTAI is partitioned in three function libraries:

- Public WTAI

- Network Common WTAI
- Network Specific WTAI

Services that execute in the standard WAE user-agent only have access to the Public WTAI function library which make it possible for third party applications to use simple functions like making a call. These functions require explicit user verification. A more detailed description will be presented in section A.6.1.

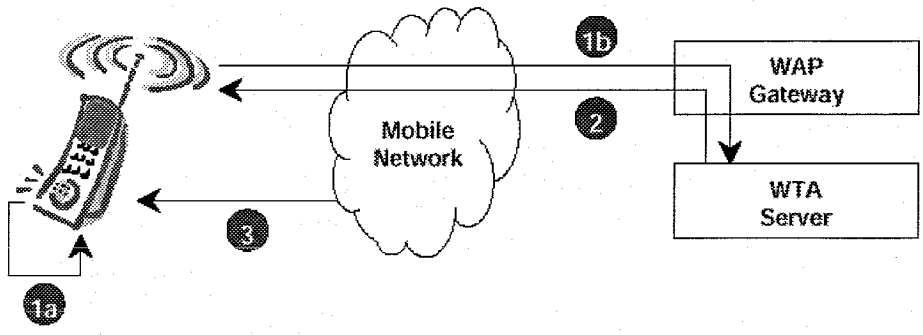
A.3.4 Repository

The repository is a persistent storage module within the mobile terminal that may be used to eliminate the need for network access when loading and executing frequently used WTA services. The repository also addresses the issue of how a WTA service developer ensures that time-critical WTA events are handled in a timely manner.

The content can be loaded into the repository as a result of a Service Indication (push) or by an end-user request to the WTA server (pull).

Only WTA applications (that is, content loaded or otherwise received from the WTA server) may access the repository. A more detailed description will be presented in section A.6.2.

A.4 WTA Services



Explanations of numbered items:

1a Access to a URL (via the repository)
1b Access to a URL (via the WTA server)
2 Service Indication (Push)
3 Network event (transformed to WTA event in client)

Figure 13. Initiation of WTA services [18]

WTA services are what the end-user ultimately experiences from using the WTA framework. A WTA service appears to the client in the form of various content formats, e.g. WTA-WML, WMLScript etc. The WTA user-agent executes content that is persistently stored in the client's repository or content

retrieved from a WTA server. The framework also allows the WTA user-agent to act on events from the mobile network.

Figure 13 shows the possible ways of initiating a WTA service in the WTA user-agent. A Service Indication contains a URL for a service that the user can choose to start immediately, or postpone for later handling. A typical example would be an end-user that is notified about an incoming e-mail with an URL that she can choose to open directly or later.

A.5 WTA Security

Telephony services have a stronger security requirement than other types of WAP services. Because of the nature of the operations possible when executing WTA content it must be ensured that only authorized WTA services are allowed to execute. The following architectural decisions to ensure security are taken:

- The WTA user-agent uses a secure WDP port on the WAP gateway for the WSP session it needs. The WTA user agent is also required to discard any content received outside a session established over a non-secure port. The term for a WSP session established over the secure port is now a WTA session
- The WAP gateway has to ensure a secure link between itself and the WTA server, either using a private network or by means of strong authentication mechanisms
- The user configures the mobile phone to specify user permissions for the execution of different WTAI library functions. There are three user permission types.

The user permission types are summarized in the table below. An executable is any entity, which calls WTAI functions.

Permission	Description
Blanket permission	The user giver blanket permission to the executable for the specified WTAI function and the executable subsequently uses the user's original permission for the identified subsequent WTAI functions whenever the executable is running.
Context permission	The user gives permission to the executable for the specified WTAI function during a specific run time context of an executable, and the executable subsequently uses the user's permission for the identified subsequent WTAI functions whilst the executable context is still running.
Single action permission	The user gives a single permission to the executable for the specified WTAI function; if the executable subsequently wishes to repeat the WTAI function it must again request the user's permission for the identified subsequent WTAI function.

Table 1. User Permissions

The duty of providing an acceptable level of security lies with the mobile network operator. It is possible to do this only if:

- The mobile network operator has administrative control over the WTA server or at least has a trust relationship with the provider of the WTA server
- The mobile network operator controls or at least has a trust relationship with the provider of a WAP gateway that is used by its subscribers

A.6 WTA Interfaces

As already mentioned in section A.3.3, WTAI provides a means to create telephony services. WTAI is partitioned in three function libraries:

- Public WTAI
- Network Common WTAI
- Network Specific WTAI

A.6.1 Public WTAI Functions

Public WTAI are simple functions that are available to third party applications executing using the standard WAE user-agent. These functions require explicit user verification. The public WTAI library includes the functions listed in Table 2 below.

WTAI Library	Available Functions
Public WTAI	Make a call Send DTMF tones Add a new phonebook entry

Table 2. Public WTAI functions

A.6.2 Network Common WTAI

Network Common WTAI covers the most common features that are available in all networks. They are only accessible from the WTA user-agent. The functions included in the Network Common library are listed in Table 3.

WTAI Library	Available Functions
Voice Call Control	Set-up a new call Accept an incoming call Release a call Send DTMF sequence Retrieve parameters for a specific call
Network Messages	Send network text

WTAI Library	Available Functions
	List network messages Remove network message Mark message as read
Phonebook	Write phonebook entry Search phonebook entry Remove phonebook entry Change an existing phonebook entry
Call Logs	Read "last dialled numbers" log Read "missed calls" log Read "received calls" log
Miscellaneous	Change logical indicator Terminate user-agent context Read context protection mode Set context protection mode

Table 3. Network common WTAI

A.6.3 Network Specific

Network specific functions are only available in certain types of networks. Operator specific functions may also reside in this set. Currently there are libraries for ANSI 136, PDC and GSM. The GSM specific library is shown in Table 4 below. For GSM devices, the GSM specific library is mandatory. The same rule obtains for ANSI 136 and PDC devices. In this document the availability of the GSM specific library is assumed because in Europe the current and future networks are based on GSM.

WTAI Library	Available functions
GSM Specific	Put a call on hold Make a held call active again Transfer an active call Deflect an unanswered call Join/create a multiparty call Retrieve a party from a multiparty call Send an USSD message Get network information

Table 4. GSM Specific WTAI

A.7 The Repository

The repository is a persistent module within the client that may be used to eliminate the need for network access when loading and executing frequently used WTA services. This forms an important component of the WTA framework because it removes the latency associated with pulling the content over the network before executing it.

The repository consists of channels and resources. A channel contains a set of links to resources. Resources are data that have been downloaded to the client. The relationship is shown in Figure 14.

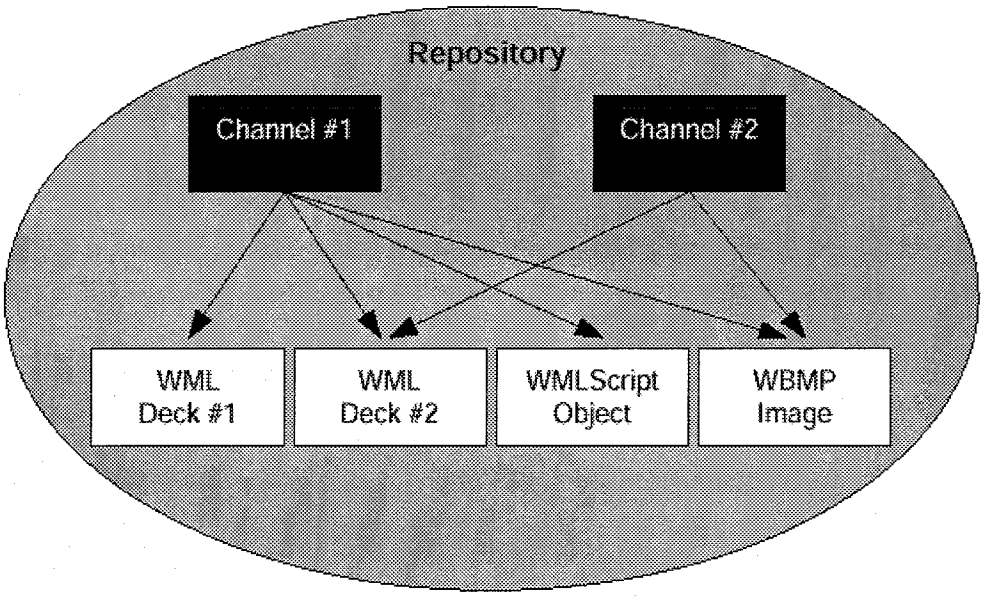


Figure 14. The Repository [18]

A resource may be referenced by more than one channel. Channels have a freshness lifetime beyond which time they are considered stale. A channel is accessible to the WTA user-agent only after all the resources referenced by it are available in the repository.

The channels can be loaded into the repository as a result of a service indication or by an end-use request to the WTA server. The minimum size of the repository is not yet specified. The WTA-group is trying to define a suitable size.

A.8 WTA Events

Network events are events that originate from the mobile network. Network events must be transformed from their abstract form into a predefined format called WTA events.

The WTA user-agent can be programmed to take certain actions when receiving a specific WTA event. Considering these events may require a real time response, event-handling content should be executed from the repository. A WTA event can be associated with any content in the repository.

A WTA user-agent can respond to a WTA event in using two different methods:

- **Global bindings** – The repository establishes a link between a WTA event and a service. When an event occurs, the service is initiated from the repository.

- **Temporary bindings** – If content is executing and an event occurs, this event is handled dynamically within existing context. A temporary binding overrides a global binding the event may have.

The following events are available:

WTAI Library	**Area**	
Public WTAI	None	
Network common	Voice Calls	Incoming call Call cleared Call connected Outgoing call Call alerting DTMF sent
	Network messages	Message sent status Incoming message
	Miscellaneous	Networkstatus changed
Network specific	GSM	Call held Call active USSD received

Table 5. WTAI events

B SIM Application Toolkit

B.1 Overview

As its name suggests, the SIM Application Toolkit (SAT) is based around the use of a smart card, namely the Subscriber Identity Module (SIM) card in a GSM phone. The SAT is a protocol that enables an application running on a SIM to interact with a mobile device. The commonly implemented features supported by the protocol are listed below:

- SMS point to point
- Extension of the mobile device menu
- Proactive handling of the mobile device MMI
- Proactive polling
- Call set up

The following features are optional depending on the device class:

- Call control
- Proactive request of location information
- Refresh of the GSM session
- USSD handling
- Event handling
- Timers
- Handling of multiple cards

The SAT protocol is part of the GSM phase 2+ specification with optional levels of functionality. The service environment has converged with a common lowest level of functionality of implementation. It may be referred to as an undefined industry standard for SAT devices. An overview of the common service environment obtained with SAT is given by Figure 15.

The SAT enables operators to send applications over the air as SMS or broadcast messages to update SIM cards with adapted or new services. Because with SAT the application is implemented in a SIM, this provides an even more limited environment than the mobile device.

The SIM interacts with the mobile device via proactive sessions. A proactive session is initiated by the SIM. The SIM indicates that a proactive session is to be started and the mobile device executes it by fetching the first proactive

command. This procedure is then repeated after each command response from the device until the SIM ends the session.

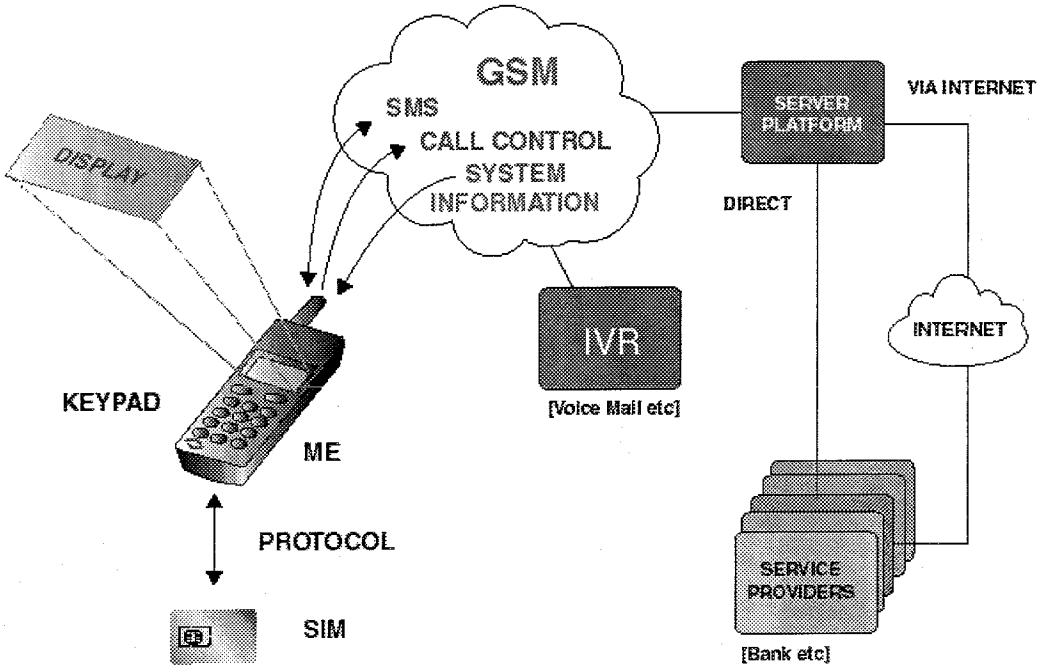


Figure 15. SAT service environment [7]

The most common way to start applications on the card is based on the envelope command. The command is used to transfer information to the SIM. Several functions of the envelope command are specified. The information sent is used as stimuli to start the application.

The SAT provides functionality to supply a number of items to be integrated in the mobile device menu system. This gives the user the opportunity to select one of the menu items to start an application. The update of the menu is usually being done during the initial phase when the terminal profile is sent to the SIM.

The man-machine interface features accessible via SAT are limited. However, to some extent they are found suitable enough for the limited mobile device. The main functions are:

- Presentation of a text with or without user input
- Presentation of a list of text items for user selection
- Playing of a tone

B.2 Call control

The WTA user-agent can fully control calls. The SAT offers the call control functionality as listed in Table 6.

Function	Description
Set up call	Initiates a voice call
Send DTMF	Sends a DTMF sequence
Provide Local Information	Sends current local information to the SIM

Table 6. SAT Call Control functions

In contrast to WTA the SAT does not support incoming call control and GSM specific functions like putting a call on hold and creating multiparty calls.

With SAT it is possible to use call control. When this service is activated by the SIM, all dialed digit strings, supplementary service control strings and USSD strings are first passed to the SIM before the mobile device sets up the call, the supplementary service operation or the USSD operation. The mobile device shall also pass to the SIM at the same time its current serving cell. The SIM has the ability to allow, bar or modify the call, the supplementary service operation or the USSD operation. The SIM also has the ability to replace a call request, a supplementary service operation or a USSD operation by another call request or supplementary service operation or USSD operation. This function is not available in WTA and may not be relevant for implementations in WTA.

Both WTA and SAT support provisioning of positioning information.

B.3 Messaging control

Table 7 below shows the SAT Messaging Control functions.

Function	Description
Send Short Message	Sends a short message to the network
Send USSD	Sends an Unstructured Supplementary Services Data (USSD) string to the mobile network
Send SS	Sends supplementary service control string to the network
MO Short Message Control	This feature gives the possibility to filter addresses of outgoing SMS. Every short message is passed to the SIM before it is send.

Table 7. SAT Messaging Control functions

The WTA user-agent can read, send and remove network messages. The SAT also only supports sending messages. Reading and removing messages are not specified in the SAT, but should be possible when messages are stored on the SIM. Normally the implementation platforms for SAT allow access to all SIM content. However, this is dependent on the access conditions of the SIM. Messages stored in the mobile device memory are not accessible from the SIM.

The SAT offers SMS point-to-point download. This feature gives the possibility to send SMS directly to an application on the card. This message goes directly to the application on the card. A similar function is provided by the push function in WAP.

The MO (Mobile Originated) Short Message Control function in the SAT can be used to filter outgoing SMS. A similar function is not implemented in WTA. Sending SS messages is also not possible in WTA.

B.4 Feature control

The SAT offers the feature control functionality as listed Table 8.

Function	Description
Language Notification	Informs the mobile device about language being used
Play tone	Plays a tone in the mobile device
Set up menu	Adds items to the mobile device menu system

Table 8. SAT Feature Control functions

In contrast to WTA SAT offers a function to play a tone in the mobile device. The SAT offers a function to add items to the mobile device menu system. This gives the possibility to address applications directly from the mobile device menu. This functionality is not supported by WTA. In WTA the user first has to open a WML deck containing a link to the service before starting a service.

Manipulating the phonebook, which is possible with WTA, is not defined in the SAT. But if the phonebook is stored on the SIM, it should be possible to access the phonebook because applications running on the SIM can access all data on the SIM. However, this depends on the SIM and mobile device used. The same story applies for call log handling.

The SAT offers some other functions like displaying text in the mobile device terminal and getting input from the user. These functions are not supported in WTA, but are supported by the WAP user-agent. As this is not part of the WTA specification these function are not described in detail here.

B.5 Event handling

A set of events for the mobile device to monitor can be supplied by the SIM using the proactive command set up event list. If the SIM has sent this command, and an event which is part of the list subsequently occurs, the mobile device informs the SIM using the procedure below, relevant for that event. The possible events are listed in Table 9.

Event	Description
MT Call	Incoming call
Call connected	Call connected

Event	Description
Call disconnected	Call disconnected
Location status	Location status has changed
User activity	Detection of some user activity
Idle screen available	Idle state
Language selection	Change of current selected language

Table 9. SAT Events

The location status event is not supported in WTA. The user activity, idle screen available en language selection are also not supported in WTA, but these events seem not be relevant for WTA applications.

B.6 SAT versus WTA

It is not easy to compare SAT with WTA because of the different techniques. It also depends on the application which technique is the preferred one. Below some advantages and disadvantages are described to position SAT versus WTA.

A disadvantage of the SAT is the closed system issue. Because there are many different classes of the SAT protocol not all handset allow all applications. And because of the vendor specific implementations of the SIM with proprietary solutions, applications have to be built according to each type of card. The WAP environment provides a web-centric/thin-client environment with a standardized framework, which is easier to manage and maintain. There are moves to address this interoperability issue.

The SAT offers an advantage in the security area. But WAP is making up ground. The current WAP specifications include enhanced security features for the wireless transport layer section (WTLS). As this is outside the scope of WTA this item will not be discussed here.

At this moment there are more handsets that support SAT than handsets that support WAP. This means that SAT has a bigger installed base, which is an advantage. But this advantage depends on how long it will take WAP-enabled mobile devices to penetrate the market. More and more vendors start selling WAP-enabled mobile devices. Ericsson for example has announced that from 2001 all new mobile devices will be WAP-enabled.

The last disadvantage of SAT is the limited bearer. The SAT only supports SMS, while WTA is independent of the bearer.

As said in the beginning of this subsection, the SAT is not a real alternative for WTA. It depends on the application, which of the two techniques is more suitable. Services that remain stable and that need lot of security could be run on a card. But given the limitations of the bandwidth, the limited run time environment, the closed system item and the smaller functionality, WAP with WTA offers more functionality.

Prepared (also subject responsible if other)		No.		
ETM/BL/IB Peter de Vries		ETM/BL/IB-01:0117 Uen		
Approved	Checked	Date	Rev	Reference
ETM/BL/IB Patrick W.J. Essers		12/06/01	A	H:\etmptvr

C Mobile Execution Environment

This appendix describes the Mobile Execution Environment (MExE) in more detail. Some parts of this text, especially the descriptions of the techniques, are directly copied from the specification [1].

C.1 Overview

MExE is a 3GPP standard [1] that specifies the application environment in mobile phones, allowing applications to run in a browser environment or as a stand-alone application in the mobile terminal.

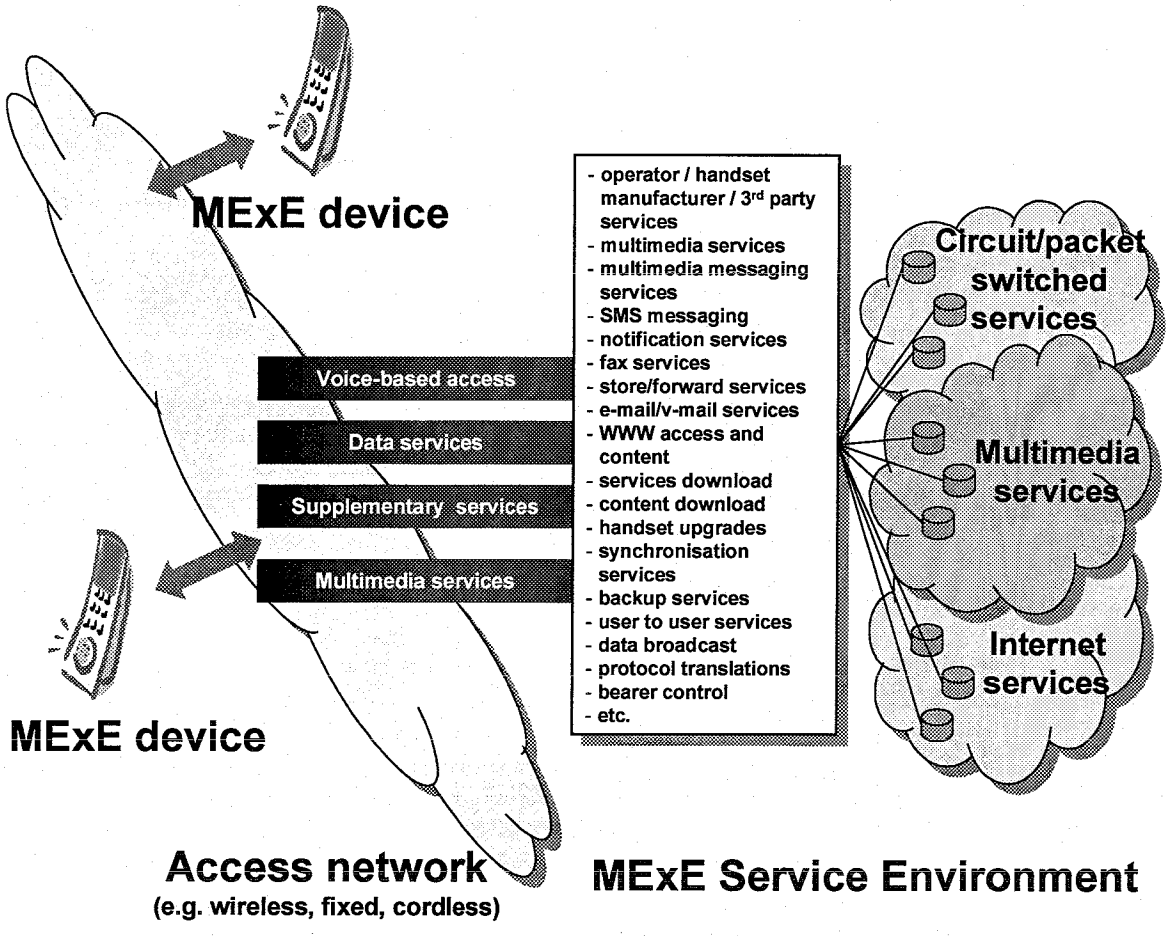


Figure 16: Generic MExE architecture [1]

The architectural model above shows an example of how standardized transport mechanisms are used to transfer MExE services between the MExE device and the MExE service environment, or to support the interaction between two MExE devices executing a MExE service.

The MExE service environment could, as shown in Figure 16, consist of several service nodes each providing MExE services that can be transferred to the MExE device using mechanisms such as mobile network protocols, Bluetooth, infrared, serial links, wireless optimized protocols, standard

Internet protocols. These service nodes may exist in the circuit switched domain, packet switched domain, IP multimedia core network subsystem or in the internet space (e.g. SMS service centers, multimedia messaging servers, internet servers etc.). The MExE service environment may also include a proxy server to translate content defined in standard Internet protocols into their wireless optimized derivatives.

The MExE specification categorizes devices by giving them different MExE classmarks. Currently there are three classmarks specified:

- MExE classmark 1 - based on WAP - requires limited input and output facilities on the client side, and is designed to provide quick and cheap information access even over narrow and slow data connections. Because this classmark only refers to the WAP specification this classmark introduces nothing extra to the WTA functionality and is therefore not discussed further in this section.

- MExE classmark 2 - based on PersonalJava - provides and utilizes a run-time system requiring more processing, storage, display and network resources, but supports more powerful applications and more flexible MMIs.

- MExE classmark 3 – based on J2ME CLDC and MIDP environment – supports Java applications running on resource constrained devices.

Content negotiation allows for flexible choice of formats available from a server or adaptation of a service to the actual classmark of a specific client device. Capability negotiation between the MExE Service Environment and MExE device (including MExE classmark), supports the transfer of capabilities between the client and the server.

C.2 MExE Classmark 2

Classmark 2 specifies Personal Java enabled devices with the addition of the JavaPhone API.

The Personal Java application environment is the standard Java environment optimized for consumer electronic devices designed to support World Wide Web content including Java applets. The Personal Java API is a feature level subset of J2SE with some Java packages optional and some API modifications necessary for the needs of small portable devices (for example an optimized version of the Abstract Windowing Toolkit targeted to small displays).

MExE classmark 2 devices are based on the API for PersonalJava, which defines the required and optional components of PersonalJava/JavaPhone APIs that are used to realize a classmark 2 compliant MExE device. The APIs primarily define the functions available to a Personal Java based MExE device such that services specified in the form of Java classes and interfaces can control such a MExE device in a standardized way.

Many aspects of the MExE classmark 2 API specification are optional. Services and applications shall be able to determine the presence of optional parts of the functionality. When optional parts of the functionality are implemented, the API shall be supported.

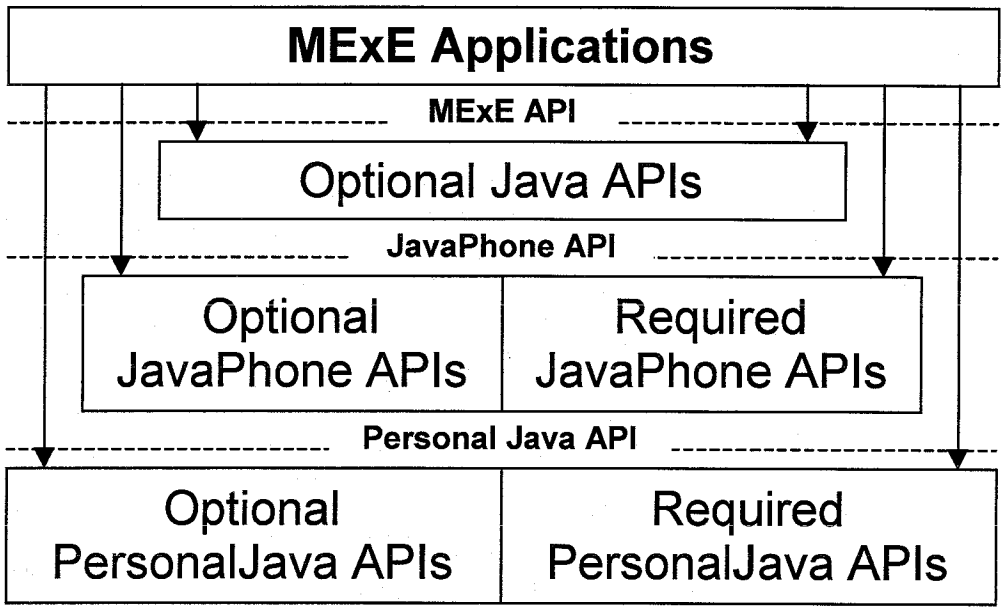


Figure 17. Overview of APIs used by MExE classmark 2 [1].

The functional architecture of a Java MExE classmark 2 device is shown in Figure 17. Java applets, applications, and services access functionality via the MExE PersonalJava API. The MExE PersonalJava API is based on a combination of optional Java APIs approved by Sun Microsystems and the Wireless Profile of the JavaPhone API [12] as defined by the JavaPhone Expert Group. The JavaPhone API is based on the PersonalJava API [14] defined by Sun Microsystems.

The JavaPhone APIs extend the PersonalJava APIs to provide functionality unique to telephony devices. Java MExE devices shall support all APIs specified as required by the Wireless Profile in the JavaPhone API specification. All APIs that are optional in the Wireless Profile shall be optional in Java MExE devices.

A Java MExE device shall not be required to support any other Java APIs.

To provide backward compatibility to MExE classmark 1, i.e. allow access to services designed for MExE classmark 1 devices, classmark 2 devices must feature a pre-installed or pre-loaded WAP browser.

C.2.1 Call control

Table 10 shows the call control functions that JavaPhone, and thus the MExE API, offers. Only the core and a part of the mobile package are mandatory. The optional packages are written in italic.

JTAPI package	Interface	Available Functions
Core Package		Set-up a new call Answer incoming call Release a call
Call Control Package	*CallControlCall*	*Set-up conference call* *Transfer call* *Release all calls at once* *Set-up consultation call*
	CallControlAddress	*Set call forwarding* *Set do not disturb* *Set message waiting*
	CallControlConnection	*Accept a call* *Reject a call* *Redirect a call* *Transfer a call* *Park a call*
	CallControlTerminal	*Retrieve a call previously parked*
	CallControlTerminal-Connection	*Put a call on hold* *Join a multiparty call* *Retrieve a party from a multiparty call*
Mobile	Mobile Provider	Get provider-information
	Mobile Address	Get/set callwaiting option
	Mobile Terminal	Get terminal information Send DTMF tones
	Mobile Network	Get network information
	Mobile Selection	*Get network selection info* *Set network selection mode*
	Mobile Radio	*Radio management* *Get signal strength*

Table 10. JTAPI call control functions

The complete JTAPI provides more functions than WTA, but most functions are optional. All WTA functions are covered by the (complete) JTAPI.

C.2.2 Messaging control

The table below shows the messaging-related functions.

API package	Interface	Available Functions
Network Datagram		Send message Receive message Read message

Table 11 MExE messaging control functions

JavaPhone offers the Network Datagram API, which provides for transport independent addressing and delivery of messages. Applications send and

receive messages using addresses consisting of service name and service location. The physical network or bearer, used to deliver the message is selected by the device when the message is sent. For example, on a wireless phone the GSM short message service (SMS) can be used.

Compared to WTA the same functions are possible, except that it is not possible to send USSD messages. While in the WTA framework functions to list and remove messages are present, in MExE these functions have to be implemented.

C.2.3 Feature control

The JavaPhone APIs offer many device-related functions, which will not be described in detail here. Instead of describing every function, an overview of the feature-related APIs is given in Table 12. Optional APIs are written in italic.

JavaPhone API	Description
Addressbook	Provides storage for and access to business card information. An application using the AddressBook API can locate and update contact information such as phone numbers, postal addresses, email addresses, and other information about individuals, groups, and organizations. In GSM phones, for example, the AddressBook API provides access to name and address information stored on the SIM card.
User Profile	Provides an application with information about the current user or owner of the device. For example, in a GSM phone, the User Profile API provides access to the SIM card information for the owner, which might include the user's name and other attributes that identify the user.
Calendar	Provides storage for and access to schedule information such as dates and task entries. CalendarEntries provide access to descriptions, dates, and repeat information. ToDo entries provide for access to notes and items to be done. Scheduling data stored in a device implemented with the Calendar API is accessible to a user who can easily locate dates and times or make modifications to this information.
Install	The JavaPhone extension includes an interface to install and remove applications.
Power Monitor	Monitors the available power levels of the device and notifies when the device is changing power states. Using the Power Monitor API, the application can tailor its activities to the available power. The Power Monitor API allows an application to determine the state of power on the device. It allows an application to retrieve the current battery level, an estimate of remaining battery life, and whether an external power source is being used. The API also provides notification when

JavaPhone API	Description
	battery power is running low and a service is about to be terminated as a result.
Power Management	Applications can use the Power Management API to determine the power state of the device. For example, the ON power state can be identified. The API allows an application to be aware of transitions in the power states, and allows it to respond to various conditions such as full power, power managed for efficiency, suspend, and sleep states.
Communications	*The Communication API extension to the PersonalJava platform is used by applications to access serial or parallel ports on the device.*
Phone	*The Phone Extension Package defines a number of components standard to traditional telephone-set hardware. These components include speakerphone, microphone, display, buttons, ringer, handset, and lamp. Each component exports a standard interface to control its attributes. For example, the ringer permits applications to control its volume, while the buttons permit applications to simulate pressing buttons.*

Table 12. MExE Feature control functions

As can be seen in Table 12, the JavaPhone APIs offers much more feature control functions than WTA. However some functions, like displaying text, are covered by the WAP user-agent. The JavaPhone APIs cover all WTA functions.

C.2.4 Event handling

Table 13 presents the possible events in JavaPhone. Optional packages are written in italic.

JavaPhone API	Area	Event
Core Events	Call-related	Call active Invalid call
	Connection-related	Alerting Connected New connection object created Disconnected Connection failed Connection in progress Connection state unknown
	Provider-related	Provider in service Provider out of service Provider shutdown

JavaPhone API	Area	Event
	Terminal-related	Connection created Connection active Connection dropped Connection passive Connection ringing Connection unknown
Call control	*Address-related*	*Do Not Disturb feature changed* *Call Forward feature changed* *Message Waiting feature changed*
	Call-related	
	Connection-related	*Alerting* *Dialling* *Disconnected* *Connection established* *Connection failed* *Initiate connection* *Network alerting* *Network reached* *Offering* *Connection queued* *Connection unknown*
	Terminalconnection -related	*Connection bridged* *Connection dropped* *Connection held* *Connection in use* *Connection ringing* *Connection talking* *Connection unknown*
	Terminal-related	*Do Not Disturb feature changed*

Table 13. JavaPhone events

The optional Phone Events package defines the specific event transitions associated with the physical terminal characteristics like pressing a button.

Just like the other characteristics the possible events cover the possible events in WTA.

C.3 MExE Classmark 3

MExE Classmark 3 devices are based on the J2ME Connected Limited Device Configuration (CLDC) with the Mobile Information Device Profile (MIDP). All APIs defined by CLDC and MIDP shall be supported by a MExE classmark 3 device.

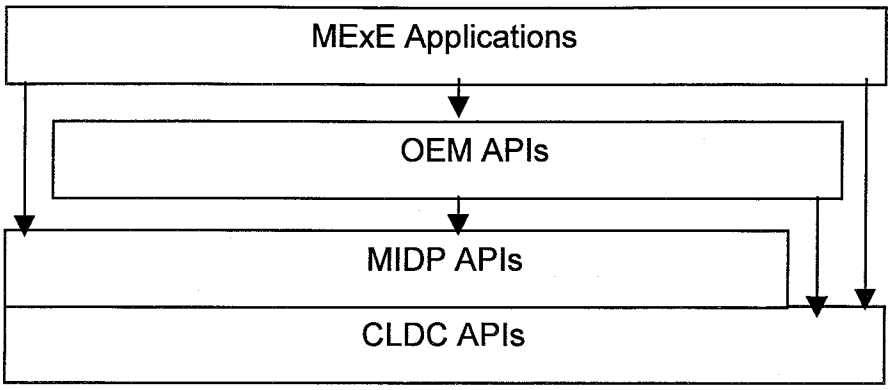


Figure 18. Functional architecture of a Classmark 3 MExE device [1].

The functional architecture of a Classmark 3 MExE device is shown in Figure 18. The MExE API is based on the combination of CLDC APIs and MIDP APIs. OEM specific APIs are outside the scope of MExE specification. CLDC and MIDP APIs are defined in J2ME specified by Sun Microsystems [10,13].

J2ME CLDC and MIDP addresses a large market of resource-constrained devices and is aimed to provide complete end-to-end solution for creating dynamically extensible networked products and applications. It allows the use of Java programming language as the standard platform for secure delivery of dynamic content for the extensible next-generation devices.

In order to fit into various types of the devices and support extensibility, J2ME defines in a configuration a minimum platform with a virtual machine and minimum libraries, which are available on all devices of the similar class. In a profile J2ME addresses the specific demand of a certain category of the devices allowing additional APIs. A profile is implemented on top of configuration. Classmark 3 MExE device shall be based on the following types of configuration and profile: Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP).

However, these configuration and profile do not specify telephony functions and therefore this classmark is not discussed in more detail here.

C.4 MExE versus WTA

As can be read in the sections above, MExE offers a much richer set of functions for application programming than WTA. Both classmarks 1 and 2 cover all the WTA functionality. MExE classmark 3 devices do not support telephony functions according to the standard. Whereas WAP incorporates some scripting, graphics, animation and text, MExE allows full application programming which requires significant processing resources on the mobile client. That's why MExE is primarily aimed at the next generation of powerful smart phones.

D Detailed Descriptions Applications

This appendix describes the applications discussed in chapter 1 in more detail. The first applications are described in more detail than the latter ones to get an idea how these applications can be implemented. For the latter applications it is enough to understand the concept.

D.1 Incoming Call Selection

D.1.1 Description

The Incoming Call Selection service is started when an incoming call is detected in the client. When an incoming call is detected, the service presents a menu with various call handling options to the user. The menu could for the example look like this:

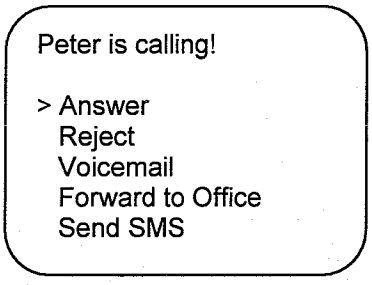


Figure 19. Example of an Incoming Call Selection Menu

D.1.2 Architecture

The service can be implemented as follows. The menu used to display the call handling options is a channel which resides in the repository of the mobile client and which has `wtaev-cc/ic` as `eventid` value. On the occurrence of the incoming call event (`wtaev-cc/ic`), this channel is processed and the content specified by the first `href` link in this channel is processed.

The menu is a WML deck with the various call handling options as links and a WML script to replace the caller id with the name, if this id is stored in the phonebook. Otherwise the caller id is displayed. In the example below the user answers an incoming call.

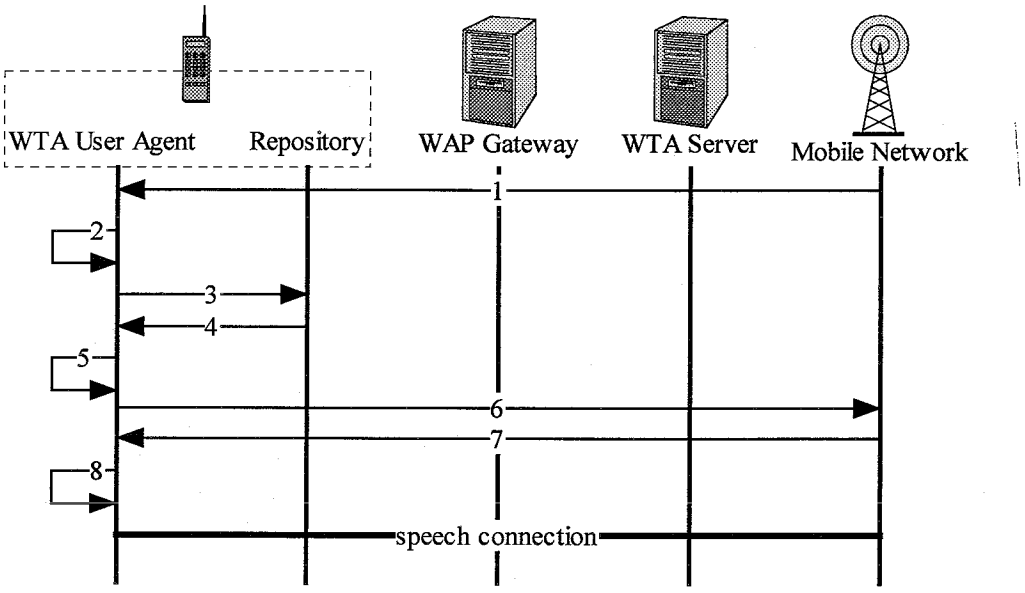


Figure 20. Incoming Call Selection trace.

1. The mobile network receives an incoming call and sends a "Call indication" to the mobile subscriber

2. In the client, the incoming call WTA event (`wtaev-cc/ic`) is generated. The repository is consulted in order to find a dedicated channel. The channel provides the URL to the "Incoming Call Selection Menu" stored in the repository

3. The user-agent requests the content from the repository

4. The repository returns the requested content

5. The content is loaded and starts executing. The service presents the list of options to the user from which the user can choose how to proceed with the call in progress. In this example the user selects the answer option.

6. A "Connect" request is sent to the mobile network (the invoked WTAI function communicates with the mobile network).

7. A "Connect Acknowledgement" is generated in the mobile network. A result code indicating the outcome of the call is generated internally in the phone.

8. A speech connection between the mobile network and the client is established.

The objects that have to be implemented to make this service work are the WML deck that describes the menu and a WMLScript file that contains functions to handle the call.

D.1.2.1 WML deck

The implementation of the WML deck contains a `menuCard`, which describes the menu by implementing links to handle the call. The answer and reject options refer respectively to the `answerCall` and `rejectCall` functions. The forward to voicemail and forward to other number options both use the `forwardCall` function, but in the first case the voicemail number is passed as parameter and in the second case a card asking for a number is displayed before the function `forwardCall` is called. The send SMS option displays a card to enter a text and then sends this text by calling the `sendSMS` function. When the menu card is displayed, the function `findName` is called to set the parameter `whoCalls`, which shows the calling id. The WML deck also contains cards to display the state of the call handling and to display error messages, for example cards that display "answering call" and "call connected" messages.

D.1.2.2 Scriptfile

The scriptfile contains the following functions:

- `answerCall` calls the `WTAVoiceCall.accept` function and sends the browser to the `answeringCall` card
- `rejectCall` calls the `WTAVoiceCall.release` function and sends the browser to the `rejectingCall` card.
- `forwardCall` calls the `WTAGSM.deflect` function and sends the browser to the `forwardedCall` card.
- `sendSMS` calls first calls the `rejectCall` function and after that the `WTANetText.send` function and sends the browser to the `messageSent` card.
- `findName` uses the `WTAPhoneBook.search` function to search in the "number" field for the `callerId` value and replaces the `whoCalls` parameter in the `menuCard` with the corresponding name if found or with `callerId` otherwise.

In this implementation the role of the WTA server is to store the service. The user needs to access the WTA server to install the channel on the mobile terminal. Theoretically, the WTA server is not necessary anymore for the usage of this service after the initial installation. The WTA is necessary when the user wants to change the menu, as also can be read in the next section.

D.1.3 Implementation Issues

According to the WTA specification, it should be possible to implement the service as described above. However the `WTAGSM.deflect` function has to use a corresponding function in the bearer layer. In case of the usage of GSM as bearer, this function should make use of the GSM function `deflect` which

is optional in the GSM specification and will therefore not be available in all GSM networks.

Because it takes a long time on most telephones to enter a message, especially within the WAP environment, where no predictive text input system is used, it would be easier if the user could select from a number of predefined messages. Predefined messages could for example be strings like "I am in a meeting now", "I'll call you back later" or "Please try again within an hour". This could be implemented by using a WML card containing these strings. This implementation has the disadvantage that it is not easy for the user to change these strings. To change one or more of the predefined strings, the WML card in the repository that contains the string has to be replaced. This means that the user has to access the WTA server to generate a new WML card and then load this card in the repository. It would be a better option to store these strings in a separate locally stored file such that only this file needs to be updated.

D.2 Call Screening Service

D.2.1 Description

The Call Screening Service is designed to automatically screen incoming calls on a mobile terminal based on a screen list that is maintained by the subscriber. Incoming calls that are screened can be forwarded to the voicemail depending on the screen list. With this service users can be interrupted only by calls from certain people. For example during a meeting a manager can forward all calls to his voicemail, except calls from his secretary.

D.2.2 Architecture

The service can be implemented by using a channel which resides in the repository of the mobile client and which has `wtaev-cc/ic` as `eventid` value. On the occurrence of the incoming call event (`wtaev-cc/ic`), this channel is processed and the content specified by the first `href` link in this channel is processed. This link refers to the `CallScreening` WML deck, which contains a WMLScript file `CallScreen`. The diagram below shows an example where an incoming call is forwarded.

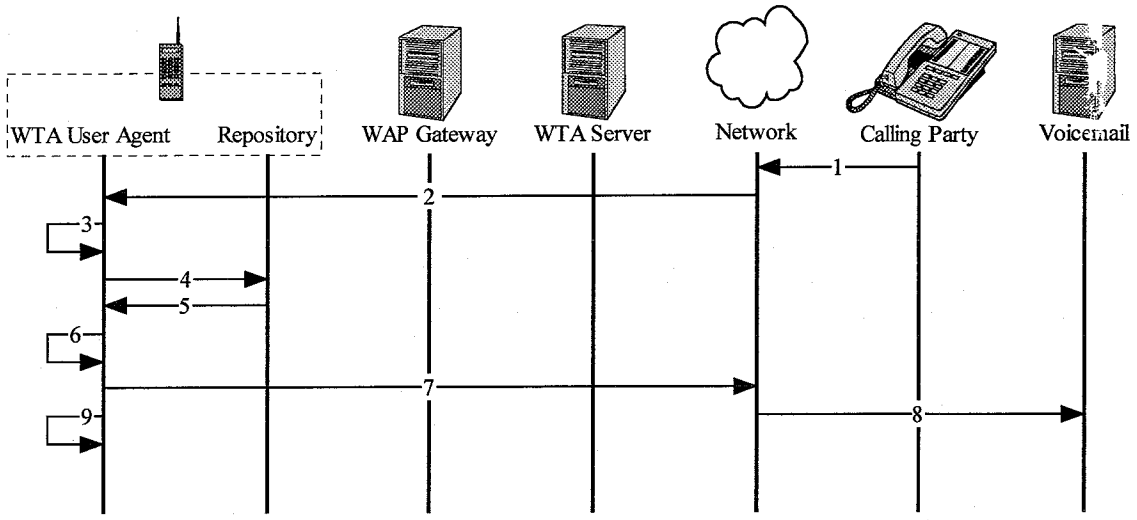


Figure 21. Call Screening trace.

1. The calling party dials the number of the mobile subscriber.
2. The mobile network receives the incoming call request and sends a "Call indication" to the mobile subscriber.
3. In the client, the incoming call WTA event (`wtaev-cc/ic`) is generated. The repository is consulted in order to find a dedicated channel. The channel provides the URL to the "Call Screening Service" stored in the repository.
4. The user-agent requests the content from the repository.
5. The repository returns the requested content.
6. The Call Screen script compares the incoming call id with that of the screen list and determines that the call should be forwarded.
7. A call forward request is sent to the network (the invoked WTAI function communicates with the network).
8. The mobile network forwards the call request to the voicemail system.
9. The Call Screen script updates the status of the screen and displays a message to the subscriber.

The implementation of this service consists of two objects in the client and an application on the WTA server, which are described below. In this section the application on the server is often denoted by the word servlet. It is to be noted that these applications could be servlets, but do not necessarily have to.

D.2.2.1 WML deck

The WML deck contains a card `CallScreening`. This card calls the function `screenCall` that is described in the next section. Furthermore the WML

deck contains cards to display the result messages of the `screenCall` function. An example of these messages is "1 Call Forwarded". Another result of the `screenCall` function is the `incomingCall` card, which displays options to answer the incoming call.

D.2.2.2 Scriptfile

The scriptfile contains the function `screenCall`. This function checks whether the passed call id has to be forwarded. When the incoming call has to be forwarded, `screenCall` calls the function `WTAGSM.deflect` and sends the browser to the resultpage. In the other case the browser is sent to the `incomingCall` card.

D.2.2.3 Servlet

The application on the server is used to install and change the service. To change the service, this means updating the screen list in this context, the user should be able to access the service on the server via a WAP session and edit the screen list. This means that the current screen list should be available on the server. When the user changes this list a new script has to be generated and installed on the mobile client.

D.2.3 Implementation Issues

It should be possible to implement the service as explained above. However, just like with the Incoming Call Selection the usage of the `WTAGSM.deflect` function depends on the implementation in the GSM network.

It might be interesting for the user to have a list of numbers that are allowed to interrupt. To implement this feature this list has to be encoded in the script. This can not be marked as a beautiful solution as to update the list a new script has to be generated and installed on the terminal. This implies that one of the server's tasks is generating scripts. It would be a better solution when one could define a table containing the numbers and store this table in the repository. The script should be able to access and update the table such that no communication with the WTA server is needed. This is nevertheless not possible.

It is possible to screen calls based on the presence of the number in the phonebook. This means that when a calling number is "unknown" it is forwarded to the voicemail. This can be implemented using the WTA phonebook library. However it is not possible to define a selection from the phonebook. For example: the phonebook contains the entries A, B, C and D. The user wants to forward all calls from A and B to the voicemail. It is not possible to link this service to the phonebook, because nor additional fields can be defined in the phonebook neither a table can be stored containing the index of the phonebook and an attribute.

Another interesting feature might be the usage of profiles. The user could for example define three profiles: @home, @work and @meeting. The @home profile forwards all calls except from friends to the voicemail. The @work profile forwards all calls from friends to the voicemail and the @meeting

profile forwards all calls except from the secretary to the voicemail. Using these profiles the user should be able to change quickly from one profile to another. In the current implementation support of profiles is possible. The profiles should reside on the server, and a change of profile implies a new channel installation. The change of profiles could be done automatically by using a schedule on the server. When the profile should be changed according to the schedule, a service indication could be sent to the client to install a new channel.

The call screening function as described above can also be extended to support forwarding to different numbers, based on the caller id. For example caller A is forwarded to voicemail A, caller B is forwarded to a colleague and other callers are forwarded to voicemail C. Combining this functionality with a schedule leads to the Automatic Diverting service. The Automatic Diverting service is described in the next section.

D.3 Automatic Scheduled Divert

D.3.1 Description

The Automatic Scheduled Divert service is designed to automatically divert and cancel diverts on a subscriber's mobile unit. The service is intended to remove the need for manual phone diversion before attending scheduled events. Using this service the users simply do not need to bother about setting and resetting diverts during meetings.

D.3.2 Architecture

The service exists of two main components: the calendar functionality and the divert functionality.

D.3.2.1 Calendar

The design of the calendar functionality is outside the scope of this report. The user uses the calendar component like any other scheduler to plan their day and to be aware of any appointments the user has. The typical actions performed on the calendar are adding, rescheduling and cancelling appointments. The calendar holds the subscriber's schedule, and generates events in the beginning and end of the appointments. Because not all mobile clients have a built-in calendar, this component should reside on the server. The user might want to integrate this calendar with his existing personal schedule system, like for example Microsoft Outlook or Lotus Organizer. In this section it is assumed that this calendar is able to initiate a communication to the client.

D.3.2.2 Divert functionality

The divert functionality can be implemented in two different ways, as a network based divert functionality or as a client based divert functionality.

D.3.2.3 Network based divert

The first manner is based on the ordinary way of diverting calls by using the network unconditional forward service. This works as follows: when the calendar system generates a beginning of a meeting event, a service indication is sent to the client containing a link to the `setForward` script. The user agent processes this script, which sends the string to enable the service to the network. At the end of the meeting the same procedure is carried out, but using the `resetForward` script instead of the `setForward` script. This design has a number of limitations: it is not possible to forward calls to different numbers and it is not possible to allow calls from specific numbers.

The figure below shows what happens at the beginning of a meeting.

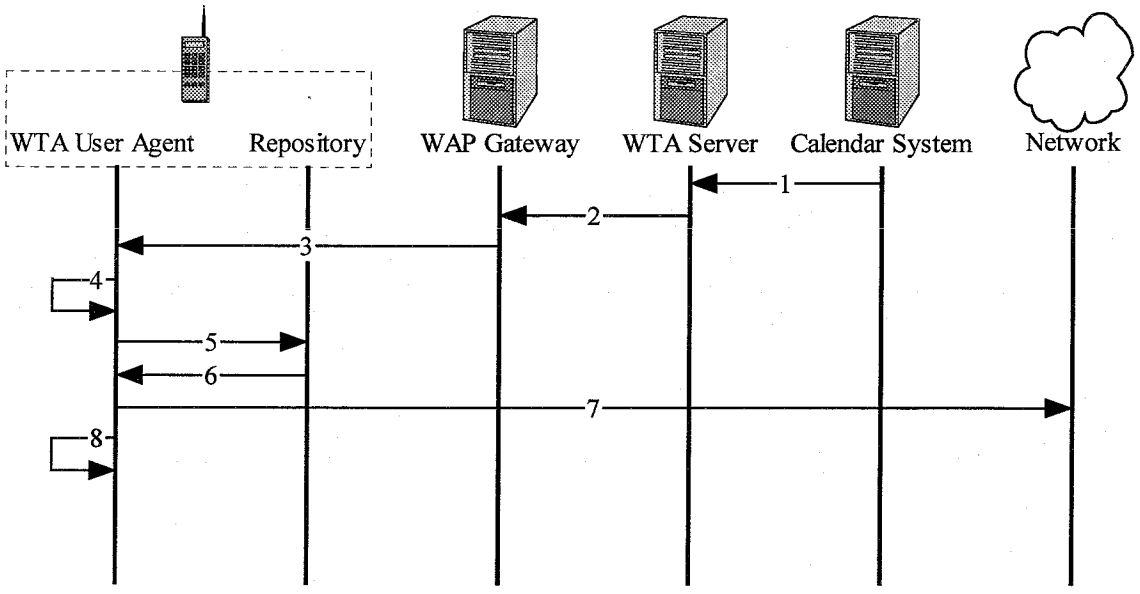


Figure 22. Network based Automatic Scheduled Divert trace.

1. The calendar system generates an "beginning of a meeting" event and sends the WTA server a message to activate the divert service.

2. The WTA server sends a push message containing a service indication to the WAP gateway.

3. The WAP gateway sends the service indication containing the URL to divert the mobile client.

4. The mobile client receives the service indication, beeps and shows the message "Activate divert" to the user. The user accepts the service indication.

5. The WTA user agent asks the repository for the URL (in this example it is assumed that the WML deck containing the `setForward` function is stored in the repository. Otherwise this page should be loaded from the WTA server).

6. The repository returns the WML deck containing the `setForward` script.

7. The string to set the unconditional forward is sent to the network using the `WTAGSM.ussd` function.

8. The message "Divert active" is shown to the user.

When someone now tries to call the subscriber the call will be diverted in the network as shown in the figure below.

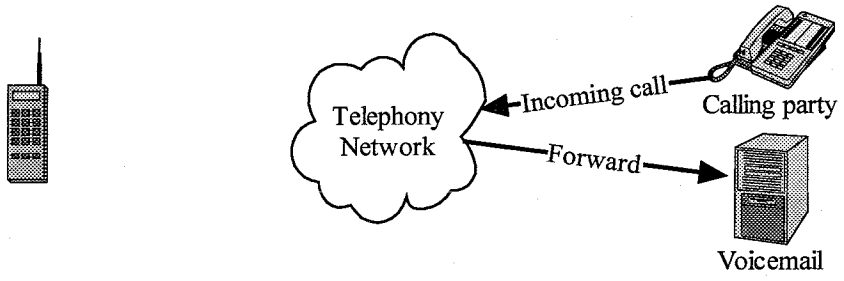


Figure 23: Network divert.

D.3.2.4 Client based divert

The second way of implementing the divert functionality is based on the `deflect` function. When the calendar generates a beginning of a meeting event, a service indication to install the `divertService` is sent. This script is linked to the incoming call event (`wtaev-cc/ic`). When an incoming call is detected, the client generates an incoming call event and starts processing the `divertService` script. This script checks whether the call has to be forwarded (see also call screening) and to which number. After this the script may call the `WTAGSM.deflect` function to forward the call. This design gets rid of some limitations of the prior design. It is now possible to forward to different numbers and to allow calls from some specific numbers. But unfortunately this design introduces a new limitation: this implementation assumes that the mobile terminal is always active and available. This may not be true, for example when the user is travelling through a tunnel.

The figure below shows what happens when a meeting starts.

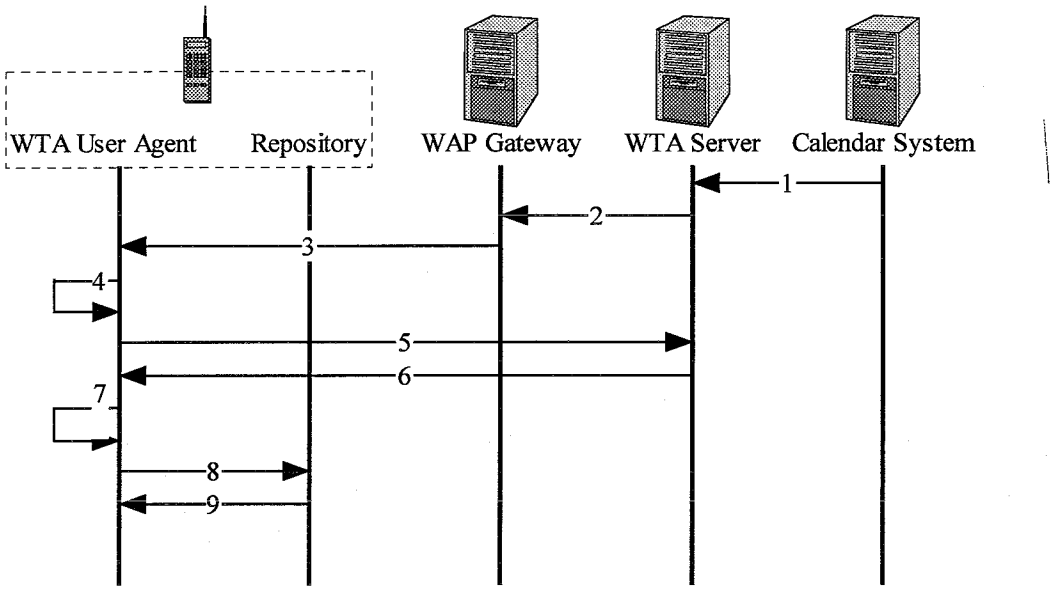


Figure 24. Client based Automatic Scheduled Divert trace.

1. The calendar system generates a "meeting start" event and sends a message to the WTA server to activate the divert service.

2. The WTA server sends a push message to the WAP gateway containing a service indication to install the divert service channel.

3. The WAP gateway sends the service indication to the mobile terminal.

4. The WTA user agents receives the service indication, "beeps" and asks to activate the divert service. The user accepts the service

5. The WTA user agent initiates a WTA session to the WTA server to find the channel indicated by the service indication.

6. The WTA server returns the requested channel.

7. The WTA user agent processes the channel page.

8. The WTA user agent instructs the repository to install the channel.

9. The repository tells that the installation has succeeded.

After this installation, the divert service is active. In contrast with the network based divert service, the incoming call reaches the mobile client as can be seen in the figure below.

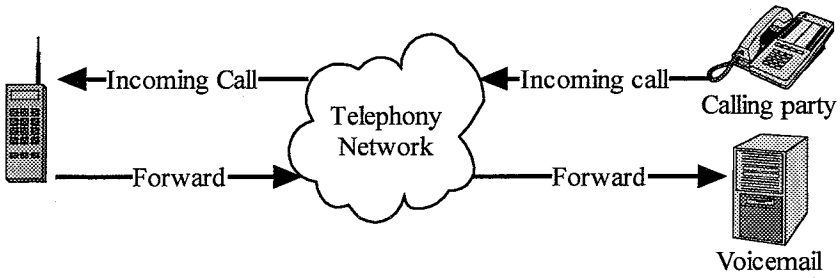


Figure 25: Client divert

The forwarding service works exactly the same as the Call Screening service as described in the previous section and will therefore not be discussed here again.

D.3.3 Implementation Issues

Both implementations have the big disadvantage that a user intervention is required to activate or deactivate the divert service. This means that the user still has to activate the service manually. This limitation follows from the WTA specification. For security reasons it is not possible to send a service load instead of a service indication to the WTA user agent to overcome this problem.

In the client based divert proposal, all incoming calls will be noticed by the client. The user might want to keep a list of numbers that have tried to reach the terminal. It is not possible to store this list somewhere in the client. After the call forwarding this information could be stored on the WTA server. When the user wants to see this list, the user had to access the WTA server.

In case of the client based divert proposal, the same implementation issues as with the call screening apply.

D.4 Location Information Services

D.4.1 Description

This service is designed to easily provide subscribers of local phone number information, based on the subscriber's current position. The user can define his own profile and the content will be generated based on his position. For example a user could define the following personal profile as shown in Figure 26.

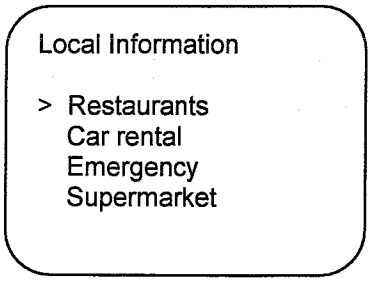


Figure 26. Screenshot of the Location Information Service.

When the user accesses for example the first option, restaurants, a list of all Chinese restaurants within the area will be displayed. When the user selects one of these restaurants, a call will be set up. The user also has the option to store the phone number in his local phonebook.

D.4.2 Architecture

This implementation is based on the `WTAGSM.info` function, which provides the current network information of the GSM terminal.

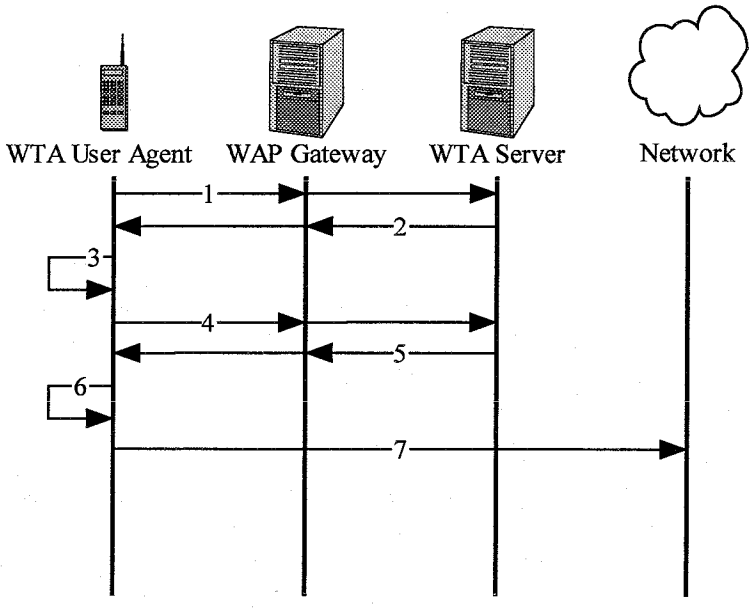


Figure 27. Location Information Services trace.

1. The user accesses the WTA server and enters his user id.
2. After selecting the Location Info Service, a WML deck containing a link to a WML script is loaded into the WTA user agent.
3. The user follows this link. The WML script calls the `WTAGSM.info` function and generates the link to post to the WTA server.
4. A request for the link is sent to the WTA server.
5. The WTA server receives the request and generates a WML deck based on the position information provided by the user agent.
6. The WML deck with the categories as defined in the user profile is displayed to the user. The user selects the restaurants and a list with the names of local restaurants is displayed to the user. The user selects one of the restaurants.
7. A call is set up using the `WTAPublic.makeCall` function.

This implementation is a service centric application. The WTA server needs access to a database containing all the entries, their phone numbers and their location. This location could be based on the positions of the operator's base stations or based on geographical co-ordinates. In the second case, another database is necessary to translate the network positioning information to the geographical co-ordinates.

D.4.3 Implementation Issues

With the network information provided by the `WTAGSM.netinfo` function it is possible to calculate the subscriber's position. When this information is passed to the server, the server can calculate the user's position, based on the position information of the operator's base stations. The information about the cells is thus operator dependent, because every operator uses its own physical network with its own base stations, which are usually not placed at the same locations as other operator's base stations. Thus to implement an operator independent solution, one need to know the geographical locations of the base stations of all operators. But operators are not generous in distributing this information. Other systems for providing positioning information are network-based systems. Network-based systems get the position information from the base stations. These systems have the advantage that the position information stays within the operator domain and they do not require new terminals supporting this option.

An alternative to this service is the Local Numbers Download. With this service a number of predefined entries is automatically update based on the user's position. For example the entry "Hospital" always represents the closest hospital. When the user's location changes the hospital number will be automatically updated.

D.5 Friends Locator

D.5.1 Description

The Friends Locator service is designed to allow subscribers to be notified when friends enter or leave the same area. A subscriber's friend is kept up in a table. When a friend enters the local area of the subscriber a message is sent to the subscriber.

D.5.2 Architecture

The Friends Locator service has both client and server components. The mobile terminal is responsible for obtaining and broadcasting its position information. The server is responsible for comparing the subscriber's current location to their friend's location. The following components need to be implemented for this service:

D.5.2.1 Friend Management

The Friend Management component maintains the list of who is whose friend. When a user A wants to add friend B to his contact list, user A accesses the WTA server and enters the mobile number of user B. After this a service

indication is sent to user B asking for permission. Only when user B gives permission, he will be added to the contactlist of user A. When user B is added to the contactlist of user A, user A receives a notification of this event.

D.5.2.2 Location Management

The Location Management component checks, which friends are in the same area. This service should be run on the server. When the Location Management detects that two or more friends are in the same area, a notification is sent to these users. This component should therefore maintain the location info of all friends. When the location of a user is updated, the location management component should first check the locations of the user's friends on the contactlist and then inform the friends who have the user on their contactlist.

D.5.2.3 Location Broadcast

The Location Broadcast component provides the location information to the Location Management component on the server. These updates should be performed regularly. The simplest form is based on the cell id. When moving from one cell to another, the new cell id should be passed to the Location Management.

D.5.3 Implementation Issues

The biggest problem with this service is that no "movement event" exists. This means that it is not possible in WTA to detect when the client moves. The only event that can give some information is the NetworkStatus event. This event is generated when the network status has changed, for example when the terminal has been handed over to another cell.

The service is operator dependent. Users from different operators cannot locate each other, because they are connected to different radio networks with different cell ids. Two users may also be physically close to each other, but be in different cells, or physically far from each other and in the same cell. The definition of close depends on the size of cells.

When using GSM or GPRS class B or C as bearer, it is not possible to update the location information during a call, because it is not possible to have two connections simultaneously.

D.6 Automatic Retry

D.6.1 Description

The Automatic Retry application is used when the called party's phone number is busy. This service keeps trying to set up the call until the call is setup. It is not a beautiful solution, because this would keep the calling party's phone line busy.

D.6.2 Architecture

The Automatic Retry application can be implemented on the mobile terminal. The application consists of a WML deck and some WML scripts. These components can be installed in the repository.

The application first asks for the phone number to call. After the user has given this number, the application starts trying to set up the call using the `WTAPublic.makeCall` function. If this function returns busy, the function is called again until the result is "connected".

D.6.2.1 WML deck

The WML deck contains a number of cards to handle the call process. First of all it contains a card to ask the user for the telephone number to call. After this the next card shows the message that the service is trying to connect. And after all a card that is displayed when the call is connected. To go to the correct card displaying the current call state, the Call Connected and Outgoing Call events are used.

D.6.2.2 WML scripts

The application uses a script. This script consists of a while loop with the `WTAPublic.makeCall` inside the loop.

D.6.3 Implementation Issues

As described above, it is a dirty application, because this application keeps the calling party's line busy. It is a better option to implement this service such that when the calling party is free, the calling party is notified. This service can be implemented as described in the next section.

The user should keep watching the terminal screen, as there exists no function to notify the user by playing a sound or something like that.

D.7 Camp On Callback

D.7.1 Description

The Camp On Callback service is used when a called party is busy. When the user, who has installed this service on his mobile client, tries to set up a call to a busy number, the Camp On Callback service will ask whether or not to activate the service. When the user activates the service, the user will be notified when the called party is not busy anymore.

D.7.2 Architecture

The Camp On Callback service needs two components, one on the mobile client and one on the WTA server. The local installed component asks the user, when a call setup fails, whether or not to activate the service. When the user accepts to activate the service, a message is sent to the WTA server which forwards this message to the network. When the called party is free,

the network informs the WTA server. On his turn, the WTA server sends a service indication to the mobile client that the called party is free. The user can now easily call this party from the service indication.

D.7.2.1 Client component

The client component consist of a channel that is processed on when the call cleared (`wtaev-cc/cl`) is generated. If the result parameter of this event is "busy", a card asking whether or not to activate the Camp On Callback service is displayed to the user. In the other result cases, a result message is shown to the user. The accept-link, references to the Camp On Callback service on the WTA server and should include the called party telephone number.

D.7.2.2 Server component

The server component is a servlet running on the WTA server. When the user calls this servlet, the servlet passes the camp on callback request to the network. When the network detects that the called party is free, it returns a message to the WTA server. Now a service indication is sent to the user that activated the camp on callback service.

D.7.3 Implementation Issues

This service requires a tight integration between the WTA server and the telephony network to monitor the availability of the called party. Limitations that exist for the telephony network will exist for this service too.

D.8 Corporate Directory

D.8.1 Description

The Corporate Directory service uses the WTA server as a big database for phone numbers. The user can access the server and search for a number. After the search has been performed, a list with results is returned. Now the user can scroll through the list and setup a call or store the number in the local phonebook simply by selecting an entry in the list. When the user has the appropriate rights the user can even update the entry.

D.8.2 Architecture

The Corporate Directory only resides on the server. The server contains a database, or has access to an external database containing the entries. When the user accesses the service, an input form is displayed to the user. After the user has entered for example the name, the server script performs a search on the database and generates a WML deck on the results. This WML deck should contain WTA links such that the user can make a call by selecting an entry or add the entry to the phonebook.

D.8.3 Implementation Issues

The service as described above can be implemented according to the WTA specifications. But it might be interesting to extend this basic functionality with the following options:

When the service on the WTA server is also accessible from a desktop PC via a web session, the user might want to select an entry that he would like to call and send this number to his mobile terminal using a service indication. When his mobile terminal receives the service indication, the user can immediately call or store the entry from the service indication. This might be a good alternative as it is much easier performing a search using a desktop PC. It is not possible to automatically initiate a call from the mobile client, because always a user intervention is required.

Because all numbers are now stored centrally, it is easier to update these numbers. It can be useful to synchronize the local phonebook with the numbers on the server. Using the WTA phonebook functions it should be possible to synchronize the phonebook. But as the mobile client does not maintain a last edited time stamp it is not always possible to resolve conflicts based on time. This also implies that the entire contents of the phonebook must be sent to the server to find out what modifications are made. This is however not an attractive option.

Using the basic functionality implies that there exist two phonebooks that are not linked. The user might want to combine these phonebooks, such that the local phonebook is used for frequently called numbers, a kind of cache. One would like to build a new phonebook on top of the existing phonebook and the server phonebook such that when the user enters a name first the local phonebook is consulted. If the entry does not exist in the local phonebook, the phonebook on the server is consulted. It is possible to implement a service like this, but there are some limitations. The local phonebook usually has a much better interface to perform a search on. For example, it is not possible in WML script to generate a local WML deck with the search results.

D.9 Top Ten Numbers

D.9.1 Description

The Top Ten Numbers service provides a quick mobile unit list for subscribers based on the numbers that they call most often. The most called numbers appear first on the list. When a numbers is called, the counter will be updated and if necessary also the list will be updated.

The Top Ten list offers an alternative for locating a number without having to scroll or search through a large address book. Once located, the WTA service will dial the number for the subscriber at the touch of a button.

D.9.2 Architecture

The Top Ten Numbers service is a server centric design. The processing of the counters occurs on the WTA server. The mobile unit notifies the WTA

server of a call attempt and then receives the update top ten list from the server.

The role of the WTA server is updating the call counters after every successful and unsuccessful call as can be seen in the diagram below.

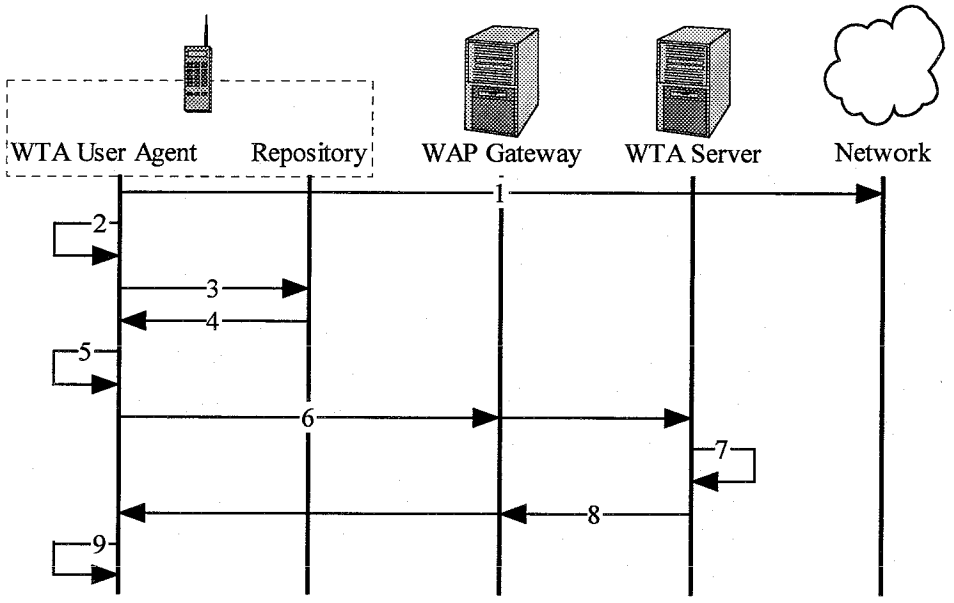


Figure 28. Top Ten Numbers trace.

1. The subscriber initiates a call.
2. The user agent generates the OutgoingCall event (`wtaev-co/oc`). The repository is consulted in order to find a dedicated channel. The channel provides the URL to the "TopTen Script" stored in the repository.
3. The user-agent requests the content from the repository.
4. The repository returns the requested content.
5. The TopTen script reads the called number from the OutgoingCall event and sends the browser to the WTA server to update the counter.
6. The browser accesses the TopTen service on the WTA server.
7. The WTA server updates the counter for the passed number and generates a result WML page. In this example the counter update results in a change in the order of the TopTen numbers list.
8. The new top ten numbers list is returned to the user agent as a channel to be installed.
9. The user agent installs the updated Top Ten Numbers channel.

The service as described above requires three components: the TopTen script, the TopTen list and a servlet running on the server to update and maintain the TopTen list.

D.9.2.1 TopTen script

The TopTen script is stored in the repository of the mobile terminal and linked to the WTA OutgoingCall event. When an outgoing call event is generated, this script will be processed and will send the WTA user agent to the WTA server to update the counters.

D.9.2.2 TopTen list

The TopTen list is the current Top Ten implemented in a WML deck, stored in the repository. When scrolling through the list, the user can click on one of the entries, resulting in setting up a call.

D.9.2.3 TopTen Servlet

The servlet is running on the WTA server. When accessing the servlet on the WTA server and passing a number, the counter belonging to that specific number is increased by one. After this update a new top ten list will be generated and if it is changed compared to the previous one, it is sent to the subscriber.

D.9.3 Implementation Issues

In the implementation above, the design is server centric. A client centric approach could be an alternative. But this is not possible, because it is not possible to store the counters locally. One might want to have the possibility to store this information as an extra field in the phonebook. However, it is possible to have additional fields in the phonebook, but these are specified by the terminal manufacturer. So it is not possible to define additional fields from the application.

With GSM as bearer, it is not possible to update the counters while a call is in progress. This means that the update should be postponed until the call is completed.

It should be possible to scroll from the top ten numbers automatically to the phonebook. This is not possible, because it is not possible to display the entries in a locally generated WML page.

D.10 Incoming Call Information

D.10.1 Description

The Incoming Call Information service is designed to extend the information presented with an incoming call. Most mobile phones show the calling party number when an incoming call is detected, or when the number is stored in the phonebook, the name of the calling party. With the Incoming Call Information service, the WTA server is contacted when an incoming call is detected to download additional information. Because the WTA server has much bigger storage capabilities, all kind of information can be provided. In this example the user name, company and a picture will be displayed.

D.10.2 Architecture

The Incoming Call Information service has both client and server components. The client component is started on an incoming call and sends a request to the WTA server for information about the calling party. The server component is used to store the information.

The diagram below illustrates how the service works.

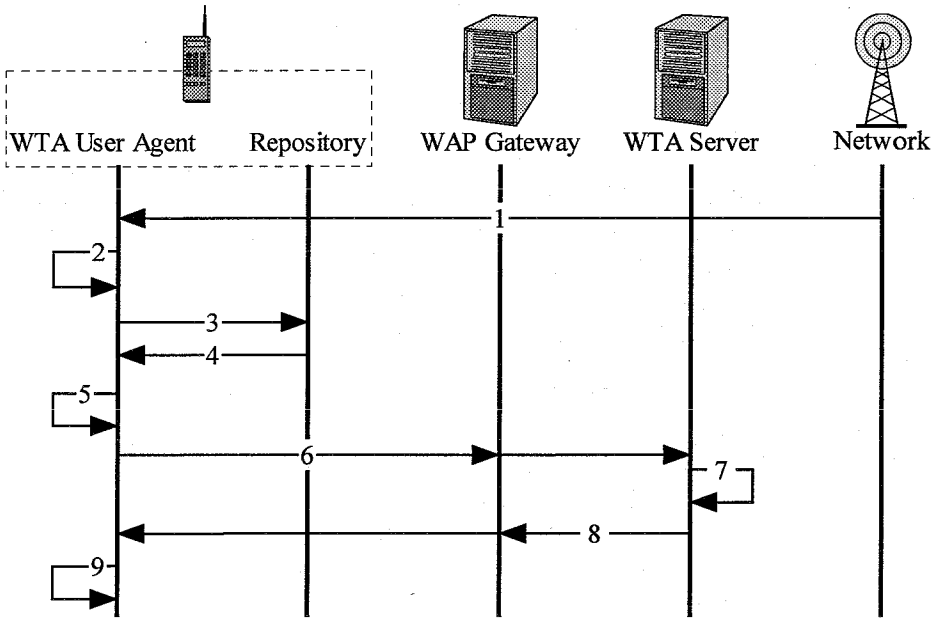


Figure 29. Incoming Call Information trace.

1. The mobile network receives an incoming call and sends a "Call indication" to the mobile subscriber

2. In the client, the incoming call WTA event (`wtaev-cc/ic`) is generated. The repository is consulted in order to find a dedicated channel. The channel provides the URL to the "Incoming Call Information Service" stored in the repository

3. The user-agent requests the content from the repository

4. The repository returns the requested content

5. The User Agent processes the Incoming Call Information script. This script retrieves the calling party number from the IncomingCall event and sends the browser to the WTA server.

6. The User Agent requests the WTA server for information about the calling party.

7. The WTA server searches in the database for information on the given number and generates a page with this information.

8. The generated page is returned to the User Agent.

9. Finally the user information is presented to the user.

D.10.3 Implementation Issues

The information that is stored on the WTA server can be all kind of information. It might be interesting to display the name of the caller and the company he works for. It is also possible to display a picture of the calling party. Other functionalities can be implemented when using scripts. These scripts can be integrated in the information that will be downloaded and can for example be used to deflect the incoming call to another number or to send a message automatically. The difference with other incoming call depended services is that the information about how to handle the call is not stored locally, but on the WTA server. But to make this work, it must be possible to have two connections simultaneously, one for the incoming call and one for the WAP session. This is possible in the GPRS voice/data class (class A), but not in GSM. Another issue is the speed of the connection. The time between the detection of the incoming call and the information retrieval should be as short as possible to prevent that the calling party aborts the call. It should also be possible to answer the call immediately before the additional information is retrieved.

Another application based on this idea is using the WTA server for priority purposes. When a user initiates a call, the user selects for example urgent. The called party consults the WTA server on the incoming call and finds the incoming call is an urgent call.

D.11 Visitcard Exchange

D.11.1 Description

The Visitcard Exchange service is designed to easily exchange phonebook entries between two telephones. Usually when two users exchange name and phone number both users have to enter name and number. With this service only one of the users has to enter the phone number after which both names and numbers are exchanged and stored in their local phonebooks.

D.11.2 Architecture

The Visitcard Exchange service can be implemented in two ways: with and without using the WTA server.

D.11.2.1 Without WTA server

The implementation without the use of the WTA server assumes that both the sender and the receiver have a script available that can encode and decode the visitcard.

The first user (user A) enters the number of the second user (user B). The name and phone number of user A are encoded and sent to user B using the `WTANetText.send` function. When user B receives a message, the channel

linked to the `wtaev-nt/it` event will be processed. This channel contains a script that check whether or not the message is a visitcard message. If so, the script decodes the message and stores the name and number after confirmation in the phonebook. After this, the script creates a new message containing the encoded name and number of user B and returns this message to user A.

This implementation requires that both users have installed the scripts to encode and decode visitcards.

D.11.2.2 With WTA server

This implementation uses two components. One component stored locally on the client and one component stored on the WTA server. The component stored on the client is used to initiate the service. The component on the server generates the service indications that will be sent to the users. The diagram below shows what happens when the service is initiated.

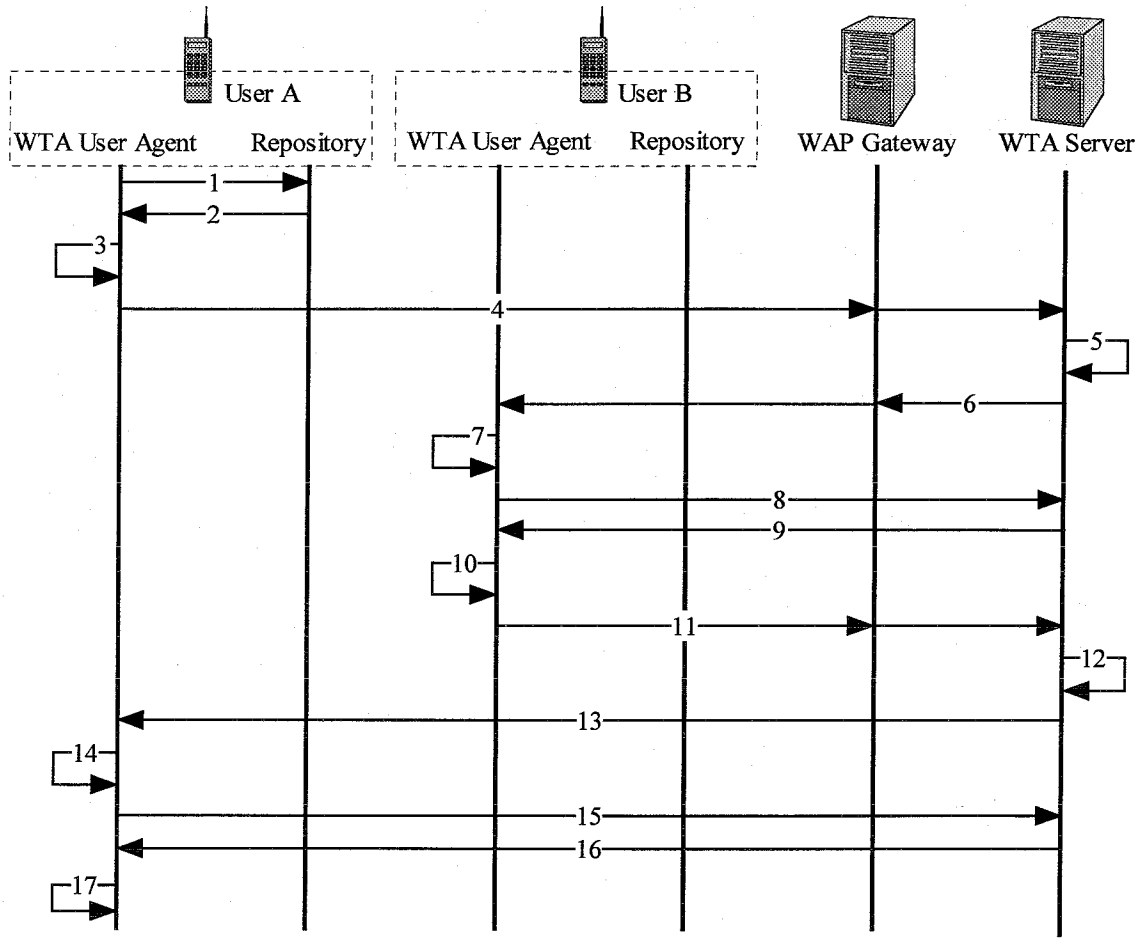


Figure 30. Visitcard Exchange using a WTA server.

1. User A starts the Visitcard Exchange service by requesting the repository.

2. The Visitcard Exchange service is stored in the repository as a channel and contains a WML deck with an input field for the number to send the visitcard to.

3. The Visitcard WML deck is showed to the user. User A enters the phone number of user B.

4. The WTA user agent sends the name and number of user A and the number of user B to the WTA server.

5. The WTA server receives the information and generates a service indication message containing a link to a WML deck and a script to store the name and number of user A.

6. The service indication is sent to the terminal of user B using WAP push.

7. The service indication is displayed to the user. When the user accepts the user indication, the user agent accesses the WTA server to download the WML deck containing the user information of user A.

8. The user agent requests the information as specified in the service indication.

9. The information is returned to the user agent.

10. The user agent executes the script. This script adds the name and number of user A to the phonebook of user B.

11. After this it sends the name and number of user B back to the server.

12. The WTA server generates a second service indication to store the name and number of user B in the phonebook of user A.

13. The service indication is sent to user A using WAP push

14. After the user accepts the service indication, the WML browser will follow the link as specified in the service indication

15. The user agent requests the content as specified in the service indication.

16. The WTA server returns the requested content.

17. The user agent processes the script and adds the name and number of user B to the local phonebook of user A.

In this example it is assumed that the Visitcard exchange service is stored in the repository of user A. This is not necessary, but it has one advantage: the name and number of user A can be stored in this page such that it is not necessary to enter this information manually.

D.11.3 Implementation Issues

Both users must support WTA functionality.

The application can also be simplified such that the service is used to send an entry from the phonebook to another user. This still requires that both users have WTA support.

D.12 Context Aware Dialling

D.12.1 Description

The Context Aware Dialing service is based on the way people start a conversation in situations where they meet physically. In these situations the initiator can see the situation and he can decide, depending on the situation to start or postpone the conversation. For example compare the following two situations. In the first situation user A is in an important meeting. User B wants to talk about the movie that was on television the night before. User B sees that user A is in a meeting and decides it is not important enough to interrupt the meeting and he decides to try it later. In the second situation user B has very important information about the subject discussed in the meeting. In this situation it may be desired to interrupt user A. With a telephone conversation user B cannot see the situation of user A, so the only option user A has is to allow all calls or to block all calls. When using the Context Aware Dialing service, user B can first check the situation of user A after deciding to initiate a call.

D.12.2 Architecture

The WTA server contains a database to store the current context of a user. The user has access to read and write the state. Other users only have read access. When the user wants to initiate a context aware call, the user first accesses the WTA server and enters the number he would like to call. The WTA server returns the current context of the called party, for example "In meeting, do not disturb". Now the user can decide to call or not.

D.12.3 Implementation Issues

The user has to enter the number manually. It is not possible to use the service from the normal phonebook interface. Another option is to implement a search form using the WTA phonebook.search function. But this gives a less good interface than the built-in phonebook, because of the limitations of WML and WMLScript.

The service should work for every call. This can be implemented by using the outgoing call event. When an outgoing call event is generated, the context aware dialing service is started and immediately the call will be cleared. Now first the context information is requested from the server and after this a call can be initiated. By using a temporary binding for the outgoing call, the call can now be completed. This is not a beautiful implementation. Other problems may rise when the user retries to setup the call. Now for every retry the WTA server is consulted.

D.13 Menu Selection

D.13.1 Description

The Menu Selection service is designed to replace the existing menu selection by using interactive voice response systems. Instead of listening to all the menu options, the options are now displayed in a menu on the mobile terminal and after making a choice a call is set up to the appropriate person.

D.13.2 Architecture

The menu selection can be implemented in different ways. The simplest one is using the `makeCall` function. Using this method, the WTA user agent accesses the WTA server to get the menu. The user now navigates through the menu and finally the WTA session is terminated and a voice call is initiated to the number specified in the menu. This architecture requires that all numbers are directly accessible from outside.

Another option to implement the service is as follows. The user can navigate through the menu. When the user had made some choices, the system reminds what the user has chosen, and when the user calls the system, the system recognizes the user and routes the call to the appropriate person.

D.13.3 Implementation Issues

The first implementation is in fact just a skin over the existing interactive voice systems. The second implementation requires a tight integration between the interactive voice system and the WTA server. This integration is outside the scope of this study.

D.14 Message Manager

D.14.1 Description

The Message Manager Service is designed to allow subscribers to receive notifications and manage their email, voice mail and fax messages. The Message Manager will be integrated with a unified messaging system that provides access to all of the subscriber's messages. The message manager makes use of the WTA especially during the management of their messages. The subscriber can forward emails, faxes and voice mails to other subscribers selected from the phonebook. In fact the use of WTA makes it possible to offer a new interface for accessing the subscriber's mailbox.

D.14.2 Architecture

The message manager is designed to provide access to a subscriber's unified messaging system. The role of the WTA server and the WTA client is minimal. The major function of handling and processing of the messages occurs within the unified messaging system. The WTA server acts as a conduit for the unified messaging system to the mobile unit, but does not perform any major processing. The WTA client is used to navigate through the messages.

In the diagram below the user is notified that he has received new voice mails and the user chooses to listen to one of them.

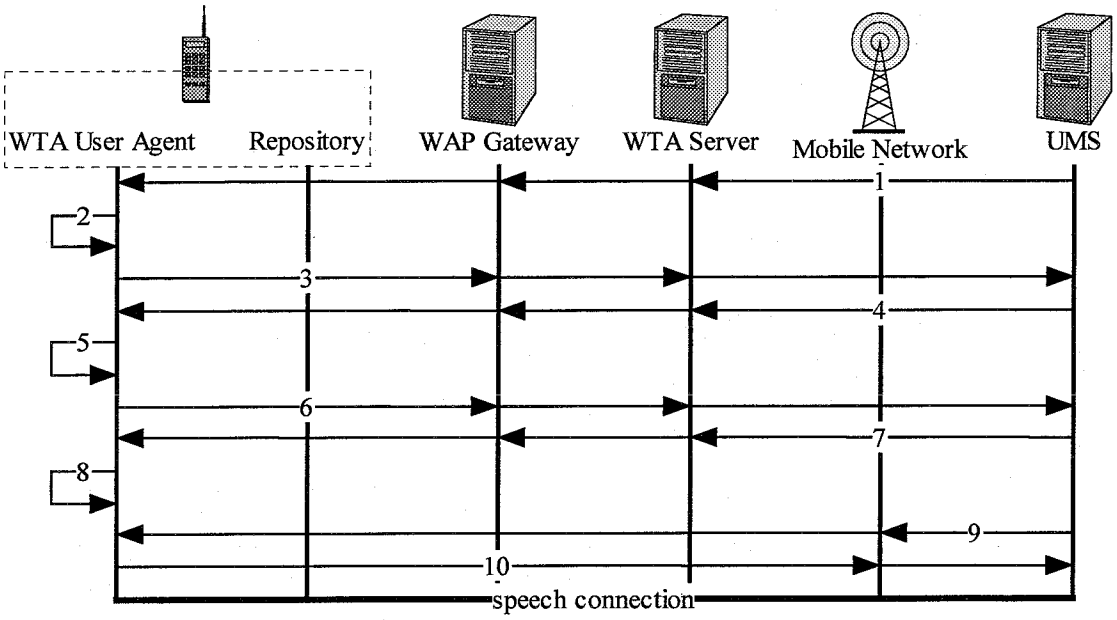


Figure 31. Message Manager trace.

1. The unified messaging system indicates the WTA server that there are new voice mails. The WTA server sends a push message to the WAP Gateway, which forwards the message to the WTA user agent.

2. The WTA user agent processes the service indication and displays the message "You have 4 new voice mails" to the user. The user accepts the message.

3. The WTA user agent requests the content as specified in the service indication.

4. The unified message system returns a page containing a list of messages stored in the subscriber's mailbox.

5. The subscriber chooses to listen one of the messages.

6. A request to play one of the messages is sent to the unified messaging system.

7. The unified messaging system returns a page containing only a link to temporarily bind the incoming call event such that the subsequent call from the unified messaging system will be answered automatically.

8. The page is processed and the incoming call event is temporarily bound. In order to avoid that the message manager answers a call from someone else that the unified messaging system, the calling party's phone number should be checked.

9. The unified messaging system initiates a call to the subscriber.

10. The WTA user agent generates an incoming call event, which is bound to the messaging manager, such that the user agent automatically answers the call and the voice message is played to the user.

The WTA functionality will be used for voice mails only. Other types of messages, like fax and email do not need WTA functionalities.

In the implementation as described above, there should exist a connection between the WTA server and the unified messaging. But unfortunately no standard solutions exist for this yet.

D.14.3 Implementation Issues

When using the Ericsson unified messaging system, all messages are stored as emails. To access the messages using the message manager, a front-end application is needed which converts the emails to WML decks. This application also should be able to initiate a call to play voicemail messages.

To use this service the bearer must support a WAP session and a voice call simultaneously. This is possible with GPRS as the bearer.

D.15 Call Information Service

D.15.1 Description

The Call Information Service provides the subscriber with information about his recent calls. This information is obtained from the mobile network. The subscriber requests the service using a WAP menu selection. The information can be displayed as follows:

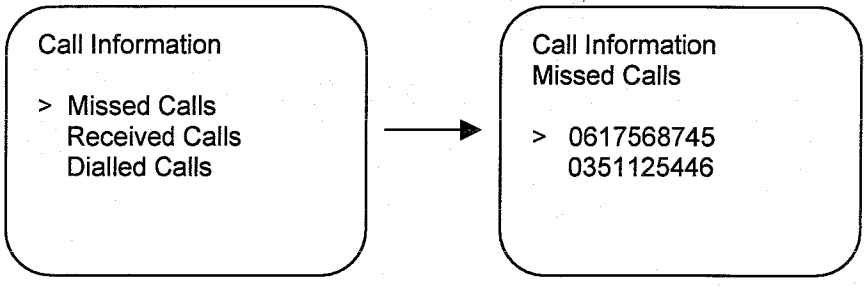


Figure 32. Screenshot of the Call Information Service.

The options when selecting a number can be time of call, duration and the WTA functions to call and save the phone number.

D.15.2 Architecture

The Call Information Service can be implemented using a server centric design. To implement this service, it is assumed that the WTA server can obtain information about call logs from the mobile network.

The following diagram shows how the service works:

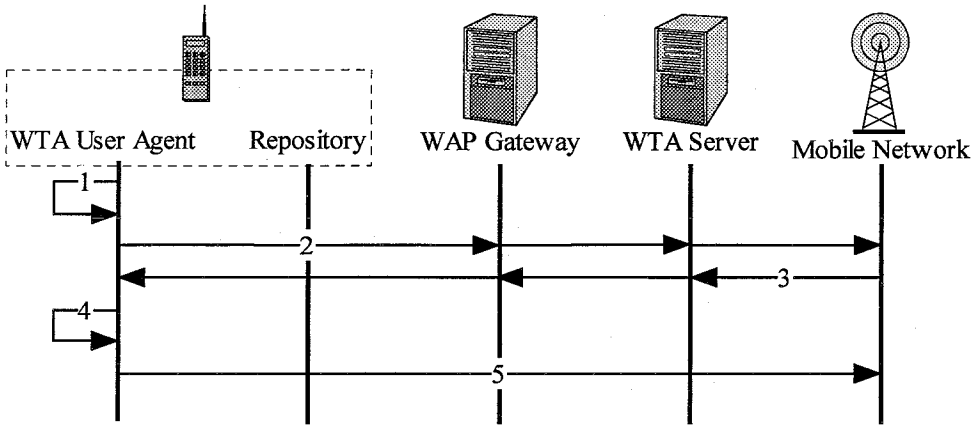


Figure 33. Call Information Service trace.

1. The subscriber selects the call information service.

2. The WTA user agent requests the content. This request is passed to the WTA server. The WTA server requests the information from the mobile network.

3. The mobile network returns the call information to the WTA server. The WTA server transforms the information from the mobile network to WML decks and passes this to the WTA user agent.

4. The user agent processes the content and the call information is displayed to the user.

5. The user chooses to call back one of the numbers. The `WTAVoicecall:makeCall` function is used to initiate a call.

D.15.3 Implementation Issues

The WTA functionalities used in this application are calling and storing a phone number from a list.

E Detailed Description Prototype: I See You

In this appendix the implementation of the I See You application is discussed in more detail.

E.1 Introduction

Nowadays, mobile phones are able to display the number of the calling party, the caller id, on an incoming call. When this number is stored in the local phonebook, most phones can display the name instead of the number. "I See You", or abbreviated to ICU, goes one step further by making it possible to display the name and a picture of the calling party.

Every user of the ICU service has an own account on the ICU server containing the contact details and a picture of the user. Every user also has a contactlist on the server. The contactlist is used to find a name and number on an incoming call. The user can also use the contactlist to find contactdetails and, when using the WAP interface, to initiate a call by using WTA functionality.

Users can add two kinds of contacts to this contactlist: buddies and personal contacts. Buddies are other registered users on the ICU server. When a user adds a buddy to the contactlist, the contactdetails as entered by the buddy himself are used. So the user does not have to look for a picture and when the buddy updates his contact details, these details are automatically updated in the contactlist too. Buddies can be found by searching in a public database. For privacy reasons users can choose whether or not their contactdetails are publicly available to other users. A user can also choose whether other users must request authorization to add the user to their contactlist.

Personal contacts, on the other hand, are contacts created by the user himself. These contacts can for example be used when the contact is not a registered contact. The user has to enter the contactdetails and upload a picture himself.

To use the service, a channel must be installed on the user's mobile phone. This channel contacts the ICU service on the server on every incoming call, and sends the number of the calling party to the server. When this number is found in the contactlist, the name and picture of the calling party are displayed together with the options to accept, reject or forward the call. If the number is not found on the contact list, the service queries the public database. When still no entry can be found only the number of the incoming call and the options to accept or reject the call are displayed. In the latter case, the user is asked whether he wants to add this number to the contactlist after the call has been ended.

E.2 Architecture

E.2.1 Overview

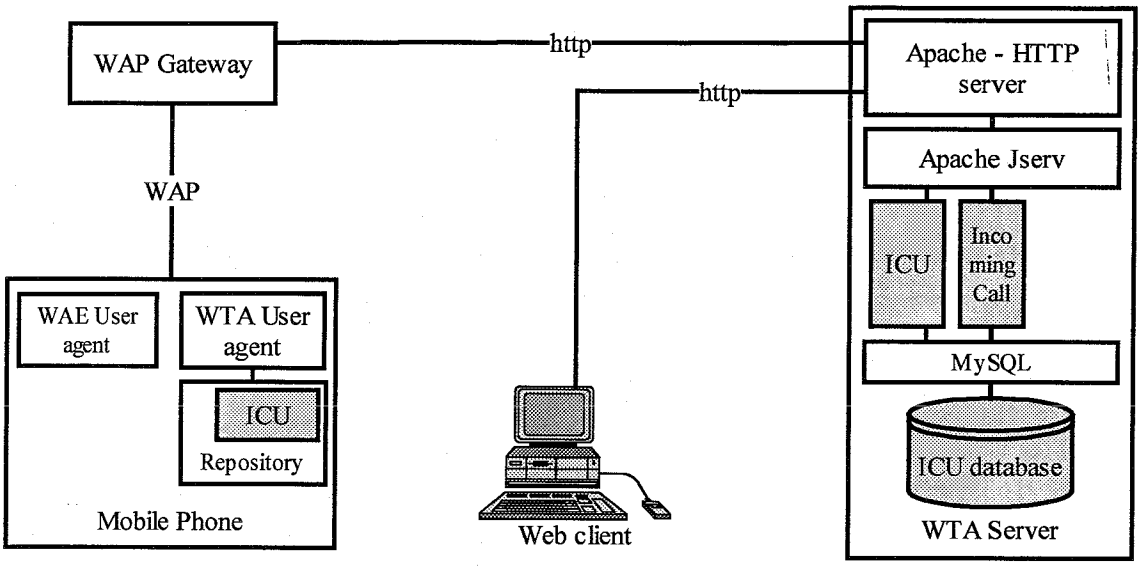


Figure 34. Overview of the I See You architecture.

Figure 34 gives an overview of the architecture of the I See You service. This figure shows the components that are used in the ICU service. The four shaded components are the components especially designed for the ICU application. These components will be introduced here, but discussed in more detail below. The ICU service can be divided in two parts: the client part and the server part. The client part is the WTA supporting mobile phone or an ordinary web browser. The mobile phone contains the ICU channel in the repository. On every incoming call, the WTA user agent processes this channel and requests the appropriate page from the server.

The following steps are involved in the trace described in Figure 35:

1. The mobile network receives an incoming call and sends a "Call indication" to the mobile subscriber

2. In the client, the incoming call WTA event (`wtaev-cc/ic`) is generated. The repository is consulted in order to find a dedicated channel. The channel provides the URL to the "IncomingCall deck" stored in the repository

3. The user-agent requests the content from the repository

4. The repository returns the requested content

5. The User Agent processes the IncomingCall script. This script retrieves the calling party number from the IncomingCall event and sends the browser to the WTA server.

6. The User Agent requests the WTA server for information about the calling party.

7. The WTA server searches in the database for the picture belonging to the given number and generates a page with this picture.

8. The generated page is returned to the User Agent.

9. Finally the picture is presented to the user.

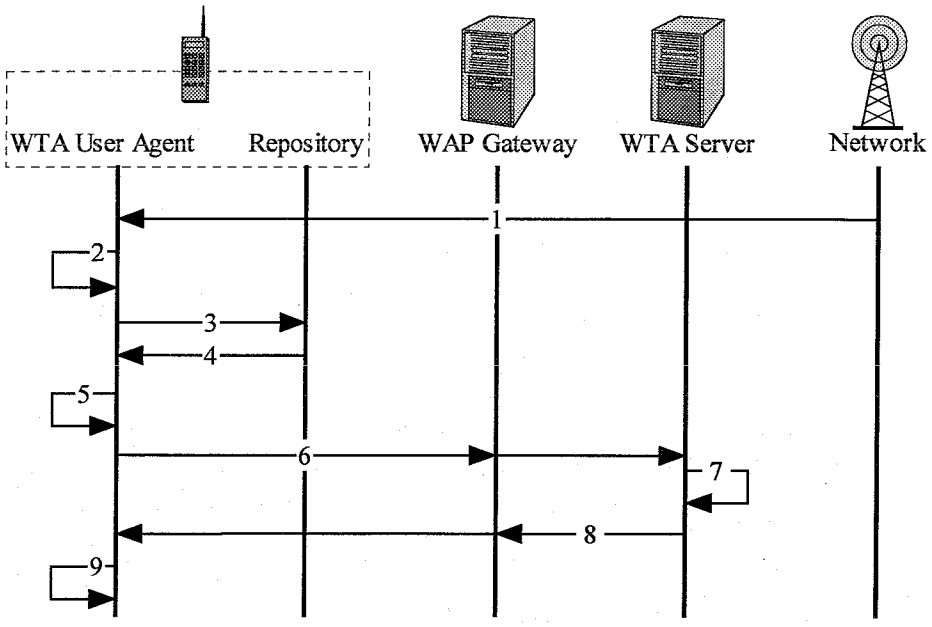


Figure 35. Trace of an incoming call reaching the mobile phone with the ICU service installed.

The server part is based on the architecture of the Ericsson WAP Application Server. By using this architecture it should be relatively easy to use the application on the WAP Application Server such that the application can be used for demonstrations purposes on the WAP Application Server too. The WAP Application Server consists of an Apache HTTP-server with an Apache Jserv servlet engine, a database and some APIs for fast development of WAP applications. The Common Gateway Interface (CGI) makes it possible to connect a URL on a webserver to an external program. The Ericsson WAP Application Server uses Java servlets to implement these external programs. Java servlets offer the some functionalities for easy application development, like session tracking.

The main ICU application and the application that generates the incoming call pages are implemented as Java servlets. All the contactdetails are stored in a database, using MySQL.

E.2.2 Java Servlets

Java Servlets are protocol- and platform-independent server side components, written in Java, which dynamically extend Java enabled servers.

They provide a general framework for services built using the request-response paradigm. Their initial use is to provide secure web-based access to data, which is presented using HTML web pages, interactively viewing or modifying that data using dynamic web page generation techniques.

Since servlets run inside servers, they do not need a graphical user interface. Otherwise, they are the server side counterpart to applets (which are used only on the client side of systems): they are Java application components, which are downloaded, on demand, to the part of the system which needs them. [11]

The Java Servlet API offers functions to easily implement servlets. The most important function that is used in this prototype is the use of sessions. The HTTP-protocol is a stateless protocol. The Java Servlet API offers functions to create a session object such that the servlet can store information about the session in this object. The session id of this session is sent to the server with every request.

More information about the Java Servlet techniques can be found on `http://java.sun.com/`.

E.2.3 Database

A database is used to store all the contactdetails for the ICU service. The database is described below using an entity-relationship diagram.

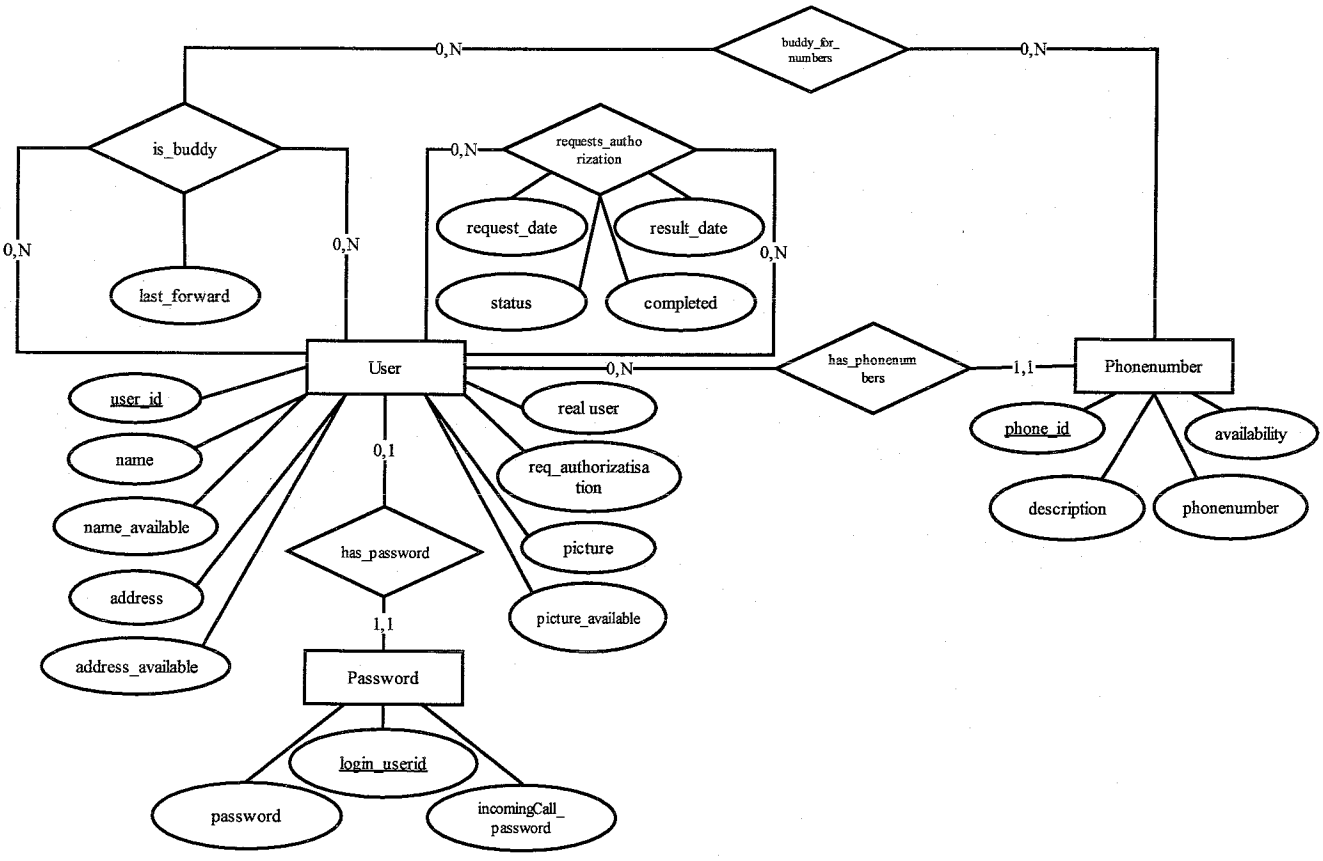


Figure 36. E-R diagram of the ICU database.

The database contains three entities: user, phonenumber and password. The user entity is the central entity. Every user has a unique user_id the key. Every user also has a name and address attribute, which are so-called composite attributes. This means these attributes describe a group of simple attributes like for example name might be composed of firstname, middlename and lastname. For simplicity only these composite attributes are shown in the diagram. The realuser attribute describes whether the user is a real user and the req_authorization attribute describes whether other users must request authorization before they can add the user as buddy to their contactlist. The picture attribute contains the filename of the picture. Finally, a user also has some *attribute*_available attributes. These attributes describe whether these attributes are publicly available, that is available to other users without an is_buddy relation.

Every user has zero or more phonenumbers. A phonenumber belongs to exactly one user. A phonenumber has a phone_id as key. Furthermore a phonenumber contains a phonenumber, a description describing this phonenumber and an availability attribute. The availability attribute describes, like the available attributes of a user, whether this phonenumber is publicly available to users that have no is_buddy relation for this particular phonenumber.

The password entity contains information to login to the service. The password entity contains a unique login_userid as key. The value of this attribute is used to login to the service. The password attribute contains the password to login. The incomingcall_password contains the login password that is used in the incomingCall servlet. Every real user has a password and a password belongs to exactly one user.

Every user of the service can maintain a contactlist, which is implemented by relations. The contactdetails of user appear in another user's contactlist if and only if an is_buddy relation between these users exists. Every user can have zero or more buddies and every user can be the buddy of zero or more users. The buddy_for_numbers relation defines which phonenumbers appear on the contactlist. Only the phonenumbers that have a buddy_for_numbers relation with an is_buddy relation appear on the user's contactlist. By using this relation a buddy can decide to hide some numbers for a particular user. The last_forward attribute of the is_buddy relation contains the phone numbers the buddy has been recently diverted to.

Finally the request_authorization relation defines requests for authorization. A request_authorization entry represents a request for authorization to add a user as buddy to another user's contactlist or to see more phonenumbers of a buddy. The request_date and result_date respectively represent the date when the request for authorization is created and completed. The completed attribute describes whether the buddy has processed the authorization_request and the status attribute describes the result of the processing, for example "rejected" or "accepted".

The database contains a database constraint that cannot be derived from the diagram. A `buddy_for_numbers` relation is valid if and only if the `user` to whom the `phonenumber` belongs is the same user as the buddy in the `is_buddy` relation.

In this database model both buddies and personal contacts are stored as users. Personal contacts are represented by `users` with the following properties:

- The `realuser` attribute has the value 'no'.
- All the `_available` attributes have the value 'no'.

And:

- The `availability` attributes of all `phonenumbers` belonging to this `user` have the value 'no'.
- The `user` belongs to exactly one `is_buddy` relation such that the `user` is the buddy of the (real) user who created the personal contact.

E.2.4 ICU Servlet

E.2.4.1 Overview

The ICU servlet is the server side application to maintain the contactlist, process authorization requests and to configure the service. As can be seen in the class diagram below, the `ICU` class extends the `HttpServlet` class. The ICU class uses some helper classes of the WAP Application Server,

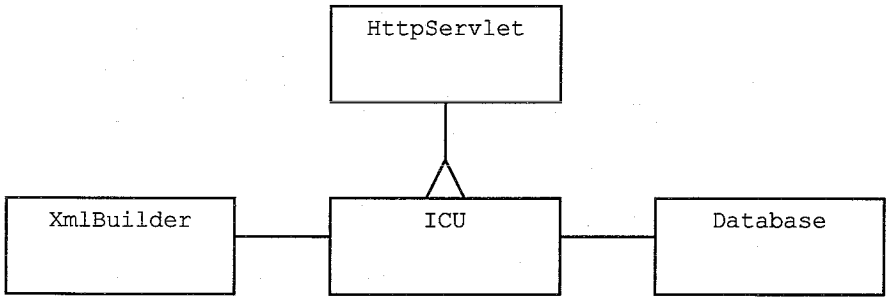


Figure 37. Class diagram of the ICU servlet.

namely the `Database` class and the `XmlBuilder` class. Besides these classes, some other standard Java classes are used. These classes are not described here.

The `Database` class provides an API for performing sessions with the database used in the WAP Application Server. The class provides methods to query the database.

The `XmlBuilder` class is used to build documents based on dynamic data. The `XmlBuilder` gets the path and file name for the template that should be

displayed, based on the information in a request. The file located may have both static and dynamic content. The dynamic content is created with the `XmlBuilder` commands. `XmlBuilder` goes through the commands in the template, applies data to it, and creates a static document that can be displayed in the terminal.

The `ICU` class is the main class where the most of the work takes place. The idea behind the servlet is quite easy: the first time a user accesses the servlet, a sessionobject is created. By using this sessionobject the server can store information about this session in a session object. When the user is authorized, the user can send a command parameter to the servlet, which will call a corresponding function to process the command. For some commands the servlet will use the database class to execute queries. After this the `XmlBuilder` is used to select a template, create a responsepage and return this page to the ICU servlet. Finally the ICU servlet will return the page to the user.

E.2.4.2 Flowdiagrams

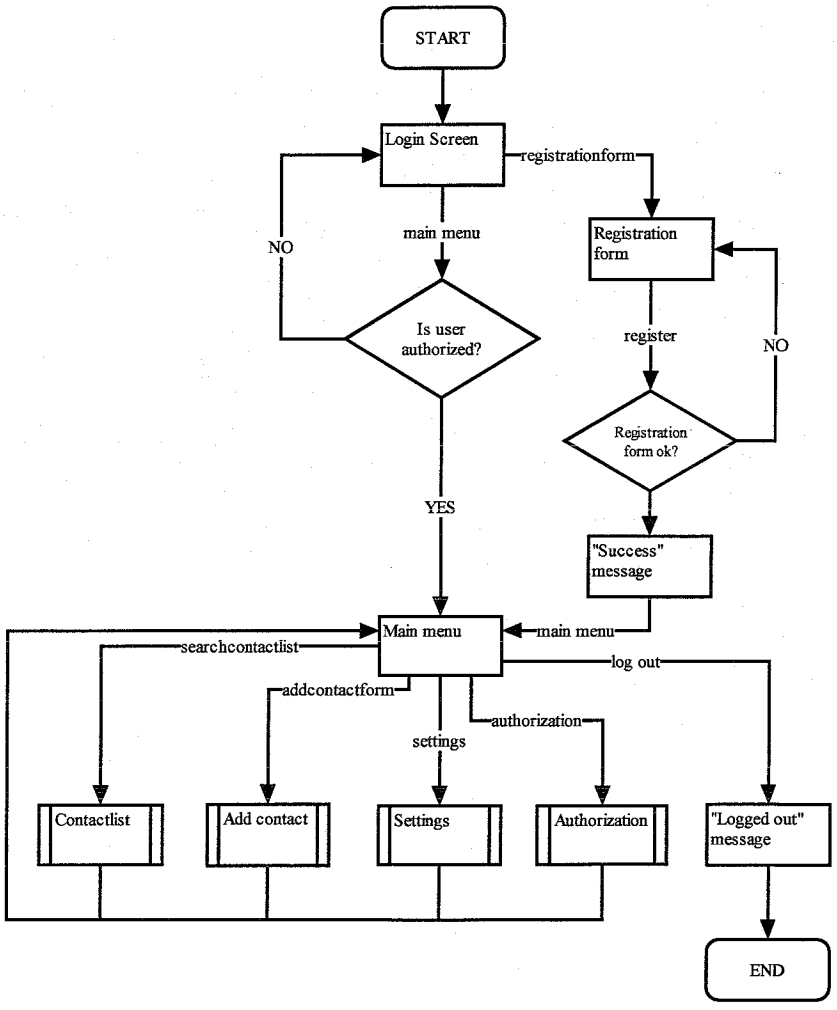


Figure 38. High level flowdiagram of the ICU servlet.

The user's flow through the ICU servlet can be described best by a kind of flowdiagram. In this diagram the boxes represent output pages and the arrows between boxes represent hyperlinks to the servlet, named with the corresponding command parameter. This means a box can have multiple outgoing arrows. A decision symbol is only drawn if the servlet has to make a choice that can result in different responsepages. Because these pages are displayed in a browser, it is always possible to use the "back" button of the browser to go back. It is also possible to go to the main menu from every page. These two links are only explicitly drawn in the diagram when necessary for the flow. A dotted line means that the link is only available for either WAP or web clients.

Figure 38 shows the high level flowdiagram of the ICU servlet. When the user connects to the server a loginscreen is shown and a sessionobject is created. The user can choose to log in with an existing userid and password, or fill in the registrationform to create a new account. Once the user has entered the correct userid and password combination, this is stored in the sessionobject and the main menu is shown. The main menu has four subflows that are shown below. The other option is to log out, which means that the session will be invalidated. During the user session, the session id is used to identify the session object.

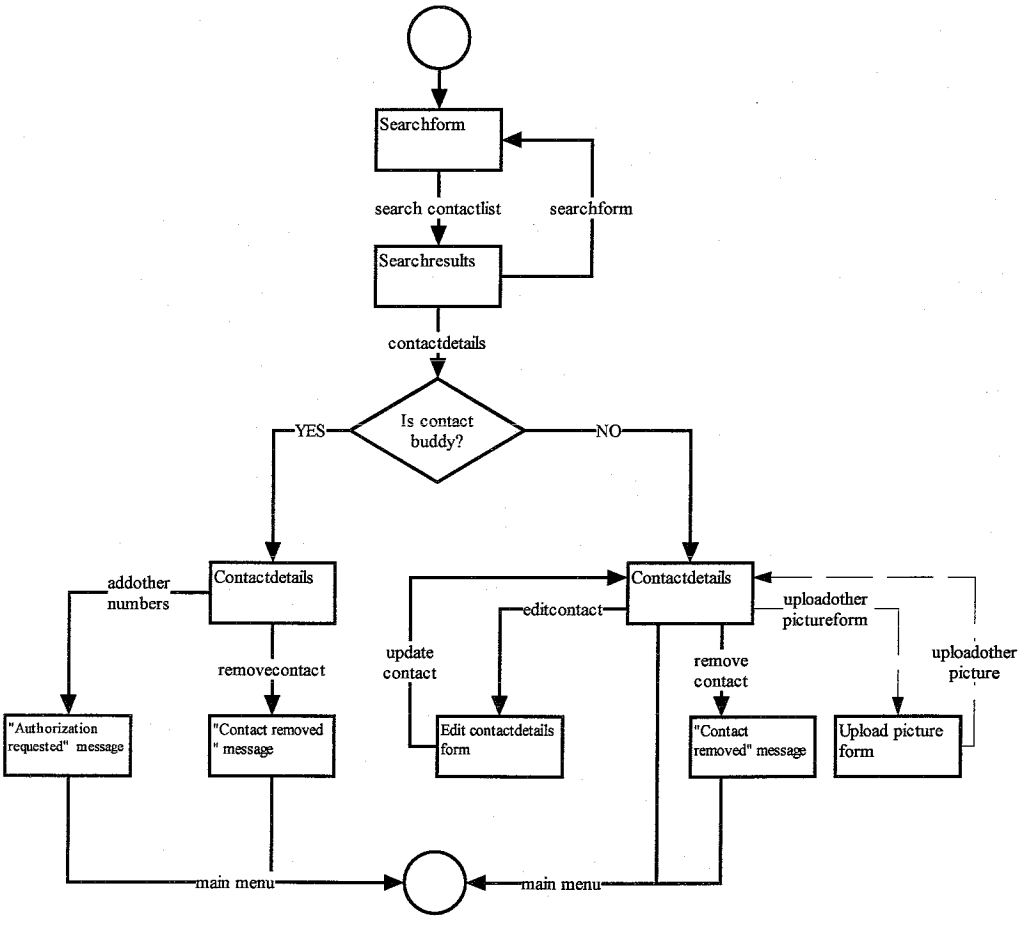


Figure 39. The contactlist subflowdiagram.

When the user selects the contactlist option in the main menu, a searchform is displayed. The user can search for a name in his contactlist. When no name is given, all contacts in the contactlist are shown in the searchresult. When one or more contacts are shown in the searchresult, the user can select one of these contacts to display the contactdetails of that contact. The operations that are possible on a contact depend on the contact. If the contact is a buddy, the user can remove the contact, or add possibly available phone numbers. When the contact is a personal contact, the user can edit the contactdetails, upload a picture to this contact or remove the contact from the contactlist. Uploading a picture is only possible from a web client. When the contactdetails are shown in a WML browser, the user can initiate a call using the public WTA function `makeCall`.

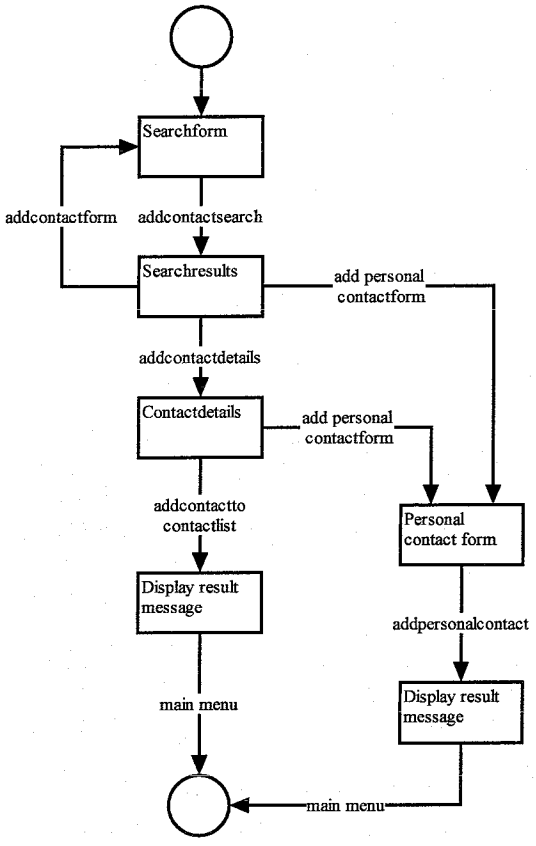


Figure 40. The add contact subflowdiagram.

When the user wants to add a contact to his contactlist, a search form is shown. The user can search for a by name and city. When one or more contacts are found, these contacts are displayed. The user can select one of the results to see the contactdetails or decide to create a personal contact by filling in the personal contact form. Only the contactdetails that are publicly available are shown to the user. The user can add the contact to his contactlist. In that case, the servlet checks whether authorization is needed. If authorization is needed, an authorization request is created otherwise the contact is added to the contactlist immediately. All publicly available phone numbers are added to the contactlist as well. If the contact has some private numbers, an authorization request for these numbers is created.

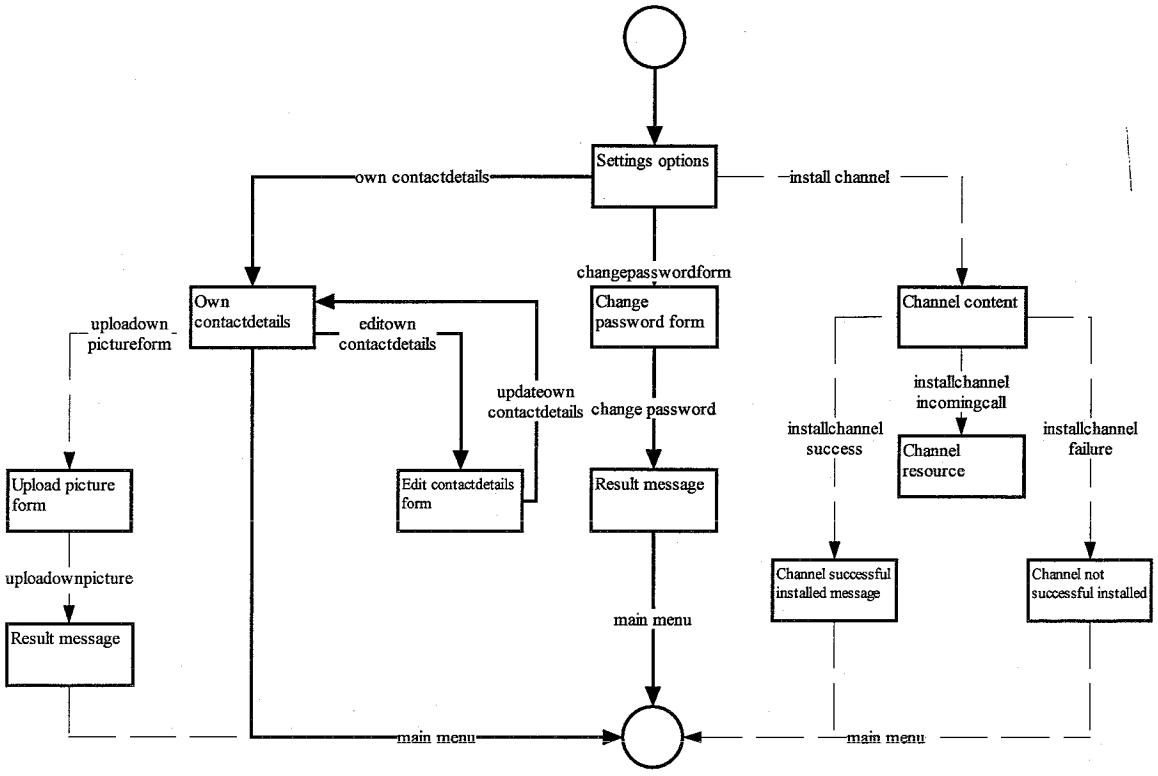


Figure 41. The settings subflowdiagram.

In the settings menu, the user can view and edit his contactdetails. If the user uses a web client, the user can also upload a picture to his account. The user can also change his password in the settings menu. When the user accesses the service from a WTA capable mobile phone, the user can install the ICU channel that is necessary to use the service.

When the user chooses to install the ICU channel, the servlet generates the channel and a new password to use the IncomingCall servlet. The channel references to the WTA-WML deck that must be installed in the repository. That's why this page, as can be seen in the subflowdiagram has no outgoing arrows. This page is only created to be installed. After the installation of the channel, either the success or the failure page will be requested. When the success page is requested the new password is updated in the database.

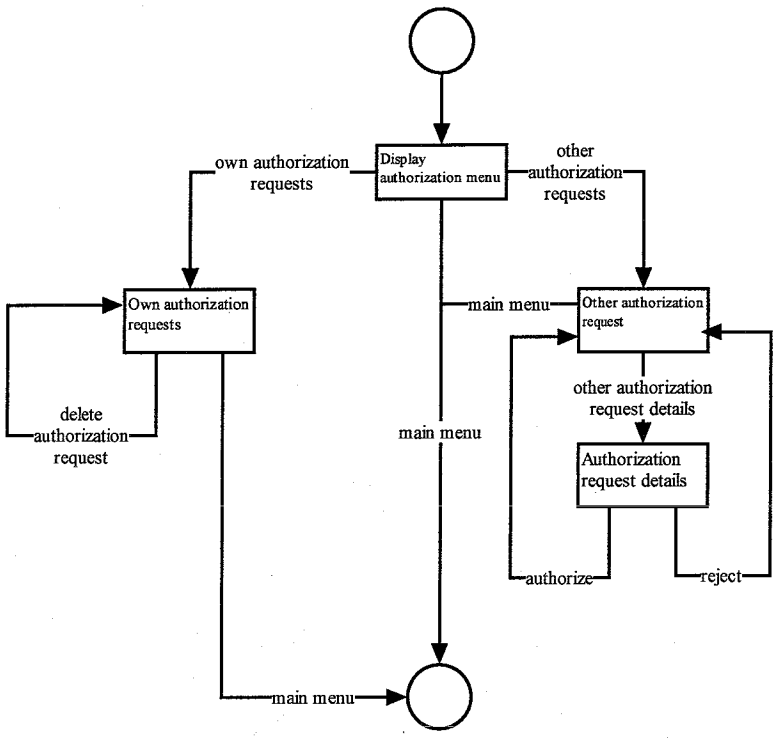


Figure 42. The authorization subflowdiagram.

Finally, when the user selects the authorization option in the menu, two options are displayed. The user can choose to view and delete his own authorization requests or to view authorization requests from others. When the authorization request from others are shown, the user can select to display the details of a request. When the details are displayed to the user, the user can select to reject the request or to authorize the request. Besides this choice, the user can also select, which phone numbers are visible to the other user. Only phone numbers that are not yet visible can be made visible. It is not possible to make a phone number invisible for a user once authorization is given to that user.

E.2.5 ICU channel

The purpose of the ICU channel is to take care that the appropriate picture is downloaded from the server. In fact this is the component that makes this service possible. This channel references to a WTA-WML deck, the `incomingcall` deck, which is stored in the repository after the channel installation. When an incoming call reaches the mobile phone, the WTA incoming call event is generated. This channel is bound to the WTA incoming call event and therefore the `incomingcall` deck, as specified in the channel will be loaded in the WTA browser. The `incomingcall` deck immediately redirects the browser to the `incomingCall` servlet on the server. Between this request and the response to this request some time elapses. The user should always be able to answer the call during this delay. Therefore the `incomingcall` deck contains temporary bindings for some events. When the user chooses to answer the call, a corresponding event is thrown. In that case the request is aborted and the browser sends a new request containing

the new state to the `IncomingCall` servlet. The user doesn't see anything of this.

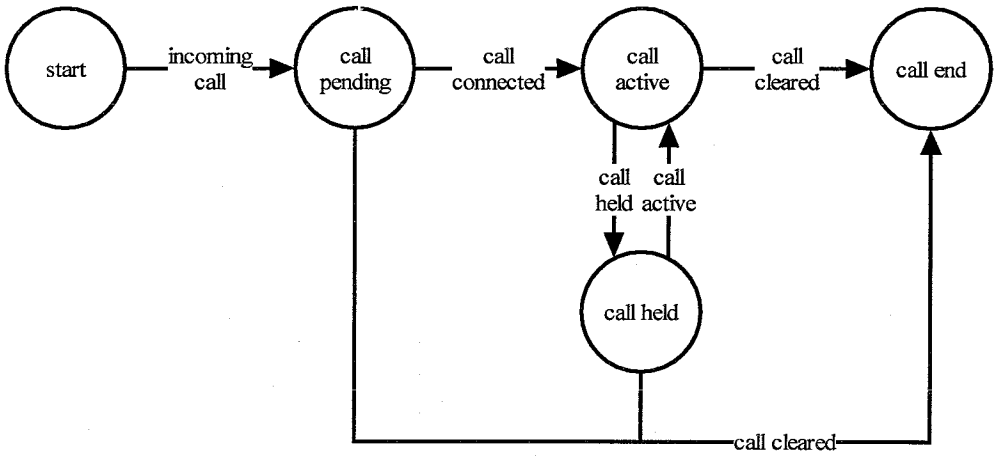


Figure 43. GSM call states [17]. The circles represent the possible call states. The arrows represent the events that are thrown when the state changes.

Figure 43 shows the possible call states and the events that are thrown when the state changes. The responsepages that are generated by the IncomingCall servlet use these states as well.

Besides the state, the incomingCall deck also contains a userid and password, which are sent to the IncomingCall servlet on an incoming call. For security reasons, this password is different from the password used in the ICU servlet. This password is generated during the installation of the channel in the repository.

E.2.6 IncomingCall servlet

E.2.6.1 Overview

The IncomingCall servlet generates the page that contains the picture of the calling party. The architecture of this servlet is very similar to the ICU servlet as can be seen in Figure 44.

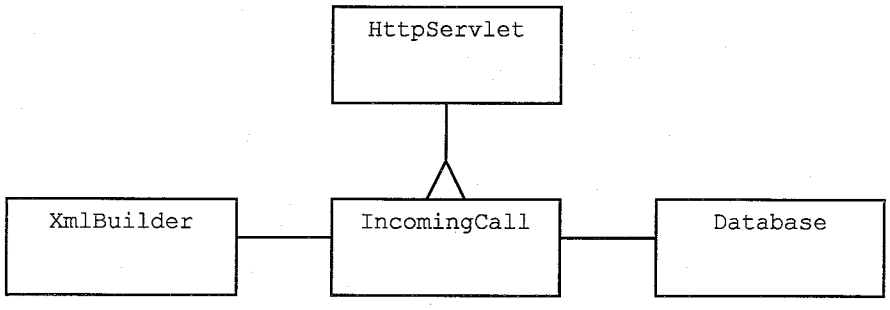


Figure 44. Class diagram of the IncomingCall servlet.

The `IncomingCall` class extends the `HttpServlet` class. The `XmlBuilder` class and the `Database` class are provided by the WAP

Application Server and are already shortly described in the previous section about the ICU servlet (E.2.4.1).

The `IncomingCall` class is the main class where the most of the work takes place. Globally, the user sends a command parameter to the servlet, which will call a corresponding function to process the command. For some commands the servlet will use the `Database` class to execute queries. After this the `XmlBuilder` is used to select a template, create a responsepage and return this page to the `IncomingCall` servlet. Finally the `IncomingCall` servlet returns the page to the user. In contrast to the ICU servlet, the IncomingCall servlet returns WTA-WML pages because these pages contain WTA functions. This also implies that the IncomingCall servlet is only available for mobile phones with WTA support.

E.2.6.2 Flowdiagram

The IncomingCall servlet is described using a kind of flowdiagram. In this diagram the boxes represent pages generated by the IncomingCall servlet. The arrows between boxes represent requests to the servlet. The text in an arrow represents the command parameter that is sent to the servlet. Finally diamonds are used to describe choices in the servlet that can result in different responsepages.

The incomingCall servlet first checks the user id and password. When the user is authorized, the incomingCall servlet first searches for the phone numbers in the contactlist that match the caller id. If one match is found, a WTA-WML deck containing the name and picture of the matching contact is displayed on the user's browser. When multiple matches are found, only the possible names are displayed. When no match is found, the incomingCall servlet will search the public phone numbers list. When more than one buddy are displayed, the user can select one of these buddies to get the details of that buddy. When the user forwards a call from a buddy, the forward attributes in the database are updated.

When one match is found in the public phone numbers list and some contactdetails are publicly available, these contactdetails will be shown to the user. In case more than one match is found, only the names will be shown to the user. When the user selects one of these names, the contactdetails of that contact will be shown. In case that the contactdetails of one contact are shown, the user can add this contact to his contactlist after the call has ended.

Finally when no match is found, only the number of the calling party will be shown. When this call has ended, the user can create a personal contact for this number.

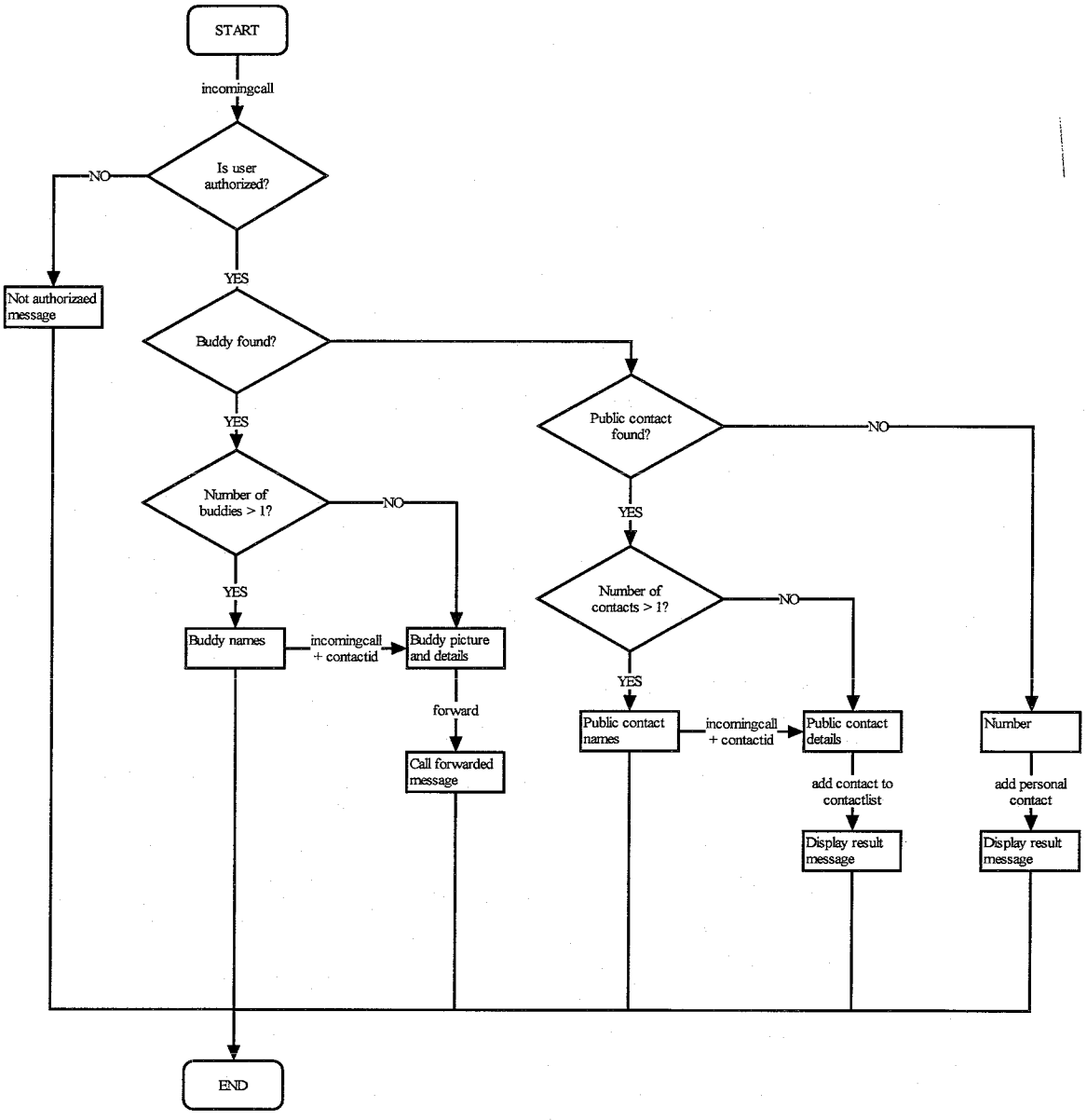


Figure 45. The flowdiagram of the IncomingCall servlet.

E.2.6.3 Response pages

The response pages do not only contain the name and picture, but also WTA functions to handle the call. Functions that can be used are for example accepting, rejecting and ending the call. When a call is ended, the user can add the contact to his contactlist, if the user is not in the contactlist yet. To use these call handling functions the decks contain multiple cards. These cards correspond with the possible call states. The following call states are possible: pending, active, held and end. The navigation between these cards is triggered by WTA events. See also Figure 43. By using these different states, only the useful options for a particular state are shown to the user. For example during the pending state of a call, the user can accept, forward or reject the call, while during the active state the options hold and end are

shown. To know the current state, the state is passed from the ICU channel to the IncomingCall servlet.

E.3 Implementation

E.3.1 Database

The database described in the E-R diagram above is implemented in MySQL using the following tables. An asterix (*) marks the key of a table. The types described in the tables are not exactly the types used in the MySQL implementations. For example the `boolean` type does not exist in MySQL. The `boolean` type is implemented as a `char`.

E.3.1.1 User

The user table as described below implements the entity user.

Column	Type	Description
`userid*`	`integer`	User id
`firstname`	`string`	Firstname
`middlename`	`string`	Middlename
`lastname`	`string`	Lastname
`company`	`string`	Company
`street`	`string`	Street
`zipcode`	`string`	Zipcode
`city`	`string`	City
`country`	`string`	Country
`email`	`string`	E-mail address
`picture`	`string`	Filename of the picture
`real_user`	`boolean`	`true` if user is real user, `false` else
`request_ authorization`	`boolean`	`true` if user is real user, `false` else
`name_ available`	`boolean`	`true` if name and company are publicly available, `false` else
`address_ available`	`boolean`	`true` if street and zipcode are publicly available, `false` else
`city_ available`	`boolean`	`true` if city is publicly available, `false` else
`country_ available`	`boolean`	`true` if country is publicly available, `false` else
`email_ available`	`boolean`	`true` if email ispublicly available, `false` else
`picture_ available`	`boolean`	`true` if picture is publicly available, `false` else

Table 14. The `user` table.

E.3.1.2 Phonenumber

The following `phonenumber` table implements the `phonenumber` entity. The relation `has_phonenumbers` is implemented by the column `userid`.

Column	Type	Description
`phoneid*`	`integer`	Phone id
`userid`	`integer`	Userid where this phonenumber belongs to
`phonenumber`	`string`	Phonenumber
`description`	`string`	Description of the phonenumber
`available`	`boolean`	`true` if the phonenumber is publicly available, `false` else

Table 15. The `phonenumber` table.

E.3.1.3 Password

The `password` table implements the `password` entity. The `has_password` relation is implemented by the `userid` column in the `password` table.

Column	Type	Description
`userid*`	`integer`	Userid where this password belongs to.
`loginuserid`	`string`	Userid used to log in to the service. This field must have a unique value in the table.
`password`	`string`	Password for the ICU servlet
`incomingcall password`	`string`	Password for the incomingCall servlet

Table 16. The `password` table.

E.3.1.4 `Is_buddy` relation

The `is_buddy` relation is implemented by the `buddy_relation` table.

Column	Type	Description
`buddy_ relationid*`	`integer`	Buddyrelationid for this relation
`userid`	`integer`	Userid of the user to whom this buddyrelation belongs
`buddyid`	`interger`	Userid of the buddy
`lastforward1`	`string`	Most recent phone number to which the buddy is forwarded.
`lastforward2`	`string`	Second most recent phone number to which the buddy is forwarded.

lastforward3	string	Third recent phone number to which the buddy is forwarded.

Table 17. The buddy_relation table.

The buddy_for_numbers relation is implemented by the buddy_relation_phonenumber table.

Column	Type	Description
buddy_ relationid*	integer	Buddyrelationid of the buddy_relation where this relation belongs to
phoneid*	integer	Phoneid of the phonenumber that belongs to the buddy_relation

Table 18. The buddy_relation_phonenumber table.

E.3.1.5 Request_authorization relation

The request_authorization relation is implemented by the authorization_request table.

Column	Type	Description
authorization _requestid*	integer	Authorizatoinrequestid of this request for authorization
userid	integer	Userid of the user who the authorization request belongs to
buddyid	integer	Userid of the user the authorization request is meant for
request_date	date	Date of request
response_date	date	Date of response
status	string	Status of the authorization request
completed	boolean	true if the request is processed, false else

Table 19. The buddy_relation_phonenumber table.

E.3.2 ICU servlet

The ICU class contains the following functions. The servlet engine calls the init() function when the servlet is called for the first time. When a get or post request is sent to the servlet, the doGet() or doPost() functions are called. These standard servlet functions both call the doAll() function such that every request, independent of whether it is a get or post request, calls the doAll() function. The destroy() function is called when the servlet engine wants to remove the servlet.

E.3.2.1 Attributes

The ICU class has the following global variables. Besides these variables the class contains some constants as well, but these are not discussed here.

- `Database dbICU`
  `dbICU` contains a instance of the Database object. `dbICU` contains the database as described above.

- `HttpSession session`
  `session` contains the session object for the current request.

E.3.2.2 Init

```
public void init(ServletConfig config)
```

`init()` first calls the `init()` function of the super class and creates a instance of the `Database` class. If the creation is successful, it assigns the instance to `dbICU`.

E.3.2.3 Destroy

```
public void destroy()
```

`destroy()` calls the `destroy()` function of the Database `dbICU` and calls the `destroy()` function of the super class.

E.3.2.4 DoAll

```
void doAll(HttpServletRequest req,
  HttpServletResponse res)
```

`doAll()` is called on every request. First the sessionobject is requested from `req`. If the sessionobject is new, the user-agent type is determined to determine the outputtype. This can be wml or html. This outputtype is stored in the sessionobject. If the session is not new, the outputtype is retrieved from the sessionobject.

When the sessionobject is available (just created or already available), the authentication is checked. If a userid is stored in the sessionobject, the user is authenticated. Otherwise `logInScreen()` is called to return the loginscreen. When the `loginuserid` and `password` parameters are included in the request, `authenticate()` is called. If this function return a userid, this means that the user is authorized to use the service. Therefore this userid is stored in the sessionobject. Otherwise `loginScreen()` is called.

Now the user is authenticated. If the contenttype of `req` is `multipart/form-data` this is a picture upload request and `uploadPicture()` is called. Otherwise the command parameter is retrieved from the request `req`. Now the function corresponding to the command is

called. For example if the command is mainmenu, `mainMenu()` is called. All these functions return a string, which is returned to the client.

E.3.2.5 Authenticate

```
private String authenticate(String logInUserid,
  String logInPassword)
```

`authenticate()` queries the `password` table in the database with the parameters `logInUserid` and `logInPassword`. If a result is found, the userid of this user is returned otherwise `authenticate()` returns `null`.

E.3.2.6 LogInScreen

```
private String logInScreen(HttpServletRequest req,
  HttpServletResponse res, String outputType)
```

`loginScreen()` returns the `LogInScreen` template.

E.3.2.7 LoggedOutScreen

```
private String loggedOutScreen(HttpServletRequest req,
  HttpServletResponse res, String outputType)
```

`loggedOutScreen()` returns the `LoggedOutScreen` template.

E.3.2.8 ShowMessage

```
private String showMessage(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String title, String message)
```

`showMessage()` returns a standard message page. `title` contains the title of this page and `message` contains the message.

E.3.2.9 RegistrationForm

```
private String registrationForm(HttpServletRequest req,
  HttpServletResponse res, String outputType)
```

`registrationForm()` returns the registrationform. The `registrationForm` template is used for this page. The template contains form variables that must have an initial value. Therefore all these variables are initialized with "".

E.3.2.10 Register

```
private String register(HttpServletRequest req,
  HttpServletResponse res, String outputType)
```

`register()` registers a user in the database. First `register()` checks whether the user has chosen a unique `loginuserid`. When `loginuserid`

already exists in the `password` table, all the parameters that the user has filled in yet are sent to the `registrationForm` template including a message containing the message about the failure. So the user does not have to fill in all the details again.

If the `loginuserid` does not exist yet, the two password given by the user are compared to assure that the user has not made a typing error. When both passwords are equal, the user is registered to the service. Otherwise the `registrationForm` is returned with an error message.

To register a user to the service, first a new record is added to the `user` table. Then a record is added to the `password` table with the chosen `loginuserid` and `password`. After this, for every phone number a record is added to the `phonenumber` table. Finally the `userid` is saved in the sessionobject such that the user is authenticated and `showMessage()` is called.

E.3.2.11 MainMenu

```
private String mainMenu(HttpServletRequest req,
  HttpServletResponse res, String outputType)
```

`mainMenu()` returns the `mainMenu` template.

E.3.2.12 SearchContactlist

```
private void searchContactlist(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`searchContactList()` retrieves the `searchstring` parameter from `req` and uses this value to find contacts in the contactlist. If `searchstring` is an empty string, all contacts in the contactlist will be returned. The result of the query is stored in the sessionobject.

E.3.2.13 Contactlist

```
private String contactlist(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`contactList()` returns the contactlist. The input for the contactlist is stored in the sessionobject. If the contactlist is too long, the list is divided in a number of pages and the appropriate contacts are sent to the `XmlBuilder`. The `XmlBuilder` uses the `contactList` template to create a responsepage.

E.3.2.14 ContactDetails

```
private String contactDetails(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`contactDetails()` queries the database for the requested `contactid`. If a `is_buddy` exists between the user and the contact, this contact is sent to the `XmlBuilder`. Then all `phonenumbers` that are related to this `is_buddy` relation are sent to the `XmlBuilder` as well. After this `makeTemporaryPicture()` is called to make a temporary copy of the picture file. Finally `XmlBuilder` uses the `contactDetails` template to create a responsepage.

E.3.2.15 EditContactList

```
private String editContactlist(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`editContactList()` first checks whether the given `contactid` really is a personal contact. If it is a personal contact, the contactdetails are requested from the database. Then the phone numbers belonging to this contact are requested from the database. Finally the `editForm` template is used to create a form. In this form the fields are filled with the contactdetails.

E.3.2.16 UpdateContactList

```
private String updateContact(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`updateContactList()` processes the `editForm` as created in `editConactList()`. `updateContactList()` first checks whether the given `contactid` actually is a personal contact of the current user. If this is true, the user record in the database is updated with the possibly changed values. After this for every the `phoneid` of every phone number in the `editContactForm` is checked. If it is a new phone number, this number is added to the database. If it is changed, the database is updated, and if the field is empty, the `phonenumber` and the `is_buddy_phonenumber` relation are removed.

E.3.2.17 RemoveContactConfirmation

```
private String removeContactConfirmation(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`removeContactConfirmation()` returns the `removeContactConfirmation` template.

E.3.2.18 RemoveContact

```
private String removeContact(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`removeContact()` removes the contact with the given `contactid` from the user's `contactlist`. First the database is queried to assure that a `buddyrelation` exists. Then `removeContact()` checks whether the contact is a real user. If the contact is a real user, first all the `buddyrelation_phonenumbers` for this `buddyrelation` and the contact's `phonenumbers` are removed. After this the `buddyrelation` is removed.

If the contact is not a real user, thus a personal contact, some additional actions are performed. The `phonenumbers` for this contact are removed, the `user` record is removed and if a picture is available, this picture is removed as well.

E.3.2.19 AddContactForm

```
private String addContactForm(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid )
```

`addContactForm()` returns the `addContactForm` template.

E.3.2.20 AddPersonalContactForm

```
private String addPersonalContactForm(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid )
```

`addPersonalContactForm()` returns the `addPersonalContactForm` template.

E.3.2.21 SearchPublicContacts

```
private void searchPublicContacts(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`SearchPublicContacts()` retrieves the search parameter from `req` and uses these parameters to find contacts in the user table. Besides these search parameters, the contacts must be real users and at least their name must be public available. The result of the query is stored in the sessionobject.

E.3.2.22 AddContactSearchResults

```
private String addContactSearchResults(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`AddContactSearchResults()` returns the searchresults. The results are stored in the sessionobject by `addContactSearch()`. If the list is too long, the list is divided in a number of pages and the appropriate contacts are sent

to the `XmlBuilder`. The `XmlBuilder` uses the `addContactSearchResults` template to create a responsepage.

E.3.2.23 AddContactDetails

```
private String addContactDetails(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`addContactDetails()` queries the database for the contactdetails for the given `contactid`. If a result is found, all the publicly available information is passed to the `XmlBuilder`. After this all the `phonenumbers` that belong to this contact and that are publicly available are sent to the `XmlBuilder`. Finally the `XmlBuilder` creates a responsepage using the `addContactDetails` template.

E.3.2.24 AddContactToContactlist

```
private String addContactToContactlist(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`addContactToContactlist()` adds a contact to the user's contactlist. First `addContactToContactlist()` checks whether the given `contactid` is a real user. After this the database is queried to check whether a `buddyrelation` already exists. If this is true, an error message is shown. When no `buddyrelation` exists yet and the user requires `authorization`, a `request_authorization` record is created. If the user does not require `authorization`, a `buddyrelation` for this user is created. When the `buddyrelation` is created, for every publicly available `phonenumber` that belongs to this buddy, a `buddyrelation_phonenumber` entry is created. If the buddy has some private phone number as well, a `request_authorization` is created. Finally `showMessage()` is called to display the result.

E.3.2.25 AddPersonalContact

```
private String addPersonalContact(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid )
```

`addPersonalContact()` processes the `addPersonalContactForm`. First a new `user` is created with the given parameters. This `user` is a personal user, so the `real_user` and all the `_available` fields get the value `'n'`. After this for every `phonenumber` that is not empty, a `phonenumber` record is created. After this a `buddyrelation` record is inserted in the database and finally for every created `phonenumber` a `buddyrelation_phonenumber` record is inserted.

E.3.2.26 SettingsMenu

```
private String settingsMenu(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`settingsMenu()` returns the `settingsMenu` template.

E.3.2.27 OwnContactDetails

```
private String ownContactDetails(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`ownContactDetails()` requests all the contactdetails that belong to the current user and sends these details to the `XmlBuilder`. The `makeTemporaryPicture()` function is called to make a temporary copy of the picture. The `XmlBuilder` uses these details in the `ownContactDetails` template.

E.3.2.28 EditOwnContactDetailsForm

```
private String editOwnContactDetailsForm(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`editOwnContactDetailsForm()` requests the user's contactdetails from the database and sends these details to the `XmlBuilder`. The `editOwnContactDetailsForm` template is used to create a form that contains all the current user details.

E.3.2.29 UpdateOwnContactDetails

```
private String updateOwnContactDetails(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`updateOwnContactDetails()` processes the `editOwnContactDetailsForm`. The contactdetails are updated. For every phone number `updateOwnContactDetails()` checks whether it is a new phone number, an already existing number that must be updated, or an empty number that should be removed. When a phone number must be removed, all the `buddyrelation_phonenumbers` that contain the phone number are removed as well.

E.3.2.30 ChangePasswordForm

```
private String changePasswordForm(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`changePasswordForm()` returns the `changePasswordForm` template.

E.3.2.31 ChangePassword

```
private String changePassword(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`changePassword()` processes the form created by `changePasswordForm()`. First the current password is checked in the database. If this is correct, the new passwords are compared. When the new password are equal, the password is updated in the database.

E.3.2.32 UploadOwnPictureForm

```
private String uploadOwnPictureForm(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`uploadOwnPictureForm()` returns the `uploadOwnPictureForm` template.

E.3.2.33 UploadOtherPictureForm

```
private String uploadOtherPictureForm(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`uploadOtherPictureForm()` returns the `uploadOtherPictureForm` template.

E.3.2.34 UploadPicture

```
private String uploadPicture(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`uploadPicture()` processes the upload of a picture. This picture is uploaded to the servlet using the `multipart/form-data` contenttype. Therefore the command parameter, which is normally retrieved from the `req` using the `getParameter()` function can not be used now. The `uploadPicture()` function parses the data and searches for the command parameter. If the command parameter is `uploadownpicture`, the picture is saved on the server and the `picture` attribute of the `user` record is updated with the picture name. If the command parameter is `uploadotherpicture`, the `contactid` of the personal contact is checked. If this is correct, the file is saved and the `picture` field of the `user` is updated.

In this implementation the uploaded files are not checked and both the GIF and WBMP files must be uploaded.

E.3.2.35 InstallChannel

```
private String installChannel(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`installChannel()` is only available for WTA supporting clients.
`installChannel()` returns the `channel` template.

E.3.2.36 InstallChannelIncomingCall

```
private String installChannelIncomingCall(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`installChannelIncomingCall()` is called during the installation of the channel. The output of this function is the page that is installed in the repository. In this function a password is generated (not implemented in this implementation) and sent to the `XmlBuilder`. Finally the `installChannelIncomingCallWTAWML` template is used to create the response.

E.3.2.37 InstallChannelSuccess

```
private String installChannelSuccess(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`installChannelSuccess()` is called when the channel is successfully installed. Therefore the password is updated in the database. Finally the `installChannelSuccessWTAWML` template is returned.

E.3.2.38 InstallChannelFailure

```
private String installChannelFailure(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`installChannelFailure()` returns the `installChannelFailureWTAWML` template.

E.3.2.39 AuthorizationMenu

```
private String authorizationMenu(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`authorizationMenu()` returns the `authorizationMenu` template.

E.3.2.40 OtherAuthorizationRequests

```
private String otherAuthorizationRequests(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`otherAuthorizationRequests()` requests all `request_authorizations` from the database that are not completed yet, and have `userid` as `buddyid`. If this list is too long, the list is divided into smaller parts. Finally the appropriate requests are sent to the `XmlBuilder` which uses the `otherAuthorizationRequests` template to create the response.

E.3.2.41 OtherAuthorizationRequestDetails

```
private String otherAuthorizationRequestDetails(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`otherAuthorizationRequestDetails` returns the details of an authorization request. First the authorizationrequestdetails are retrieved from the database. After this the database is queried to find out whether there exists a `buddyrelation` for this contact. Then all private numbers that are not connected to this `buddyrelation` are sent to the `XmlBuilder`. Finally the `XmlBuilder` uses the `authorizationRequestDetails` template to create the responsepage.

E.3.2.42 Authorize

```
private String authorize(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`authorize()` processes an authorizationrequest displayed in the page generated by `otherAuthorizationRequestDetails().authorize()` first checks the value of the `action` parameter. If the value equals `reject`, the `status, completed` and `response_date` attributes of this `request_authorization` record are updated. If the value of the `action` parameter equals `authorize`, `authorize()` checks whether a `buddyrelation` between these `users` exists. If not, a `buddyrelation` is created. Now all numbers that are explicitly authorized in the form are added to this `buddyrelation`. Then for all publicly available numbers that are not yet related to the buddyrelation, a `buddyrelation_phonenumber` record is added to the database. Finally the `request_authorization` record is updated and `otherAuthorizationRequests()` is called.

E.3.2.43 OwnAuthorizationRequests

```
private String ownAuthorizationRequests(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`ownAuthorizationRequests()` requests all `request_authorizations` from the database that have `user_id` as `userid`. If this list is too long, the list is divided into smaller parts. Finally the appropriate requests are sent to the `XmlBuilder` which uses the `ownAuthorizationRequests` template to create the response.

E.3.2.44 DeleteAuthorizationRequests

```
private String deleteAuthorizationRequest(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`deleteAuthorizationRequest()` deletes the `request_authorization` records with the given `authorization_requestid` and calls `ownAuthorizationRequests()`.

E.3.2.45 MakeTemporaryPicture

```
private boolean makeTemporaryPicture(String sessionId,
  String pictureName)
```

`makeTemporaryPicture()` copies the picture denoted by `pictureName` to a temporary directory, identified by the current `sessionId`. Using these temporary copies, it is impossible to directly download pictures from the imagesdirectory on the server to the client. Only if a user is authorized to see a picture, and thus a copy is made, the user can see this picture.

E.3.2.46 SessionListener

```
class SessionListener implements
  HttpSessionBindingListener
```

The `SessionListener` class is used to destroy temporary files when a session is invalidated. An instance of the `SessionListener` is saved in every sessionobject and when a sessionobject expires, the `valueUnbound()` function is called. This function deletes all files in the temporary directory belonging to this session and finally removes the directory itself.

E.3.3 ICU channel

The ICU channel is bound to the `wtaev-ic`, the incoming call event. The channel contains a link to the `IncomingCall` deck that is installed in the repository.

The IncomingCall deck consists of one card as can be seen in the extract below:

```
<wta-wml>
<card id="pending">
  <onevent type="ontimer">
    <go href="http://icu.ericsson.nl/servlets/incomingCall"
      sendreferer="false" method="get">
```

```
      <setvar name="callerId" value="$1"/>
      <setvar name="callHandle" value="$2"/>
      <postfield name="command" value="incomingCall"/>
      <postfield name="userid" value="a userid"/>
      <postfield name="password" value="a password"/>
      <postfield name="callerid" value="$callerId"/>
      <postfield name="callhandle" value="$callHandle"/>
      <postfield name="callstatus" value="pending"/>
      </go>
    </onevent>
    <timer value="1"/>
    <onevent type="wtaev-gsm/ch">
      <go href="http://icu.ericsson.nl/servlets/incomingCall"
          sendreferer="false" method="get">
      <setvar name="callerId" value="$1"/>
      <setvar name="callHandle" value="$2"/>
      <postfield name="command" value="incomingCall"/>
      <postfield name="userid" value=" a userid"/>
      <postfield name="password" value="a password"/>
      <postfield name="callerid" value="$callerId"/>
      <postfield name="callhandle" value="$callHandle"/>
      <postfield name="callstatus" value="held"/>
      </go>
    </onevent>
    ...
    <p>
    Requesting information for incoming call from: $(callerId) ...
    </p>
  </card>
  </wta-wml>
```

This card contains a number of <onevent> tags. These tags are bound to the possible WTA-events. As an example, the call held event (wtaev-gsm/ch) is shown above. For the other events similar pieces of code exist. On the occurrence of the call held event, the browser is sent to the IncomingCall servlet, using the postfields specified in between the <go> and </go> tags to send the parameters. The "a userid" and "a password" fields are replaced by the user's userid and password at the moment of installation. The callHandle parameter is required to use the WTA functions.

The first <onevent> tag is bound to a timer such that after 1 ms the browser is redirected to the server.

E.3.4 IncomingCall servlet

The IncomingCall class contains the following functions. The servlet engine calls the init() function when the servlet is called for the first time. When a get or post request is sent to the servlet, the doGet() or doPost() functions are called. These standard servlet functions both call the doAll() function such that every request, independent of whether it is a get or post request, calls the doAll() function. The destroy() function is called when the servlet engine wants to remove the servlet.

E.3.4.1 Attributes

The `IncomingCall` class has the following global variables. Besides these variables the class contains some constants as well, but these are not discussed here.

- `Database dbICU`
  `dbICU` contains an instance of the Database object. `dbICU` contains the database as described above.

- `HttpSession session`
  `session` contains the session object for the current request.

E.3.4.2 Init

```
public void init(ServletConfig config)
```

`init()` first calls the `init()` function of the super class and creates a instance of the `Database` class. If the creation is successful, it assigns the instance to `dbICU`.

E.3.4.3 Destroy

```
public void destroy()
```

`destroy()` calls the `destroy()` function of the Database `dbICU` and calls the `destroy()` function of the super class.

E.3.4.4 DoAll

```
void doAll(HttpServletRequest req,
  HttpServletResponse res)
```

`doAll()` is called on every request to the server. `doAll()` first checks whether a sessionobject exists. If not, a new sessionobject is created and the device type of the client is stored in the sessionobject. Now a session exists and `doAll()` checks whether the session contains a userid. If the sessionobject contains an userid this implies the user is authenticated. If no userid can be found, `authenticate()` is called. If `authenticate()` returns a userid, this userid is stored in the sessionobject, and thus the user is authenticated. Otherwise `showMessage()` is called to display an error message. Now the user is authenticated. Finally `doAll()` retrieves the command parameter from `req` and calls the appropriate function.

E.3.4.5 ShowMessage

```
private String showMessage(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String title, String message)
```

`showMessage()` uses the `showMessageIncomingCall` template to generate a message to the user.

E.3.4.6 Authenticate

```
private boolean authenticate(String userid,
  String password)
```

`authenticate()` queries the `password` table in the database. If a result is found, the userid of this user is returned otherwise `authenticate()` returns `null`.

E.3.4.7 IncomingCall

```
private String incomingCall(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

`incomingCall()` first checks whether the `contactid` parameter has a value. If `contactid` equals `null`, `incomingCall()` queries the database for buddies that have the same phone number as the `callerid`. If one buddy is found, the `IncomingCallBuddy` template is returned. If more than one buddy are found, the `IncomingMultBuddy` template is returned. When no buddies are found, the public phone numbers are queried. If one public number is found, and at least the name of the user the phone number belongs to is publicly available, the `IncomingCallPublic` template is returned. When more than one public number is available, the `IncomingCallMultPublic` template is returned. Finally when no numbers are found, the `IncomingCallUnknown` template is returned.

When the `contactid` does not equal `null`, this means that a specific contact is requested. Therefore the same steps are taken, but the query has an additional criterion, namely the `userid` must match the value of `contactid`.

In the description above not the real names of the templates are used. For every named template four templates exist, namely one for every call state. For example the `incomingCallBuddy` template has the following templates: `incomingCallBuddyPending`, `incomingCallBuddyActive`, `incomingCallBuddyHeld` and `incomingCallBuddyEnd`. Based on the value of the `callStatus` parameter, the corresponding template is used. These templates are in fact the same, but the order of the cards in these templates are different, such that the card that corresponds with the current state is the first card in the file, and is therefore presented first in the browser.

E.3.4.8 Forward

```
private String forward(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid)
```

The `forward()` function is called when a call has been forwarded. First the `buddyrelation` is queried from the database. Then the `buddyrelation` record is updated in the database with the number presented by the `number` parameter. Finally the buddy's contactdetails are requested from the database and the `incomingCallBuddyForwarded` template is returned.

E.3.4.9 AddPersonalContact

```
private String addPersonalContact(HttpServletRequest req,
  HttpServletResponse res, String outputType,
  String userid )
```

`addPersonalContact()` processes the `addPersonalContactForm`, which is present in the `IncomingCallUnknown` template. First a new `user` is created with the given parameters. This `user` is a personal user, so the `real_user` and all the `_available` fields get the value `'n'`. After this for every `phonenumber` that is not empty, a `phonenumber` record is created. After this a `buddyrelation` record is inserted in the database and finally for every created `phonenumber` a `buddyrelation_phonenumber` record is inserted.

E.3.4.10 AddContactToContactList

```
private String addContactToContactlist(
  HttpServletRequest req, HttpServletResponse res,
  String outputType, String userid)
```

`addContactToContactlist()` adds a contact to the user's contactlist. First `addContactToContactlist()` checks whether the given `contactid` is a real user. After this the database is queried to check whether a `buddyrelation` already exists. If this is true, an error message is shown. When no `buddyrelation` exists yet and the user requires `authorization`, a `request_authorization` record is created. If the user does not require `authorization`, a `buddyrelation` for this user is created. When the `buddyrelation` is created, for every publicly available `phonenumber` that belongs to this buddy, a `buddyrelation_phonenumber` entry is created. If the buddy has some private phone number as well, a `request_authorization` is created. Finally `showMessage()` is called to display the result.

E.3.4.11 MakeTemporaryPicture

```
private boolean makeTemporaryPicture(String sessionId,
  String pictureName)
```

`makeTemporaryPicture()` copies the picture denoted by `pictureName` to a temporary directory, identified by the current `sessionId`. Using these temporary copies, it is impossible to directly download pictures from the imagesdirectory on the server to the client. Only if a user is authorized to see a picture, and thus a copy is made, the user can see this picture.

E.3.4.12 SessionListener

```
class SessionListener implements
  HttpSessionBindingListener
```

The `SessionListener` class is used to destroy temporary files when a session is invalidated. An instance of the `SessionListener` is saved in every sessionobject and when a sessionobject expires, the `valueUnbound()` function is called. This function deletes all files in the temporary directory belonging to this session and finally removes the directory itself.

E.3.5 Implementation Remarks

The implementation as discussed above is meant to be used as prototype for demonstration purposes only. The application is for example not very robust. When a user tries to send commands to the servlet using self-defined get requests, this might lead to problems. Also, the application does not check the input given by the user like phone number formats and picture formats. When this application is implemented for real use, these checks must be implemented.

The MySQL version that is used does not support transaction safe tables. As a result of this, the commands that use more than one table during an update can not be seen as atomic commands. When during the executing of one of these commands an error occurs, the database might become inconsistent. The use of transaction safe tables should overcome this problem.

Due to the lack of devices with WTA support, the WTA functions are not tested yet. The incomingCall servlet and the incoming call channel are built using the specifications only and therefore it is not for sure how they will behave exactly in practice.

When during a call a picture is shown to the user, the incoming call event should be handled by a temporary binding to handle second incoming calls. In this prototype this is not implemented.