WIRELESS INTERNET SOFTWARE ENGINEERING IST-2000-30028

Title:
Management of heterogeneous clients

Version: 3.2
Date : Sept. 10, 2004
Pages : 31

Author(s):
Filippo Forchino (MTCI), Mario Negro Ponzi (MTCI), Marco Cappelli (MTCI)

To:
WISE CONSORTIUM

Printed on:
17/09/2004 10.29

Status:		**Confidentiality:**	Public	
[ ]	Draft	[X]	Public	- Intended for public use
[ ]	To be reviewed	[ ]	Restricted	- Intended for WISE consortium only
[ ]	Proposal	[ ]	Confidential	- Intended for individual partner only
[X]	Final / Released			

Deliverable ID: D3

Title:

Management of heterogeneous clients

Summary / Contents:

This document describes the problems that may be faced developing software for mobile, heterogeneous clients and which are the applicable solutions.

It represents a collection of guidelines that developers shall follow in order to cope properly with such problems, and it is the result of the experience matured by the writers in the field of design and development of software for wireless devices, inside and outside the WISE project.

Change log

Vers.	Date	Authors	Description
1.0	2 Oct 2002	F. Forchino, M. Negro Ponzi	Initial description
1.1	6 Nov 2002	M. Negro Ponzi	Devices summary table, introduction to Qualcomm's BREW, merge of chapter 6 into 5 and 7.
1.2	20 Dec 2002	F. Forchino, M. Negro Ponzi	Update considering Sodalia's review.
2.0	14 May 2003	F. Forchino, M. Negro Ponzi	Re-organisation of the document in form of guideline
2.1	08 Sept 2003	M. Negro Ponzi	Adjustements
2.2	21 Oct 2003	P. Falco	Integrated comments from partners' review
2.3	30 Jan 2004	M.Cappelli	Rework of chapter 4: caching of pictures,GPRS EDGE UMTS overview, i-mode.
3.0	27 Apr 2004	M.Cappelli	Added a table of comparison of GPRS, Edge, UMTS.
3.1	18 June 2004	F. Forchino	Added guidelines about developing with GPS and Databases to integrate partners' experience.
3.2	10 Sept 2004	F. Forchino	Update after PoliTo and IESE review.

Content

1 Introduction 5
2 Classes of Devices 5
3 Packet-Switching Wireless Network Standards 8
3.1 GPRS 8
3.2 EDGE 8
3.3 UMTS 9
3.4 Comparison Table 9
4 Service architectures 10
4.1 Thin clients 10
4.1.1 HTML 10
4.1.2 WML and similar technologies 10
4.1.2.1 i-mode™ 11
4.2 Fat clients 12
4.2.1 Available Java 2 Micro Edition technologies 13
4.2.2 BREW technology 14
4.2.3 Native development 14
4.3 Summarizing architectures: advantages and disadvantages 15
5 Common problems and how to solve them 16
5.1 User interface and graphics 17
5.1.1 Downsizing desktop screens 17
5.1.2 J2ME-specific issues 17
5.1.3 Customized deployment 18
5.1.4 Caching of pictures 19
5.2 Input management 19
5.3 Network support 20
Protocols 20
5.4 Computing power 21
5.5 Memory and storage 22
5.6 Security issues 22
5.7 Hardware access 23
5.8 Debug: how good are emulators? 24
5.9 Developing position-aware applications 24
5.10 Databases 25
6 Related work 26
7 Conclusion 29

Abbreviations

AWT	Abstract Windowing Toolkit
BREW	Binary Runtime Environment for Wireless
CDC	Connected Device Configuration
CDLC	Connected Device Limited Configuration
CDMA	Code Division Multiple Access
EDGE	Enhanced Data rates for GSM Evolution
GPRS	General Packet Radio Services
GPS	Global Positioning System
GSM	Global System for Mobile communications
J2ME	Java 2 Micro Edition
JNI	Java Native Interface
MIDP	Mobile Information Device Profile
OS	Operating System
PDA	Personal Digital Assistant
PDC	Portable Digital Cellular
PDC-P	Portable Digital Cellular Packet Network
PGW	Packet Gateway transfer processing equipment
PPM	Packet local processing Module
QoS	Quality of service
SDK	Software Development Kit
TDMA	Time Division Mobile Access
UMTS	Universal Mobile Telecommunications System
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WML	Wireless Markup Language
WMLScript	ECMA-62 based scripting language for WML
WTLS	Wireless Transport Layer Security

1 Introduction

This document describes common issues when building up services targeting multiple different clients and some development approaches to solve them.

Since the same service can be developed for desktop computers and mobile clients, differences between desktops and mobiles, as well as among mobile clients, are a main concern for developers. This document will present a common approach for developing services that have to be handled by many clients with different sizes, performances, and, in general, capabilities, especially where clients are connected to the network wirelessly.

We will at first outline the more important target devices and describe two main classes of services. Then we will overview current development approaches that bridge device differences, and describe common problems faced by programmers along with solutions or good practices that can solve, avoid or mitigate them.

2 Classes of Devices

Let us begin by over viewing the kind of device this document addresses. We grouped them in a short list describing the typical capabilities of each one. The least common denominator is the possibility to connect them wireless to the Internet. Devices are listed from the biggest and faster to the slowest and smallest (at least when these two dimensions agree).

1) **Desktops / Notebooks**: as everybody knows it is possible to provide wireless connectivity to a common PC just adding a dedicated card. Service application on those clients is just limited by the network bandwidth /latency, while processing speed is not a problem; application development follows a traditional approach;
2) **TabletPC**: notebooks taken to their extreme consequences are called TabletPC and their power/capabilities range from an oversized PDA (see later) to a compressed notebook. In Figure 2-1 it is emphasized that TabletPC often have a pointing device as a touch screen. TabletPC without such pointing device are most commonly called Subnotebooks. Processor speed ranges, today, from a 200Mhz ARM processor to full last generation Pentium III/IV and memory sizes vary accordingly. Operating systems can be those of desktops or those of PDAs, depending on device.

Figure 2-1: TabletPC

3) **PDA**: entirely based on a touch screen the most common components of this family are Palm PC and PocketPC. Palm PC are developed by Palm and Visor companies and are based on Motorola Dragonball processor (up until now, at least) at a typical clock rate of no more than 20MHz. Memory size ranges from 1MB to 16MB. Palm OS is the referring Operating System (many versions available). PocketPC are developed by many companies, are based on ARM (mostly Intel StrongARM) processors with a clock frequency that range from 150MHz to 206 and above. Memory size can be 16, 32, 64 MB. They are all based on Microsoft PocketPC operating system (basically a downgrade of common desktop Microsoft OS Windows NT) and the vast majority has now a color display. PDAs are generally thought as mini-notebooks and connectivity is normally optional (although easy to add).

Figure 2-2: PDA

4) **Handheld PC:** basically PDAs with also a small keyboard.
5) **PDA phones**: an extension to PDA where connectivity to the phone network is deeply integrated in the device and not added. They follow general guidelines provided by PDAs (Operating systems have been enhanced to support the added capabilities).

Figure 2-3: PDA phones

6) **Smart phones**: same as PDA phones but coming from a cellular world rather than from the computer world. Operating systems now include SmartPhone from Microsoft (basically an enhancement to PocketPC), Symbian (a consortium formed by Motorola, Nokia, Ericsson and others) and other proprietary OSes.
7) **Cellular phones** (J2ME enabled): the electronic best sellers of those last few years have been enhanced to support the download of application and services. Today it's common to find J2ME enabled phones but in the near future other programming platforms can be available (i.e. Qualcomm is porting BREW also on non-CDMA platforms, see 4.2.2).

The above list is an overview of the main classes of devices in the wireless domain that can be reached by the same type of service, and for which developers and service providers usually want to be able to create and maintain a single piece of software. The table below summarizes the situation.

Category	Screen size and capabilities	Memory	Cpu	Connectivity	OS/Platform
Desktops/Notebooks	640x480 to 1600x1200 and more (screen from 10 to 24 inches), million color graphics	Up to some GB RAM and hundreds of GB of mass storage	Currently from 1 to about 3 GHz	Wired or wireless	Any high-end: Windows, Linux/Unix, MacOS
TabletPC	800x600 to 1280x1024) screen from 10 to 15 inches, million color graphics	Up to few GB RAM and tens of GB of mass storage	Around 1 GHz (extra low power processors)	Wired or wireless	Customized version of those for desktops
PDA	About 240*320, thousand color graphic	Up to some hundreds MB of shared RAM/mass storage memory.	Intel StrongARM and XScale, Motorola DragonBall, MIPS and other ARM-based processors	Wired, wired through a host PC, wireless	Scaled-down version of those for desktops or dedicated ones.

Handheld PC	Up to 320x120, thousand color	As for PDA	As for PDA	As for PDA	Same as PDA
PDA phones	As for PDA	As for PDA	As for PDA, scaled down for power saving	As for PDA plus cellular networks	Customized version of those for PDA
Smart phones	As for PDA	As for PDA	Mostly new ARM-based CPUs	As for PDA phones	Empowered version of those for cellular phones, Windows Mobile, Symbian.
Cellular Phones	From non-graphical, text-only to thousands colors graphics	From some KB to few MB, growing	From 16bit microcontrollers to high speed cpu	Cellular network, wireless and wired with or without a host PC	Dedicated mainly proprietary plus Symbian.

3 Packet-Switching Wireless Network Standards

Standardization and cooperation using GSM caused a huge market growth in Europe. GSM has proven itself as a good system for voice communication but its lack of bandwidth made it less useful for data transmission.

3.1 GPRS

A partial solution of the lack of bandwidth problem of GSM has come from the General Packet Radio Service (GPRS). GPRS is a service that allows for the first time the users to connect to the Internet with a mobile device. GPRS use the circuit switched[1] GSM network, overlaying it with a packet based[2] air interface. Rather than dedicating a radio channel to a single mobile data user with its packet switching system GPRS share the available radio resources between several users. This efficient use of scarce radio resources means that a single cell can potentially serve a large numbers of GPRS users that will share the same bandwidth. The number of users supported depends on the application being used and the amount of transferred data. With the efficient use of the radio resources of GPRS, there is less need to build in idle capacity that is only used in peak hours. GPRS therefore lets network operators maximize the use of their network resources in a dynamic and flexible way, along with user access to resources and revenues. GPRS will offer a connection speed of up to 115 kbps in ideal conditions and 30-40 kbps in average mobile condition.

3.2 EDGE

The Enhanced Data rates for GSM Evolution (EDGE) is a third-generation (3G) high-speed mobile data and Internet access technology it is an upgrade for GPRS networks that increase the throughput and

[1] Type of network in which a physical path is obtained for and dedicated to a single connection between two end-points in the network for the duration of the connection. An example is the ordinary voice phone service.

[2] Packet-switched describes the type of network in which relatively small units of data called *packets* are routed through a network based on the destination address contained within each packet. Breaking communication down into packets allows the same data path to be shared among many users in the network. This type of communication between sender and receiver is known as connectionless (rather than dedicated). Most traffic over the Internet uses packet switching and the Internet is basically a connectionless network.

capacity of GPRS by three to four times. EDGE, like GPRS, is a packet based service that provides to the users a constant data connection. The first commercial launch of EDGE services took place on June 30, 2003 in Indianapolis (USA) and a full-scale commercial EDGE deployments is expected to occur throughout 2004.

With an average throughput of 80-130 kbps , EDGE can support a wide range of advanced data services, including streaming audio and video, large file download and a fast Internet access.

3.3 UMTS

Universal Mobile Telecommunications System (UMTS) is the leading 3G technology today, it will offer a potential worldwide coverage. UMTS is an Internet Protocol-based technology that supports packetized voice and data that delivers peak data rates of up to 2.4 Mbps and an average speed of 300Kbps (with an increase of speed if the device is stationary). As EDGE, UMTS is designed to deliver services such as streaming multimedia, large file transfers and even video-conferencing to a variety of devices, including cell phones, PDAs and laptops. Like EDGE, UMTS is packet-based and offers an always-on connection [3] .

UMTS is compatible with EDGE and GPRS which allows users to move out of an area with UMTS coverage and automatically be switched to a an EDGE or GPRS network, it includes sophisticated quality of service (QoS) mechanisms, that will ensure that the needed amount of resources is given to each type of data service.

3.4 Comparison Table

	Average speed (kbps)	Theoretical maximum (kbps)	
GPRS	30-40	115	- Always on data connection - Packet switching - Available over GSM networks
EDGE	80-139	170	- Always on data connection - Packet switching - Available over GSM networks - Future enhancements shall include improved voice capacity, coverage and speech quality features
UMTS	300	2400	- Always on data connection - Increased speech quality - Able to switch to an EDGE or GPRS network if needed

The UMTS and EDGE’s high bandwidth will lead to a dramatic increase in wireless services and let developers choose many options previously unavailable.
It is partially possible to solve computing capability and memory problems of clients increasing their bandwidth and moving data, graphics manipulation as well as resource management on the server side. Graphical resources and databases can be moved on the server so that the client just asks for the information it needs and all the heavy duty gets performed on the server. This kind of approach relies on a low-latency and high-bandwidth link.

[3] In the sense that the mobile terminal is always connected to the internet and can send and receive data packets at any time, without the need of establishing before a dial-up connection.

4 Service architectures

Two different categories of services exist. These two map to two service architectures and therefore to two different development methods.

Depending on the needed type of service, programmers can create a kind of web site or a "real" application. That means that for some services a web-based solution is the best choice while in some other cases it's best to have a full client on the terminal. All depends on the type of interactivity and processing power the application needs, as outlined in the table below.

Service group	Poorly interactive	Highly interactive
Information	Mostly static	Highly dynamic
Information freshness	Refreshed rarely	Refreshed frequently or in real time
Data exchange	Slow exchange of small packets of data	Frequent or continuous exchange of packets of data of any size
Information display	Mostly textual	Mostly graphical

These two service categories are both useful and used. Often, it may be possible to implement a service as both a poorly-interactive or highly-interactive application, but most of the times the service requirements clearly map int one of the two categories.

Poorly interactive services tend to be implemented on **Thin Clients**, while highly interactive services usually need a **Fat client**. The choice of which client architecture to use depends almost exclusively on the type of service required and, as we will see, it is not that much influenced by the need of compatibility with desktop applications.

Section 4.1 will describe the thin client approach while section 4.2 will introduce the fat client approach of both Java Micro Edition and Qualcomm's BREW proposals.

4.1 Thin clients

Thin clients do not need or are not able to provide high computational capabilities (i.e. no interactive multimedia or complex searches on databases). They are usually not required to process complex information. Calculations are made, where needed, on servers. It means they usually presents only static information to the user and that almost any interaction between the user and the client triggers a request on a server. In order not to be dependent on different screen sizes and capabilities (e.g. colour depth), the usual approach is to adopt either HTML or a stripped down version of it, usually WML or cHTML.

4.1.1 HTML

High-end PDAs actually support an almost full HTML. It is possible that in the near future, thanks to bigger screens and to faster processors, HTML will be the common, and probably the only, language for the web. For the time being, however, full HTML support for wireless devices is available only on the higher-end PDAs and smartphones, and usually rely on specialized components to render the HTML correctly on the smaller display.

Additionally, at the moment the majority of input devices on mobile devices, are keypads without pointing (mouse) capabilities, while HTML relies heavily on the point-and click interface of desktop computers.

For this reason, poorly interactive services are not usually implemented using full HTML.

4.1.2 WML and similar technologies

In the last few years, a group of markup languages targeted to limited mobile devices has been developed.

Before proceeding we must say that WML is not the only markup language platform of its kind: cHTML and others also do exist. But today, in Europe, WML is by far the de-facto standard among limited-devices markup languages.

As stated above, many services can be deployed using some sort of WAP/WML, where WAP is the application protocol over which commonly WML lives. Those services, generally, are used to provide financial, weather, sport or other information to clients. As we can see in the traditional World Wide Web, there are plenty of such services and almost all of them can be accomplished on mobile clients using WML. WML is a language very similar to the HTML, specially designed for small clients with small screens and low bandwidth: it has a "deck/card [4]"-based approach that minimizes the activity on the network[5] allowing users to feel a much more interactive navigation. Technology's advances are relaxing bandwidth constraints while the major problem still remains the display capability.

WAP-enabled clients with WML commonly also support an enhanced Javascript-like language, called WMLScript: for example it can prove useful for client-side form-check but it also provides some other dedicated functions such, for example, digital signature see later, in 5.6).

WML and WMLScript let web developers modify their current web pages to make them suitable for wireless mobile clients. This operation normally does not need a complete service's architecture rework. Depending on the personal viewpoint and on the type of service that has to be deployed, those modification can be a little or a big price to be paid for handling heterogeneous clients.

Content-definition languages such as XML can be used to minimize the rework needed for porting a web service to mobile devices by applying the correct stylesheet (XSLT) for the target device, however, the initial hype over this approach has been mitigated by the actual cost of XSLT maintenance and development.

Maintenance costs are also very high, because every modification made to the standard "desktop" version should also be made available to the mobile version.

Dedicated services for mobile clients are also possible. This obviously clears every compatibility issue.

4.1.2.1 i-mode™

Launched on February 22,1999 by NTT DoCoMo in Japan, i-mode is a mobile phone platform born to provide a quick and continuous connection to the Internet. The i-mode has been phenomenally successful in Japan and on December 9, 2001 the i-mode subscriptions have growth to more than 29.5 million and have arisen to 39 million on July, 2003.

NTT DoCoMo's evolved the Portable Digital Cellular (PDC) network into a packet network named PDC Packet Network (PDC-P) .The PDC-P is based on the introduction of two new elements: Packet Gateway transfer processing equipment (PGW) and Packet local processing Module (PPM). The PGW act as a gateway between the mobile network and the external networks (like Internet). The PPM manages the packet exchange from PGW and the mobile device.

The i-mode at the beginning offered a packet transmission speed of 9.6 Kbps, raised to 28.8 Kbps in the middle of 2002. Even with this low speed i-mode had a great success due to the wide variety and quality of the services offered to the users. Some of those services are provided directly by NTT DoCoMo but the majority comes from external partners called content (service) providers.

The service providers are separated in two categories: official and unofficial. The official content provider's sites are server shown in the i-mode menu and are accessible after a subscription and it will be NTT DoCoMo that collects monthly information charges on behalf of i-Menu content providers (charging a 9% commission for the billing system service). The official content providers have been validated and approved by NTT DoCoMo.They

[4] WML "pages" are referred to as decks. Each deck consists of one or more cards. When the WML micro browser accesses a WML document (or deck), it reads the whole deck, and navigation between the cards in this deck is done without the need to load any more data. So once you've loaded a deck, all cards within it stays (statically) in the WML micro browser memory until the browser is instructed to reload the whole deck.

[5] As we've tested on Pilot2 of WISE project, round-trip times over GPRS can be very long, even more than 10 secs. So, if a page takes a long time to be loaded on a desktop client, its loading time on a wireless client could be unacceptable.

are directly connected to the i-mode server, which means that data is not transported over the Internet on its way from the content provider's server to the user.

The unofficial sites are accessible via the Internet directly entering their URL, it will be the i-node server that enables Internet access from i-node phones by relaying communications between the NTT DoCoMo packet network and the open Internet network. Those content providers will not participate in DoCoMo's billing system.

Both the official and unofficial sites have to be written using iHTML that is a subset of HTML, it is very similar to the Compact HTML (cHTML) defined by the W3C, with some addiction like phone number links and the access to a predefined set of icons (Emoji icons). NTT DoCoMo to integrate J2ME on mobiles phones has developed *DoJa*, a Java platform that is based on the CLDC but is not compatible with MIDP.

i-mode™ users outside Japan exceeded two million at the end of January 2004 and continue to grow rapidly. Services are currently available through at least seven i-mode operators outside Japan: BASE NV./S.A (Belgium), Bouygues Telecom (France), E-Plus (Germany), Far EasTone Telecommunications Co., Ltd. (Taiwan), KPN Mobile (The Netherlands), Telefónica Móviles España (Spain), WIND Telecomunicazioni S.p.A. (Italy), representing a market of over 60 million cellular phone users.

Developing an application for i-mode presents pretty the same problems of developing a 'normal' wireless application. The use of iHTML offers an interesting chance, as suggested by NTT DoCoMo a content provider can translate his web site from HTML to iHTML (and this will not be a challenging task) or he can use a conversion server, this server will translate the HTML pages, requested by an i-mode client, into iHTML compatible pages, after that it will deliver the pages to the i-mode client. With a conversion server the web service will be available to both i-mode and Internet users.

The lack of compatibility of WML - iHTML and MIDP - DoJa is a big problem for application development. It means choosing whether to develop a service/application for one or the other platform. Developing an application for both the platforms means develop two different applications.

4.2 Fat clients

Some services need to process a lot of data directly on the client. Common examples include games, graphical applications such as slide viewers, engineering and mathematical tools and many complex applications with local databases. Those applications share many needs with thin clients but cannot be implemented only using the above-described approach. Nevertheless they have to be deployed on many different platforms.

Today almost every device can be programmed using some sort of C-like language but C is hardly a cross-platform language and, more important, compiled C is not portable. A dedicated client should be deployed (where not also developed) for every target platform, slowing down the time-to-market of the service itself and increasing costs.

As a matter of fact, the only "powerful" commonly used language available today on many platforms is JAVA. Not only it is compatible at source-code level but every program doesn't have to be recompiled to distribute it on different platform. Especially after the release by Sun Microsystems of the J2ME specification, Java can now be easily deployed on many mobile clients providing a common ground for developers. Java also offers a protected environment providing users insurance that a malicious or bugged application cannot harm the device itself or its data. And leaves companies the ability to maintain secrecy on their Operating System's code (even API).

But J2ME is not full Java. That means that, while the language remains the same, some of the most advanced (and useful) features of Java are not available on a J2ME platform. Moreover, Java was born for desktop and server computers and its being "both" interpreted and compiled can be a constraint on slow-processors based devices like cellular phones.

Without identical libraries, particularly regarding graphics, different clients have to be developed for small devices and desktops. And since some PDAs implement Personal Java (see later), this could lead to a third version of the client. Even with those limits (three different versions to maintain) many parts of the code and of the architecture can be shared and commonalities are still very important.

Java masks many troubles related to the handling of heterogeneous clients: code can be shared among many devices without big modifications. Making software that has to be deployed also on desktop clients implies making specific software not using resources of modern computers. Some ongoing projects have as a goal to run unmodified J2ME code on a desktop machine equipped with J2SE.

A trained Java programmer can easily switch between standard Java and J2ME (MIDP or Personal Java).

4.2.1 Available Java 2 Micro Edition technologies

Many Java technologies exist at the moment. Sun groups and develops them as divided in 3 main "Editions": Java 2 Micro Edition (J2ME), Java 2 Standard Edition (J2SE) and Java 2 Enterprise Edition (J2EE). Java 2 Micro Edition (J2ME) is targeted specifically at low-end devices, and is being developed using two concepts: configurations and profiles. Configurations are Virtual-machines related issues and mainly state what a virtual machine should or should not be able to do (for example regarding memory constraints, garbage collections, etc.). Profiles on the other side are related to standardization of the API. It is possible that different profiles are available for the same configuration as well as the opposite, at least in theory.
The most common branches of J2ME include:

Embedded Java
Embedded Java provides no core-API. In place of it any vendor can implement any device-specific API to suite specific needs.

Java Card
Java Card was born for implementations on small smart cards-based devices, especially targeted for standardizing secure card communications and transactions.

CDLC + MIDP
Today CDLC (the configuration specific for cell phones) is mainly connected with MIDP. That is why we will consider them toghether. This implementation is developed on three layers and it is specifically targeted to mobile phones and similar devices, in order to provide customized applications to clients.
Main layers of this J2ME implementation are:

- K (Kilobyte) - Virtual Machine:
    - Developed to have a memory footprint between 40 and 80 kilobytes;
    - ANSI-C implementation for easy of porting
    - As fast as possible
- CLDC (Connected Limited Device Configuration):
    - No support for floating point;
    - No support for finalization;
    - No support for Java Native Interface;
    - No reflection;
    - No support for threads group;
    - No weak references.
- MIDP (Micro Java Device Profile):
    - Screen size of at least 96x54;
    - Display depth at least of 1 bit;
    - Ratio of pixel of approximately 1:1;
    - Input with keyboard, keypad or touch screen;
    - 128 KB of non-volatile memory for MIDP components;
    - 8 KB of non-volatile memory for application persistent data
    - 32 KB of volatile memory for Java Runtime (heap)
    - Two way network communication with limited bandwidth

CDC

Based on CLDC, another specification has been launched called CDC: Connected Device Configuration and differentiates itself from the CDLC because of a minimum of 512K ROM needed, 256K RAM, full connectivity to a network and a full compatibility with the standard Java Virtual Machine.

Personal Profile and Personal Java

The Personal Profile, under standardization, will be compatible with an earlier version of limited Java, called Personal Java and already available on some PDA and high-end cell phones.

This profile is a very complete Java environment lacking in practice just some heavy graphical features. For example, it includes support for reflection, JNI and AWT. It's commonly said that Personal Java is very much like Standard Java as of version 1.1.8.

4.2.2 BREW technology

BREW is a Qualcomm's solution proposed in order to deploy a common programming environment dedicated to cell phones. BREW is an open applications execution platform that resides on the wireless device. Some key features (with the Java counterpart) are discussed below:

- BREW is thin because it's developed specifically for wireless phones, while J2ME suffers for being a "downsizing" of an heavy architecture;
- BREW is fast (at least faster than Java) because its runtime is chip-based (that means that is directly supported by a dedicated hardware/firmware); on the other side new CPUs (i.e. ARM) are introducing hardware support for Java bytecode (called Jazelle for ARM);
- BREW is more flexible than Java because it supports different programming languages: from C/C++ to WML, on top of it it's provided even a J2ME compliant virtual machine (so that a BREW-enabled phone can easily support J2ME too), but using JAVA on top of BREW should be deeply investigated for performance behaviors. It's always better to build a Java Virtual Machine on top of the operating system's API.

The BREW platform also proposes a framework for deploying, testing and shipping application directly on the phone. And, much important, it takes care also of the billing perspective providing a kind of pre-packaged server-side structure. For commercial application this can be seen as a big add-on because with J2ME there is no standard already in place.

Actually the main limit of BREW is that it's provided only for CDMA platforms, while porting on other networks such as GSM/GPRS is underway.

4.2.3 Native development

Until some recent initiatives, it was practically impossible to develop native applications for mobile handsets, because manufacturers did not provide interfaces to the handset code, relied on heavily proprietary systems, and did not provide any facility to install and debug the software. Development was possible on PDAs, however, since the two main Operating System developers, Palm Inc. and Microsoft, used the customisability of the platforms and the number of available applications as an incentive for buyers, and therefore provided SDKs and tools to the developer community.

This situation has recently changed mainly for two reasons:

1. Manufacturers have started to commercialise mobile handsets based on third-party operating systems, such as Linux or Microsoft Windows Mobile. This has made the facilities of these Operating Systems available to developers
2. Manufacturers have created consortiums for the standardization of Operating Systems or native application layers, such as the Symbian OS consortium. This again has the intent of attracting the developer community to native development.

The most recent analyses, however, still show a slow progress on development of native service for mobile handsets, which tends to be very costly and too difficult to port to other devices.

4.3 Summarizing architectures: advantages and disadvantages

The table below compares the previously discussed technologies, summarizing the advantages and disadvantages of each. A brief description on their scope (typical applications for which each technology is mainly used) is included.

Technology	Advantages	Disadvantages
Thin client based on WAP/WML	Portable over millions of cell phones	Very limited GUI customization capabilities
	Easy to develop for non-programmers	Very limited and slow interaction
	Easy to port from web-applications	Little or no control over representation
	Widely supported by handset manufacturers	Provider support through WAP gateways
	WTLS security protocol supported	Unsupported by manufacturers outside the wireless industry
	No development risk	
Typical applications: news, financial updates, online address book, WEB-based applications porting		
Fat client based on proprietary solution	It is possible to develop core applications	A different application has to be developed for every possible device
	Total integration with native operating system: it is possible to use hardware dependant features.	A different application has to be deployed for every possible device
	Full GUI customization	Developers should be trusted by manufacturers (and/or have to pay license fees)
	Fastest access to the network	Usually developed in C/C++ with language idiosyncrasies
	Fastest execution speed	
Typical applications: embedded applications developed for provider-customized devices		
Fat client based on J2ME	Portable to many new generation high-end phones	Not adaptable to old phones
	Full GUI control	Need for customization for different devices' displays
	Widely known language	J2ME is a bit different from J2SE and programmers need to get in touch with it
	Strongly supported	Devices specific implementations
	New processor can provide hardware support	Proprietary non-portable APIs
	Open solution with default implementation	Interpreted language: on low-power device it can be very slow
	Supported over many general-purpose devices (not only cell phones) with various implementations.	Limited performance of the Garbage Collector
	Application secured inside a sandbox	Need for a consortium agreements on improvements: slow process
		Application deployment still not standardized
		Application security limits
		Memory-hungry
		Dependence on manufacturer's implementation for hardware access
Typical applications: interactive games, business applications developed for a wide audience, application that need a graphical processing such as map-based applications		

Fat client based on BREW	Core hardware support	CDMA-only devices (at least until today)
		Proprietary solution (with license fee)
		Specific language
		No native support
		Dependence from Qualcomm for improvements
Typical applications: committed on-demand applications developed for BREW clients on CDMA networks		
Fat client developed using native tools on Symbian OS, Palm OS or MS Pocket PC/Smartphone	Standardized solution with heavy industrial support	Three different platforms (PocketPC, Palm OS and Symbian OS)
	Deep integration with the environment	Today only deployed on very high-end devices
	Open consortium for Symbian OS, open APIs for the others	Minor customization problems among different devices: different PocketPC distributions, various GUI interface Symbian implementations, different OS versions
	Large background and history for Microsoft PocketPC/Smartphone and Palm OS	
	Programmable using many languages (from C to Java to OPL/Visual Basic/.NET)	
	Adaptable both for business and for core performance oriented applications	
Typical applications: any for enabled (high end) devices		

5 Common problems and how to solve them

What follows is a detailed list of troubles the developer can run into while developing applications for heterogeneous clients. A development team should consider every aspect listed when building the architecture of the service.

As a note we remember that what follows are notes from a programmer perspective. Many problems listed are related to developing applications, particularly using Java Micro Edition. Many of such troubles are less important or non-existent for what we called information services (mainly because they are handled by the system).

The following table presents an assessment of the impact of each of the issues outlined below for thin clients and fat clients.

Problem	Thin client	Fat client
User interface and graphics	Heavy issue	Medium issue
Input management	Heavy Issue	Heavy Issue
Network support	N/A	Medium issue
Computing power	N/A	Heavy Issue
Memory and storage	N/A	Heavy Issue
Security	Moderate Issue	Moderate Issue
Hardware access	Heavy Issue	Moderate Issue
Debug: how good are emulators?	Moderate Issue	Moderate Issue
Developing position-aware applications	N/A	Moderate Issue
Databases	Moderate Issue	Heavy Issue

Each topic will be articulately discussed in the following paragraphs. At the end of each paragraph we will summarize in a table, as quick-reference guideline:

- The scope of the problem [**PROBLEM** field, in green]
- The things that developers should keep in mind for designing their solution to the problem, i.e. potential or commonly faced issues [**ADVICE** field, in red]
- Our suggested solution; this is a usually empirical, previously experimented approach that can therefore be duplicated with relatively minimal issues [**SUGGESTED APPROACH** field, in blue]

5.1 User interface and graphics

Following sections summarize problems related to the user interface. Section 5.1.1 applies to the thin client approach while section 5.1.2 is specific for fat clients application developed using J2ME. Also section 5.1.3 is specific for J2ME but the described considerations can be easily ported to other fat client approaches (native code or BREW).

5.1.1 Downsizing desktop screens

Representation, aka User Interface, is the enemy. In fact, we cannot easily control representation on different devices because of their different screen-sizes, color and input mechanism. We all have seen that some web pages, great-looking on a big desktop web-browser aren't at all optimized for small monitors. And on the mobile things get even worse. But for informations services we are lucky since our data don't depend on screen size (mostly). As with HTML we have to be aware that the content will be displayed with client's preferences but the content itself is fully preserved.

A specific architecture could be useful in order to optimize data transfers between client and server: the deck/card solution used by WML should be used to load the most probably needed data. With HTML all those data can be merged in a single page (i.e. a big form can be split on a mobile on multiple cards while on the desktop client it can be on a single one). Anyway in the near future bandwidth will grow and latency will decrease allowing also less optimized or heavier pages to be usable.

Another important issue to consider in developing information services for the desktop as well as mobiles is the use of "heavy-weighted" pages. Today in the web there are more and more pages that rely on applets and/or components (i.e. Macromedia Flash, as on the WISE web-site). All those technologies are mostly unusable for light mobiles. Therefore compatibility means a probably less-appealing desktop solution.

Even big and/or animated images can be a problem: mobile screens can be at the most 120x240 or so, while a common display usually achieves a 96x64 resolution; this means that bigger images can hardly be displayed (not to mention the problem of the frame-rate for animations).

All those considerations depend on the real target and may change in a few years: for example some light PDA already have 400MHz processors and a relatively "big" screen. More, their integrated browser can process applets (at least those not using Java 2 features), as well as (for Windows Mobile–based clients) ActiveX.

5.1.2 J2ME-specific issues

The first issue that must be considered handling heterogeneous clients is the graphical front-end. Mobile clients are equipped with many different sized screens; this means that an application must handle with care what is displayable and what's not on each device. Use of bitmaps graphics can be difficult to sustain due to limited stretching capabilities and to the poor quality of results. This problem has always existed even on common desktop programs, but with the introduction of scroll-bars and some other graphical tricks it seemed less important. But on mobile clients scroll-bars aren't often a good solution because of difficulties of handling them (see later, under Input management).

Under J2ME it's available a kind of high-level API (the *javax.microedition.lcdui* package) that is a common layer for using buttons (menu-driven commands), menus, edit-boxes and the like. But the graphical quality of the output on the vast majority of clients is at most scarce. Programmers have no control on the display and this leads to an uncomfortable look, on which commercial applications can hardly be based.

A lot of effort has been spent to bypass this problem:

- LWT (Lightweight Windowing Toolkit) has been developed by Motorola;
- OWT (Open Windowing Toolkit) is an open source windowing project;

- kAWT is another commercial (and probably the most featured) windowing toolkit.

All those toolkits try to compensate the lack of AWT in J2ME while none recreates the original AWT (but kAWT exists also for desktops).

As we will show in section 5.1.4 later the above-mentioned toolkits are not well suited for every mobile client and are mostly targeted to PDAs or PDA-like.

Another issue concerning graphics is the difference between displays: on clients colour depth ranges from two to thousands of colours. More, screen background and foreground colours can vary widely from grey on green, dark grey on light grey and so on, thus producing different effects on different screens. Readability, particularly for pictures, can thus be perfect on some screens and completely compromised on others.

Colour customisation is, in practice, the only possible solution. It can be actively (changing resources in the code) or automatically (using some functions to check display capabilities) performed but, in any case, a preview of target devices is advisable. New clients development can lead to new releases of the code and this also lead to a continuous rework of the code (although a well-planned project can minimize this).

5.1.3 Customized deployment

Hardware limits can be a constraint for every application forcing programmers to write a program using only least common denominator attributes of every device. For example using million color graphics, while useful on a color screen, can prove meaningless on a b/w device (that is unreadable graphic and huge space loss). Big bitmaps can also be used just on some device without providing also some scrolling capability (that is not always what we want).

A common solution could be to share code without sharing resources. While Java doesn't allow conditional compilation, it is still possible to use different resource classes for every different device (or, at least, for classes of devices). So, for example, a bitmap name for a million-color display can be different to that for a b/w one. Resource classes can be selected at runtime (so that every distributable package has every possible configuration loadable), or hard-coded on a case-by-case basis. The second solution can be annoying for deployment but doesn't force even harder constraints in terms of physical storage memory on devices. Build tools as Ant can prove useful.

PROBLEM	Mobile devices display can vary in size and color depth. WML and the like do not allow a satisfactory control on content representation and are just suitable to some kind of services. Programming languages are more powerful but an application must nevertheless take in account the fact that it may be run on clients provided of very different display capabilities.
ADVICE	Keep in mind that the mobile devices displays are *small*. Size can vary from 96x64 pixel or less to, at the most, 120x240 pixel or a little more
ADVICE	Keep in mind that the mobile network may have a *limited bandwidth* (specially GSM connections) so big pictures can also take an unacceptable time to be loaded.
ADVICE	Keep in mind that the mobile devices displays *can have a little color depth* (even 2 colors). For screens supporting a very low number of colors (typically 2 or 4), all the graphics in order to look nice should be completely redrawn (color reduction algorithms of graphics programs do not produce satisfactory results).
SUGGESTED APPROACH	If you need to control the content representation or a customized GUI (not just a set of forms), the only solution is to use a programming language (Java/C++) instead of a markup language (even with scripting capabilities) as WML. It is also advisable to use an open solution as the J2ME or Symbian OS frameworks.
SUGGESTED APPROACH	Use small pictures. To have better chances that your pictures are dithered in a satisfactory way on low-color-depth devices colors, you may use contrasting colors and remark the picture edges in black (applies to logo and graphics, not to

	photo-realistic items of course).
SUGGESTED APPROACH	When developing in Java for a PDA or PDA-like devices, the use of a windowing toolkit is highly recommended if you need to show a good looking GUI.

5.1.4 Caching of pictures

In 5.1.3 we've talked about the solution of having the resources loaded at runtime (so that deploying the application without the resources will save space in the storage memory) but if we have to load a lot of pictures at runtime we will quickly fill up the memory. A first solution is building a caching system to store the runtime-loaded pictures so that when the application needs a picture it will simply ask for the picture to the cache, the cache will provide the requested picture (eventually requesting it to a server).

A better approach is to provide a caching system able to load the required graphics from the local resources or from the server. With this approach the developer can choose which pictures could be stored in the local resources and which have to be loaded from an external server, so there could be various versions of the application which differ in requested physical storage memory.

PROBLEM	Loading all pictures from a server at runtime will save space in the mobile device's storage memory, but it will use a larger amount of volatile memory.
ADVICE	Keep in mind that the mobile devices have different displays with different size and color depth.
ADVICE	Keep in mind that the mobile devices have a different amounts of storage and volatile memory.
ADVICE	Keep in mind that the mobile network may have a *limited bandwidth* (specially GSM connections) so big pictures can also take an unacceptable time to be loaded.
SUGGESTED APPROACH	If you need to include a large number of pictures in your application it is preferable to load them from a server, so the server will send pictures that fit better with the device display.
SUGGESTED APPROACH	To prevent unnecessary network traffic and to save space in the memory of the mobile devices use a caching system (the most used pictures will be requested few times and the least used pictures will not occupy the memory of the device when not needed).

5.2 Input management

Differences in input management can be harder to handle because of the variety of input devices available. But, in practice, those devices can be generally classified as full keyboards, keypads and pointing devices (such as mouses, tablets and joysticks/joy pads). Providing support for each one of those three different classes of pointing devices means covering almost every possibility.

As everybody knows inputting data into a small mobile phone isn't as easy as on desktop PCs. Just a simple SMS can take a relatively long time to be edited on a cell phone. The problem is the limited amount of space available for keyboard and the absence of a mouse. Current solutions comprise software-enhanced interfaces (such as T9-like technology for writing on phones) and touch-screens. A touch screen can be a definitive solution allowing wider screen and direct mouse-like input. Handwriting recognizing techniques have already been developed and a trained user can be as fast as with common paper handwriting. The only problem lies in the inability to use the device with a single hand and the lack of robustness of touch-screens compared to keypads.

Relying on touch screens, programmers can make "window" interfaces with buttons and all other common controls we are accustomed to on desktops. But those controls are not of practical use when having just keypads, just as using a windowed desktop only with a keyboard (that is by far easier than a keypad).

Up until today the majority of mobile clients don't use touch screens so a common application should not involve pushbuttons, combo-box and scrollbars. Today's mobile phones interface should be used as guides for graphical design. Also old-style arcade games may be used as references (but remember they had a big screen).

More exotic input solutions involve voice. While a huge number of phones support this method for making calls, it is not so common on other devices and it's not at all fail-proof. A noisy environment can be a hard obstacle to by-pass. So applications cannot rely on this input method.

A wide deployable application should be able to use the most limited set of buttons. In the near future J2ME application might be deployed on watches or even on wearables. On those devices it's likely that a very limited keypad will be present (maybe with just three or four buttons), so a very easy-to-use interface should be developed. Technologies like T9 can be a big help but are not useful for, in example, list selection.

If even the most limited handset must be taken in consideration, that doesn't mean that a full-size keyboard shouldn't be of any help. A major mapping of commands can greatly improve usability. For example, while on some devices certain preferences can be selected only navigating on menu and submenus, a shortcut could be placed for clients having full-keyboards. But development should follow a ground-up approach (from the most limited device to the most features-full) avoiding, as we've said in 5.1.2, common graphic interfaces based on buttons, list boxes, etc. that are uneasy to use without pointing devices. This is the only guarantee one can have to develop a fully usable application even on the most limited device.

Providing every possible input handling on every version of the program should be the preferred choice because, even if on most devices many of the possibilities offered will not be used the overhead in term of coding is very low. On the other hand, customizing input handling for every device can easily be a nightmare for the programmer.

PROBLEM	Mobile devices input methods can vary widely. When developing in WML you don't have to worry about this but when using a programming language (specially J2ME) you do.
ADVICE	Keep in mind that the mobile devices displays are *small*, so even text messages may not fit entirely into them. Your GUI shall have scrolling support so if the information is bigger than the (variable size) display it can be fully accessed as well by the user.
ADVICE	Low-end devices usually have a keyboard (at the most some kind of jog for navigation), high-end devices usually use touch screen. Design your GUI in a way that is handy for both approaches (and handle both approaches in the code as well).
ADVICE	Both with keyboard or virtual keyboard (on touch screen enabled devices) inputting text is much harder than on desktops. The amount of text input shall be carefully limited.
ADVICE	Voice input may be not widely supported for open solutions. In general, relying on it is discouraged.
SUGGESTED APPROACH	Use an icon-based GUI. Icons can be clicked on touch-screen displays and navigated and selected with a keyboard (imagine using Microsoft Windows without a mouse). Support both kinds of input.
SUGGESTED APPROACH	Use scrollable screens and wherever it is possible resizable screens (i.e. the content adapts to the screen size. An example is Wise Pilot2 game screen).

5.3 Network support

Protocols

J2ME specification states that, at least in the first release, only HTTP has to be supported by all producers. That is because HTTP is independent from the underlying protocols while mobile networks run on many different protocols (at least if we use a worldwide point of view). So the common Internet TCP/IP layer is not mandatory. But many producers support also TCP/IP as well as UDP/IP. This last protocol, when supported, is by far the fastest protocol (maybe also because of network packet-based infrastructure) and, despite its unreliability, is a must-choice for real-time applications. In this context real-time definition may vary depending on network speed and, until now, any wireless solution is many times slower than a wired one.

But this area is an evolving one and things are likely to change in the near future. UMTS, for example, can lead to a significant performance upgrade.

PROBLEM	Transport protocols other than HTTP are not widely supported. You may have, however, the need to use other lighter transport protocols because of the high latency and low bandwidth of the wireless networks.
ADVICE	If you use WML, you don't have to worry about these issues. You simply don't have any choice.
SUGGESTED APPROACH	HTTP is the most widely supported protocol. Use HTTP connections wherever it is possible.
SUGGESTED APPROACH	Other transport protocols may be very useful but not supported by the implementation or not well handled by the carriers' networks. In general it is advisable to push the choice of the transport protocol at the lowest layer of the application. Application's logic should not be influenced (or not rely too much on) the transport mechanism.

5.4 Computing power

It seems that Moore's law still applies also to wireless devices. With that in mind programmers can develop applications thinking they will have in few years enough computing power for almost every application as they do now for desktops. The only limit to this assumption is that wireless devices have big power constraints that can dramatically limit their growth. Even high-end PDA's batteries last nothing but a few hours before needing recharge. Mobile phones need to work for a far longer time, thus it is possible that their computing power will be limited by energy constraints.

Advances in batteries technology can obviously dramatically change this assumption.

Java is recognized as a CPU-hungry technology. Even if with J2ME a lot of features have been discarded and others have been deeply optimized, especially regarding the Java Virtual Machine, compatibility among devices has this big price to be paid: performance. Everything in a Java program should be optimized not to waste resources, so that Java speed-up in building code can be lost in optimization (which is not commonly necessary using C or other compiled languages).

PROBLEM	Computing power of wireless devices is low, and battery consumption is a problem.
ADVICE	Be aware that the performance of J2ME is, as for Java, quite low. It is a CPU and memory hungry technology, so its performances can be seriously affected by the lack of resources on the device.
SUGGESTED APPROACH	If you are using J2ME, try to optimise the code as much as possible (i.e. instantiate object only when they are really needed, do not rely too much on the efficiency of the garbage collector, etc…). J2ME programming is more similar to C approach than to Java itself.

5.5 Memory and storage

Small devices have a very limited memory capacity. So code must be optimized also for space. This is true particularly because Java stores names of classes, methods and variables directly in the bytecode. A technique known as "Obfuscation" can reduce the size of the bytecode by changing, when compilation takes place, every name in a shorter string based on statistical analysis. For example if the method called "getVariableNamedGoofy()" is the most frequently used method of a class, it will be renamed as "a" thus effectively reducing bytecode size. This approach (originally used for protecting code from decompilation) can lead to significant code reduction.

Another issue that must be considered is that stored images take a lot of overhead that can be shared if all the pictures are instead packed in a single file: compressed images carry an header with information on size, color depth and so on; if this header is shared, we have a considerable saving of space. Moreover, Huffman-like compression (used for image data) works better with big quantities of data. Using many small pictures may mean having the header same in size as the image data, so using a single image instead (and then clipping it when needed) can reduce roughly the storage size by half!

PROBLEM	The storage capacity of mobile devices is usually very low. Code and resources must be optimized for space.
ADVICE	If you don't have bandwidth problems (or you use WML), you can download images from a remote URL. Else, images have to be hardcoded in the application.
SUGGESTED APPROACH	If you are using J2ME, use a code obfuscator to reduce the size of bytecode. This also helps in making the bytecode harder to be reverse-engineered.
SUGGESTED APPROACH	If you are using J2ME, pack all the images in a single file (or in a few files) so that their storage occupation is the lowest (just one header, and more efficient compression results). You pay this with bigger occupation of run-time memory when you have the need of splitting the big file into different pictures, but in general the suggested approach is more efficient.

5.6 Security issues

Limited computing power as well as mobility have important consequences on security.

Many Internet services rely on privacy and secrecy. While communication between mobile terminals and the network is generally considered secure from third party interception, that doesn't mean that the network operator cannot easily read information sent. More, when those information exit from the private mobile network and go through the Internet, they are by default sent in a unencrypted manner.

For those reasons the entire digital communication must be considered insecure.

In the WAP environment WTLS (Wireless Transport Layer Security) has become a standard for securing communications. Security in the WAP architecture enables services to be extended over potentially mobile networks while also preserving the integrity of user data. The denial of service is also prevented. The wireless mobile networks set many requirements to the security layer. The existing Internet secure protocols cannot be used in mobile networks without adaptation.

One of the most important requirement is to support low data transfer rates. For instance, the SMS as a bearer can be as slow as 100 bit/s. The amount of overhead must be kept as small as possible because of the low bandwidth. Compared with the industry-standard Transport Layer Security (TLS), formerly known as Secure Socket Layer (SSL), datagram transport layer must also be supported because of the nature of the wireless mobile network. The protocol should handle lost, duplicated, and out-of-order datagrams without breaking the connection state.

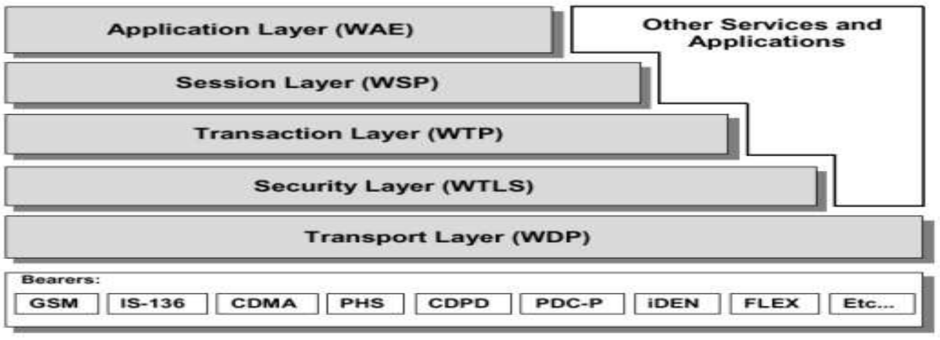

Figure 5-1 WAP Protocol Stack

But while most modern mobile terminals support WTLS on WAP, this protocol cannot be used from Java. No secure service can be provided using standard libraries. Limited processing power and the Java language's performance discourage programmers to set-up a security layer for their applications. A “simple” MD-5 calculation (for enabling integrity check) on each network packet could be overwhelming for a mobile processor. While some libraries are available (i.e. *bouncycastle,* which is also open-source) they are to be intended for use just with high-end Java-enabled PDA.

Obviously consequences affect the type of service that can be provided without reducing the number of potential customers.

Recently Motorola and Sun have standardized Bluetooth API for J2ME access. Security support within that API should be investigated.

PROBLEM	Secure transactions support requires computational speed and bandwidth availability.
ADVICE	In the WAP environment WTLS (Wireless Transport Layer Security) has become a standard for securing communications. So secure communications are possible.
SUGGESTED APPROACH	If you need secure communications, design your application with the WAP architecture. Implementing the traditional encryption algorithms on mobile devices processors can be unfeasible, so be aware of it if you choose to develop with a programming language.

5.7 Hardware access

J2ME provides no support for JNI. This means that no call can be made from Java to the underlying operating systems' APIs. Programmers can make no native extensions. This approach has been chosen in order to simplify support in the VM and not to force companies to leave a possible “hole” in their Java implementations. Even hackers have a difficult time trying to reverse engineer the C-code of the operating system.

But this also means that programmers have no access to the underlying hardware (driver) layer and that any extension could be in a few years surpassed by a new technology or standardized API released by Sun.

But basic functions implemented in J2ME aren't really enough for many application, thus many extensions have already been developed by some big industries.

Particularly important are the Siemens Game API, a set of classes developed to enhance graphics and audio support as well as phone utilities (like phone calls or SMS), file system and network access; but the most used one is the iAppli API (available only on phones for the Japanese market) that greatly enhances graphic capabilities.

Programmers who want to develop code for heterogeneous clients should port those APIs, or avoid using them. This means that a common set of API is something that matters.

PROBLEM	Using device-specific features can be done through Manufactures' provided APIs, where available.
SUGGESTED APPROACH	Try to avoid the use of manufacturers' provided APIs, unless you have a very

	specific target. Heterogeneous clients means first of all different manufacturers!
SUGGESTED APPROACH	In J2ME, keep to the standards (CLDC+MIDP), so you can say that your application complies with a widely supported standard. That's the best you can do to ensure that your application successfully hits the most of platforms.

5.8 Debug: how good are emulators?

There is a dangerous issue that arises at the end of heterogeneous clients software development: how much emulators that are commonly used during the developing phase are compliant with real devices. The fact is that, despite programmers' efforts, no emulator is 100% compliant with the device that emulates. It could be just that commonly on emulators applications run faster than on real devices, it could be a difference in memory behaviour or it could be a less traceable strange attitude in displaying graphics or in handling threads. The fact is that such problems are by definition unpredictable and commonly arise at the final stage. When tracked they can lead to a major problem, forcing developers even to rethink big parts of the application.

The only feasible solution here is to get, in the very earliest stage of the developing process, real devices for which the application is being developed (or, at least, those critical for certifying it) and to test frequently on them.

Another issue in deploying the application in real devices is that no easy debugging is possible. Some manufactures offer some "live" debugging capabilities, but the majority still lack this feature. That means that debugging on real devices should be "hand-made".

PROBLEM	Device emulators are often not compliant with real devices.
SUGGESTED APPROACH	Try to get real devices for which the application is being developed (or, at least, those critical for certifying it) and to test frequently on them. This might be necessary even in the earliest stage of the developing process, because a Manufacturer claimed feature could be, as a matter of fact, unavailable on the real device.
SUGGESTED APPROACH	When using emulators, be sure to set the emulated hardware capabilities very close to those of the real device. The results in the emulation can vary greatly.

5.9 Developing position-aware applications

The availability of GPS receivers is in continuous increase. They come in the form of a hardware card that connects to the serial port of the mobile device. Applications can see the GPS interface as a serial port on which a continuous flow of data is output. The raw data produced by the GPS hardware need to be interpreted by the application in order to calculate the device position (latitude and longitude). This requires basic trigonometric calculations.

3G mobile networks allow a network-based positioning system; since an UMTS device always listens three base stations at the same time, it is possible to determine with quite a satisfactory precision the position of the device even without a satellite.

PROBLEM	How to use GPS for positioning in an application targeted to a mobile device?
SUGGESTED APPROACH	Use a wireless SDK that allows programming with the Operating System APIs. Keep in mind that your application needs to access the serial port. In J2ME this can be done only using proprietary or at least non standard, platform-dependent API or JNI, Java-Native Interface (though serial support has been standardized on MIDP2.0). In Personal Java this is instead straightforward though you may have the bitter surprise that your Personal Java implementation does not support the

	needed package (javax.comm). Another way may be access the GPS receiver by Bluetooth connectivity. About this there is JSR82 standardized by Motorola but implemented, as far as we know, only on some Nokia Symbian mobile phones). GPS based applications are usually thought for a PDA or Notebook. Such devices generally offer development kits for their Operating Systems (i.e. Symbian OS, Windows Mobile), which allow the programmer to access and use all the system resources.

5.10 Databases

Using a database to store data can dramatically simplify the development of bigger applications and reduce the number of programming errors, thanks to a highly standardized data access model.

However, low-end wireless devices, like mass-market mobile phones, can't support a database. The CLDC/MIDP (or BREW, or the like) technology substantially creates a protected environment in which applications run. Applications can't communicate with other applications on the same device or use directly system resources. The database concept itself is too "big" for a mobile phone.

Remote database access. Data are stored remotely on a database. The client uses a standard interface (i.e. JDBC) to communicate with the remote database and retrieve/upload information. This approach is generally feasible on almost all the platforms, though is rare to find custom JDBC drivers developed for MIDP/CLDC due to the many limits of this platform for this kind of applications.

Embedded database. The mobile device is complex enough to allow some kind of database to run and be accessed locally. This generally implies the usage of a mobile operating system SDK (i.e. Symbian, Linux Embedded, Windows Mobile).

PROBLEM	How to use a real, third party database for data access in an application developed for a limited device?
ADVICE	When developing wireless internet applications targeted to low-end devices, data should be stored remotely rather than locally. For instance, J2ME offers a rather primitive support for local data storage: the RMS. It is byte-oriented, structured data access does not go any further than providing a set of records, and generally the implementations are terribly slow. It is not meant to be used dynamically but just to save a bunch of configuration data (i.e. login access information or the high score in a game). If the application is meant to have a heavy data component and this data has to be shared with other users, then a remote database accessed anyhow is the best solution.
SUGGESTED APPROACH	Using a local database on wireless devices (and for a wireless device here we do not mean a laptop with WI-FI support) is still something challenging. Windows Mobile and Linux Embedded are some of the few embedded Oss that offer the full support to connect to a database and communicate with it. The required APIs are part of the SDK. Consider using such systems if your wireless application is complex enough to need a relational or object oriented embedded database for storing data.
SUGGESTED APPROACH	For guidelines about accessing a remote database, see the next point.

PROBLEM	The application needs to access remote data stored in a database.
SUGGESTED APPROACH	Database drivers are often written with a mobile OS SDK. "Sandbox" technologies (i.e. J2ME, BREW, etc…) are generally too limited for that. However, there exist some minimal JDBC drivers written in J2ME. To develop database drivers on the mobile client, Personal Java or C/C++ in an OS SDK is preferable.

6 Related work

Some w3c working groups (mainly Device Independence WC, refer to http://www.w3.org/2001/di/) and some other organizations like www.uiml.org, are working on the concept of a device independent presentation of data for the web.

Briefly the UIML group is designing rules for creating uiml (xml) parsers for enabling representation in many areas (from Java to VoiceXML to WML) while Device Independence WC is working on the more general concept of device independence for web that enables web content to be delivered to any kind of devices.

In practice one can say that the UIML group is working on a kind of adaptation layer for data in order to deliver them to many devices. And the adaptation layer is an idea from Device Independence WG that sees it as a needed part of the whole process.

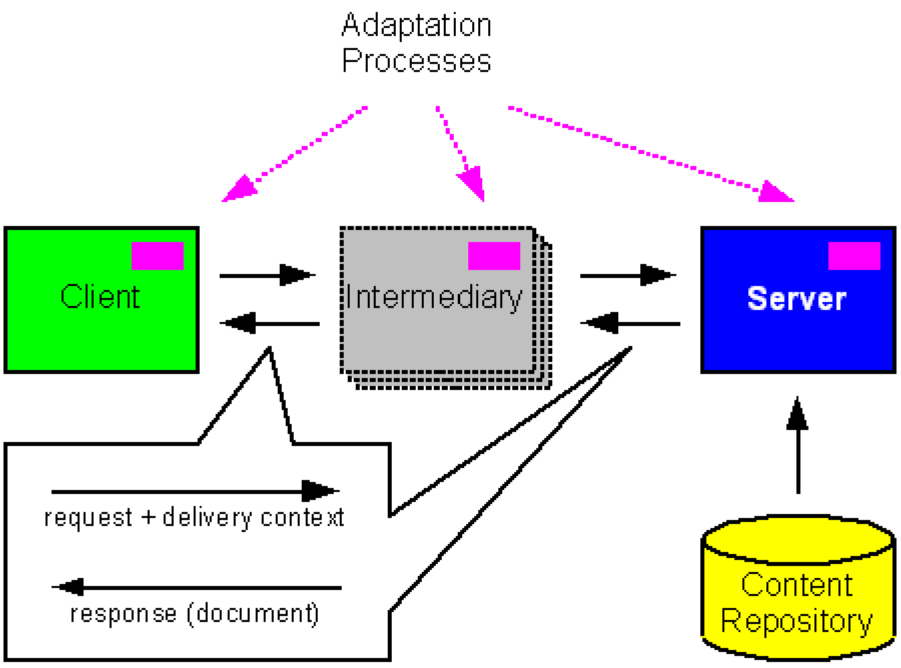

Figure 6-1: from w3c Device Independence Principles

What follows is a concept demonstration of uiml:

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC
        "-//UIT//DTD UIML 2.0 Draft//EN"
        "UIML2_0d.dtd">
<uiml>
 <interface>
   <structure>
```

```
<part name="tabledeck" class="Wml">
   <part name="testcard" class="Card">
      <!-- Sub Menu -->
     <part name="testp" class="P">

      <part name="header" class="RichText">
        <style>
          <property name="content">Table of Images</property>
        </style>
      </part>

      <part name="testtable" class="Table">
        <style>
          <property name="align">center</property>
          <property name="columns">2</property>
        </style>


        <!-- First table row -->

        <part name="row1" class="TR">
           <part name="r1c1" class="TD">
              <part name="r1c1Cont" class="RichText">
                 <style>
                   <property name="content">Floppy</property>
                 </style>
              </part>
           </part>
           <part name="r1c2" class="TD">
              <part name="r1c2Cont" class="Img">
                 <style>
                   <property name="alt">Floppy</property>
                   <property name="src">Icons/flop.bmp</property>
                 </style>
              </part>
           </part>
        </part>

        <!-- Second table row -->

        <part name="row2" class="TR">
           <part name="r2c1" class="TD">
              <part name="r2c1Cont" class="RichText">
                 <style>
                   <property name="content">Clock</property>
                 </style>
              </part>
           </part>
           <part name="r2c2" class="TD">
             <part name="r1c2Cont" class="Img">
                 <style>
                   <property name="alt">Clock</property>
                   <property name="src">Icons/clock.bmp</property>
                 </style>
              </part>
           </part>
        </part>

        <!-- Third table row -->

        <part name="row2" class="TR">
           <part name="r3c1" class="TD">
              <part name="r3c1Cont" class="RichText">
                 <style>
                   <property name="content">Email</property>
                 </style>
              </part>
           </part>
           <part name="r3c2" class="TD">
             <part name="r1c2Cont" class="Img">
                 <style>
                   <property name="alt">Email</property>
                   <property name="src">Icons/mail.bmp</property>
                 </style>
              </part>
           </part>
        </part>

     </part>
```

```
        </part>
      </part>
    </part>
  </structure>
 </interface>
</uiml>
```

The above code can be automatically translated in, say, WML obtaining:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
 "http://www.wapforum.org/DTD/wml_1.1.xml">

<!--
  This .wml file was automatically produced from a User Interface Markup
  Language (UIML) file by the UIML-to-WML renderer version 0.3c
  from Universal Interface Technologies Inc.

  See www.universalit.com and www.uiml.org for more information

  Source .uiml file:  Table.uiml
  Translation  date:  Mon Sep 11 19:46:28 EDT 2000
-->
<wml>
  <card id="testcard">
      <p>
          Table of Images
          <table align="center" columns="2">
            <tr>
              <td>
                Floppy
              </td>
              <td>
                <img alt="Floppy" src="Icons/flop.bmp" />
              </td>
            </tr>
            <tr>
              <td>
                Clock
              </td>
              <td>
                <img alt="Clock" src="Icons/clock.bmp" />
              </td>
            </tr>
            <tr>
              <td>
                Email
              </td>
              <td>
                <img alt="Email" src="Icons/mail.bmp" />
              </td>
            </tr>
          </table>
      </p>
  </card>
</wml>
```

And this lead to a representation on a mobile browser like this:

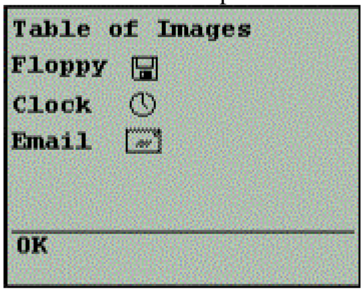
Table of Images
Floppy
Clock
Email
OK

The important fact is that a common representation (uiml) can be automatically rendered on different devices without any further human customization.

However, there can be some limits. The same class on different devices (where, for class, in uiml we intend the object of the representation, let's say a button) can be rendered very differently on different clients. The result can be poorly looking and barely usable. The creator has no control (or just a very limited one) over the final representation of the data and, while this implies a lot less work for developers, drawbacks also exist.

In the visual era, appearance counts much, particularly in some area like, for example, entertainment. For customers to buy, everybody knows that a beautiful "box" is essential.

And this common representation risks to be limited by capabilities available (and implemented) on every device.

7 Conclusion

Developing wireless applications for portable devices is an order of magnitude more difficult than developing the same services for standard desktop computers. The lack of resources, both hardware (memory, processing power) and perceptual (screen size, input devices), the lack of tools and standardization caused by the great variety of devices available and by the manufacturers' race to acquire the highest portion of the market at all costs, all make it very difficult to create a truly portable service in a short time.

Therefore, a lot of care must be taken when approaching new software development for this class of devices in order to make sure that the costs of development and maintenance are bearable and still leave a margin for profit. Unlike many other situations in software development, this is mostly a matter that needs to be tackled from a technical perspective, during the whole lifecycle of the application.

For this reason in this document we showed several development approaches, including tools, languages and platforms for both the Fat Client and Thin Client architectures, outlining pros and cons for them from a technical and economic perspective.

We then outlined the common issues faced by developers when developing services in terms of user experience (User interface and graphics, Input management), application functionality (Network support, Computing power, Memory and storage, Security, Hardware access) and development (tools and emulators), showing some of the best practices both discovered during the course of the WISE project and from other sources.

A lot of work still needs to be done in this area, and many standardization efforts are ongoing, but finding a unique approach for successful software development over wireless devices is still a research area that needs much exploration.

References

1. Boggio, D., Forchino, F., Ricciardi, S., "Pilot 2 Requirements for WISE Iteration I". Requirements Specification for Pilot 2, WP4, Deliverable D11 (part A), IST-2000-30028 Project WISE (Draft), Version 00.02, Mar. 2002.

2. Forchino F., Negro Ponzi M., Tiella R., "Architecture for Pilot 2", Deliverable D11 (part B), IST-2000-30028 Project WISE (Draft), Version 01.04, 30, Sep. 2002.

3. http://java.sun.com/j2me/ for J2ME specifications and API

4. Qusay H. Mahmoud, Learning Wireless Java, O'Reilly, January 2002, ISBN 0-596-00243-2

5. David Fox, Roman Verhovsek, Micro Java Game Development, Addison Wesley, April 2002, ISBN 0-672-32342-7

6. http://www.w3.org/TR/2001/WD-di-princ-20010918: Device Independence Principles, Working Draft, 19 September 2001

7. http://www.wapforum.org

8. http://www.w3schools.com/wap/

9. http://www.uiml.org/ and http://www.harmonia.com/resources/tutorials/index.htm

10. http://www.qualcomm.com/brew

11. http://www.mobilegprs.com and http://www.3gamericas.org and http://www.gsmworld.com

12. http://www.nttdocomo.com and http://developpeur.journaldunet.com

13. http://www.gsacom.com/

14. Windows Mobile: http://www.microsoft.com/windowsmobile/default.mspx

15. Symbian OS: http://www.symbian.com/

Copyright Notice