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

<table>
<thead>
<tr>
<th>Vers.</th>
<th>Date</th>
<th>Authors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2 Oct 2002</td>
<td>F. Forchino, M. Negro Ponzi</td>
<td>Initial description</td>
</tr>
<tr>
<td>1.1</td>
<td>6 Nov 2002</td>
<td>M. Negro Ponzi</td>
<td>Devices summary table, introduction to Qualcomm’s BREW, merge of chapter 6 into 5 and 7.</td>
</tr>
<tr>
<td>1.2</td>
<td>20 Dec 2002</td>
<td>F. Forchino, M. Negro Ponzi</td>
<td>Update considering Sodalia’s review.</td>
</tr>
<tr>
<td>2.0</td>
<td>14 May 2003</td>
<td>F. Forchino, M. Negro Ponzi</td>
<td>Re-organisation of the document in form of guideline</td>
</tr>
<tr>
<td>2.1</td>
<td>08 Sept 2003</td>
<td>M. Negro Ponzi</td>
<td>Adjustements</td>
</tr>
<tr>
<td>2.2</td>
<td>21 Oct 2003</td>
<td>P. Falco</td>
<td>Integrated comments from partners’ review</td>
</tr>
</tbody>
</table>
Management of heterogeneous clients
Deliverable ID: D3

Contents

1 Introduction ....................................................................................................................... 5
2 Classes of Devices ........................................................................................................... 5
3 Service architecture ........................................................................................................ 8
  3.1 Thin clients .................................................................................................................. 8
  3.1.1 HTML ................................................................................................................... 9
  3.1.2 WML and similar technologies .............................................................................. 9
  3.2 Fat clients .................................................................................................................... 9
  3.2.1 Available Java 2 Micro Edition technologies ......................................................... 10
  3.2.2 BREW technology ................................................................................................. 11
  3.2.3 Native development ............................................................................................... 12
  3.3 Summarizing architectures: advantages and disadvantages ...................................... 12
4 Common problems and how to solve them.................................................................... 13
  4.1 User interface and graphics ....................................................................................... 14
  4.1.1 Downsizing desktop screens .................................................................................. 14
  4.1.2 J2ME-specific issues ........................................................................................... 14
  4.1.3 Customized deployment ....................................................................................... 15
  4.2 Input management ...................................................................................................... 16
  4.3 Network support ........................................................................................................ 17
  4.3.1 Protocols ............................................................................................................... 17
  4.3.2 Standards ............................................................................................................ 18
  4.4 Computing power ....................................................................................................... 18
  4.5 Memory and storage ................................................................................................. 18
  4.6 Security issues ........................................................................................................... 19
  4.7 Hardware access ....................................................................................................... 20
  4.8 Debug: how good are emulators? .............................................................................. 20
5 Related work ................................................................................................................... 21
6 Conclusion ...................................................................................................................... 24

© Copyright WISE Consortium
Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WML</td>
<td>Wireless Markup Language</td>
</tr>
<tr>
<td>AWT</td>
<td>Abstract Windowing Toolkit</td>
</tr>
<tr>
<td>BREW</td>
<td>Binary Runtime Environment for Wireless</td>
</tr>
<tr>
<td>CDC</td>
<td>Connected Device Configuration</td>
</tr>
<tr>
<td>CDLC</td>
<td>Connected Device Limited Configuration</td>
</tr>
<tr>
<td>J2ME</td>
<td>Java 2 Micro Edition</td>
</tr>
<tr>
<td>JNI</td>
<td>Java Native Interface</td>
</tr>
<tr>
<td>MIDP</td>
<td>Mobile Information Device Profile</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>WAP</td>
<td>Wireless Application Protocol</td>
</tr>
<tr>
<td>WMLScript</td>
<td>ECMA-62 based scripting language for WML</td>
</tr>
<tr>
<td>WTLS</td>
<td>Wireless Transport Layer Security</td>
</tr>
<tr>
<td>CDMA</td>
<td>Code Division Multiple Access</td>
</tr>
</tbody>
</table>
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.
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).
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](image)

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 3.2.2).

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.

<table>
<thead>
<tr>
<th>Category</th>
<th>Screen size and capabilities</th>
<th>Memory</th>
<th>Cpu</th>
<th>Connectivity</th>
<th>OS/Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktops/Notebooks</td>
<td>640x480 to 1600x1200 and more (screen from 10 to 24 inches), million color graphics</td>
<td>Up to some GB RAM and hundreds of GB of mass storage</td>
<td>Currently from 1 to about 3 GHz</td>
<td>Wired or wireless</td>
<td>Any high-end: Windows, Linux/Unix, MacOS</td>
</tr>
<tr>
<td>TabletPC</td>
<td>800x600 to 1280x1024 screen from 10 to 15 inches, million color graphics</td>
<td>Up to few GB RAM and tens of GB of mass storage</td>
<td>Around 1 GHz (extra low power processors)</td>
<td>Wired or wireless</td>
<td>Customized version of those for desktops</td>
</tr>
<tr>
<td>PDA</td>
<td>About 240x320, thousand color graphic</td>
<td>Up to some hundreds MB of shared RAM/mass storage memory.</td>
<td>Intel StrongARM and XScale, Motorola DragonBall, MIPS and other ARM-based processors</td>
<td>Wired, wired through a host PC, wireless</td>
<td>Scaled-down version of those for desktops or dedicated ones.</td>
</tr>
</tbody>
</table>
In the following paragraph we will describe the services categories and what is the most suitable software architecture for each of them.

### 3 Service architecture

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.

<table>
<thead>
<tr>
<th>Service group</th>
<th>Poorly interactive</th>
<th>Highly interactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information</td>
<td>Mostly static</td>
<td>Highly dynamic</td>
</tr>
<tr>
<td>Information</td>
<td>Refreshed rarely</td>
<td>Refreshed frequently or in real time</td>
</tr>
<tr>
<td>freshness</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data exchange</td>
<td>Slow exchange of small packets of data</td>
<td>Frequent or continuous exchange of packets of data of any size</td>
</tr>
<tr>
<td>Information display</td>
<td>Mostly textual</td>
<td>Mostly graphical</td>
</tr>
</tbody>
</table>

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 into 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 3.1 will describe the thin client approach while section 3.2 will introduce the fat client approach of both Java Micro Edition and Qualcomm’s BREW proposals.

### 3.1 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.

### 3.1.1 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.

### 3.1.2 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”-based approach that minimizes the activity on the network 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 4.6).

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.

### 3.2 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

---

1 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.
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).

3.2.1 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 “flavors” 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 together. 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:
Management of heterogeneous clients

Deliverable ID: D3

- 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.

3.2.2 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.
3.2.3 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 CE. 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.

3.3 Summarizing architectures: advantages and disadvantages

<table>
<thead>
<tr>
<th>Technology</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thin client based on WAP/WML</td>
<td>Portable over millions of cell phones</td>
<td>Very limited GUI customization capabilities</td>
</tr>
<tr>
<td></td>
<td>Easy to develop for non-programmers</td>
<td>Very limited and slow interaction</td>
</tr>
<tr>
<td></td>
<td>Easy to port from web-applications</td>
<td>Little or no control over representation</td>
</tr>
<tr>
<td></td>
<td>Widely supported by handset manufacturers</td>
<td>Provider support through WAP gateways</td>
</tr>
<tr>
<td></td>
<td>WTLS security protocol supported</td>
<td>Unsupported by manufacturers outside the wireless industry</td>
</tr>
<tr>
<td></td>
<td>No development risk</td>
<td></td>
</tr>
</tbody>
</table>

**Typical applications:** news, financial updates, online address book, WEB-based applications porting

<table>
<thead>
<tr>
<th>Fat client based on proprietary solution</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is possible to develop core applications</td>
<td>A different application has to be developed for every possible device</td>
<td></td>
</tr>
<tr>
<td>Total integration with native operating system: it is possible to use hardware dependant features.</td>
<td>A different application has to be deployed for every possible device</td>
<td></td>
</tr>
<tr>
<td>Full GUI customization</td>
<td>Developers should be trusted by manufacturers (and/or have to pay license fees)</td>
<td></td>
</tr>
<tr>
<td>Fastest access to the network</td>
<td>Usually developed in C/C++ with language idiosyncrasies</td>
<td></td>
</tr>
<tr>
<td>Fastest execution speed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Typical applications:** embedded applications developed for provider-customized devices

<table>
<thead>
<tr>
<th>Fat client based on J2ME</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portable to many new generation high-end phones</td>
<td>Not adaptable to old phones</td>
<td></td>
</tr>
<tr>
<td>Full GUI control</td>
<td>Need for customization for different devices’ displays</td>
<td></td>
</tr>
<tr>
<td>Widely known language</td>
<td>J2ME is a bit different from J2SE and programmers need to get in touch with it</td>
<td></td>
</tr>
<tr>
<td>Strongly supported</td>
<td>Devices specific implementations</td>
<td></td>
</tr>
<tr>
<td>New processor can provide hardware support</td>
<td>Proprietary non-portable APIs</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Fat client based on BREW</th>
<th>Fat client developed using native tools on Symbian OS, Palm OS or Pocket PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core hardware support</td>
<td>Standardized solution with heavy industrial support</td>
</tr>
<tr>
<td>CDMA-only devices (at least until today)</td>
<td>Three different platforms (PocketPC, Palm OS and Symbian OS)</td>
</tr>
<tr>
<td>Proprietary solution (with license fee)</td>
<td>Deep integration with the environment</td>
</tr>
<tr>
<td>Specific language</td>
<td>Open consortium for Symbian OS, open APIs for the others</td>
</tr>
<tr>
<td>No native support</td>
<td>Large background and history for PocketPC and Palm OS</td>
</tr>
<tr>
<td>Dependence from Qualcomm for improvements</td>
<td>Adaptable both for business and for core performance oriented applications</td>
</tr>
</tbody>
</table>

**Typical applications:** committed on-demand applications developed for BREW clients on CDMA networks

**Typical applications:** any for enabled (high end) devices

4 **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.
4.1 User interface and graphics

Following sections summarize problems related to the user interface. Section 4.1.1 applies to the thin client approach while section 4.1.2 is specific for fat clients application developed using J2ME. Also section 4.1.3 is specific for J2ME but the described considerations can be easily ported to other fat client approaches (native code or BREW).

4.1.1 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 informations 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 CE-based clients) ActiveX.

4.1.2 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 4.2 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).

### 4.1.3 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 display 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.

### 4.2 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 4.1.2, 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).

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).

4.3 Network support

4.3.1 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.
4.3.2 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. A partial patch has been provided with GPRS that basically let terminals use more channel while sending and receiving data. Today the performance of a GPRS communication link is comparable with that of an analogue phone line, although still smaller. UMTS’s promise to solve this problem 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.

4.4 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).

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>Computing power of wireless devices is low, and battery consumption is a problem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVICE</td>
<td>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.</td>
</tr>
<tr>
<td>SUGGESTED APPROACH</td>
<td>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.</td>
</tr>
</tbody>
</table>

4.5 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. A similar 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 packed in a single file: think at compressed images carrying an header with information on size, color depth and so on, and
where Huffman-like compression works better with quantities of data. Every byte in some environments can be critical.

| 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. |

### 4.6 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 an 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 4-1 WAP Protocol Stack](image)

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.

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>Secure transactions support requires computational speed and bandwidth availability.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVICE</td>
<td>In the WAP environment WTLS (Wireless Transport Layer Security) has become a standard for securing communications. So secure communications are possible.</td>
</tr>
<tr>
<td>SUGGESTED APPROACH</td>
<td>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.</td>
</tr>
</tbody>
</table>

4.7 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.

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>Using device-specific features can be done through Manufactures’ provided APIs, where available.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUGGESTED APPROACH</td>
<td>Try to avoid the use of manufactures’ provided APIs, unless you have a very specific target. Heterogeneous clients means first of all different manufacturers!</td>
</tr>
<tr>
<td>SUGGESTED APPROACH</td>
<td>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.</td>
</tr>
</tbody>
</table>

4.8 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 Related work**

Some w3c working groups (mainly Device Independence WC, refer to [http://www.w3.org/2001/di/](http://www.w3.org/2001/di/)) and some other organizations like [www.uiml.org](http://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 5-1: from w3c Device Independence Principles**

What follows is a concept demonstration of uiml:

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

Deliverable ID: D3

<!ml>
<!structure>

<!-- Sub Menu -->

<!part name="tabledeck" class="Wml">

<!interface>

<!part name="testcard" class="Card">

<!part name="header" class="RichText">

<!style>

<!property name="content">Table of Images</property>

<!part>

<!part name="testtable" class="Table">

<!style>

<!property name="align">center</property>

<!property name="columns">2</property>

<!part>

<!!-- First table row -->

<!part name="row1" class="TR">

<!part name="r1c1" class="TD">

<!part name="r1c1Cont" class="RichText">

<!style>

<!property name="content">Floppy</property>

<!part>

<!part name="r1c2" class="TD">

<!part name="r1c2Cont" class="Img">

<!style>

<!property name="alt">Floppy</property>

<!property name="src">Icons/flop.bmp</property>

<!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>

<!part>

<!part name="r2c2" class="TD">

<!part name="r2c2Cont" class="Img">

<!style>

<!property name="alt">Clock</property>

<!property name="src">Icons/clock.bmp</property>

<!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>

<!part>

<!part name="r3c2" class="TD">

<!part name="r3c2Cont" class="Img">

<!style>

<!property name="alt">Email</property>

<!property name="src">Icons/mail.bmp</property>

<!part>

© Copyright WISE Consortium
The above code can be automatically translated in, say, WML obtaining:

```xml
<?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 [F]
Clock [C]
Email [E]
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.

6 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.
Management of heterogeneous clients
Deliverable ID: D3

References


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


Copyright Notice

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc.

Many of the designation used by manufacturers and sellers to distinguish their products are claimed as trademarks. Use of a term in this paper should not be regarded as affecting the validity of any trademark or service mark.