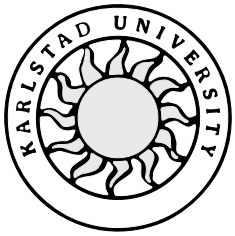

Computer Science

Mattias Mollstedt

Johan Keikkala

WAP services in UMTS

Bachelor's Project

2000:09

WAP services in UMTS

Mattias Mollstedt

Johan Keikkala

© 2000 The Authors and Karlstad University

This report is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not our own work has been identified and no material is included for which a degree has previously been conferred.

Mattias Mollstedt

Johan Keikkala

Approved 2000-05-30

Advisor: Johan Garcia

Examiner: Stefan Lindskog

Abstract

This document describes WAP in UMTS and the possibilities with services in this area. It also describes CORBA and Parlay, which can be used when creating these services. CORBA is used for distributed computing, and Parlay is an open API for networks aiming on the convergence between fixed-, wireless networks and public telephone network. A major part of the document focuses on the service area and especially on WTA services. We have tried to give the reader a general introduction to the different wireless technologies, before getting in to the service concepts. A demo is included in order to describe a possible WTA service using CORBA. In the end we examine future service evolution, both from a technical and a user point of view. The document concludes that there will be many new useful services available in the future, but how fast the progress of service development will be mainly depends on the standardization of the fundamental parts in the wireless environment and the new architecture.

Acknowledgements

Finally our work with the Bachelor's project is finished. When looking back, we feel that it has been a very interesting time working with the project, but there has been a lot of hard work on the way. We would like to take the opportunity to thank some people who have been helping us in different ways during this period. Bo Hagengren our supervisor at Ericsson Infotech, for creating such an interesting project within an area that is highly topical. Johan Garcia our supervisor at Karlstad University, for his commitment and support when we needed it the most, we really appreciate all your help. Mikael Larsson our coach at Ericsson Infotech for helping us to get started when we first arrived at Ericsson. We would also like to thank Helena Lindskog and Magnus Lindström at Ericsson Infotech, who continuously followed up our work, making sure that we had all the equipment needed.

Contents

List of Figures

List of tables

1 Introduction

1.1 General introduction

Mobile communications have grown rapidly over the past years.

The first generation systems had analogue cellular radio technology providing speech. Simply each person was assigned to a frequency to use for each call. Each country developed its own system or in cases like NMT in a couple of countries.

The desire for intercontinental functionality and data transfer possibilities led to the GSM specification in Europe 1990. It is defined as a second-generation mobile system (2G). It enabled roaming which makes it possible to use services in different GSM service areas. It also provided possibilities to transfer data over the air with the speed of 9.6kb/s . The data transfer gives digital data features such as Short Messaging Service (SMS) and by connecting to a laptop enabling it to send or receive e-mail, faxes, browse the Internet or access company LAN/intranet. Later 57.6kb/s became possible with HSCSD (High Speed Circuit Switched Data).

Today's systems are envisioned to be extended with packet data service beside the circuit-switched. This uses the bandwidth more effectively and is easier to connect with fixed data networks. For example, GPRS (General Packet Radio Service) is a packet-switched data technology that is being developed for GSM networks and will initially give 115Kb/s-transmission speed for data. GPRS and HSCSD will eventually converge at EDGE (Enhanced Data rates for Global Evolution), which will provide better use of bandwidth increasing the data rate of the existing GSM systems by a factor of three. This upgrades the throughput to possible 384kb/s rendering the possibility to create almost 3G-like services.

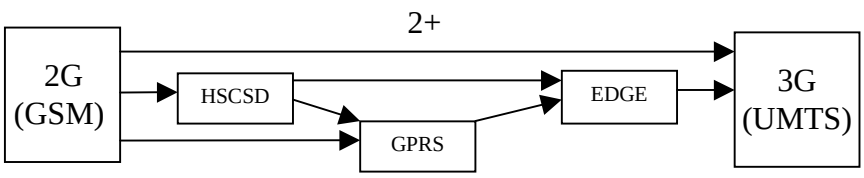

Figure 1-1: Development from 2G (GSM) towards 3G (UMTS).

UMTS, which stands for Universal Mobile Telecommunications System [16], is one of the major third-generation mobile systems (3G). Third generation offer higher efficiency by using new radio access technology. It will give data rates spanning from 144kb/s to 2Mb/s making it possible to use multimedia services such as video conferencing. Third generation systems integrate packet and circuit switched data transmission and aims for the convergence of wireless data and the Internet.

The Wireless Application Protocol (WAP**)** [18] is a standard protocol for wireless communication and its transition to the Internet. Apart from the protocol definition it also defines a unified architecture for application environments in handheld devices and includes a general interface and functionality for telephony services. WAP is the strongest candidate for the future convergence between the wireless data and the Internet.

The combination of the third-generation mobile system UMTS and WAP is a good platform for creation of new services that could take advantage of both Internet and telephony functionalities and adapt them for wireless users. These services could work independent of network, access or terminal.

1.2 Background

In the beginning of mobile telephony the only end-user service provided were telephony itself. As time past new services such as telephony services like call hold, call waiting and multiparty were added along with other practical services such as SMS, voicemail and positioning. The number of services increased, but the network operator controlled all the network functionalities behind the services.

Today the network operator has begun to give others access to some of the network functionalities, with the belief that it would accelerate the development of services based on these and thereby increase the use of them.

Because of this and WAP it is now possible to create a variety of such services which can be used from a mobile device on a standardized way. But what techniques are there to use when implementing these services and what services can be created?

1.3 Purpose

One of the purposes with this report is to give the reader a technical view within the area of WAP and UMTS. Because these two techniques are very large areas we have tried to summarize the most important parts of them.
Our main area has been the service part within WAP and telephony services. The purpose has been to show different techniques that makes it possible to implement the services in a distributed, relatively simple and secure way. The techniques also make the implementation platform and program language independent.

The techniques we have mentioned are developed by cooperation between numbers of different companies and are deemed to be strong candidates to a future standardization. We have implemented a demo, demonstrating a possible service, implemented by one of the techniques. There is also an ambition to show a possible evolution of future services.

1.4 Scope

Our description of UMTS is very abstract and does not discuss how the technical details such as radio techniques works, but rather describe the high level changes it will result in and some concepts it may introduce. Concerning WAP, we discuss the purpose, give an overview of its architecture and make a brief presentation of two related languages, WML and WMLScript.

We explain the application server concept by giving only an introduction to its functionality. Some general features that are common for most application servers are described. Three different application servers are mentioned and we examine if they support the techniques we use, without going into how it is done.

Concerning the implementation techniques we have chosen to use CORBA [7] as an object broker and we are using Parlay [10] as a network API. Other techniques that could be used are for example DCOM or RMI for object handling, and for network API one could for example use Jain or IEEE P1520. In the report we do not discuss any of these other techniques or compare them with the ones we use. When describing the CORBA and Parlay implementation we do not discuss the pre-created files, only the implementation we have made ourselves.

We have chosen to describe Internet services and WAP services on the user level, no underlying technical details are described. However concerning WTA [19] services, the concept, how it works and its client architecture are described, but no more detailed technicalities are brought up.

There is also a chapter about service evolution, where possible future services are brought up, but there is no description on how to implement them.

1.5 Overview

The report mainly contains five parts, a technical overview describing the two telecommunication standards UMTS and WAP, an overview of the involved implementation techniques, a review of the different service environments, a description of the implementation done in the demo and one part where we discuss future possibilities.

In the technical overview UMTS is introduced and describes the radio technique UTRA and two concepts that are used in UMTS. An introduction to WAP is also included in this chapter describing what it is, its architecture and two related languages, WML and WMLScript.

The chapter describing implementation techniques introduces the reader to the concept of application servers and compares the features of 3 different application servers that can be used. The object broker CORBA and the Parlay network API is described in this chapter, were we explain how they work and what benefits they provides. We do not go further into implementations in these parts.

In the service section we first provide a very light overview of Internet services. The WAP service section is only described on the user level and mainly compares the WAP services with the Internet services. In the WTA section we explain what it is and show how the steps in the service procedure. For each of the parts in this chapter a number of example services are listed.

The service implementation chapter describes a demonstration that we have made which illustrates a possible voice mail and WTA service. First the chapter contains a figure and a description of the step sequence done in the demo. Next a brief explanation of the code is provided, dealing with a couple of rows at the time.

In chapter 6 we discuss future possibilities with WAP services in UMTS, from both a technical and a user point of view.

2 Technical overview

2.1 Brief description

This report focuses on WAP, WTA and its communication towards the mobile devices and the network operator. A number of components are involved in providing this and it can be difficult to see the context. Therefore we have made a figure of the scenario with a brief description, in order to give a better understanding of how the different parts are related to each other, although the parts themselves have not been described.

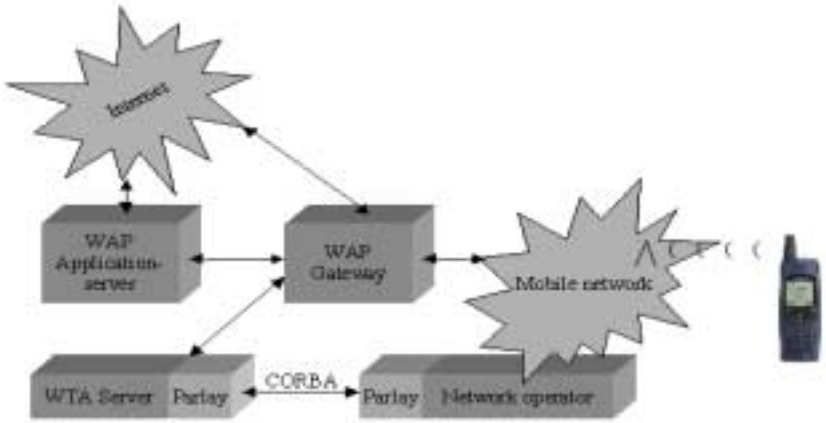

Figure 2-1: Overview WAP/WTA using Parlay and CORBA

As shown in Figure 2-1 the WAP application server and the WTA server communicate with the mobile network through a gateway. The WTA server also connects to the network operator to access the available services. This communication between the WTA server and the Network operator is in our case handled by CORBA. How this works is discussed in chapter 3.2. Parlay is used as an interface to make sure that the access is made in a safe and secure manner. Parlay is described in chapter 3.3.

UMTS as we talk about in next section covers the mobile network part in the figure and to a certain degree also some parts that are connected to it.

2.2 UMTS

2.2.1 General

Universal Mobile Telecommunications System (UMTS) is one of the major third-generation (3G) mobile systems. It has been developed within the framework of IMT-2000 [4] defined by the International Telecommunication Union. UMTS builds on and extends today's mobile, cordless and satellite [15] technologies by obtaining increased data capability and much more services by using innovative radio access scheme and an enhanced, evolving core network. UMTS is being standardized by the European Telecommunication Standard Institute (ETSI). The purpose is to produce a detailed standard to achieve global roaming and service availability. UMTS will most probably be in commercial use in Japan 2001 [17] (Europe by the beginning of year 2002). UMTS have a potential to support 2Mbit/s data rate and Internet Protocol (IP), these two qualities makes a powerful combination when it comes to deliver interactive multimedia services as well as video telephony and other wideband applications.

Most of the 2G systems only support circuit switched technology for data transmission, but UMTS integrates both packet and circuit switched data.

This gives several benefits [15]:

- Virtual connectivity to the network all time, this means that the user feels like being connected all the time.
- Asymmetric bandwidth in the uplink and downlink. This is a wanted ability by data services, that may in one direction carry only simple commands and in the other carries content rich, bandwidth intensive traffic (for example web browsing and video transmission).
- Alternative ways of billing. Pay by month, pay per bit, per session etc.

The UMTS service concept is built upon standardized services, which are common to all UMTS user and radio environments. This have the effect that a personal user will have the same set of services, even when roaming from ones personal network to another network operator. This is called a Virtual Home Environment (VHE), meaning that the personal user will always feel like being in ones home network. VHE will assure that the same services are available independent of the user location. The VHE will enable terminals to provide this by negotiating with visited networks, and if some services may be missing, the terminals will download the software. In the end the goal is to keep all the connection, signaling and

registration, among other things invisible for the end user so that they will experience a simple user-friendly environment.

2.2.2 Technologies of UMTS

UTRA

UMTS is built on two radio techniques that were assembled into one standard: the UMTS Terrestrial Radio Access (UTRA), this was done by ETSI. The two technologies are W-CDMA for paired spectrum bands and TD-CDMA unpaired bands. The combination provides an optimal solution for the different operation environment and service needs.

Transmission rates of UTRA:

- At least 144 Kbit/s for full mobility in all application.
- 348 Kbit/s for limed mobility in micro and macro environment
- 2.048 Mbit/s for low mobility applications in micro and pico environment. The rate may also be available for packet applications in the macro cellular environment.

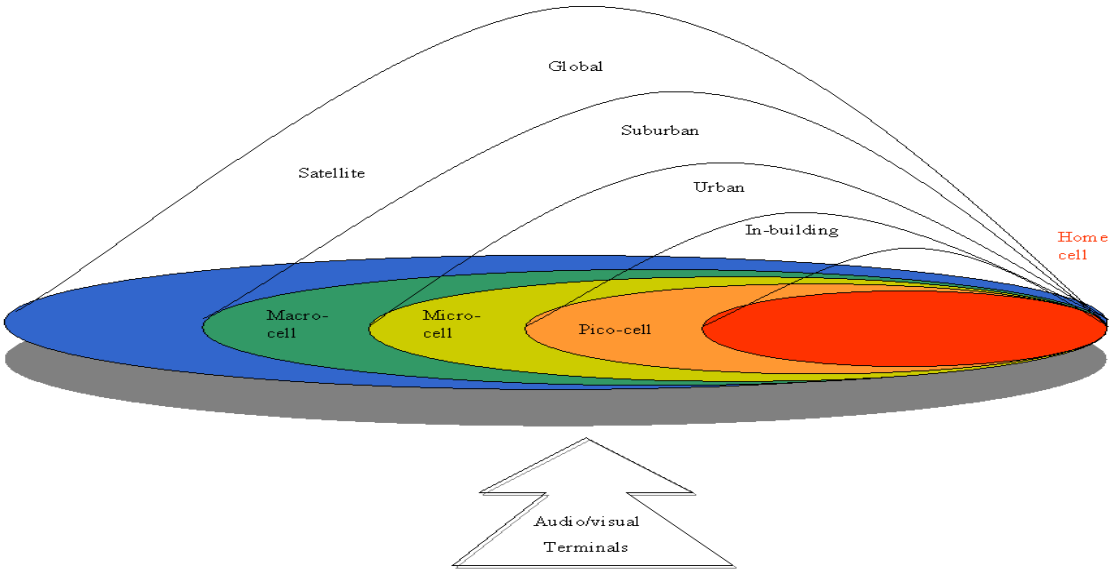

Figure 2-2: Inter-Network roaming

SIM cards

With GSM came the Subscriber Identity Module (SIM), also known as smart card. The original cards provided the user with the necessary authentication to access the network and storing the GSM encryption algorithms that ensured speech security. By 2002, when UMTS will be up and running the SIM cards will probably be provided with greater memory, contact-less operation [15], faster CPU performance and better encryption. These advances will help the UMTS Subscriber Identity Module (USIM) to provide high security data storage and transmission for the users. The SIM Toolkit, which is a standard set of program tools stored on the chip within the SIM card will extend the role of the SIM card, making it a key interface between the mobile terminal and the network. Using the SIM Toolkit, the SIM card can be programmed to carry out new functions. These include the ability to manipulate the menu structure of the mobile terminal to provide new, tailored options – for instance the handset could provide a menu for personal use and a menu for business use. Either way, the phone becomes personalised to the individual and therefore user-friendly. These tools, combined with an application dependent code, can be set to run remote applications downloaded by the operator and accessed via the phone.

API

The UMTS Application Programming Interface (API) [15] is something that UMTS forum will try to get as a standard for the UMTS networks. The UMTS API is still under development.

The API will be an abstraction of both the terminals and the network, making a general way for the application to interface the terminals and the networks.

The UMTS API will support:

- Security
- Billing
- SIM management
- Service management
- Call management

The API is planned to build upon and extend such technologies as:

- Java
- WAP
- GSM
- Internet technologies

All these above-mentioned properties seem to make UMTS well prepared for future demands and needs.

2.3 WAP

2.3.1 Introduction

The Wireless Application Protocol (WAP) is a standard for content delivery and telephony services on wireless devices. The specifications were drafted by the WAP Forum which was founded by Motorola, Ericsson, Nokia and Phone.com (Formerly Unwired Planet) in June - 97. Since then, more than four hundred new companies have joined.

The reason for the creation of WAP was the ambition to let mobile devices get access to Internet services and to get a general specification for telephony services that would work across all wireless network technologies. The WAP solution has considered the technical differences between the wireless and the fixed networks. The wireless communication brings difficulties like less predictable availability, stability and higher latency that must be handled. Also to be considered were the limitations of the mobile terminals such as the limited CPU, small memory, the need to restrict power consumption due to short battery life and smaller displays of the wireless devices. Beside that the wireless networks also gives a lower bandwidth. This requires that the wireless protocol is more compact using less data and the data-packets had to be smaller.

Other important requirements were that the solution had to be:

- Interoperable - terminals from different manufacturers must be able to communicate with services in the mobile network.
- Scaleable - mobile network operators should be able to scale services to customer needs.
- Efficient - will provide quality of service suited to the behavior and characteristics of the mobile network.
- Reliable - will provide a consistent and predictable platform for deploying services.
- Secure - enabling services to be extended over potentially unprotected mobile networks while still preserving the integrity of user data. It must also protect the devices and services from security problems such as denial of service.

The specification was made to use existing standards and technologies wherever possible and aligning its technology closely with WWW and Internet technologies. It leverage and extends technologies such as digital data networks, IP, HTTP, URL’s, XML, SSL, Scripting etc. This makes it easy to adopt for users familiar to these technologies. For example the

WML which is used for content delivery in WAP is very similar to HTML and the WMLScript is based on JavaScript. Both WML and WMLScript are optimized to fit the wireless environment

WAP uses proxy technology to handle the communication between the wireless domain and the Internet. A WAP proxy contains the following functionality:

- Protocol Gateway translating requests from the WAP protocol stack (WSP, WTP, WTLS, and WDP) to the WWW protocol stack (HTTP and TCP/IP).
- Content Encoders and Decoders that translates WAP content into compact encoded formats to reduce the size of data over the network.

These conversions makes it possible for mobile terminals to reach contents and applications hosted on standard WWW servers developed using proven WWW technologies such as CGI scripting.

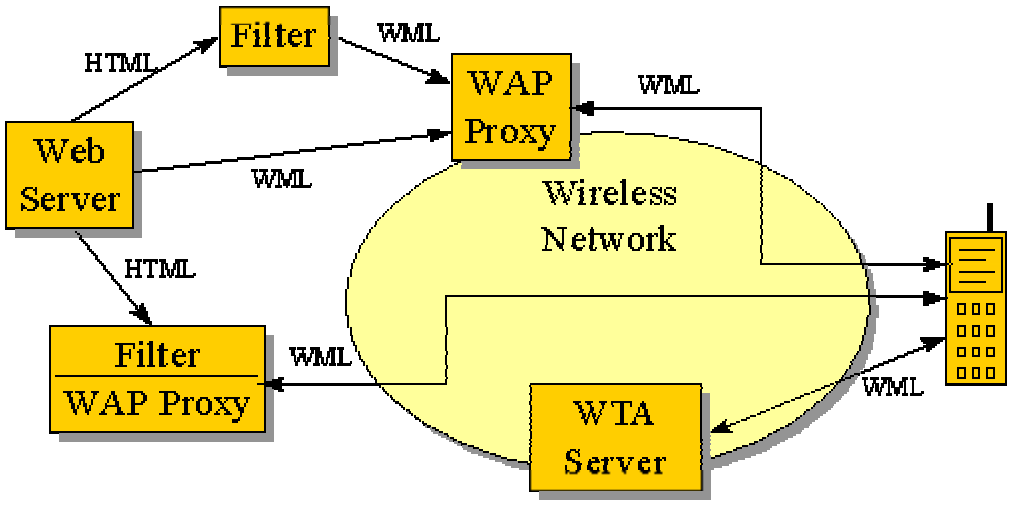

Figure 2-3: An example of a WAP network (Figure from WAP-forum).

The WAP client communicates with two kinds of servers in the wireless network.

- The WAP proxy – Translates WAP requests to WWW requests then forwards them to the webserver. The proxy also encodes the responses from the web server into the compact format understood by the WAP client. If the webserver does not provide WAP content (e.g. WML), the WAP proxy uses an HTML filter to translate HTML to WML.

- The WTA (Wireless Telephony Application) server - responds to requests from the WAP client directly. The WTA server is used to provide WAP access to features of the wireless network provider's telecommunications infrastructure. WTA allows access to telephony functionality such as call control, phone book access and messaging from within WMLScript applets. This makes it possible for operators to develop secure telephony applications integrated into WML/WMLScript services. For example, services such as Call Forwarding may provide a user interface that prompts the user to make a choice between accepting a call, forwarding it to another person or forwarding it to voicemail.

2.3.2 The layers in the WAP architecture

The WAP architecture like the Internet protocol is constructed with a layered design of the protocol stack (Figure 2-4).

Despite of the layer design, the layers can be reached by all layers above and is accessible by other applications.

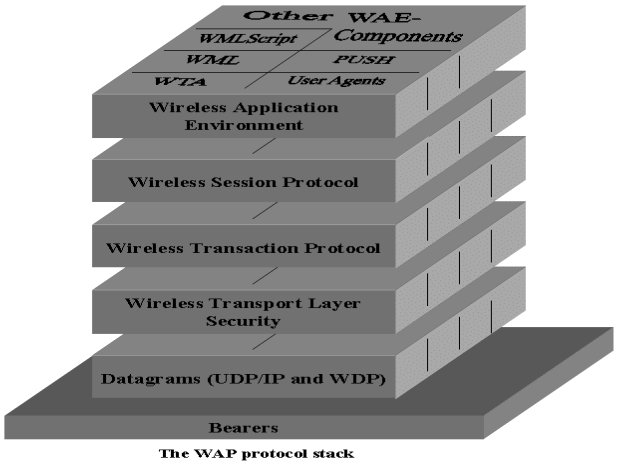

Figure 2-4: The WAP layers

- **The Wireless Datagram Protocol (WDP)** layer operates above the data capable bearer dealing with the actual transport of the content. This is the only layer that needs to be concerned itself about the type of underlying wireless network. It provides the upper layers with a common interface independent of network. Observe that carriers running mobile IP to a handset (CDPD, iDEN, or circuit-switched PPP etc.) have to use UDP (User Datagram Protocol) instead, which is not in the scope of WAP.

- **Wireless Transport Layer Security (WTLS)** is a security protocol based on the industry-standard Transport Layer Security (TLS). Providing data integrity, privacy, authentication and denial-of-service detection. WTLS is optimized for narrow-band communication.
- **Wireless Transaction Protocol (WTP)** controls the transaction (request/response) of information with the objective to balance the reliability against the cost of delivering. It runs on top of a datagram service and optionally through the security layer.
- **The Wireless Session Protocol (WSP)** provides the application layer of WAP with session handling. It is optimized for networks with long latency and supports session suspend and resume with session migration[3] to manage the use of mobile devices. It is also designed to allow a WAP proxy to connect a client using WSP to a standard HTTP server.
- **Wireless Application Environment (WAE)** offers an interoperable environment for creating applications and services that will be compatible with different wireless platforms. It is designed to consider the thin-client architecture with centralized management of the services at the origin server. The major elements of WAE are:
  - User Agents to interpret network content (WML, WMLScript, WTAI or other resources) referenced by a URL.
  - Content generators to create standard content formats to requests from User Agents in the mobile terminal, usually these are CGI scripts or similar techniques.
  - Standard Content Encoding to encode formats (WML, WMLScript, images etc) so that the user agent can utilize the content.
  - Wireless Telephony Applications (WTA) which is telephony specific feature control mechanisms that provide authors advanced mobile network services.

[3] *mobile terminal changing from one base station to another*

2.3.3 Bearers

The WAP protocols are designed to operate over a variety of different bearers, handling their differing levels of Quality of Service (QoS) with respect to throughput, error rate, and delays. WAP is designed to manage different bearer services including short message, circuit-switched data, and packet data. The WDP layer converges between the bearer service and the rest of the WAP stack. The WDP specification lists the bearers that are supported and the techniques used to allow WAP protocols to run over each bearer. The list of supported bearers will change over time with new bearers being added as the wireless market evolves.

2.3.4 WML

WML (Wireless Markup Language) is used to create documents for limited capability devices. WML is specified as an XML document type. The WML pages are fetched to the terminal as a 'deck', which consists of several cards. Individual cards are executed to fill the screen, simular to an HTML page. Cards can refer to each other via 'anchors' in the pages. These anchors simply consist of the deck URL followed by the card index number called the 'fragment'. The first card usually executes automatically when a deck arrives to the terminal.

2.3.5 WMLScript

WMLScript is a lightweight procedural scripting language. It supplies the WML pages with more interactive capabilities and helps it to utilise the device and its peripherals. It is based on ECMAScript, which is an ancestor to JavaScript, making it easy for developers to adopt.

2.3.6 Push

In general the user requests contents that will be initiated by the terminal user agent this action is called pull. An example of pull is when the user request Internet content. But there are mechanisms in WTA that allows content to be pushed in to the terminal by the Push Proxy Gateway (PPG) without asking the user first. This functionality is called push and can be useful in several occasions. The feature of course demands some security. The client and the PPG can establish trust, and the PPG may be listed in the client's list of trusted proxies. Only the listed PPGs are allowed to push to that client. A trusted PPG then handles the entire authentication and security part towards the push initiator. The push utility can be used for

example for multicasting or broadcasting information over the mobile radio network. Today the push initiator in most cases is the network operator.

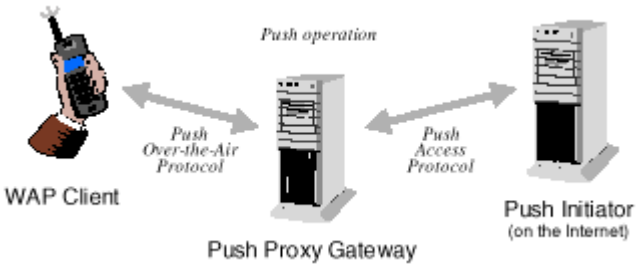

Figure 2-5: Push framework and protocols (Figure from WAP forum).

3 Technical overview, implementation techniques

3.1 Application Server Concept

3.1.1 Background

The application servers of today are most probably a result of two independent lines of development. One of the lines was the development of network development tools. The first development tools were built in c, c++ or the program that was available, these tools were developed as they were needed.

The other line was a natural evolution from the client/server paradigm, called the n-tier paradigm. The three-tier application builds upon a back-end system, a user interface, and a middle-tier consisting of a server executing the business logic. At the beginning the middle-tier was written and rewritten depending on the applications needs.

What we today call an application server have probably arisen from either of these two separate lines.

3.1.2 What is an application server

An application server is a server program, running in a computer in a distributed network that provides the logic for an application program. For example an applications server could be used running advanced programs and services for a thin client. The applications server is known as the 'middleware', that is because the application server is situated in the middle of a three-tier application. It is placed between the client and the database or legacy server (Figure 3-1)

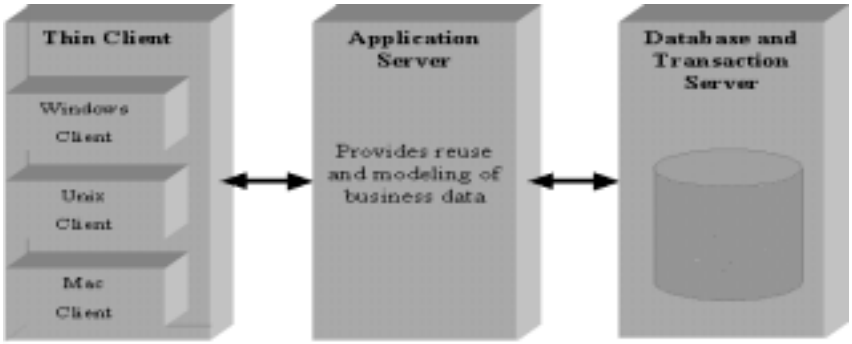

Figure 3-1: Three-tier model

Each part in the three-tier model could be developed independent of each other, languages, and development groups. This gives the ability to continually involve new applications as new needs and opportunities arise. The three-tier application is consistent with the ideas of distributed object-oriented programming (for information, see section 3.2.1). One of the benefits with an application server is that it provides the reuse and modeling of business logic, this means that the clients do not need to provide the logic (thin clients).

In the traditional two-tier model (Figure 3-2), the client had to contain all the logic (fat clients) that is included in the application server. One also loses the modularity that the three-tier model gives.

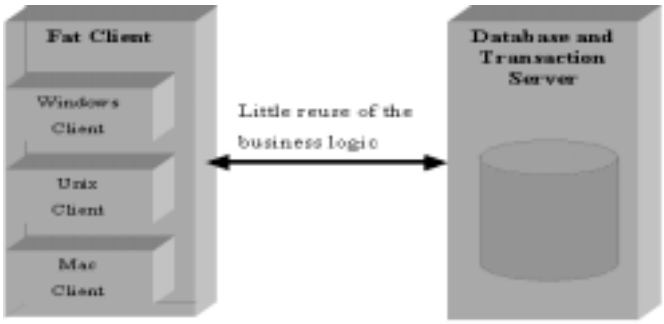

Figure 3-2: Two-tier model

Recently there have been a rapidly growth of using applications servers in web applications, thanks to all the possibilities the application servers offer. One of the big benefits of using an application server in web services is that it provides the possibility for clients to share the same source and maintain stateful connections. A stateful connection is a connection where the server and client keep a stable open connection for several transactions. This differs from the two-tier model, where the client for every new transaction has to establish a new connection.

3.1.3 Application servers

3.1.3.1 General features

There are over 30 different manufacturers of application servers, but most of the application servers have some things in common. These things are:

- Tools for development, management, deployment and monitoring.
- JSP/Servlet engine.
- Database engine/driver
- Application frameworks

3.1.3.2 Three different application servers

In the following selection of application servers we have only examined if they support the implementation techniques we use. For further information on application servers see Bachelor's project Evaluation of Web Application servers [12].

IBM WebSphere Application Server 2.03.

WebSphere [3] is a product from IBM. It is a Java oriented server, available in three different versions. The first one, which is the standard version is a simpler version made for building personalized and dynamic web content and is simply a servlet engine. The advanced edition and enterprise edition adds EJB and CORBA support.

BEA WebLogic server 4.5.1

BEAWebLogic [1] server is an enterprise level application server with advanced Java support. The server is a product from BEA Systems inc. It supports both EJB and CORBA.

Sybase Enterprise Application Server 3.0

Sybase Enterprise Application Server 3.0 [14] is an industrial strength application server from Sybase Inc. The server supports both EJB and CORBA.

3.2 CORBA

3.2.1 What is CORBA

A big problem with the current applications is the need to integrate a great number of work elements, so that the enterprise can use existing hardware and software to solve different business problems. For example an information company that works with changing information from many different sources would have a great benefit of using a common way to handle the information. One solution for this problem may be Common Object Request Broker Architecture (CORBA). CORBA is a specification developed by the Object Management Group (OMG) and describes an object-oriented architecture used in the development of applications. The first description of CORBA was published in a document called Object Management Architecture Guide in 1990, by the OMG group. After the first edition there have been two new versions, adding implementation details and more specification information.

Integration of applications in CORBA is based on the object-oriented model. The object-oriented model provides techniques for building software that is extensible, reusable, and easier to develop and maintain. The reuse of software is an accelerating trend in software engineering. CORBA makes this possible by distributed object computing, which combines distributed computing[1] with object oriented computing. CORBA relies on a protocol called the Internet Inter-Orb Protocol [8] (IIOP) and supports both synchronous and asynchronous communication styles. Synchronous communication means for example that a client sends a request to a server and waits for the server to reply, with asynchronous communication the client continue with the tasks without waiting for the answer.

[1] Distributed computing is simply two or more pieces of software that are sharing information with each other.

3.2.2 The Broker

In traditional server/client applications, a server and a client have fixed relationships. The client always asks for a task to be performed and the server performs the task.

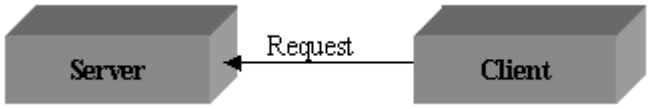

Figure 3-3: Traditional server/client application

CORBA adds a new layer between the server and client, called broker. The broker will provide intelligence between the client and server, mapping certain service request to a certain server implementation. The client and server do not need to have much knowledge of each other, instead they will find their way through the broker. The broker also makes it possible for multiple servers to work with one client or multiple clients to work with multiple servers.

A client application can locate and interact with new objects and servers during runtime.

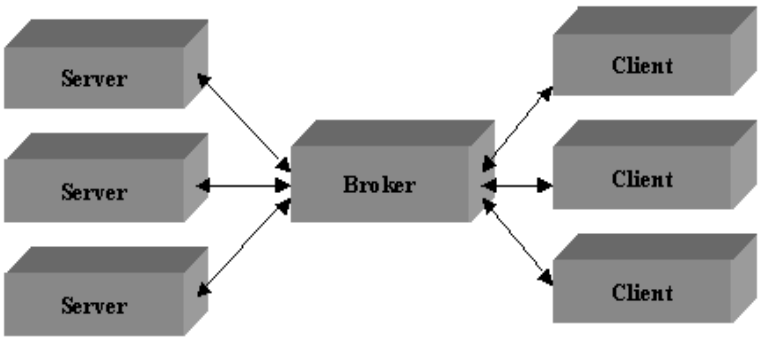

Figure 3-4: CORBA server/client application

3.2.3 OMG Interface Definition Language

Server and client need to know some basic information about each other, such as information about available operations in the server that the client can request and the argument necessary to perform the operation.

CORBA includes this information in interfaces. A CORBA interface defines the characteristics and behavior of an object, including operations that the client can perform on this object. Definition of an interface is made in OMGs Interface Definition Language (IDL). Before writing a CORBA client/server application, you must create or access an IDL file containing the definition of the interfaces the client or server implementation will support. OMG IDL is a definition language, not a programming language. One uses OMG IDL to define interfaces and datastructures, but not to write algorithms. OMG IDL is used to generate source code for the desired programming language. For example in our implementation (section 5.2) we have compiled the IDL definition into Java code. The compiler generates clients-side stubs and server-side skeleton.

The client stub provides definitions and other CORBA vendor-specific information. The client stub contains routines that a client application calls to invoke a request. The generated client stub should be treated as a black box that one compiles and links in to the application.

A server skeleton is the entry point into the distributed object. It provides any code necessary to dispatch a request to the appropriate method. The developer only has to implement a Java class that extends the generated skeleton and do not have to be concerned about the inside of it. In our implementation (section 5.2.5) we have extended a skeleton named MailImplBase in code section Mail_impl.java.

The use of a interface definition language frees CORBA from being tied up to a specific programming language.

3.2.4 Object adapters

3.2.4.1 General

An object adapter is a framework for implementing and managing CORBA objects on the server. It provides an API that the implementation use for different low level services. The object adapter is responsible for:

- Providing a 'binding' between an object's interface and a server's implementation in the object.
- Method invocation

- Security of interactions
- Registration of implementations

3.2.4.2 Basic Object Adapter (BOA)

The Basic Object Adapter (BOA) is the most commonly used object adapter. One of the main things for the BOA is to support on-demand object activation. When a client wants to make a request, the BOA checks if the object is currently running. If this is the case, it sends the request to the object. If the object is not running the BOA activates the object before it sends the request.

3.2.5 Architectural overview

The OMG Object Management Architecture Guide has defined an object-oriented architecture for applications called the Object Management Architecture (OMA). The OMA is the basis of CORBA.

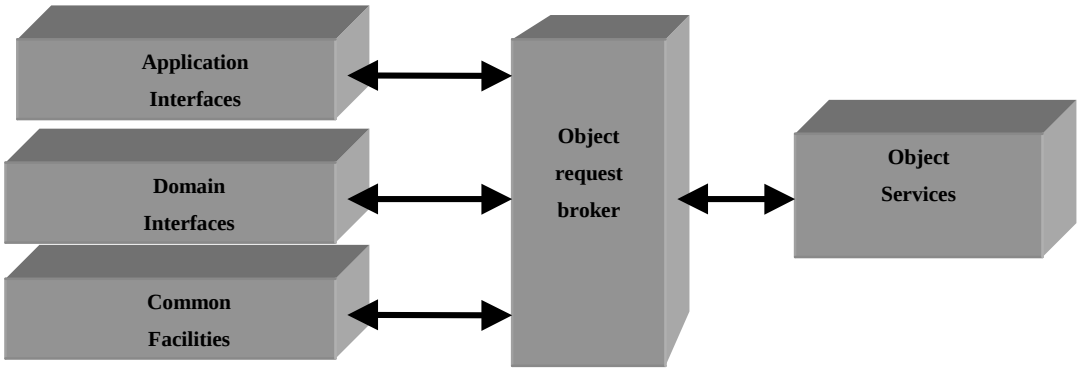

Figure 3-5: Object Management Architecture

- Object Request Broker - Central in the OMA and CORBA has inherited its name from the ORB. The ORB is the communication hub for all objects, it provides a mechanism for transparently sending a clients request to the target object implementation. One of the key roles for the ORB is to separate the server from the client, so that the server and the client do not need to contain information about each other. The ORB itself is defined by its interfaces.
- Object Services - Provides a set of standard functions to create objects, control access to objects, track objects, etc. This allows developers to call object service functions instead of writing and calling their own private service functions. Examples of object

services are naming service (find objects based on name), object lifecycle management (for controlling of an objects creation), trading service (find objects based on their properties).

- Application Services - The interfaces situated here are developed for a specific application and not for OMG standard interfaces. For example a vendor could design an interface to handle a specific WTA service.
- Domain interfaces - These interfaces provides functionality of direct interest to end user in a particular domain. Domain interfaces are designed to do specific tasks for users within a certain market, for example Product Data Management (PDM) or Telecommunications.
- Common facilitates - Provides a set of general-purpose applications for use in a wide variation of end user applications, such as compound document management, accessing databases and time synchronization in a distributed environment.

3.2.6 What benefits does CORBA offer

CORBA supports multiple language mappings for OMG IDL, so that different components in a system/application can be implemented in different languages. All interactions between the programs are through interfaces, which are specified independently of the program language they are implemented in. This makes it possible to implement the application in the programming language that is most suitable for implementing the application. CORBA can run on any platform as long as there is an ORB installed. It has been proven to be high performing [6], which makes it suitable for heavy loaded servers.

CORBA also provides location transparency, which means that an object can be located independent of its actual physical location. One could physically change an object location without damaging the application. The IDL compiler and ORB runtime systems frees the programmer from programming a number of low level and repetitious efforts, such as opening/closing network connections, setting up servers for listening on sockets for incoming data and forwarding it to the implemented objects.

3.3 The Parlay API

3.3.1 Introduction to Network API

A network API can make it possible for network customers to access network capabilities that are currently controlled by network operators. Such functions are:

- Network Operator Applications, e.g. personal numbering, voice messaging etc.
- Service Components, e.g. billing, routing, authentication, configuration etc.
- Networks, e.g. ISDN, PSTN, IP etc.

When these functions are exclusively used by the network operator it usually results in incompatibilities between applications made for different networks, or slow development of compatible applications, caused by different security and integrity standards.

An API can make it possible for independent developers to access network capabilities and still retain the integrity. It helps the network operators to maximise the value of their network technology by directly passing on that functionality to third parties in a safe and controlled manner. This is made by encapsulating the network capabilities that are useful for enterprise applications in order to make them visible in a way that ensures the security of the network.

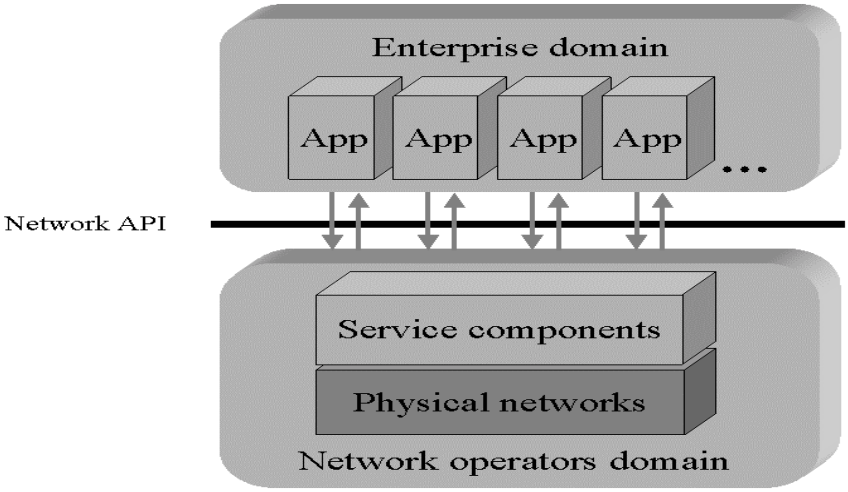

Figure 3-6: Network API

There will be an opportunity for more developers to create services without concerning about the network technologies, because the API allows the network functionality to be

implemented in a normal software environment. This concept enables third part developers to create services that aim towards small user groups.

3.3.2 The Parlay API background

The project of developing the Parlay API started in March 1998 with the goal to give enterprises access to network information and allow them to control a range of network capabilities.

The specification of the first phase was completed in November 1998 primary handling addressed call control, messaging and security.

The Parlay Group originally was consisting of BT, Microsoft, Nortel Networks, Siemens and Ulticom (Formerly DGM&S Telecom). In the early 1999 the Parlay group expanded with six new members: AT&T, Cegetel, Cisco, Ericsson, IBM and Lucent. Phase two started in June 1999 with focus on expanding the API functionality with concentration on wireless communication, IP and the convergence between these and public switched telephone networks. As an example giving one a touch of the future, the Parlay API in phase two will support creation of a service that integrates location information obtained from a wireless network or Global Positioning System and the Internet [11]. Using information about the location of a customer's wireless handset, an application could for example search the Internet to find the nearest theater ticket office and connect the customer to make a reservation.

Some of the important characteristics the Parlay group has aimed for the API to be are:

- Object oriented
- Multi-media supportive
- Manageable
- Secure (The issue considered most important by the Parlay Group)
- Simple
- Extensible
- Network independent
- Supports a wide range of service capabilities

3.3.3 The Parlay API technology

The Parlay API architecture consists of two categories of interfaces:

- Service Interface Set - Common functions that deliver whole complex services or sub-components of services (micro services).
- Framework Interface Set - Common functions that are required to enable services to work together in a coherent way and will make the service interface secure, resilient, locatable and manageable.

For example the framework in Parlay handles the security and the service discovery when one first access the Parlay API. This procedure is shown in Figure 3-7

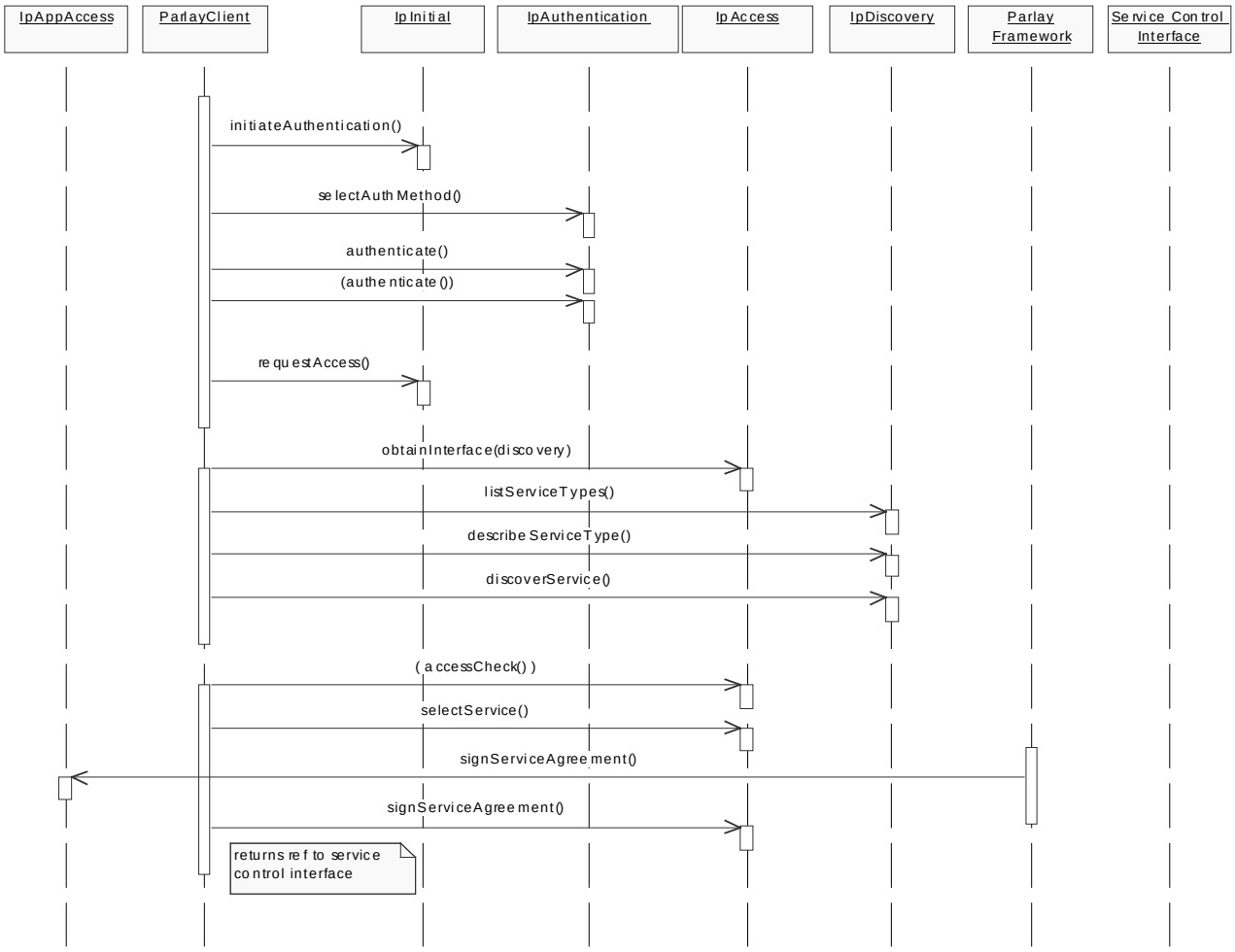

Figure 3-7: Procedure when accessing the Parlay API

In Parlay 2.0 they have divided the specification in to six parts. These parts are:

- Framework
- Common data
- Connectivity manager
- Call processing
- Messaging
- Mobility

The framework is described above. Common data contains all the common data types that are used in the Parlay API. The last four parts includes the service components. The connectivity manager handles the connection functions, such as finding site locations, receiving information about certain connections etc. Call processing includes the telephony functionalities such as divert calls, demanding pin code when dialing a certain number etc. Messaging handles functions concerning for example mailboxes. The mobility part manages for example the retrieval of terminal position, what kind of terminal that is used, if it moves etc.

One must keep in mind that Parlay has not implemented these functions. The Parlay API only makes it possible for network operators to give others access to such functionalities in the network.

The Parlay API is supposed to be technology independent and hence the main specification is defined in the Uniformed Modeling Language (UML). However, while keeping the network API technology independent, implementation issues that could affect the realization of the API is to be considered.

There is no specification for the interface towards the network-based resources. The underlying network software, physical components or protocols are also not in the scope of the API. It is up to the industry to design and implement the API. The Parlay group has however to date (Jan-2000) generated, from the main UML specification, technology dependent IDL specifications in Microsoft IDL and CORBA IDL.

4 Service concept

4.1 Internet based services

4.1.1 Background of Internet

From the beginning (by the end of the 60's) the Internet was a smaller network called ARPANET founded by American Research Project Agency (ARPA) of the U.S Department of Defense. ARPANET connected some American universities, military and other governmental research institution that required a high quality information source and needed to share information with each other. In the beginning of the 70's the agency started a project which purpose was to study different ways to send data packages in the network, the project were called the Internetting project. With the introduction of the World Wide Web by a research team in CERN in the early 90's, today's Internet became a fact.

4.1.2 Services

Companies saw the big market that Internet introduced and started to advertise their products on the Internet. Many companies also started to develop new Internet based services. The Internet based services are growing rapidly, and today there is a wide range of available services.

A few examples of the services are:

End user communication.

- E-mail
- Chat
- Video conferencing

E business

- Bank business
- Trading products with (more or less) secure payment over Internet.

Distribution of content

- Personal homepages
- Distributed databases
- Distance education

4.2 WAP Services

4.2.1 Introduction

The arrival of a general standard that brings the growing possibilities of Internet to the mobile devices has created a new expanding market for services. But the boom hasn't occurred yet. The limitations of the small devices still restrain the progress. Another problem that complicates the development is the many changes that have been made in the standard recently and more changes will come before it will be in line with the market. Therefore the WAP functionality in today's devices probably will fall short within a year or so. Despite this there is a notable interest in the mobile terminals compatible with WAP.

The future scenario for WAP services heavily depends on the expected progress of technology, for example the evolution of semiconductors or better data compression for multimedia.

4.2.2 The WAP service user

One must keep in mind that there probably are a couple of things that differs the wireless user from the usual fixed Internet user. The users of a wireless device are in most cases on the move, needing to retrieve necessary information fast rather than to lean back and browse the Internet. The devices are in general supposed to reach ordinary people with no technical experience. This requires that the terminals and services are easy to use, demanding only a few clicks from the user to perform a task. Of course this makes it more difficult to create advanced services.

4.2.3 Services

In large the possibility exists to develop the same services to wireless users as to the Internet user. In the future the wireless user may in principle be able to achieve the same services and more if bandwidth- and device limitation problems can be solved.

WAP services extend the services invented for Internet with the possibility to combine it with features from the wireless network through the WTA (Wireless Telephony Application) interface. (More about WTA in section 4.3)

At this point only essential or very time limited tasks will be made unwired since the small market in the introduction of the concept makes the use of services quite expensive. In time one will pay for services according to data sent, not time spent which is used today. This will

of course make it cheaper to use data services where one does not use the connection all the time during a session.

Here follows a list of services which are, or soon will be, in use:

- E-Business, will achieve the same possibilities as the fixed Internet. One example that is being developed is the possibility to pay with your device directly in the shop like a credit card using SIM cards (see section 2.2.2).
- Retrieve information, from Internet through purpose made WML pages, or special made applications pushing or receiving broadcasted information of your choice into your device.
- End user communications that are in line with Internet such as Email, chat, video telephony etc.

With access to network functionalities through network APIs one have the possibility to create applications using specific sets of telephony functions, positioning, billing (sorry but features costs) new levels of authentication etc. These can be combined in many ways to create useful services, as described in the next section.

4.3 WTA

4.3.1 Introduction to WTA

WTA services are implemented in the WAP application layer (for further information about the application layer, see in section 2.3.2). WTA is a framework for telephony services. One could say that WTA is a telephony extension for WAP services that provides a range of different phone services in a mobile network, merging the data networks with the service voice network. WTA framework could for instance allow real-time processing of information that are of importance for the end user while browsing.

A WTA service could appear in the WTA user-agent in different forms such as WML and WMLScript. The asked service (either temporary stored in client's repository or retrieved from the WTA server) will be executed by the WTA user-agent. The Wireless Telephony Application Interface (WTAI) is an interface from WML and WMLScript to a specific set of local, telephony related functions in the client (Figure 4-1).

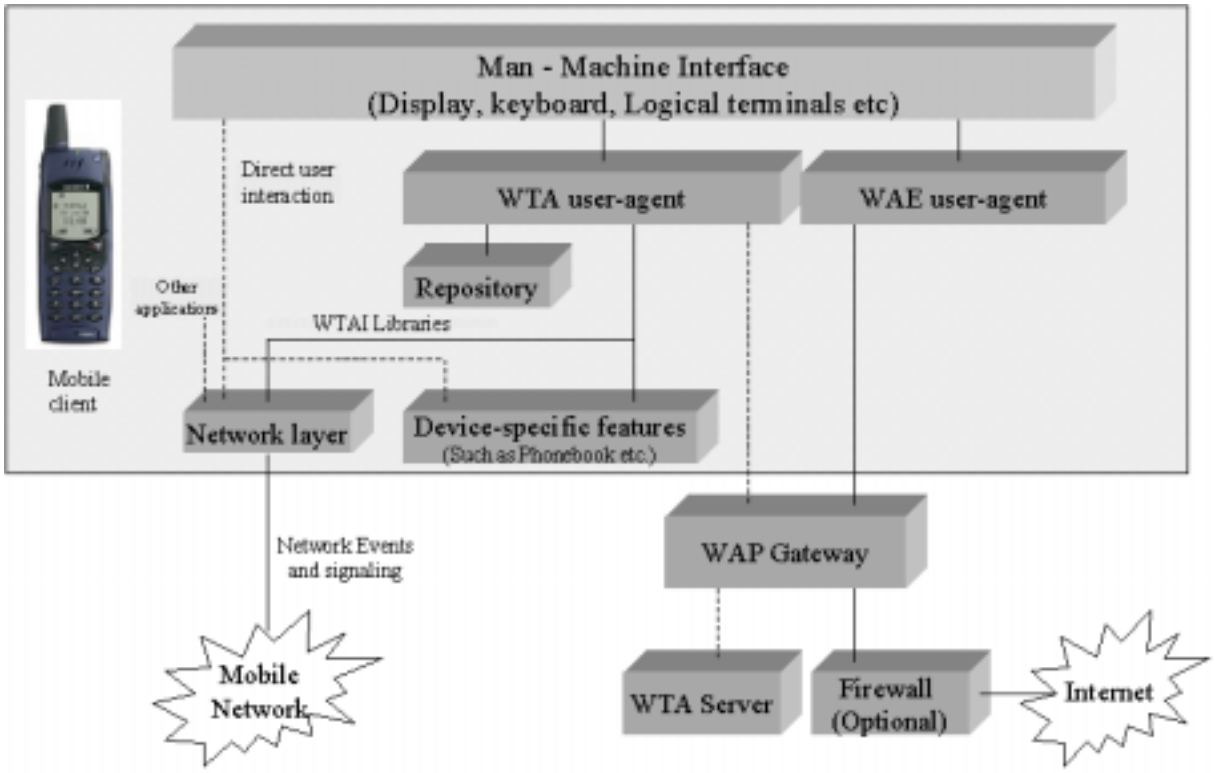

Figure 4-1: WTA client architecture

- Man-Machine Interface - Interacts with the user (display, keyboard, etc)

- WTA user-agent - Retrieves data from repository, execute contents, WTAI enables the WTA user-agent to interact with mobile network functions (setting up a call) and device specific features (searching a number in the phonebook).
- WAE user-agent - Retrieves its contents through the WAP gateway and accesses the WTAI Public library functions. Executes WAP contents.
- Repository - A persistent storage in the mobile client, accessible via the WTA user-agent.
- Networks layer - Handles such things as call control, etc.
- WTA server - It's similar to a web server that is delivering the contents requested by the client.

4.3.2 WTA Services

WTA services could be initiated in WTA user-agent in several different ways. The figure below illustrates the different scenarios.

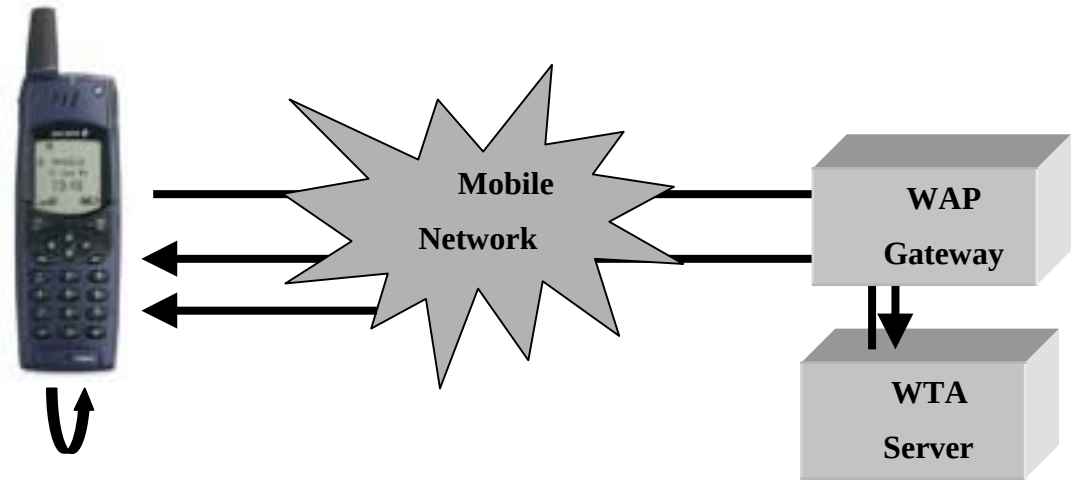

Figure 4-2: Initiating WTA services

Number	Explanation
1.1	Access to a URL (Via the WTA server)
1.2	Access to a URL (Via the repository)
2	Service indication (Push)
3	Network event (Transformed to WTA event in client)

Table 4-1: Different cases of initiating a WTA service

WTA services is an area that is constantly growing, there are many possible services that have not yet been developed. WTA services are using WML and WMLScript. Both WML and WMLScripts access their telephony functions through the Wireless Telephony Applications Interface (WTAI). WTAI provides access to telephony functions from WML using URIs (Uniformed Resource Identifier). URI creates a standard naming model to identify features independent of the internal structure on the client and the network. The WTA framework makes it possible to do quick updates in the client, without interacting with the user. It also allows personalizing of services for a single person or a group. The WTA services could be common network services, meaning that they are independent of the network they are running on, or else the services could be network specific services, meaning that they are unique for that specific network.

WTA services can be client centric or server centric based on where it is most convenient to manage the service. Which architecture to be chosen depends on if it is more handy to locate most of the service to the mobile client or if it needs to be located on the WTA server because of the need for more space or the need to be accessed at all time. There is a wide range of possible WTA services. Below we have stated a few examples (the examples are mainly based on an EIN internal report [2]):

- Address Finder - A user searching for a specific address could get a road map that is based on the current position.
- Advertisement service - Companies could send special offer messages to a user that is in the neighborhood of the specific company.
- Auto schedule Divert - The terminal could automatically divert all calls to for example voice mail, based on notes in the schedule agent.
- Current Position - One could get the address displayed for the current position.
- Forward Call - A user could forward an incoming call to for an example a voice mail or the office.
- Friend Locator - Notice a user by an alert that a friend is in the same area.
- Incoming Call - The user sees the name or number of the person who is calling on the display.
- Local Number download - The terminal phone book is getting updated when roaming. It could be updated with local numbers for different services, such as airport, car rental, agencies, restaurant, taxi, tourist information, etc.

- Message Manager - The user gets a notification when a new message has arrived. A message could be e-mail, voice mail or a fax.
- Proximity locator - Locates people from a certain group in a certain area. For example could an operator in a taxi company locate the nearest taxi in a specific area.
- Top 10 numbers - Save the users most frequently used numbers in a quick selection list.
- User Finder - Allowing the user to broadcast its position to others.

5 Service implementation

5.1 Service demonstration overview

We have tried to illustrate a scenario where messages are recorded in a voice mail node situated in the telecom network. When a message is received, information about it is sent to the WTA server. The information is buffered in the server until a pre-chosen number of messages have been reached. This number is chosen by the user. When the amount of messages have been achieved, a WML page is created and pushed out to the client, showing the names (as links) of the persons who left a voice mail message. The user can browse the list of names and listen to the messages that are of interest.

In our example we have realized the scenario with a client (voice mail node) reading the messages from a text file one by one and then sends the information to the server. Instead of creating WML pages when the pre-chosen amount of messages have been reached, our demo creates HTML pages and sends them to a monitor instead of pushing them to the mobile terminal.

We have used ORBacus 3.0, which is a fully CORBA compliant Object Request Broker developed by Object Oriented Concept Inc [9]. The version of ORBacus we have been using is royalty-free and for non-commercial use. The ORB is used in the two computers, one representing the voice mail node and the other representing the WTA server. Both computers are connected through a network.

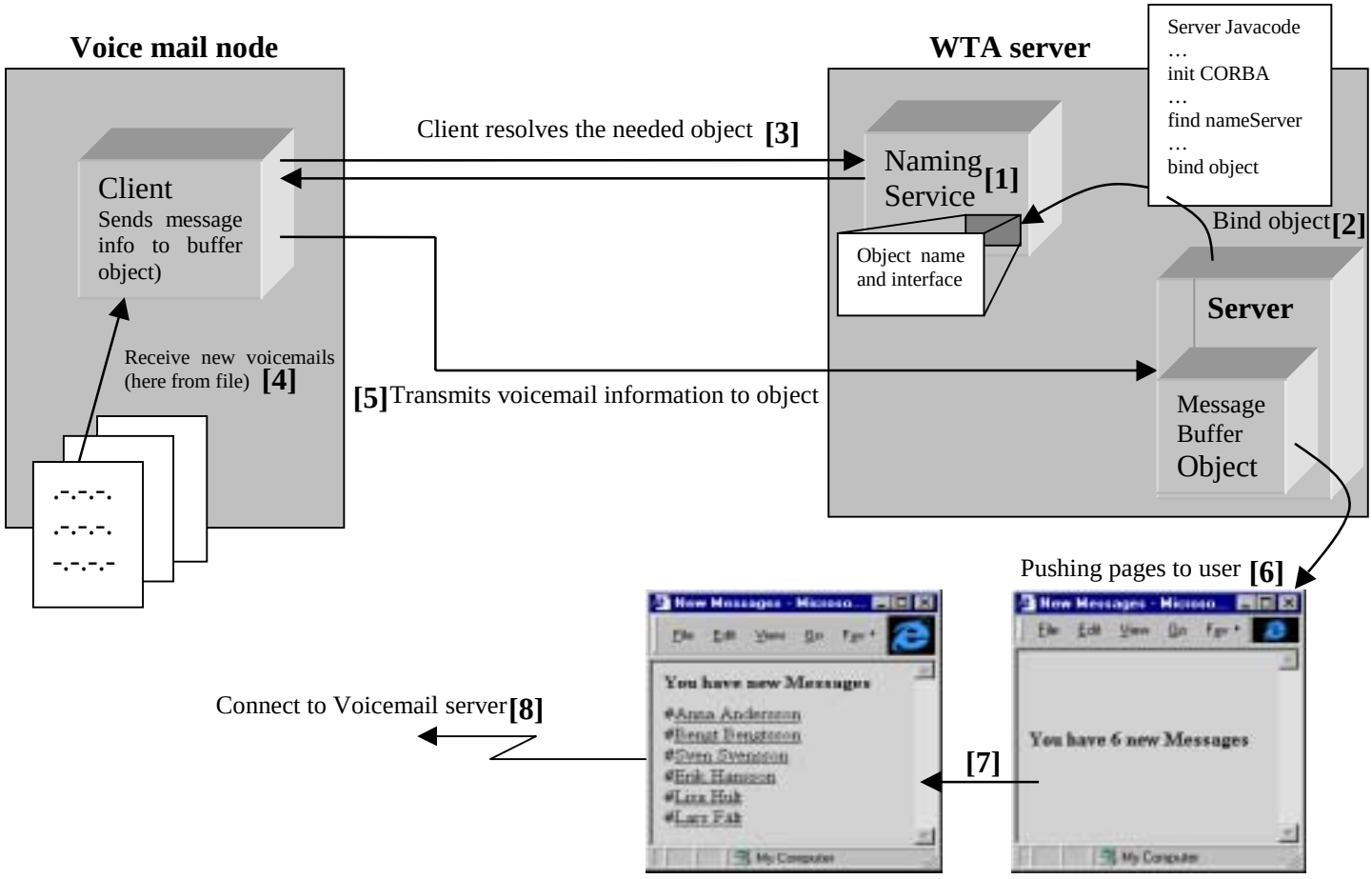

Figure 5-1: Demo of an application using CORBA

The procedure is as follows:

(1) The namingservice is started and bound to a specific port.

(2) A server program is started. This server connects to the namingservice and registers the objects it contains. The operation is called bind, where the server submits the names and interfaces of it's objects and what "context maps" they should be placed under, in this case there is only one object called mail. The objects are reachable only when the server is active.

(3) Then the client is started. It connects to the namingservice and resolves the needed object by submitting name, context and interface of the needed object. The server matches the query and replies.

(4) Now all the co-working parts are active and the client begins to receive the voicemails. In the demonstration the voicemail information is read from files and contains only the names of the persons who left the messages since we are not connected to a telecom network.

(5) The client transfers the data to the object through the appropriate interface. The object buffers the information until a specific number of messages have been received.

(6) When this quantity is reached the object push the information to the user device where a text appears informing the user that there are new messages. It is up to the user to decide the quantity of messages that will be buffered in the server object before it pushes the pages to the device. In the demonstration this is put in to practice by a client (not shown in figure) which takes the wanted buffer size as input from the keyboard and changes the size in the object through a purpose made interface. Further the demonstration only pushes HTML pages to the monitor through a browser instead of pushing WML decks to a mobile device.

(7) If the user chooses to check the messages a page appears containing the names of those who have left a message and the user can choose which of these messages he wants to listen to.

(8) When the user chooses a message to check he is connected to the voicemail server via a reference that follows with the message information. This is not happening in the demonstration since no connection to a real voicemail exists.

5.2 Implementation

We have made an example of a hypothetical WTA service, using CORBA as a distributor between the client and server, as shown in section 5.1. Here follows a description of the implemented files. These are only the files that we have created. The files that come from the IDL to Java compilation are not mentioned since they are just auto-created from our IDL file.

5.2.1 Mail.idl

This is the file that contains the IDL definitions, which will be compiled to Java skeletons and stubs.

3 An interface with the name Mail is defined, an interface in IDL is conceptually equivalent to an interface in Java or a class in C++.

5-7 Declares the reachable operations, the operations can be found in mail_impl.

5.2.2 Server.java

Here we have written a class that holds the server's main method. We have called this class 'Server', it can be found in appendix under Server.java. Among the first things the main program does is starting the BOA. This is done because the Objet Adapter that is on top of the BOA connects the CORBA server object implementations to the CORBA ORB. Next we contact and bind our server object to the naming service. When the object is bound to the naming service it becomes possible for the client to find and use it.

15-19 These properties are necessary to use ORBacus ORB with JDK 1.2.

25-26 Initiating the BOA and the ORB.

27 An instance of Mail_impl is created.

31-41 Tries to connect to the naming service, if there is no naming service available an InvalidName exception is thrown.

43-47 If the naming service object is null, there is something wrong with the naming service reference.

49-56 The object is checked and narrowed in order to verify that it is a naming service instance. If the narrow operation returns null reference, then the object returned is not a naming service instance.

62-66 Creates a new context named mail_context in the naming service.

68-75 Binds the object p that is an instance of mail_impl to the naming service under the earlier created context.

79 Confirm to the BOA that everything is ready and waiting for requests.

81-82 Before exiting the program we unbind all bindings.

84-135 Catching up different exceptions.

5.2.3 Client.java in voice mail node

This is a class that holds the client's main method. The client's main method is similar to the servers main. First we look up the NameService and obtain a CORBA object reference. Next the returned CORBA object is being narrowed down to naming context. We create a NameComponent and narrow it down to the server object by resolving the names with the naming context, received earlier. Last the text file that contains the messages is sent to the server. The class is called Client and can be found in the appendix under Client.java in voice mail node.

15-19 These properties are necessary to use ORBacus ORB with JDK 1.2.

24 Initializing the ORB, the BOA does not need to be initialized in the client.

28-38 Try to connect to the naming service, if InvalidName is thrown there is no naming service available.

40-44 If the naming service object is null, there is something wrong with the naming service reference.

46-53 The object is narrowed down to the naming context and checked in order to verify that it is a naming service instance. If the narrow operation returns null reference, then the object returned is not a naming service instance.

60-71 Creates a NameComponent consisting of a context and an object, the NameComponent is resolved and narrowed in order to verify that it is a naming service instance.

75-86 Opening the text file and reading line by line.

87 Calling the function get_mail in the server with the line in the text file.

89-134 Catching up different exceptions.

5.2.4 Client.java in server

This client is used in the server for changing maximum number of messages in the message buffer object. The structure is similar to the client in the voice mail node. The reference code can be found in the appendix under Client.java in server

19-23 These properties are necessary to use ORBacus ORB with JDK 1.2.
Initializing the ORB, the BOA does not need to be initialized in the client.

35-44 Try to connect to the naming service, if InvalidName is thrown there is no naming service available.

45-49 If the naming service object is null, there is something wrong with the naming service reference.

51-58 The object is narrowed down to the naming context and checked in order to verify that it is a naming service instance. If the narrow operation returns null reference, then the object returned is not a naming service instance.

61-77 Creates description of a NameComponent consisting of a context and an object name, it resolves with the naming service if this component exists and narrows it in order to verify that it is a naming service instance.

83-91 Gets user input (max number of messages) and calls msg. buffer object sending data to the object.

97-140 Catching up different exceptions.

5.2.5 Mail_impl.java

This is the file that contains the application specific operations. It receives the message information and buffers it until a chosen number is reached, this number is changed through the change max function. It then creates HTML pages, which presents the information on the screen. The CORBA Server object extends _MailImplBase class. _MailImplBase class is a skeleton generated by the CORBA IDL compiler. The Mail_impl class and the methods get_mail(String newMessage), makeHtml(), changeMax(short max) are declared public so

that they will can be accessed from outside the package. Mail_impl class includes all of the operations declared in our Mail.idl file.

13-21 The function get_mail(String newMessage) buffers each message until maximum numbers of messages are reached. When this happens the function makeHtml() is called.

89-134 MakeHtm( ), generates the html pages that are sent to the user, presenting information about the messages.

67-69 changeMax(short max), receives and changes maximum number of messages.

6 Service evolution

6.1 From a technical point of view

There is a large not yet explored market where many new services can be created and combined. The development of these services is depended on the existence of accepted open standards, for example standardized network APIs. This would also open a market for third party developers to create different services. Other things that will affect the service evolution are the technical advancements in radio technique and the progress in terminal technologies. For example battery life is expected to double over the next three years and processor power will continue to double every 18 months [5].

Early WTA services may be client centric using WML because it is easier to develop and it does not require complex integration with the network. In the future when the services get more advanced a larger part of the service may be placed on the server. If services are to be accessed at all time it is always more convenient to have them placed on the server, since one can never guarantee that the terminal is reachable.

As of the future it is assumed that low-cost backbone networks will support the UMTS infrastructure for distant communication [5]. Software downloadable applications and services will probably be used when mobile devices are capable of extended software management and standards for security are set. The expectations of UMTS are mainly centered towards the introduction of multimedia and real time interactivity, but the question is if the mobile network can handle that amount of data and guarantee real interactivity despite the latency and unpredictability of the mobile network. To handle this, development towards robustness in network to lower the error rate and invention of better data compression is still needed.

6.2 From a user point of view

The new mobile technologies introduce new possibilities, and with them there will be a built up need for the user to communicate, access information and manage personal business while on the move and thereby turn the dead time into product time. In the nearest future the usage of mobile services will be aimed at fast services, retrieving information fast, making

adjustments in personal settings, booking etc. This may be because of people's general view of the mobile device as equipment that is used only when necessary. Another cause may be the rather high cost that comes with the services, at least in the beginning when they are charged per time and not per data unit. Thereto the limitation of the terminals restricts the service possibilities. When these limitations are minimized and people get more familiar to the usage of the mobile terminals in the daily life, more advanced services can be utilized.

A possible scenario:
A businesswoman is going to meet a customer in another city and she has decided to travel by train. Arriving at the station, her mobile terminal beeps and she gets a popup message, which informs her that the train is on time and stands on platform 2b. On her way to the platform she downloads a map over the city on which she can localize the customer. Once she finds her seat in the train the terminal receives another message. This time it is from the customer who wants to postpone the meeting by one hour. She finds this most suitable because she is hungry and this gives her the possibility to have lunch before the meeting. Arriving to the city the terminal does a 'local number download', meaning that the terminal receives local numbers for different services (ex. taxi, car rental etc). She makes a request on the mobile terminal for the closest lunch restaurant. The terminal receives a map showing her the way calculated from her position. After lunch she looks up taxi in her phone book and pushes the call button, the terminal calls a local taxi company. When the meeting begins the terminal does a 'auto scheduler divert', diverting all incoming calls to a pre-chosen number during the scheduled meeting.

When mobile terminals and bandwidth allows it, more multimedia usage will be integrated in the services. The devices may be used for amusement such as games, movies, music, etc.
Of course this can be combined with other services. Let's say you are a tourist in a city and pass by a church. You would like to know more about the church's history and connects to the local tourist office, when connected you send a query including your position, which is received by a 'current position call'. A short movie sequence is retrieved, it describes the history of the church. After the lecture you feel satisfied and head towards the next monument.

7 Summary

The transfer speed has always been a big problem for the development of mobile applications. UMTS will decrease these problems by introducing an increment of data rate. This will result in new possibilities for mobile services. The WAP architecture, which now is settled as a standard, is a base for future development. A very interesting area in WAP is the WTA services, combining the data networks with the public telephone network, which can lead to many new useful services in the near future.

An important issue for now is which APIs that will be used as interfaces towards wireless application servers and telecom networks. A standardization of these would make it possible to obtain services that are interoperable. Parlay 2.0 has worked towards creating an API that addresses the multiple domains of IP, public telephone network and the wireless network. Some benefits with the Parlay API are the open and technology independent specification.

We found CORBA to be a good way to handle distributed objects in application server environments since it provides scalability and has been proven to be fast performing. It is platform independent and was also easy to learn and use. We had problems running Parlay on CORBA since there was no documentation available at this point. The only documentation available was a user-guide for Parlay on DCOM, which differs from using Parlay on CORBA.

We conclude that the new open architecture with the APIs will open a market where more developers will participate in the creation of new useful services. There will be a fast progress in the evolution of services.

Abbreviations

3G	Third Generation mobile system
API	Application Programming Interface
ARPA	American Research Project Agency
ARPANET	American Research Project Agency NETwork
BOA	Basic Object Adapter
CDPD	Cellular Digital Packet Data
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
EJB	Enterprise Java Beans
ETSI	European Telecommunications Standards Institute
GPRS	General Packet Radio Service
GSM	Global System for Mobile communication
HSCSD	High Speed Circuit Switched Data
HTTP	HyperText Transfer Protocol
IDEN	Integrated Dispatch Enhanced Network
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IMT-2000	International Mobile Telephone 2000
IIOP	Internet Inter-Orb Protocol
ISDN	Integrated Services Digital Network
ITU	International Telecommunication Union
IP	Internet Protocol
JAIN	Java in Advanced Intelligent Networks
JDK	Java Development Kit
JSP	Java Server Pages
LAN	Local Area Network
NMT	Nordic Mobile Telephone
OMA	Object Management Adapter
OMG	Object Management Group
ORB	Object Request Broker
PDM	Product Data Management

PPG	Push Proxy Gateway
PSTN	Public Switched Telephone Network
RMI	Remote Method Invocation
SIM	Subscriber Identity Module
SMS	Short Messaging Service
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TD-CDMA	Time Duplex Code Division Multiple Access
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
URI	Uniformed Resource Identifier
URL	Uniformed Resource Locator
USIM	UMTS Subscriber Identity Module
UTRA	UMTS Terrestrial Radio Access
WML	Wireless Markup Language
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
W-CDMA	Wideband Code Division Multiple Access
WDP	Wireless Datagram Protocol
VHE	Virtual Home Environment
WML	Wireless Markup Language
WSP	Wireless Session Protocol
WTA	Wireless Telephony Application
WTAI	Wireless Telephony Application Interface
WTLS	Wireless Transport Layer Security
WTP	Wireless Transaction Protocol
WWW	World Wide Web
XML	Extensible Markup Language

References

[1] BEA Systems Inc, BEA WebLogic Server, http://www.bea.com/products/weblogic/server/index.html.

[2] ETOCK, ETOTGK, ETOLAS, ERAROSK, Ericsson WTA prestudy, Ericsson As, June 30, 1999

[3] IBM, IBM WebSphere, http://www-4.ibm.com/software/webservers/

[4] International Telecommunication Union, IMT-2000, http://www.itu.int/imt/, March 9, 2000.

[5] J. Hjelm, Final report on advanced mobile multimedia applications, Bonnier Interaktiv AB June 30, 1998.

[6] Morgan B, CORBA meets Java, JavaWorld October 1998

[7] Object Management Group Inc, CORBA, http://www.corba.org/

[8] Object Management Group Inc, The Common Object Request Broker: Architecture and Specification, http://sisyphus.omg.org/gettingstarted/corbafaq.htm#WhatIsIt October 8, 1999.

[9] Object oriented concepts Inc, ORBacus, http://www.ooc.com/products/orbacus.html

[10] Parlay Group, Parlay, http://www.parlay.org/ January 25, 2000.

[11] Parlay Group, Press release: Expansion into IP and wireless, http://www.parlay.org/Papers/Web Release.htm June 15, 1999.

[12] Per-Anders Johansson and Magnus Nilsson, Bachelor's project Evaluation of web application servers June, 2000.

[13] Randy Otte, P Patrick and M Roy, Understanding CORBA, Prentice Hall PTR 1996.

[14] Sybase Inc, Sybase enterprise application server, http://www.sybase.com/products/application_servers/

[15] UMTS forum, The path towards the UMTS - Technologies for the information society http://www.umts-forum.org/reports/report2.doc.

[16] UMTS Forum, UMTS, http://www.umts-forum.org/ April 28, 2000.

[17] UMTS forum, UMTS/IMT-2000 Spectrum, http://www.umts-forum.org/reports/report6.doc.

[18] WAP Forum, WAP, http://www.wapforum.org/

[19] Wireless Application Protocol forum Ltd, WTA, http://www1.wapforum.org/tech/documents/SPEC-WTA-19991108.pdf November 8, 1999.

Appendix

```
1     // Client.java in voice mail node
2
3          package Mail_syst;
4
5          import org.omg.CORBA.*;
6          import org.omg.CosNaming.*;
7          import org.omg.CosNaming.NamingContextPackage.*;
8          import java.io.*;
9          import java.util.*;
10
11         public class Client
12         {
13                   public static void main(String[] args)
14                   {
15                   java.util.Properties props = System.getProperties();
16                   props.put("org.omg.CORBA.ORBClass",
                     "com.ooc.CORBA.ORB");
17                   props.put("org.omg.CORBA.ORBSingletonClass",
18                             "com.ooc.CORBA.ORBSingleton");
19                             System.setProperties(props);
20
21                   try
22                   {
24                             org.omg.CORBA.ORB orb =
                     org.omg.CORBA.ORB.init(args, props);
25
26         // Get naming service
27
28         org.omg.CORBA.Object namingobj = null;
29
30         try
31         {
32         namingobj = orb.resolve_initial_references("NameService");
33         }
34         catch(org.omg.CORBA.ORBPackage.InvalidName ex)
35         {
36                   System.err.println("Can't resolve 'NameService'");
37         System.exit(0);
38         }
39
40        if(namingobj == null)
41        {
42                   System.err.println("'NameService' is a nil object
reference");
43                   System.exit(0);
44         }
45
46         NamingContext nc = NamingContextHelper.narrow(namingobj);
47
48                   if(nc == null)
49                   {
50                   System.err.println("'NameService' is not " +
51                             "a NamingContext object reference");
52                   System.exit(0);
53                   }
54
55          try
56          {
57
58                             // Resolve names with the Naming Service
59
```

```
60                                          NameComponent[] mail = new
NameComponent[2];
61            mail[0] = new NameComponent();
62            mail[0].id = "mail_c";
63            mail[0].kind = "";
64
65                                          mail[1] = new NameComponent();
66            mail[1].id = "mail_obj";
67            mail[1].kind = "";
68            org.omg.CORBA.Object obj = nc.resolve(mail);
69                Mail p = MailHelper.narrow(obj);
70                    if(p == null)
71                                                      throw new
RuntimeException();
72
73                                                      //Reading the
textfile
74
75                                                      String Message =
null;
76                                                                try
77                                                                {
78
          String Mailfile = "voicemail.txt";
79
          java.io.BufferedReader in =

          new java.io.BufferedReader(new

          java.io.FileReader(Mailfile));
80
          Message = in.readLine();
81                                                                }
82
          catch(java.io.IOException ex)
83                                                                {
84
          System.err.println("Can't read from '

          "+ ex.getMessage() + "'");
85
          System.exit(1);
86                                                                }
87
          p.get_mail(Message);
88                                          }
89                              catch(NotFound ex)
90                              {
91                                                      System.err.print("Got
a 'NotFound' exception (");
92                                          switch(ex.why.value())
93                                                      {
94                                                                case
NotFoundReason._missing_node:
95
          System.err.print("missing node");
96                                                                break;
97
98                                                                case
NotFoundReason._not_context:
```

```
99
          System.err.print("not context");
100                                                  break;
101
102                                                  case
NotFoundReason._not_object:
103                                                  System.err.print("not
object");
104                                                  break;
105                                         }
106                                         System.err.println(")");
107                                         ex.printStackTrace();
108                                         System.exit(0);
109                                         }
110                                         catch(CannotProceed ex)
111                                         {
112
          System.err.println("Got a ‘CannotProceed’ exception");
113                                         ex.printStackTrace();
114                                         System.exit(0);
115                                         }
116                               catch(InvalidName ex)
118                               {
120                                         System.err.println("Got an
‘InvalidName’ exception");
121                                         ex.printStackTrace();
122                                         System.exit(0);
123                                         }
124                                                  }
125                                                  catch(COMM_FAILURE
ex)
126                                                  {
127
          System.err.println(ex.getMessage());
128                                                  ex.printStackTrace();
129                                                  System.exit(1);
130                                                  }
131
132                                                  System.exit(0);
133       }
134       }
```

```
1          // Client.Java in WTA server
2          //
3          // Client for changing max number of messages in message
4          // buffer object.
5
6          package Mail_syst;
7
8          import org.omg.CORBA.*;
9          import org.omg.CosNaming.*;
10         import org.omg.CosNaming.NamingContextPackage.*;
11         import java.io.*;
12         import java.util.*;
13         import java.lang.*;
14
15         public class Client extends Thread
16         {
17                    public static void main(String[] args)
18                    {
19                              java.util.Properties props =
System.getProperties();
20                              props.put("org.omg.CORBA.ORBClass",
"com.ooc.CORBA.ORB");
21
           props.put("org.omg.CORBA.ORBSingletonClass",
22                              "com.ooc.CORBA.ORBSingleton");
23                              System.setProperties(props);
24
25                              try
26                              {
27
28                                        org.omg.CORBA.ORB orb =
org.omg.CORBA.ORB.init(args, props);
29
30                    //
31       // Get naming service
32       //
33         org.omg.CORBA.Object namingobj = null;
34
35       try
36       {
37                    namingobj =
orb.resolve_initial_references("NameService");
38         }
39         catch(org.omg.CORBA.ORBPackage.InvalidName ex)
40         {
41                    System.err.println("Can't resolve 'NameService'");
42                    System.exit(0);
43         }
44
45         if(namingobj == null)
46         {
47                    System.err.println("'NameService' is a nil object
reference");
48                    System.exit(0);
49         }
50
51         NamingContext nc = NamingContextHelper.narrow(namingobj);
52
53       if(nc == null)
54         {
55                    System.err.println("'NameService' is not " +
```

```
56         "a NamingContext object reference");
57                   System.exit(0);
58         }
59
60
61         try
62       {
63                   //
64                   // Resolve names with the Naming Service
65                                        //
66                                        NameComponent[] mail = new
NameComponent[2];
67          mail[0] = new NameComponent();
68          mail[0].id = "mail_c";
69          mail[0].kind = "";
70
71                                        mail[1] = new NameComponent();
72          mail[1].id = "mail_obj";
73          mail[1].kind = "";
74          org.omg.CORBA.Object obj = nc.resolve(mail);
75                   Mail p = MailHelper.narrow(obj);
76                 if(p == null)
77                   throw new RuntimeException();
78
79                   //
80                   // Get user input and call msg buffer object
81                   //
82
83                   try{
84                   String maxString = new String();
85                             BufferedReader in = new BufferedReader(new
                     InputStreamReader(System.in));
86                   System.out.println("Enter max Messages before send:
");
87                   maxString = in.readLine();
88                   short max = Short.parseShort(maxString);
89                   p.changeMax(max);
90                   in.close();
91                   }catch(java.io.IOException ex){
92         System.err.println("Can't read from '" + ex.getMessage() + "'");
93                                                  System.exit(1);
94                   }
95
96                   }
97                   catch(NotFound ex)
98     {
99         System.err.print("Got a 'NotFound' exception (");
100        switch(ex.why.value())
101        {
102                  case NotFoundReason._missing_node:
103         System.err.print("missing node");
104         break;
105
106         case NotFoundReason._not_context:
107         System.err.print("not context");
108         break;
109
110         case NotFoundReason._not_object:
111         System.err.print("not object");
112         break;
113      }
```

```
114      System.err.println(")");
115      ex.printStackTrace();
116      System.exit(0);
117     }
118     catch(CannotProceed ex)
119     {
120       System.err.println("Got a ‘CannotProceed’ exception");
121       ex.printStackTrace();
122       System.exit(0);
123     }
124     catch(InvalidName ex)
125     {
126       System.err.println("Got an ‘InvalidName’ exception");
127       ex.printStackTrace();
128       System.exit(0);
129     }
130                              }
131                              catch(COMM_FAILURE ex)
132                              {
133                  System.err.println(ex.getMessage());
134                  ex.printStackTrace();
135                  System.exit(1);
136                              }
137
138                              System.exit(0);
139  }
140       }
```

```
1 //Mail.idl
2
3 interface Mail
4 {
5          void get_mail(in string obj);
6          void makeHtml();
7          void changeMax(in short max);
8 };
```

```
1          //Mail_imp.java
2
3          package Mail_syst;
4
5          public class Mail_impl extends _MailImplBase
6          {
7                     private int BuffertSize = 50;
8                     private String txt = new String();
9                     private int messages = 0;
10                    private short Max_Number_Of_Msg = 5;
11                    private String[] msg = new String[BuffertSize];
12
13                    public void get_mail(String newMessage)
14         {
15                              msg[messages] = newMessage;
16                              messages += 1;
17                              txt = "New Voicemail recieved. " +
messages + " Message(s)                                        Buffered";
18                              System.out.println(txt);
19                              if ( messages >= Max_Number_Of_Msg )
20                                        makeHtml();
21         }
22
23                    public void makeHtml(){
24                    try{
25                    java.io.PrintWriter out = new java.io.PrintWriter(new
                      java.io.FileWriter(".\\Mail_syst\\msg.html"));
26                    out.println("<HTML>");
27                    out.println("<HEAD>");
28                    out.println("<TITLE>New Messages</TITLE>");
29                    out.println("<script language=\"JavaScript\">");
30                    out.println("function popUp() {");
31              out.println("props=window.open('c:\\\\WTAlink\\\\link.html',
                              'poppage', 'toolbars=0, scrollbars=0,
location=0, statusbars=0, menubars=0, resizable=0, width=200,
           height=300 left = 300, top = 100');}");
32                    out.println("</script>");
33                    out.println("</HEAD>");
34                    out.println("<BODY onLoad=\"resizeTo(220,350)\"
                      BGCOLOR=\"#D3D3D3\" LINK=\"#000000\">");
35                    out.println("<DIV style=\"position: absolute;
top:10px; left:10px; width:200px; height:25px\"><B>You have new
                              Messages</B><BR></DIV>");
36
37                    String Line = new String();
38                    for(int i = 0; i < Max_Number_Of_Msg ; i++){
39                    int top = 40+20*i;
40                    Line = "<DIV style=\"position: absolute; top:" +
                      top + "px; left:10px; width:200px;
                      height:25px\"><A>#</A><A
                      HREF=\"javascript:popUp()\">" + msg[i] +
                      "</A><BR></DIV>";
41                    out.println(Line);
42                    }
43                    out.println("</BODY>");
44                    out.println("</HTML>");
45                    out.close();
46                    java.io.PrintWriter out2 = new
java.io.PrintWriter(new
           java.io.FileWriter(".\\Mail_syst\\info.html"));
47                                        out2.println("<HTML>");
```

```
48                                          out2.println("<HEAD>");
49                                          out2.println("<TITLE>New
Messages</TITLE>");
50                                          out2.println("</HEAD>");
51                                          out2.println("<BODY
onLoad=\"resizeTo(220,350)\"
          BGCOLOR=\"#D3D3D3\" LINK=\"#000000\">");
52                                          out2.println("<DIV
style=\"position: absolute; top:10px;
          left:10px; width:200px; height:25px\"><B>You have " +
                              Max_Number_Of_Msg  + " new
Messages</B><BR></DIV>");
53                                          out2.println("<DIV
style=\"position: absolute; top:30px;
          left:10px; width:200px; height:25px\"><A

          HREF=\"c:\\ooc\\Mail_syst\\msg.html\"> Read
                                                    </A><BR></DIV>");
54                                          out2.println("<DIV
style=\"position: absolute; top:45px;
          left:10px; width:200px;

          height:25px\"><A>Close</A><BR></DIV>");
55                                          out2.println("</BODY>");
56                                          out2.println("</HTML>");
57                                          out2.close();
58
59                                          Process proc =


          java.lang.Runtime.getRuntime().exec("iexplore

          c:/ooc/Mail_syst/info.html");
60                                          messages = 0;
61                              }catch(java.io.IOException e){
62                                          System.err.println(e);
63                                          return;
64                              }
65                  }
66
67                  public void changeMax(short max){
68                              Max_Number_Of_Msg = max;
69                  }
70
71
72
73        }
```

```
1          // Server.java
2
3          package Mail_syst;
4
5          import org.omg.CORBA.*;
6          import org.omg.CosNaming.*;
7          import org.omg.CosNaming.NamingContextPackage.*;
8          import java.io.*;
9          import java.util.*;
10
11         public class Server
12         {
13                   public static void main(String args[])
14                   {
15                             java.util.Properties props =
System.getProperties();
16                             props.put("org.omg.CORBA.ORBClass",
"com.ooc.CORBA.ORB");
17
           props.put("org.omg.CORBA.ORBSingletonClass",
18                             "com.ooc.CORBA.ORBSingleton");
19                             System.setProperties(props);
20                             try
21                             {
22
23                                       // Create ORB and BOA
24
25                                       org.omg.CORBA.ORB orb =
                                         org.omg.CORBA.ORB.init(args,
props);
26                                       org.omg.CORBA.BOA boa =

           ((com.ooc.CORBA.ORB)orb).BOA_init(args, props);
27                                       Mail_impl p = new Mail_impl();
28
29                                       // Get naming service
30
31                             org.omg.CORBA.Object obj = null;
32
33                             try
34                             {
35                             obj =


           orb.resolve_initial_references("NameService");
36                             }
37                             catch(org.omg.CORBA.ORBPackage.InvalidName
ex)
38                             {
39                             System.out.println("Can’t resolve
‘NameService’");
40                             System.exit(1);
41                             }
42
43                             if(obj == null)
44                             {
45                             System.out.println("‘NameService’ is a nil
object

                      reference");
46                             System.exit(1);
```

```
47                              }
48
49                              NamingContext nc =
NamingContextHelper.narrow(obj);
50
51                              if(nc == null)
52                              {
53                              System.out.println("'NameService' is not "
+
54                                        "a NamingContext object
reference");
55                              System.exit(1);
56                              }
57
58                              try
59                              {
60                                        // Bind name with the Naming
Service
61
62                                                  NameComponent[]
mail_context = new
                                NameComponent[1];
63                              mail_context[0] = new NameComponent();
64                              mail_context[0].id = "mail_c";
65                              mail_context[0].kind = "";
66                              NamingContext mail_c =

          nc.bind_new_context(mail_context);
67
68                              NameComponent[] mail = new
NameComponent[2];
69                                                  mail[0] = new
NameComponent();
70                              mail[0].id = "mail_c";

71                              mail[0].kind = "";

72                                        mail[1] = new NameComponent();
73                              mail[1].id = "mail_obj";
74                              mail[1].kind = "";
75                              nc.bind(mail, p);
76
77                                                  // Run implementation
78
79                              boa.impl_is_ready(null);
80
81                              nc.unbind(mail);
82                              nc.unbind(mail_context);
83                                        }
84                              catch(NotFound ex)
85                              {
86                              System.err.print("Got a 'NotFound'
exception (");
87                              switch(ex.why.value())
88                              {
89                              case NotFoundReason._missing_node:
90                              System.err.print("missing node");
91                                                  break;
92
93                                                  case
NotFoundReason._not_context:
```

```
94                                                  System.err.print("not
context");
95                                                  break;
96
97                                          case NotFoundReason._not_object:
98                                                  System.err.print("not
object");
99                                                  break;
100                 }
101
102                                         System.err.println(")");
103                 ex.printStackTrace();
104                 System.exit(1);
105                 }
106                 catch(CannotProceed ex)
107                 {
108                 System.err.println("Got a ‘CannotProceed’
exception");
109                 ex.printStackTrace();
110                 System.exit(1);
111                 }
112                 catch(InvalidName ex)
113                 {
114                 System.err.println("Got an ‘InvalidName’ exception");
115                 ex.printStackTrace();
116                 System.exit(1);
117                 }
118                 catch(AlreadyBound ex)
119                 {
120                 System.err.println("Got an ‘AlreadyBound’
exception");
121                 ex.printStackTrace();
122                 System.exit(1);
123                 }
124
125                           }
126                           catch(SystemException ex)
127                           {
128
          System.err.println(ex.getMessage());
129                                         ex.printStackTrace();
130                                         System.exit(1);
131                           }
132
133                           System.exit(0);
134         }
135         }
```