ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δυναμική ενσωμάτωση κινητών τερματικών σε σύστημα πολύ-επίπεδης αρχιτεκτονικής με χρήση της πλατφόρμας J2ME και XML τεχνολογιών.

Dynamically integrating mobile clients into a multi-tier architecture system, using the J2ME platform and XML technologies.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτρης Μ. Κυριαζάνος

Επιβλέπων : Γεώργιος Ι. Στασινόπουλος
Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2005

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δυναμική ενσωμάτωση κινητών τερματικών σε σύστημα πολύ-επίπεδης αρχιτεκτονικής με χρήση της πλατφόρμας J2ME και XML τεχνολογιών.

Dynamically integrating mobile clients into a multi-tier architecture system, using the J2ME platform and XML technologies.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτρης Μ. Κυριαζάνος

Επιβλέπων : Γεώργιος Ι. Στασινόπουλος
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Ιουλίου 2005.

....	....	....
Στασινόπουλος Γεώργιος	Θεολόγου Μιχαήλ	Συκάς Ευστάθιος
Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Μήνας Έτος

....

Δημήτρης Μ. Κυριαζάνος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Δημήτρης Μ. Κυριαζάνος. 2005.

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Στους γονείς μου

ΠΕΡΙΛΗΨΗ

Ο σκοπός της διατριβής ήταν η σχεδίαση και ανάπτυξη μιας ασύρματης client εφαρμογής ανταλλαγής δεδομένων μέσω XML, με χρήση της πλατφόρμας Java 2, Micro Edition (J2ME). Η εφαρμογή λειτουργεί βάσει του Mobile Information Device Profile (MIDP), ένα σύνολο από προδιαγεγραμμένες βιβλιοθήκες - APIs (Application Programming Interfaces) που προορίζονται για χρήση από κινητές συσκευές όπως κινητά τηλέφωνα και PDAs (Personal Digital Assistants) και ως εκ τούτου ονομάζεται MIDlet. Το MIDP έχει σχεδιαστεί για χρήση με το Connected Limited Devices Configuration (CLDC), ένα σύνολο προδιαγραφών για συσκευές με περιορισμένα αποθέματα μνήμης, ενέργειας και επεξεργαστικής δύναμης, που ορίστηκε από τους σχεδιαστές της J2ME.

Το συγκεκριμένο MIDlet ήταν εξαρχής προορισμένο να αποτελέσει έναν MIDP client σε μία ήδη αναπτυγμένη πολυεπίπεδη (multi-tier) αρχιτεκτονική (Database – Server – Clients). Το XML Schema –το οποίο χρησιμοποιούταν στην ανταλλαγή δεδομένων και στην επικοινωνία client-server στο υπάρχον σύστημα- έθετε ένα ενδιαφέρον πρόβλημα σχετικά με την ανάγνωση-parsing του XML και το κτίσιμο των διεπιφανειών χρήστη γνωστό ως UI (UI: User Interface) στην J2ME συσκευή.

Τελικώς, η εφαρμογή –η οποία ονομάστηκε XMIDlet- αναπτύχθηκε και υλοποιήθηκε επιτυχώς χωρίς να χρειαστεί κανένα επιπλέον φορτίο ή αλλαγή στην πλευρά του server (όπως π.χ. προσθήκη μιας μονάδας μετατροπής XML). Χρησιμοποιώντας διαθέσιμα στοιχεία GUI (Graphical- Γραφικές UI) των βιβλιοθηκών του MIDP, το MIDlet είναι ικανό να αναπαριστά γραφικά πλήρως και με φιλικό προς το χρήστη τρόπο, την δομή των εισερχόμενων δεδομένων. Επιπλέον το GUI αρχικοποιείται και δημιουργείται δυναμικά (σε αντίθεση με τα συνήθη στατικά GUI) καθώς το εισερχόμενο XML διαβάζεται και κάθε κόμβος της δενδρικής δομής (Document Object Model - DOM) σχετίζεται με το κατάλληλο στοιχείο GUI.

Το XMIDlet χάρη στη σχεδίαση του και στην παραμετροποιησιμότητα της XML είναι ένας client που μπορεί να χρησιμοποιηθεί από πληθώρα συστημάτων και εφαρμογών (generic client). Για λόγους βέβαια παρουσίασης σε αυτή τη διατριβή στήθηκε ένα σύστημα ανταλλαγής και διαχείρισης ιατρικών δεδομένων ασθενών. Η όλο και αυξανόμενη οικειότητα των χρηστών με τα κινητά τηλέφωνα και τα PDAs καθιστά τους MIDP client – και κάθε MIDP εφαρμογή- φορείς νέων υπηρεσιών και ευκολιών προς τους χρήστες. Η ιδιαίτερη εξάρτηση δε που έχει ο ιατρικός κόσμος με τις ασύρματες προσωπικές συσκευές αναδεικνύει το XMIDlet ως μια σημαντικότατη εφαρμογή σε ένα τέτοιο σύστημα.

ABSTRACT

The goal of this thesis was the development of a wireless client application capable of exchanging data via XML, using the Java 2 Platform, Micro Edition (J2ME). This application runs under the Mobile Information Device Profile (MIDP), a set of APIs (Application Programming Interfaces) for use by mobile devices such as cell-phones and PDAs (Personal Digital Assistants) and is therefore also called a MIDlet. The MIDP is designed for use with the Connected Limited Devices Configuration (CLDC), a configuration of devices with low memory budget and processing power, created by the J2ME designers.

Specifically, the MIDlet was intended from the beginning to fit in as a MIDP client in an already developed multi-tier architecture generic system (Database – Server – Clients). The XML Schema -already in use for data exchange and client-server communication in the pre-existent system- set a challenging problem regarding the parsing of XML and UI building (UI: User Interface) in the J2ME device.

Finally, the application (named XMIDlet) was successfully developed without requiring any further load on the server side (such as transformations of the XML). Using standard GUI (Graphical UI) elements of the MIDP APIs, the MIDlet is able to fully represent graphically and in a user-friendly way the structure of incoming data. Furthermore, the GUI is instantiated and created dynamically (rather than the typical static GUI) as the incoming XML is parsed, while each node of the Document Object Model (DOM) of the XML gets related to a proper GUI component.

The XMIDlet as part of a generic system is also a generic client. However and for purposes of presentation, in this thesis the system runs as a medical data exchange system. The usefulness of the MIDP client –and any MIDP application- derives from the ever-increasing familiarity of users with cell-phones and PDAs as well as the ubiquity of these devices in our days. The special dependency that people of the medical world have to their wireless devices pinpoints the significant part that XMIDlet holds in such a system.

TABLE OF CONTENTS

1. Introduction 10

1.1 Preface 10
1.2 The Field of Interest 11
1.3 Goals and Motivations 12
1.4 Organization 12

2. Introducing Java 2 Platform, Micro Edition (J2ME) 14

2.1 Overview of J2ME 14
2.1.1 What Is J2ME? 14
2.1.2 The J2ME Architecture 15
2.1.2.1 Configurations 15
2.1.2.2 Profiles 15
2.1.2.3 Optional Packages 16
2.2 The Connected Limited Device Configuration (CLDC) 18
2.2.1 Examining the CLDC 18
2.2.1.1 Device Specifications 18
2.2.1.2 Functionalities 18
2.2.1.3 What About the Java Virtual Machine? 19
2.2.1.4 The KVM 19
2.2.2 CLDC Libraries 20
2.2.2.1 Classes Derived from Java 2 Standard Edition 20
2.2.2.2 CLDC Specific Classes 21
2.3 The Mobile Information Device Profile (MIDP) 22
2.3.1 Hardware Requirements 22
2.3.2 Class Additions 22
2.3.3 MIDlets 24
2.3.3.1 Java Application Descriptor (JAD) 25
2.3.3.2 Creating MIDlets 25

3. Programming With the MIDP 27

3.1 MIDP GUI 27
3.1.1 The High-Level API 28
3.1.2 The Low-Level API 28
3.1.3 The MIDP GUI Model 28
3.2 MIDP Events 29
3.3 Networking 29
3.3.1 Generic Connections 30
3.3.2 MIDP Connectivity 30
3.3.3 The HTTP Programming Model 31

4. The J2ME Wireless Toolkit 32

4.1 Overview 32

4.1.1 Compilation and Prefabrication 34
4.1.2 Running and Debugging 34
4.1.3 Packaging 35
4.1.4 Packaging Obfuscated Source Code 36
4.2 The KToolBar Application 36
4.2.1 KToolBar Projects 37
4.2.2 The Emulator Application 40

5. J2ME and XML 42

5.1 Introducing XML 42
5.1.1 A Brief History of Markup 43
5.1.2 Extensible Markup Language (XML) 44
5.1.2.1 XML Parsers 46
5.1.3 Using XML 46
5.2 Parsing XML in J2ME 47
5.2.1 Multi-tier System Architecture 48
5.2.2 Parser Roundup 50
5.2.3 Performance Considerations 52

6. The HARP Project 54

6.1 Introduction 54
6.2 Platform Overview 55
6.3 Platform Architecture 56
6.3.1 Runtime Environment 56
6.3.2 Set-up environment 57
6.3.3 Data Structures 59
6.3.3.1 Information structure 59
6.3.3.2 Profile structure 60
6.3.3.3 Data structure 61
6.3.3.4 Associations information structure 61
6.3.3.5 Platform security 62
6.3.4 Platform applicability 63
6.4 The Harp Applet 64

7. XMIDlet, the MIDP Client for HARP 67

7.1 The MIDP client for HARP platform 67
7.1.1 Specifications of XMIDlet 68
7.1.2 XMIDlet networking 69
7.1.3 Parsing XML 69
7.1.3.1 SAX and DOM 69
7.1.3.2 XMIDlet and XML 70
7.1.5 The XMIDlet GUI 72
7.2 Conclusions – the future of MIDP and XMIDlet 81

References 83
Appendix A: CLDC classes inherited from J2SE 84
Appendix B: CLDC specific classes 90

1. Introduction

1.1 Preface

Internet technologies during the last years met a rapid and breathtaking progress. As desktop computers grew more powerful and Internet connections caught up with more acceptable speeds, the quality and amount of services provided through networks increased in an impressive way.

At the mid-1990's however, as mobile communications and handheld devices were already widely accepted as a necessary part in our everyday life, a lot of investment and efforts were placed on developing architectures and protocols in order to nominate devices such as the ubiquitous cell-phone as a tempting way to enjoy services and Internet technologies - once designed strictly for desktop computers. While manufacturers introduced more and more power on our desk, a resource-constrained device attracted specialists as yet another way of offering services to customers.

Small memory footprints, limited CPU power and bandwidth (add expensive to the last one too) are only some of the constraints designers of appropriate protocols have to face. What dazzles a customer on a 19' high-resolution TFT screen is usually out of the question for designers when it comes to the small display of a cell-phone.

One of the most widely known efforts during that period was the Wireless Application Protocol (WAP). Developing wireless applications using WAP technologies is similar to developing Web pages with a markup language (e.g. HTML or XML) because WAP technologies are browser-based. With WAP, it is possible to use Java servlets and JavaServer Pages to generate Wireless Markup Language (WML) pages dynamically. WAP

did not meet the success that investors expected, for various reasons. WAP applications could not easily attract cell phone users, the main market share, due to poor display of most devices at the time and of the narrow capabilities of WAP applications. WAP services were also relatively expensive to use. Users willing to indulge themselves used WAP a few times but even they eventually found that an old-fashioned phone call or Web surfing in an Internet Cafeι was better, faster, cheaper and handier.

Another approach to developing wireless applications is to use the Java 2 Platform, Micro Edition (J2ME) introduced by Sun Microsystems, Inc. at 1999. With J2ME, it is possible to write applications in Java and store them directly on a cell phone. Developers soon realized that, by resulting into small components written in Java, J2ME added a whole new dimension to wireless programming.

1.2 The Field of Interest

The goal of this thesis was the development of a wireless client application capable of exchanging data via XML, using J2ME.

Since J2ME was the key technology used for the development, this thesis examines the J2ME Platform as well as the CLDC/MIDP APIs required for programming in J2ME and in general elaborates with the capabilities provided for GUI, event handling and networking.

As its name implies, J2ME is a cut-down version of Java. Besides, most J2ME developers, if not all, are very familiar with the Standard Edition of Java. Therefore, it was considered helpful to study the similarities and crucial differences of the two editions. J2ME designers went through a great deal of considerations which standard Java classes to subset and this thesis presents some cases.

Integration of a J2ME client (or MIDP client as it is usually referred to) into an existing system would require –some years ago- many changes on the code of the server, since data exchange between the server and the new client would need some sort of filtering and special handling. However, with the introduction of the eXtensible Markup Language (XML) such tasks were greatly relieved of hard-coding programming on the server side, since data and programming logic were separated. All sorts of data are exchanged via XML

and all a new client is required to perform in order to fit in is ability to communicate using the appropriate XML documents. This is a well-known motivation among developers for using XML as data transport.

Finally, this thesis presents the efforts made by various developers for convergence of XML and J2ME and deals with the challenges of developing in such a small environment.

1.3 Goals and Motivations

The motivation for developing a MIDP client came up as an extension to a multi-tier architecture system, developed during the HARP (Harmonisation for secuRity and aPplications – IST –1999-10923) research program [2],[9],[10], funded by the European Union. This research program was part of the general domain of telemedicine and specialized on the field of medical care and access to medical data of patients.

Based on these facts, the development of the MIDP client aroused as a necessary tool to be provided on medical staff. Access to data by a cell phone is sure handy. In the medical world however it can evolve to a necessity.

The result of such considerations was the start of the research needed for this thesis, which at the beginning had to answer for whether a small MIDP client could fit in without further load on the server and also whether XML is easily handled and supported in such a resource-constrained environment. GUI was also an issue, as J2ME does not support Swing –in use by the existing Applet- or AWT.

All these issues set the goals of the thesis and the successful accomplishment of these goals is presented in the following chapters along with the related theoretical and research material.

1.4 Organization

This thesis consists of 7 chapters, as follows:

Chapter 1, Introduction

Provides a broad overview of the technologies related to this thesis. Also presents the field of interest and the motivations that led to the elaboration of the thesis as well as the goals that were set.

Chapter 2, Introducing Java 2 Platform, Micro Edition (J2ME)

Describes the J2ME architecture and also explains configurations and profiles with details for the CLDC and MIDP that concern this thesis.

Chapter 3, Programming with the MIDP

Discusses the programming models of the MIDP, specifically those concerning the XMIDlet application development, and also presents development decisions concerning GUI, event handling and network connections.

Chapter 4, The J2ME Wireless Toolkit

Presents the J2ME Wireless Toolkit, an integrated development environment, which was used for the development and deployment of XMIDlet.

Chapter 5, J2ME and XML

Focuses on the convergence of J2ME and XML, introducing XML and discussing the needs and benefits from all the efforts made so far.

Chapter 6, The HARP Project

Presents the pre-existing system with focus on the Applet client application and on the way XML is used as transport for any data within the HARP system.

Chapter 7, XMIDlet, the MIDP client for HARP

Presents the MIDP client application developed during the elaboration of this diploma thesis, providing instructions and examples of use. Reviews the design decisions that were taken according to the needs of the system on the one hand and to the limitations imposed by J2ME on the other.

2. Introducing Java 2 Platform, Micro Edition (J2ME)

2.1 Overview of J2ME

Sun Microsystems, Inc. introduced J2ME at the JavaOne conference in June 1999 as the younger sibling of both the Java 2 Standard Edition (J2SE) and the Java 2 Enterprise Edition (J2EE). At the time, distributed programming was taking the Java Developer Community by storm, so most of the participants at the show were more interested in what J2EE had to offer. However, as the years passed developers realized that there was also tremendous value in having small components running in Java. Two years later, Sun devoted an entire track for individuals seeking to master the J2ME. At the most recent JavaOne conference held in San Francisco, most of the focus was on mobile technologies in a conference that was characterized by Sun as a conference "full of J2ME".

2.1.1 What Is J2ME?

The Java 2 Platform, Micro Edition (J2ME) [1],[3],[11] is the Java platform for consumer and embedded devices such as mobile phones, PDAs, TV set-top boxes, in-vehicle telematics systems, and a broad range of embedded devices. Like its enterprise (J2EE), desktop (J2SE) [4].[8] and smart card (Java Card) counterparts, the J2ME platform is a set

of standard Java APIs defined through the Java Community Process program by expert groups that include leading device manufacturers, software vendors and service providers.

The J2ME platform delivers the power and benefits of Java technology tailored for consumer and embedded devices – including a flexible user interface, robust security model, broad range of built-in network protocols, and support for networked and disconnected applications. With J2ME, applications are written once for a wide range of devices, are downloaded dynamically, and leverage each device's native capabilities.

2.1.2 The J2ME Architecture

The J2ME architecture defines configurations, profiles and optional packages as elements for building complete Java runtime environments that meet the requirements for a broad range of devices and target markets. Each combination is optimized for the memory, processing power, and I/O capabilities of a related category of devices.

2.1.2.1 Configurations

Configurations are composed of virtual machine and a minimal set of class libraries. They provide the base functionality for a particular range of devices that share similar characteristics, such as network connectivity and memory footprint. Currently, there are two J2ME configurations: the Connected Limited Device Configuration (CLDC), and the Connected Device Configuration (CDC).

2.1.2.2 Profiles

In order to provide a complete runtime environment targeted at a specific device categories, configurations must be combined with a set of higher level APIs, or profiles, that further define the application life-cycle model, the user interface, and access to device specific

properties. Existing profiles include the CDC Foundation Profile, Personal Profile and Personal Basis Profile, while CLDC is combined with the MIDP Profile to provide a complete java runtime environment.

2.1.2.3 Optional Packages

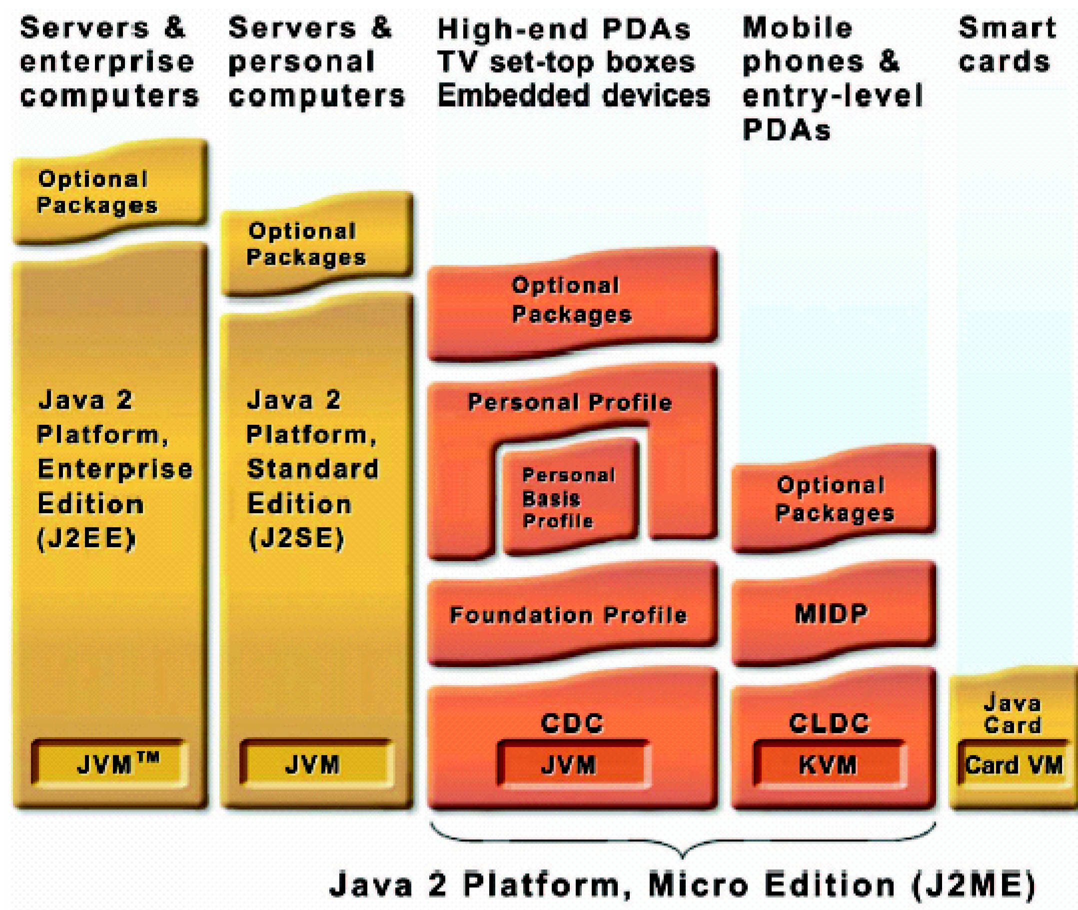
The initial flurry of J2ME specification activity was focused on the development of new profiles. As the platform evolved, however, it became obvious that developers needed a third category of J2ME component, the optional package.

Optional packages provide optionally extensions for the J2ME platform, by combining them with CDC, CLDC and their corresponding profiles. Created to address very specific market requirements, optional packages offer standard APIs for using both existing and emerging technologies such as Bluetooth, Web services, wireless messaging, multimedia and database connectivity.

This chapter goes through details only for the CLDC and the MIDP profile. The client application was developed on the environment provided by this combination, targeted for the most resource-constrained devices such as cell-phones.

As a graphical review, Figure 2-1 describes the J2ME architecture along with its counterparts.

Figure 2-1. The J2ME Architecture



2.2 The Connected Limited Device Configuration (CLDC)

2.2.1 Examining the CLDC

CLDC is the smaller of the two available configurations, designed for devices with intermittent network connections, slow processors and limited memory devices - such as mobile phones, two-way pagers and PDAs.

The latest version of the CLDC specification is 1.1 (release March 2003) and details presented here are about this version.

2.2.1.1 Device Specifications

According to the specification the devices targeted by the CLDC have the following characteristics:

1. 160 KB to 512 KB of total memory
2. 16-bit or32-bit processor with at least 25Mhz speed
3. Connectivity to some kind of networking
4. Low power consumption

2.2.1.2 Functionalities

Given the constraints listed above, the CLDC currently provides the following functionality to its devices:

1. A subset of Java language and virtual machine features
2. A subset of core Java libraries (java.lang and java.util)
3. Basic input/output (java.io)
4. Basic networking support (javax.microedition.io)
5. Security

Application life cycle management, UIs, event handling and interaction between user and application fall into the domain of profiles (MIDP), which are implemented on top of CLDC and add to its functionality.

2.2.1.3 What About the Java Virtual Machine?

The CLDC has no optional features, i.e. everything that the CLDC provides is usable on the devices that support it. As a result, a number of features have been eliminated from Java Virtual Machines (VMs) that support the CLDC.

According to the specification, features have been eliminated from a VM conforming to CLDC because the Java libraries included in CLDC are substantially more limited than the class libraries of Java 2 Standard Edition, and/or the presence of those features would have posed security problems in the absence of the full J2SE security model. The eliminated features include:

1 User-defined class loaders

2 Thread groups and daemon threads

3 Finalization of class instances

4 Asynchronous exceptions

In addition, a VM conforming to CLDC has a significantly more limited set of error classes than a full J2SE VM.

2.2.1.4 The KVM

The virtual machine for the CLDC is called the Kilo Virtual Machine (KVM), which was created at the Sun Microsystems Laboratories with contributions from industry leaders such as 3COM, Bull, Fujitsu, Panasonic, Mitsubishi Electric, NEC, NTT DoCoMo and Siemens. The KVM is a complete Java runtime environment for small devices. KVM is suitable for

devices with 16/32-bit RISC/CISC microprocessors/controllers, and with as little as 160 KB of total memory available for the Java technology stack.

2.2.2 CLDC Libraries

A general goal for designing the libraries for the Connected Limited Device Configuration is to provide a minimum useful set of libraries for practical application development and profile definition for a variety of small devices. As explained earlier, CLDC is a "lowest common denominator" standard that includes only the minimal Java platform features and APIs for a wide range of consumer devices.

To ensure upward compatibility with larger editions of the Java 2 Platform, the majority of the libraries included in CLDC are a subset of Java 2 Standard Edition and Java 2 Enterprise Edition. While upward compatibility is a very desirable goal, J2SE and J2EE libraries have strong internal dependencies that make subsetting them difficult in important areas such as security, input/output, user interface definition, networking and storage. For this reason, some libraries had to be redesigned, especially in the area of networking

The CLDC libraries defined by the CLDC Specification can be divided into two categories:

1 Classes that are a subset of standard J2SE libraries,

2 Classes that are specific to CLDC (but which can be mapped onto J2SE).

2.2.2.1 Classes Derived from Java 2 Standard Edition

CLDC supports a number of classes that have been derived from Java 2 Standard Edition, version 1.3.1. The rules for J2ME configurations mandate that each class that has the same name and package name as a J2SE class must be identical to or a subset of the corresponding J2SE class. The semantics of the classes and their methods included in the subset shall not be changed. The classes shall not add any public or protected methods or fields that are not available in the corresponding J2SE classes.
Classes belonging to this category are located in packages java.lang.*, java.util, and java.io. A detailed list of these classes is presented in Appendix A "CLDC Classes Inherited from Java 2 Standard Edition."

2.2.2.2 CLDC Specific Classes

.
The CLDC inherits some of the classes in the java.io package. However the major difference is that it does not inherit classes related to file I/O (like the popular FileInput/OutputStream and FileReader/Writer classes). This is because not all CLDC devices support the concept of a file system.

As for the java.net package, the J2SE provides several classes for network connectivity. However, none of these classes have been inherited because not all devices require TCP/IP or UDP/IP.

Differences in I/O and networking drove the CLDC expert group to define a more generic set of classes for J2ME I/O and network connectivity. These classes are known as the Generic Connection Framework, and are found in the javax.microedition.io package.

These classes are presented in detail in Appendix B “CLDC Specific Classes”.

2.3 The Mobile Information Device Profile (MIDP)

The Mobile Information Device Profile (MIDP) is built on top of the CLDC, and defines an open application development environment for what Sun calls Mobile Information Devices (MIDs).

2.3.1 Hardware Requirements

The MIDP standard defines a MID as a device that should have the following minimum characteristics:

Display:

- Screen-size: 96x54
- Display depth: 1-bit
- Pixel shape (aspect ratio): approximately 1:1

Input:

- One or more of the following user-input mechanisms: one-handed keyboard, two-handed keyboard, or touch screen

Memory:

- 256 kilobytes of non-volatile memory for the MIDP implementation, beyond what's required for CLDC.
- 8 kilobytes of non-volatile memory for application-created persistent data
- 128 kilobytes of volatile memory for the Java runtime (e.g., the Java heap)

Networking:

- Two-way, wireless, possibly intermittent, with limited bandwidth

Sound:

- The ability to play tones, either via dedicated hardware, or via software algorithm.

2.3.2 Class Additions

The MIDP adds the following packages to those available through the CLDC, as shown in table 2-1. They are presented here briefly just for purposes of getting an overview of the environment provided by the MIDP. There are currently two available versions of the MIDP profile. All java-enabled cell-phones at the moment are MIDP 1.0 compatible. MIDP 2.0, released in 2002 by Java Specification Request 118 (JSR 118) Expert Group, added new packages and features (table 2-1), most emphasis placed on networking, security and gaming - multimedia. However, no device supports it so far.

As a result, the XMIDlet is MIDP 1.0 compliant and since MIDP 2.0 is backward compatible, the MIDlet will also be able to run on future MIDP 2.0 devices.

Table 2-1. MIDP packages summary

Package Summary	
User Interface Package **javax.microedition.lcdui**	The UI API provides a set of features for implementation of user interfaces for MIDP applications
javax.microedition.lcdui.game (MIDP 2.0)	The Game API package provides a series of classes that enable the development of rich gaming content for wireless devices.
Application Lifecycle Package **javax.microedition.midlet**	The MIDlet package defines Mobile Information Device Profile applications and the interactions between the application and the environment in which the application runs.
Persistence Package **javax.microedition.rms**	The MIDP provides a mechanism for MIDlets to persistently store data and later retrieve it.
Networking Package **javax.microedition.io**	MIDP includes networking support based on the Generic Connection framework from the CLDC.
Public Key Package **javax.microedition.pki (MIDP 2.0)**	Certificates are used to authenticate information for secure Connections.
Sound and Tone Media **javax.microedition.media (MIDP 2.0)**	The MIDP 2.0 Media API is a directly compatible building block of the Mobile Media API specification.
javax.microedition.media.control (MIDP 2.0)	This package defines the specific Control types that can be used with a Player.
Core Packages	

java.lang	MIDP Language Classes included from Java 2 Standard Edition.
java.util	MIDP Utility Classes included from Java 2 Standard Edition.

2.3.3 MIDlets

Applications that run on MIDP devices are called MIDlets. MIDlets are written as one or more Java classes whose objects are compressed into a Java Archive JAR file. Like Java applets, MIDP applications have an application life cycle while running on a mobile device.

Specifically, a MIDlet can be in one of three states:

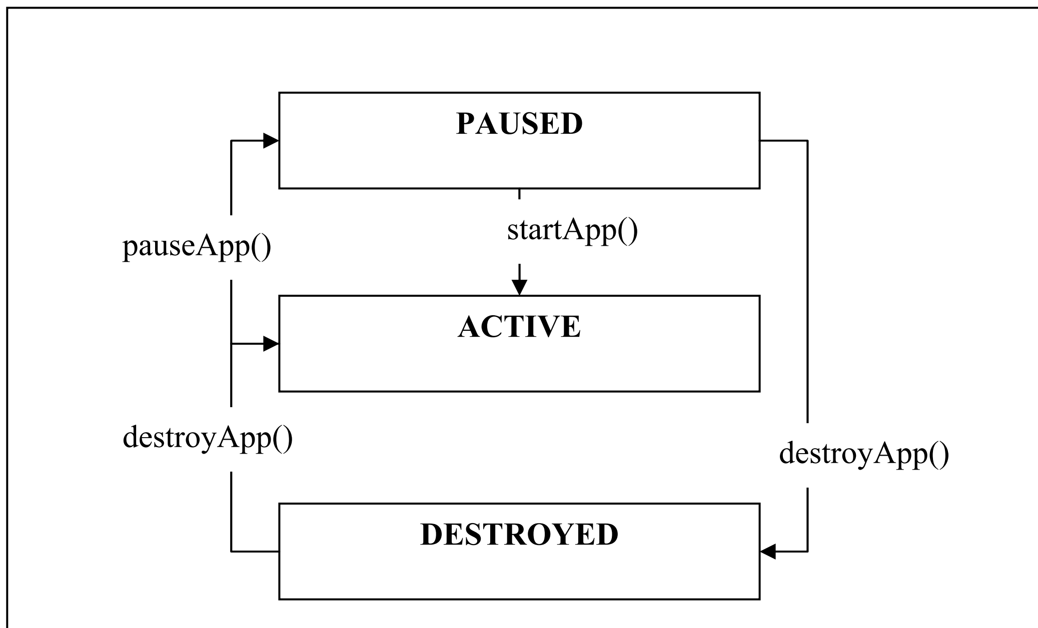
2 Paused: placed in this state when first started, can be placed any time by the MIDP system or the controlling software

3 Active: MIDlet is running and user can interact with it.

4 Destroyed: the MIDlet releases all of the resources it currently has back to the MIDP system

Figure 2-2 shows the rules for transitioning between states.

Figure 2-2. MIDlet transition states



2.3.3.1 Java Application Descriptor (JAD)

A JAD file is a text file that consists of a series of attributes used to describe a MIDlet suite. A MIDlet suite is simply two or more MIDlets that are packaged in a JAR file and which can use the classes and resources contained in the JAR file, much like any standard Java application, the classes of which are loaded by the same class loader.
The JAR file of a MIDlet often contains a manifest file with MIDlet attributes defined. These attributes describe the contents of the JAR file, which is in turn used by the application management software to identify and install the MIDlet suite. So, the JAD file is quite similar to a manifest. However, unlike a manifest, it is not packaged in the JAR file. MIDP specification defined the Java Application Descriptor to avoid unnecessary downloads of JAR files only for the user to discover deployment is not possible.

2.3.3.2 Creating MIDlets

This section briefly presents the development life cycle of a MIDlet. This involves the following five steps:

1 **Write the MIDlet.**

2 **Compile the Source Code.**

3 **Preverify the MIDlet's class file.**

In the J2SE Java virtual machine, the class verifier is responsible for rejecting invalid class files at runtime. A JVM supporting CLDC must be able to reject invalid classes files as well. However class verification typically takes from 35 to 110 KB of runtime memory when the target size of the KVM is 50 to 80 KB of memory. Including a class verifier inside it would violate its size constraints. So, KVM designers decided to move most of the verification work off the device and onto the desktop. This step (off-device verification) is referred to as *preverification*.

4 **Package the application in a JAR file.**

In order to enable dynamic downloading of MIDP applications, the application must be packaged in a JAR file.

5 **Create a JAD file.**

If a device can't handle the MIDlet , there is no reason to attempt the download. To accomplish this, the developer must create the Java Application Descriptor and manually specify some pre-downloaded properties, including the size of the MIDlet and its storage requirements.

The above is the standard development life cycle of a MIDlet and it can be accomplished from the command line. However, this is an alternative. Such a way should only be used in order to understand what is happening behind the scenes. An integrated development environment, such as the J2ME Wireless Toolkit, described in chapter 4, can be used to simplify the development and deployment of MIDlets.

3. Programming with the MIDP

Examining the J2ME Platform, as this is defined by CLDC, MIDP and KVM specifications, one can realize that certain programming models in such a platform will differ potentially from corresponding models of Java 2 Standard Edition. This chapter elaborates with such differences, especially in the domains related to the development of the MIDP client application for HARP.

Specifically, the MIDP GUI model, Event Handling and Networking were issues that had to be examined in order to develop a MIDP client that could provide the functionalities defined by HARP's client requirements. These requirements, which set the challenges for the development of XMIDlet, along with the decisions that were taken in order to meet them, will be presented on following chapters. At this point however, it is useful to present the programming models available for development XMIDlet according to the MIDP.

3.1 MIDP GUI

Any Java Standard Edition programmer with relative experience in GUI programming is familiar with the Abstract Windows Toolkit (AWT) and Swing classes. However, it was decided that these classes wouldn't be used for J2ME as well since AWT is designed and optimized for desktop computers. User interaction models assumed by AWT are bound to devices such as the mouse or other pointing devices. Most handheld devices though, have only a keypad for user input. There are also other reasons for such a decision and most of them have to do with limited CPU and memory or limited display size of the handheld devices. Therefore, the MIDP contains its own abbreviated GUI. The MIDP GUI consists of both high-level and low-level APIs, each with their own set of events.

3.1.1 The High-Level API

The high-level API is designed for applications where portability between mobile information devices is important. In order to achieve portability, the API employs a high-level abstraction and gives to the programmer little control over the design and visual representation (such as shape, color or font) of GUI components. Instead, most interactions with the components are encapsulated by the implementation. Consequently, the underlying implementation does the necessary adaptation to the device's hardware and native user interface style. Classes that implement the high-level API all inherit the javax.microedition.lcdui.Screen class.

3.1.2 The Low-Level API

On the other hand, the low-level API is designed for those applications that need precise placement and control of graphic elements, as well as to access low-level input events (such as the pressing or releasing of specific numbers in the keypad). Games using graphics are an example of such applications. The javax.microedition.lcdui.Canvas and javax.microedition.lcdui.graphics classes implement the low-level API. Since this API provides mechanisms to access details that are specific to a particular device, portability is not guaranteed.

3.1.3 The MIDP GUI Model

The MIDP GUI model is based on the relationship between display and screens. Screens act as containers for one or more graphic components and are presented on the display of the device. There are three types of screens in the MIDP GUI:

6 Screens that entirely encapsulate a complex user interface and have a predefined structure, such as the List or TextBox component.

7 Generic screens that use a Form component. These screens actually act as containers for related UI components.

8 Screens using low-level GUI API.

There can be several screens in an application, but only one screen at a time can be visible in a display, and the user can interact only with items on that screen. The application switches from screen to screen according to user events, using the display manager of the MIDlet (the Display class). Figures 3-1 to 3-3 show various screen examples containing high-level and low-level GUI components.

3.2 MIDP Events

Events and GUI are issues that usually go together, since events are generated when a user interacts with an application (as in AWT and Swing). This is also true for the MIDP model. Therefore, the distinction that was mentioned before between high-level and low-level also applies to events. Namely, there are two kinds of events as there are two MIDP user interface APIs: high-level (such as selecting an item from a list / high-level GUI component) and low-level (such as pressing a key on the device).

The high-level approach is to use commands and implement the available Command class. Command objects contain the basic information needed about the command, not the functionality of the command. Such command objects can be appended to screens (forms, lists etc.). When the MIDlet executes, the device assigns a visual representation of the command and chooses its placement based on the command information the application provided. In this way again, portability is ensured.

In order to bind commands to functionalities and therefore high-level events to specific actions (commonly known as event handling), use of a command listener is necessary. By implementing the CommandListener interface and registering it with the corresponding screen, the user is able e.g. to navigate through the different screens using the appropriate and familiar back, exit, goto or submit commands.

On the other hand, handling low-level events has some differences and is a bit more constrained. These constraints and differences relate to the way the Canvas class allows the applications to register listeners for low-level events. However, the Canvas class provides ways and methods to handle game actions, key events and pointer events.

3.3 Networking

As discussed briefly in Chapter 2, an entirely new networking library was created for the CLDC, known as the CLDC Generic Connection Framework. This was necessary, as the java.io and java.net packages of the J2SE are not suitable for handheld devices with a small

memory footprint. Variation on mechanisms for communication between handheld devices makes designing network facilities for the CLDC quite a challenge. Device manufacturers who work with circuit-switched networks require stream-based connections such as the Transport Control Protocol (TCP) while others who work with packet-switched networks require datagram-based connections such as the User Datagram Protocol (UDP). In the Standard Edition, the java.net package provides over 20 networking classes for different forms of communication. In the CLDC however, space is of the essence and such an approach isn’t feasible.

3.3.1 Generic Connections

In the Generic Connection Framework, all connections are created using the static open() methods from a single class: javax.microedition.Connector. If successful, these methods return an object which implements one of the generic connection interfaces. Since this is just the configuration level, no implementations are provided. The idea was to provide a set of related abstractions that can be used at the programming level instead of using different abstractions for different forms of communications.

3.3.2 MIDP Connectivity

The MIDP extends the CLDC Generic Connection Framework to provide support for the HTTP protocol. HTTP was chosen as it can be implemented using both IP protocols (such as TCP/IP) and non-IP protocols (such as WAP and I-mode). In the latter case, the device would have to utilize a gateway that could perform URL naming resolution to access the Internet.

Since all of the MIDP 1.0 implementations must provide support for the HTTP protocol, using the HTTP protocol allows the application to be portable across all mobile information devices. The issue of portability is quite important when programming with the MIDP. Later, on chapter 7, where the XMIDlet application is presented, HTTP networking and high-level GUI components will be some of the development decisions based on the need for portability.

3.3.3 The HTTP Programming Model

This section briefly presents the HTTP Programming Model for readers not experienced with HTTP. The HTTP is a request-response application protocol. HTTP stands for Hypertext Transfer Protocol. It's the network protocol used to deliver virtually all files and other data (collectively called resources) on the World Wide Web, whether they're HTML files, image files, query results, or anything else.

A browser is an HTTP client because it sends requests to an HTTP server (Web server), which then sends responses back to the client.

Like most network protocols, HTTP uses the client-server model: An HTTP client opens a connection and sends a request message to an HTTP server; the server then returns a response message, usually containing the resource that was requested. After delivering the response, the server closes the connection (making HTTP a stateless protocol, i.e. not maintaining any connection information between transactions)

There are two request methods to send data from e.g. a form on a web page to a servlet hosted by the HTTP server. These methods are GET and POST. Each of these has a different way of sending data to the server.

1. For the GET method, the input values are sent as part of the URL (message header).
2. For the POST method, data is sent as an input stream (message body).

The POST method is naturally more secure and one can send more data using it.

4. The J2ME Wireless Toolkit

Chapter 2 presented the development life cycle of a MIDlet. This cycle consists of compiling, preverifying, creating the JAR and JAD files and run MIDlets from the command line. Knowing this procedure helps better understanding of the J2ME infrastructure. However development in this way is tiresome and time-consuming. After all it is an alternative since it is possible to simplify the development. This can be achieved by using an integrated development environment, which is distributed freely by Sun Microsystems. This is known as the **J2ME Wireless Toolkit** (J2MEWT). Many emulators exist for running MIDlets on your desktop PC but the J2MEWT was the first free tool providing a complete yet easy-to-use solution for developing mobile applications.

This chapter briefly presents the features of the J2ME Wireless Toolkit and shows how greatly the development cycle is simplified.

4.1 Overview

The J2ME Wireless Toolkit supports a number of ways to develop MIDP applications [12]. You can carry out the development process by running the tools from the command line or by using development environments that automate a large part of this process.

The KToolBar, included with the J2ME Wireless Toolkit, is a minimal development environment with a GUI for compiling, packaging, and executing MIDP applications (described in section 4.2). The only other tools you need are a third-party editor for your Java source files and a debugger.

An IDE compatible with the J2ME Wireless Toolkit provides even more convenience. For example, when you use the Sun ONE Studio 4, Mobile Edition, (formerly Forte™ for Java™) you can edit, compile, package, and execute or debug MIDP applications, all within the same environment.

For a list of other IDEs that are compatible with the Wireless ToolKit, see http://java.sun.com/products/j2mewtoolkit/, the Java™ 2 Platform Micro Edition, Wireless Toolkit web page.

This section describes the phases of MIDP application development outside of editing, and how the toolkit contributes to these phases. The phases are illustrated in the following diagrams.

Figure 4-1. Developing and Testing an Application

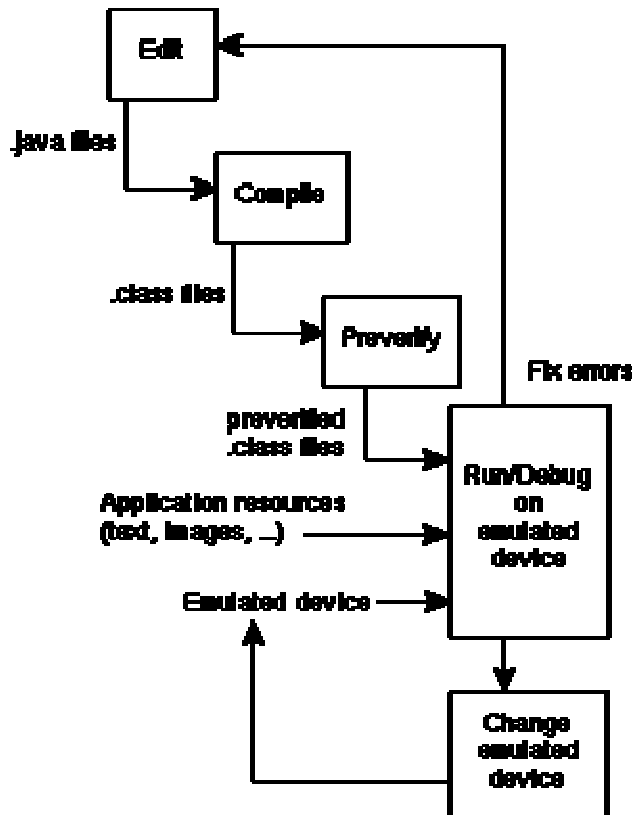
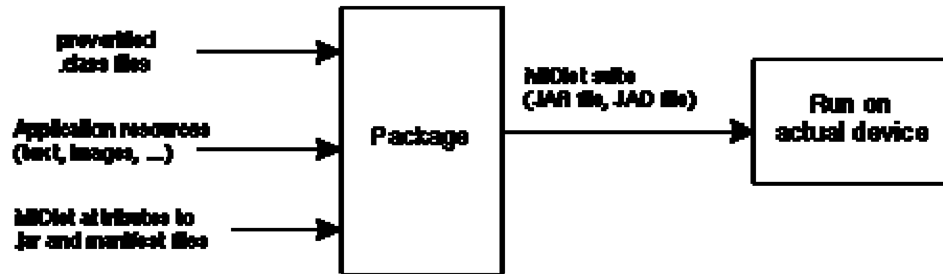


Figure 4-2. Packaging an Application



4.1.1 Compilation and Prefabrication

When you use KToolbar or a toolkit-compatible environment, such as the Sun ONE Studio 4, Mobile Edition, the environment compiles your source files for you, using the Java 2 SDK, Standard Edition (J2SE™ SDK) compiler.

After compiling the sources, the development environment passes the generated class files to the Preverifier. This tool rearranges bytecodes in the classes to simplify the final stage of bytecode verification on the CLDC virtual machine. It also checks for the use of virtual machine features that are not supported by the CLDC.

4.1.2 Running and Debugging

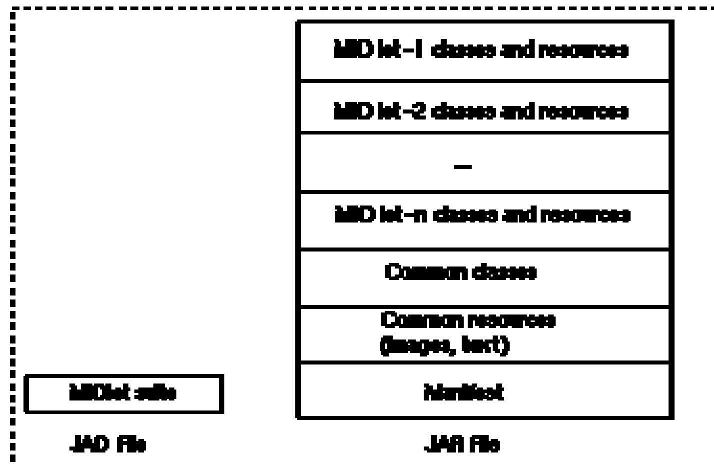
When you use KToolbar or a toolkit-compatible environment such as the Sun ONE Studio 4, Mobile Edition, you can run and debug applications within the environment using the Emulator, which simulates the execution of the application on different target devices.

The Emulator enables you to approximate the experience a user has with an application on a particular device, and to test the portability of the application across different devices.

4.1.3 Packaging

MIDP applications, or MIDlets, are packaged into a MIDlet suite, a grouping of MIDlets that can share resources at runtime. The following diagram illustrates how a MIDlet suite is organized.

Figure 4-3. MIDlet Suite Components



More formally, a MIDlet suite includes:

1. A Java Application Descriptor (JAD) file. This file contains a predefined set of attributes (denoted by names that begin with "MIDlet-") that allow application management software to identify, retrieve, and install the MIDlets. All attributes appearing in the JAD file are made available to the MIDlets. You can define your own application-specific attributes and add them to the JAD file.
2. A Java Archive (JAR) file. The JAR file contains:
    - Java classes for each MIDlet in the suite.

    1 Java classes shared between MIDlets.

    2 Resource files used by the MIDlets (for example, image files).
3. A manifest file describing the JAR contents and specifying attributes used by application management software to identify and install the MIDlet suite.

Development environments such as KToolBar and the Sun ONE Studio 4, Mobile Edition automate the packaging of MIDlet suites. On the other hand, in order to package MIDlet

suites from the command line, unlucky (or unwise) developers would need the J2SE SDK JAR tool to create JAR files, and a text editor for creating JAD files.

4.1.4 Packaging Obfuscated Source Code

An additional feature of the J2ME Wireless Toolkit is the ability to build an obfuscated package. You are required to obtain a code obfuscator plug-in to use this feature. The JAR file for the code obfuscator should be placed in the *j2mewtk.dir*\bin directory. Creating an obfuscated package is only available through the KToolBar.

Obfuscation removes extraneous class information, such as local variable names. Classes, methods, interfaces, and such are renamed so as to make them ambiguous. An obfuscated package protects your project files from decompilation and reverse engineering. In addition to protecting your source code, the obfuscation process reduces the size of the classes resulting in smaller JAR files. The details of how code is obfuscated is dependent on the specific code obfuscator you choose to use.

When creating an obfuscated package, preverification is done after the code has been obfuscated rather than immediately after compilation.

4.2 The KToolBar Application

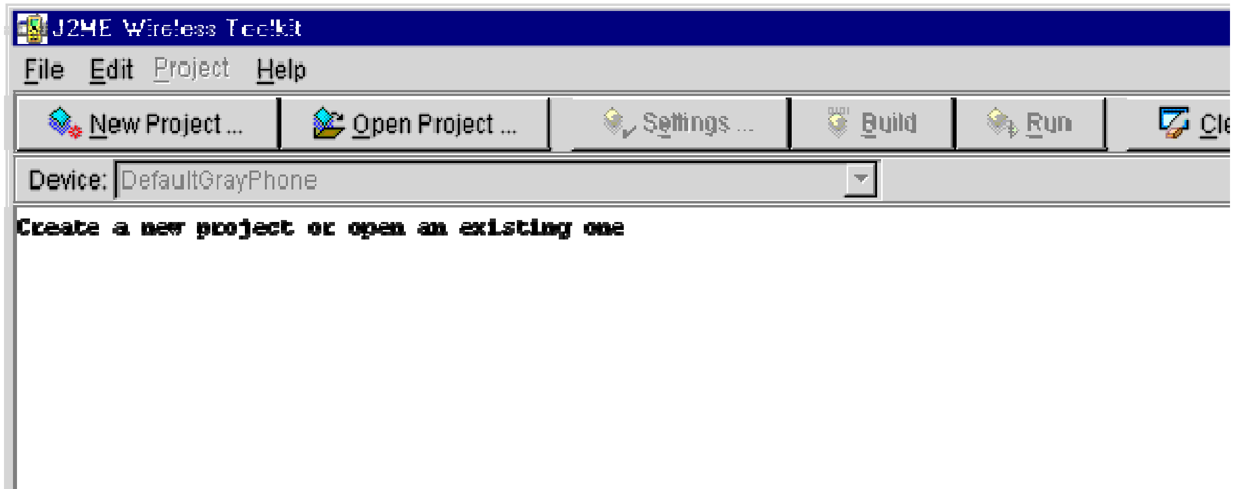
In this section, the KToolBar application isbriefly presented, as it is by far the most important application in the J2MEWT.

KToolBar is a minimal development environment for developing MIDlet suites that comes with the J2MEWT. From the KToolBar, it is possible to:

1 Create a new project or open an existing one
2 Build (includes compiling and preverifying), run, and debug your MIDlet
3 Fine tune your MIDlet application
4 Package your project files
5 Modify the attributes of your MIDlet suite

Figure 4-4 shows the application's main window.

Figure 4-4. KToolBar Main Window



4.2.1 KToolBar Projects

A KToolBar project is associated with a MIDlet suite. The project contains the suite's source, resource and binary files, as well as the JAD and manifest files that contain the suite's attributes.

Project files are located in project subdirectories under the Wireless Toolkit's installation directory, {*j2mewtk.dir*}. Table 4-1 shows how files are organized within the directory for the project, {*project.name*}.

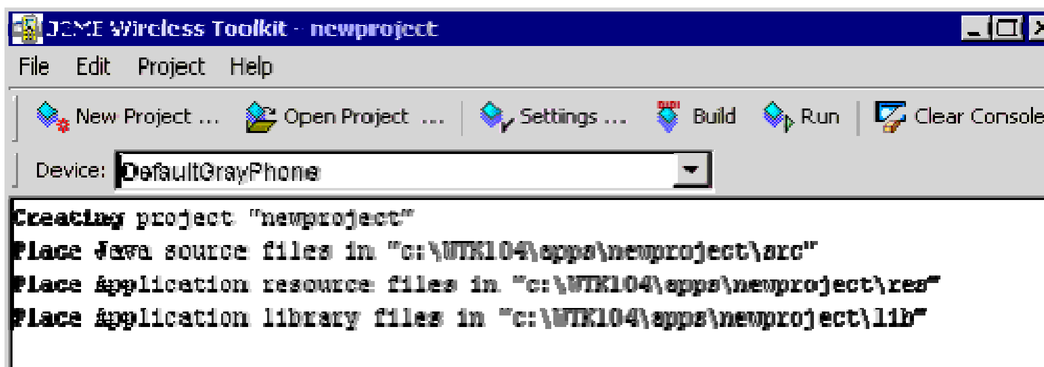
Table 4-1. Project File Organization

Directory	Description
{*j2mewtk.dir*}\apps\{*project.name*}	Contains all source, resource, and binary files of the project
{*j2mewtk.dir*}\apps\{*project.name*}\src	Contains all the source files.
{*j2mewtk.dir*}\apps\{*project.name*}\res	Contains all the resource files.
{*j2mewtk.dir*}\apps\{*project.name*}\bin	Contains the JAR, JAD, and unpacked manifest files.
{*j2mewtk.dir*}\apps\{*project.name*}\lib	Contains external class libraries, in JAR or ZIP format for a specific project.
{*j2mewtk.dir*}\apps\lib	Contains external class libraries, in JAR or ZIP format for all KToolBar projects.

Adding external class libraries to a project increases the size of the MIDlet suite's JAR file. Large JAR files take longer to load onto a device, and might be unusable on devices with low memory. This was an issue that had to be considered when developping XMIDlet as external files such as the required XML parser would add to the size of the suite's JAR file. Class libraries for use with KToolBar should be compatible with the CLDC and MIDP APIs and should be packaged in .jar or .zip format. KToolBar provides ways to develop with class libraries, both on a per project and on a global basis.

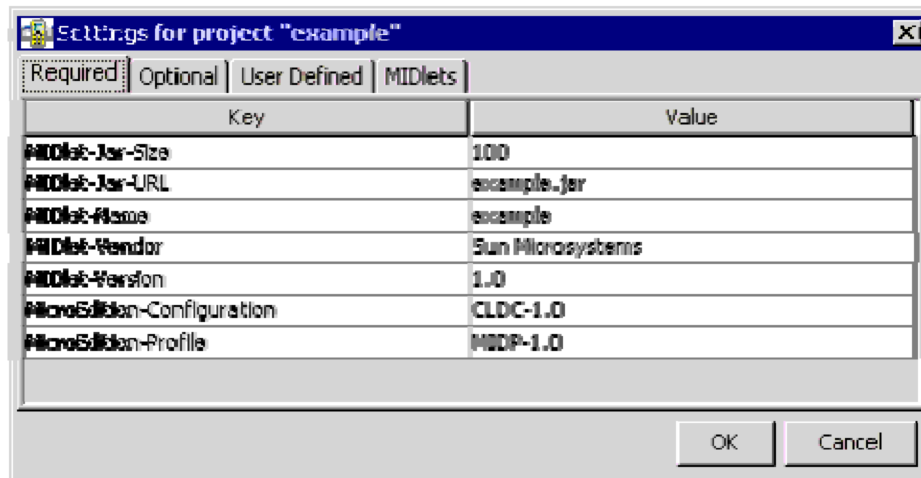
When creating a new project the console indicates where to place your source, resource, and library files. The locations are consistent with the project file organization outlined in Table 4-1. The main window's title changes to include the name of the new project, as shown by Figure 4-5.

Figure 4-5. Console Output After Creating a Project



As shown on section 4.1, the KToolBar relieves the user from the writing of the JAD file via text editor. Modifying the JAD file can be achieved in a faster and mistype-free way, as shown on figure 4-6.

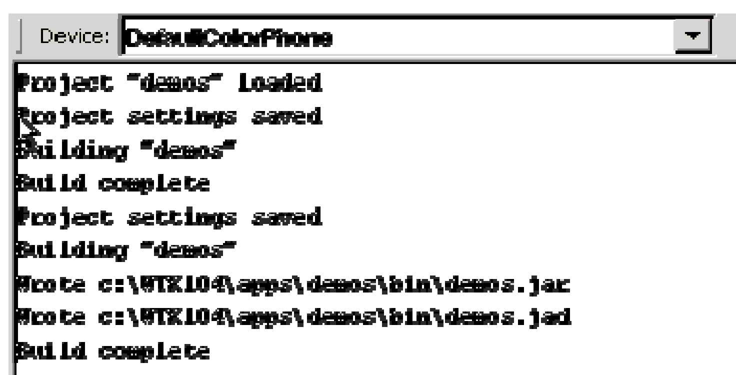
Figure 4-6. Project Settings Dialog



KToolBar gives the choice of creating a package of project files or create an obfuscated package to protect the project code from possible decompilation. Another benefit to creating an obfuscated package is that the obfuscation process reduces the size of the Java bytecode, resulting in a smaller JAR file and possibly faster download times.

The J2ME Wireless Toolkit contains a support framework for byte code obfuscators. It also contains a plug-in for the RetroGuard byte code obfuscator. You can obtain the retroguard.jar file from http://www.retrologic.com. The retroguard.jar file must be placed in the Wireless Toolkit's bin directory: {*j2mewtk.dir*}\bin.

Figure 4-7. Console Output After Packaging



4.2.2 The Emulator Application

Along with the Wireless Toolkit comes the Emulator application, embedded in the KToolBar application. The Emulator shows, on the computer, how MIDP applications operate on a variety of mobile devices. Consequently, developers can test their applications using the same platform they use to develop them, and defer testing on real devices until later in the development process.

The Emulator supports testing with several key features:

A variety of devices. The Emulator can simulate several devices that take on a variety of form factors: cell phones, pagers, and palmtops. This helps learning what type of experience users can expect and verify the portability of an application across different devices.

Tracing and debugging capabilities. The Emulator supports run-time logging of various events, such as garbage collection, class loading, method calls, and exceptions. Developers can also perform source-level debugging with an IDE while an application runs in the Emulator.

Performance tuning. The Emulator enables collection of information used to optimize the performance of MIDP applications. Performance tuning utilities include profiling

methods and monitoring memory usage and network traffic. Also included is the ability to adjust the speed settings for graphic rendering and refreshing as well as VM speed emulation and network throughput.

Next section briefly ennumerates the tools provided for perfomance tuning.

4.2.3 Performance Tuning Applications

The Wireless Toolkit enables optimization of performance with the following features:

1 **Profiler**. Enables developers to examine the execution time and the frequency of use of the methods in your application.

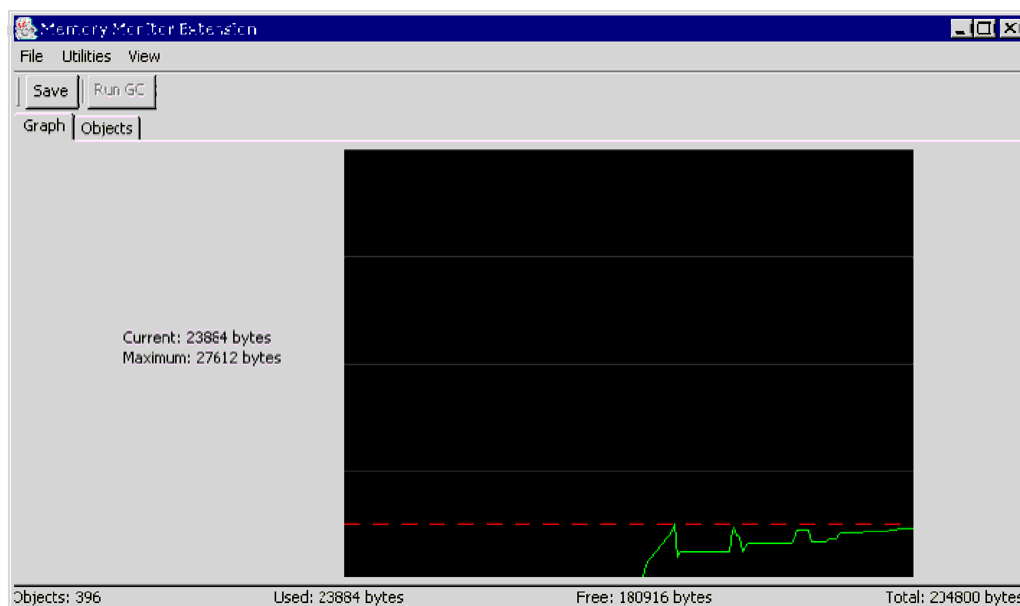
2 **Memory Monitor**. Enables developers to examine memory usage in their application.

3 **Network Monitor**. Lets developers monitor transmissions between their device and the network.

4 **Speed Emulation**. Enables developers to adjust drawing speed to refine graphics rendering. It also enables to adjust the speed of byte code execution and data transfer across the network to give a sense of how quickly an application runs on a device.

All these features can be accessed from the KToolBar application where the developer sets the **Preferences** for his project.

Figure 4-9. Memory Monitor Graph



5. J2ME and XML

To begin with, this chapter introduces the eXtensible Markup Language (XML) and explains the use of this technology, demonstrating the great benefits when used in data exchange applications.

Furthermore, this chapter proceeds into explaining the benefits – and resource-constraints considerations - from J2ME and XML convergence while presenting the related XML resources available to developers when programming in the challenging J2ME environment.

5.1 Introducing XML

Extensible Markup Language (XML) [5]-[7] has become a buzzword on the Internet, being a technology with powerful real-world applications, particularly for the management, display and organization of data.

Since XML is a technology concerned with the description and structuring of data, it is necessary to understand how data is stored and accessed by computers. For these purposes, there are two kinds of data files that are understood by computers: **text files** and **binary files**.

A binary file, at its simplest, is just a stream of bits (1's and 0's). It's up to the application which created a binary file to understand what all of the bits mean. That's why binary files can only be read and produced by certain computer programs, which have been specifically written to understand them.

On the other hand, like binary files, text files are also streams of bits. However, in a text file these bits are grouped together in standardized ways, so that the always form numbers. These numbers are further mapped to characters. Because of these standards, text files can be read by many, many applications, and can even be read by humans, using a simple text editor.

The advantage of binary file formats is that it is easy for computers to understand these binary codes, meaning that they can be processed much faster, and they are very efficient for storing **metadata** (information about information, e.g stating which part of the text is in bold). The disadvantage is that binary files are "proprietary". Binary files created by one application may not be readable by other applications, or even by the same application running on another platform. This fact does not apply to text files, which, as it was written above, are universally interchangeable. The disadvantage of text files, however, is that it's more difficult and bulky to add other information, metadata in other words. For example, consider saving a word document in simple text form. The format keeps the words of the text but none of the formatting.

Considering the pros and cons of binary and text file formats, wouldn't it be ideal if there were a format that combined the universality of text files with the efficiency and rich information storage capabilities of binary files?

5.1.1 A Brief History of Markup

This idea of universal data is not new. In fact, for as long as computers have been around, programmers have been trying to find ways to exchange information between different computer programs. An early attempt to combine a universally interchangeable data format with rich information storage capabilities was **SGML** (**Standard Generalized Markup Language**). This is a text-based language that can be used to mark up data – that is, add metadata – in a way which is self-describing (self-describing will be explained later).

SGML was designed to be a standard way of marking up data for any purpose, and took off mostly in large document management systems. It turns out that when it comes to huge amount of complex data there are a lot of considerations to take into account and, as a result, SGML is a very complicated language. But with that complexity comes power.

The best known application of SGML is **HyperText Markup Language**, or **HTML**. Because the rules for creating SGML documents are so well laid out, and because SGML has been used so extensively in document management systems, it was a good fit to produce a specific vocabulary –HTML- to be a universal markup language for the display of information, and the linking of different pieces of information (**hyperlinks**). The idea was that any HTML document (or **web page**) would be presentable in any application that was capable of understanding HTML (termed a **web browser**). Furthermore, since HTML is text-based, anyone can create an HTML page using a simple text editor, or any number of web page editors.

HTML has been incredibly succesful, yet it is also limited in its scope: it is only intended for displaying documents in a browser. An HTML document can display information about a person but cannot state which piece of information relates to the person's first name, for example, as HTML doesn't have any facilities to describe this kind of specialized information. Relying on SGML for this task is also not an option, as SGML is such a complicated language that it's not well suited for data interchange over the web.

These needs were met with the creation of XML.

5.1.2 Extensible Markup Language (XML)

XML is a subset of SGML, with the same goals (markup of any kind of data), but with as much of the complexity eliminated as possible. XML was designed to be fully compatible with SGML, which means that an XML document is also an SGML doument. However this doesn't go both ways.

It is important to realize that XML is not really a "language" at all, but a standard for creating languages which meet the XML criteria. In other words, XML describes a syntax which you use to create your own languages.

For example, suppose a need for sharing data about a name with other users, while on the same time using that information in a computer program is also required. Instead of just creating a text file like this:

Dimitris Kyriazanos

or an HTML file like this:

```
<HTML>
<HEAD><TITLE>Name</TITLE></HEAD>
<BODY>
<P>Dimitris Kyriazanos</P>
</BODY>
</HTML>
```

it is possible to create an XML file like this:

```
<name>
	<first>Dimitris</first>
	<last>Kyriazanos</last>
</name>
```

Even from this simple example, it is understandable why XML (and markup languages like SGML) is called self describing. Looking at the data, one can easily tell that this is information about a name; one can see there is data called first and more data called last. Giving things in XML meaningful names isn’t obligatory, but it helps in order to use XML in a right way.

XML version of this information is much bigger than the plain-text version. Using XML to mark up data will add to its size, sometimes enormously, but small file sizes aren’t one of the primary goals of XML; it is only about making it easier to write software that accesses the information, by giving structure to data. However, this larger file should not deter programmers from using XML. The advantages of easier-to-write code far outweigh the disadvantages of larger bandwidth issues. And, if bandwidth is a critical issue for the application, there is always the option of compressing the XML documents before sending them across the network - compressing text files yields very good results.

So is XML all about just structuring data? XML is much more than that. Programmers have been structuring data in an infinite variety of ways, and with every new way of structuring data comes a new methodology for pulling out the information needed.With those new methodologies comes much experimentation and testing. If the data changes, methodologies change as well, and new tests and experiments begin. Therefore, with use of XML data can be structured in many ways -in order to serve the specific purposes of each application-, **but there is a standardized way to get the information needed**.

The example given here is quite simple. However the more complex the data one has to work with, the more complex the logic needed to do that work. It is in the larger applications where XML is appreciated most..

5.1.2.1 XML Parsers

As stated on the previous section, what XML really provides is a standard for creating our own language in order to mark up data. On the previous section it was also promised that – provided all rules specified by XML are followed- the information marked up in such a way will be easy to access. This is because there are programs written, called **parsers**, which are able to read XML syntax and get the information needed. Programmers can use these parsers within their own programs, meaning their applications never have to even look at XML directly; a large part of the workload has been done for them.
With XML, data is clearly seperated from the programming logic of our system. XML format may change while the existing code may be left unchanged, since the parser takes care of the work of getting data out of the document for us. On the other hand, other applications may also make use of the same information, each for its own puproses.

Because it's so flexible, XML is targeted to be the basis for defining data exchange languages, expecially for communication over the Internet.

5.1.3 Using XML

XML is platform and language independent, which means it doesn't matter that one computer may be using, for example, Visual Basic on a Microsoft operating system, and the other is a Unix machine with Java code. Really, any time one computer program needs to communicate with another program, XML is a potential fit for the exchange format.

Where XML can be used is really quite a question and this section highlights just a few typical examples:

1 **Reducing Server Load**

   Web-based applications can use XML to reduce load on the web servers. This can be done by keeping all information on the client for as long as possible and then sending the information to those servers in one big XML document.

2 **Web Site Content**

The W3C (World Wide Web Consortium) uses XML to write their specifications. These XML documents can then be transformed to HTML for display (by XSLT – eXtensible Style sheet Language for Transformations), or transformed to a number of other presentation formats.

3 **Remote Procedure Calls**

XML is also used as a protocol for Remote Procedure Calls (RPC). RPC is a protocol which allows objects on one computer to call objects on another computer to do work, allowing distributed computing.

4 **E-commerce**

Companies are discovering that by communicating via the Internet, instead of by more traditional methods (such as faxing, human-to-human communication etc.), they can streamline the processes, decreasing costs and increasing response times. Whenever one company needs to send data to another, XML is the perfect fit for the exchange format. There are potential uses for XML in either business to business (B2B) e-commerce as well as in business to consumer (B2C) transactions.

5.2 Parsing XML in J2ME

The convergence of J2ME and XML is currently a handful of open source parsers. This section focuses on parsing XML in a MIDP client application, starting from system architecture and the motivation for using XML as a data transport. Then a short description of some available XML parsers follows. Finally, the challenges of developing in a small environment are presented.

5.2.1 Multi-tier System Architecture

To understand why one might want to parse XML on a J2ME device, let's first examine the architecture of a typical multi-tier application. *Multi-tier* is one of those ubiquitous terms that means something different to just about everyone. Let's examine the term with a fairly specific architecture, as shown in the following figure:

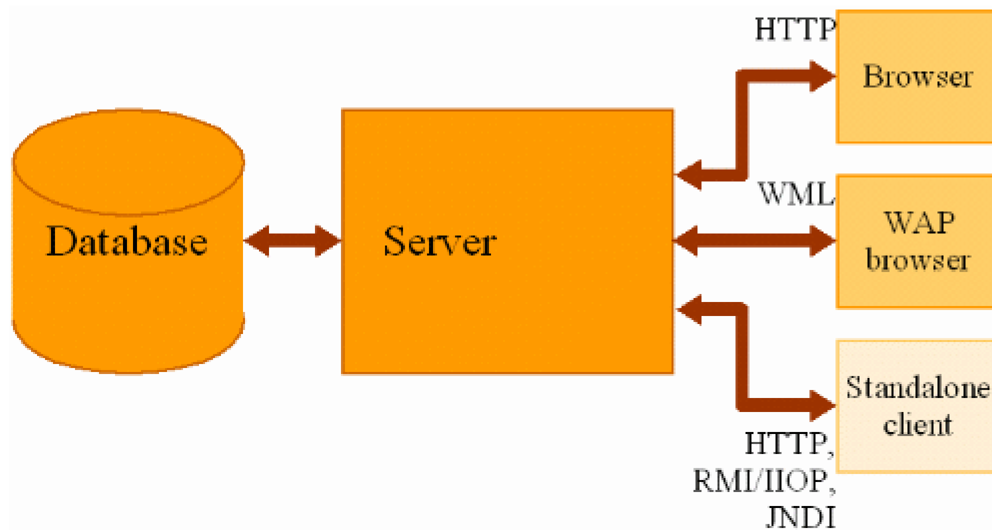


Figure 5-1. A typical three-tier architecture.

The current world is web-centric, so systems are often designed with HTML browsers as the clients. The client performs very little of the application processing and functions as a fancy kind of terminal. The bulk of the application runs on a server, which uses a database for persistent storage.

As the wireless world began expanding, server vendors found that they could conveniently support wireless devices by adding support for WAP browsers. The underlying paradigm of the browser as front end to the application remains unchanged; the server is just serving WML (Wireless Markup Language) over WAP in addition to HTML over HTTP.

The diagram also shows a standalone client, which could communicate with the application on the server in several different ways. The client could make HTTP connections, use RMI to manipulate remote objects, or implement a customized protocol. The chief advantage of having a standalone client in place of a browser is the chance to provide a richer user interface. The main disadvantage is the difficulty of client installation and maintenance.

Where do MIDP clients fit in this picture? Keep in mind that with MIDP devices, everything is small, and this affects their utility as application clients.

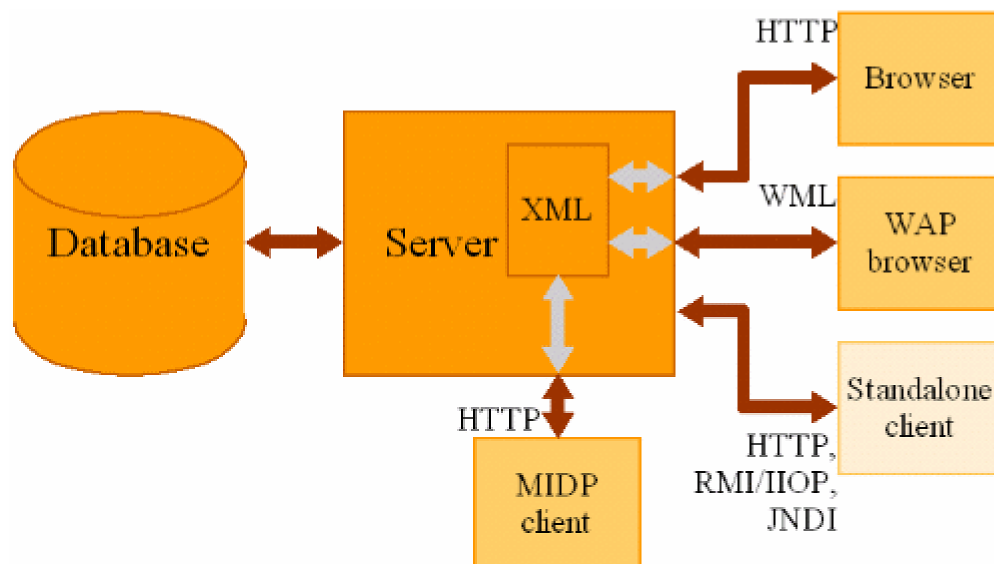
1. Network connection setup is slow.
2. Data rates are slow.
3. The processor is slow.
4. Memory is scarce.

Because of these constraints, MIDP client applications should be designed to be as small as possible. At the same time, they can feature a smooth and capable user interface that goes far beyond the user experience offered by a WAP browser.

MIDP client have one other important characteristic as compared to WAP applications; the application can run offline and make updates to the server periodically. This is especially important with wireless networks, which are slower and less reliable than the desktop network you may be accustomed to. WAP applications, by contrast, require a network connection as the user moves from screen to screen.

The following figure shows one possible implementation of a multi-tier system that supports HTML browsers, WAP browsers, standalone clients, and MIDP clients.

Figure 5-2. Including MIDP client in the picture.



The figure shows one way of supporting multiple client types. Instead of creating custom server-side code for each client type, you write generic code that returns data as XML documents. Then you create transformations (using XSLT) that change the basic XML documents into whatever is required for the client device. Once set of transformations produces HTML for desktop browsers, while another set might produce the WML to support WAP browsers.

But what kind of data do you send to the MIDP client? You send whatever you want, of course, anything from plain text to binary data. If you're using XML on the server side, however, you may consider XML itself as a data exchange format. It makes your life pretty easy on the server side, for one thing. You might be able to send your basic XML documents unchanged, or you could create some simple transformations to send a more terse XML format to the MIDP device.

Sending XML from client to server offers XML's usual advantages: the data is self-describing and offers the opportunity to loosely couple the client and server.

Sending XML to the client has another advantage. During the development cycle, you can use validating XML parsers in emulated clients to ensure that the documents the server generates are clean. By the time you try running the application on a real MIDP device, you'll be pretty sure that the data it's getting is good.

The downside of XML is that it's not a very efficient way of expressing data. On slow wireless networks, every byte counts. On today's wireless networks, latency is usually more of an issue than data transfer rate, so you may notice the larger message size of XML versus a binary format.

5.2.2 Parser Roundup

If there's a slogan for XML parsers in the MIDP world, it might be "Don't Supersize Me." Parsers are traditionally bulky, featuring lots of code and hefty runtime memory requirements. In MIDP devices, the memory available for code is usually small and individual applications may have a maximum code size. The Motorola iDEN phones, for example, have an upper limit of 50 kB on the size of a MIDlet suite JAR file. Aside from

code size, the amount of memory available at runtime is also small. What you need is a parser that's designed to be small and light.

Open source parsers are attractive because they give programmers lots of control. It is possible to customize a parser for additional features, or to fix the parser if it has bugs.

There are three fundamental parser types, according to the needs of the application and the types of documents the programmer expects to be parsed.

1. A *model* parser reads an entire document and creates a representation of the document in memory. Model parsers use significantly more memory than other types of parsers.
2. A *push* parser reads through an entire document. As it encounters various parts of the document, it notifies a listener object. (This is how the popular SAX (Simple API for XML) API operates.)
3. A *pull* parser reads a little bit of a document at once. The application drives the parser through the document by repeatedly requesting the next piece.

The following table summarizes the current offering of small XML parsers that are appropriate for mobile applications.

Name	Size	MIDP support	Type
ASXMLP 020308	6 KB	Yes	Push, model
KXML 2.0 alpha	9KB	Yes	Pull
KXML 1.2	16KB	Yes	Pull
MinML 1.7	14KB	No	Push
NanoXML 1.6.4	10KB	Patch	Model
TinyXML 0.7	12KB	No	Model
Xparse-J 1.1	6KB	Yes	Model

It's fairly simple to incorporate a parser into your MIDlet suite using the J2ME Wireless Toolkit. If the parser is distributed as source code *.java* files, you can place these files into the *src* directory of your J2MEWTK project. If the parser is distributed as a *.jar* or *.zip*

archive of *.class* files, you can place the archive in the *lib* directory of the J2MEWTK project.

The parsers shown in the table represent the current offerings in the MIDP 1.0 world. Standardization efforts are underway and the landscape is shifting rapidly (JSR 118, MIDP Next Generation and JSR 172, J2ME Web Services Specification).

5.2.3 Performance Considerations

In this section I'll describe some optimizations you can use to make your MIDlet code run well in a constrained environment. The techniques described here apply to any MIDP development, not just XML parsing. The reason I'm describing them here is because the use of an XML parser is likely to make your code significantly bigger and slower; you will probably want to optimize your application before delivering it to users.

The optimizations presented here fall into three categories:

1. Runtime performance
2. User perception
3. Deployment code size

Achieving good runtime performance is related to your XML document design. On the one hand, it takes a long time to set up a network connection. This means you should make each document contain as much useful data as possible. You might even want to aggregate documents on the server side and send one larger document rather than several smaller ones. On the other hand, the data transfer rate is slow. If you make your documents too large, the user will be left waiting a long time for each document to be loaded. In the end, you will need to find a balance between avoiding connection setup times and minimizing download wait times. One thing is for sure: XML documents that are sent to a MIDlet should not contain extra information. You don't want to waste precious wireless bandwidth transferring data you will only throw away.

Another way you can improve your application is to improve the user experience. This is not really an optimization--you're not making anything run faster or leaner--but it makes the application look a lot better to a user. The basic technique is simple: parsing, like network activity, should go in its own thread. (For several strategies for network threading, see

Networking, User Experience, and Threads .) You don't want to lock up the user interface while the MIDlet is parsing an XML document or reading the document from the network. Ideally, you can allow the user to perform other offline tasks *at the same time* that network activity and parsing is occurring. If that is not possible, you should at least try to show parsed data as soon as it is available. Note that you will need a push or pull parser to accomplish this; a model parser won't give you any data until the entire document is parsed.

Finally, you may be concerned about the size of your MIDlet suite JAR. There are two reasons this might be a problem. As I mentioned, there's not much space on MIDP devices, and carriers or manufacturers may impose limits on your code size. Second, users may download your application over the wireless network itself, which is slow. Making the MIDlet JAR small will minimize the pain of downloading and installing your software.

What's in the MIDlet suite JAR, and how can you reduce its size? The MIDlet suite JAR contains classfiles, images, icons, and whatever other resource files you may have included. Assuming you've removed all the resources you don't need, you are now ready to use something called an *obfuscator* to cut down on the classfiles.

Not all obfuscators are equal, but an obfuscator usually includes some of the following features:

1. Removes unused classes
2. Removes unused methods and variables
3. Renames classes, packages, methods, and variables
4. Adds illegal stuff to classfiles to confuse decompilers

Features 1, 2, and 3 are fine and will reduce the size of your MIDlet suite JAR, sometimes dramatically. If you have incorporated an XML parser in your MIDlet project, there may be parts of the parser that your application never uses. An obfuscator is good for pruning out the stuff you don't need.

Watch out for feature 4. Obfuscators were originally designed to make it hard for other people to decompile your classfiles. Some obfuscators do nasty things to the classfiles in order to confound decompilers. This may mess up either the class preverifier or the MIDP device's classloader, so avoid this feature if possible.

6. The HARP Project

This chapter presents the pre-existing system with focus on the Applet client application and on the way XML is used as transport for any data within the HARP system. XML holds a crucial role in the HARP platform. Understanding this role helps understand the needs and requirements the mobile client had to meet in order to be part of the platform, without having to cut down any UI features if possible.

6.1 Introduction

In our days, applications that collect data from various data sources and make them available over the Internet rely on a typical architecture that consists of four logical layers. The bottom layer is the *database layer* where all application data are stored. On the top of this layer is the *data objects layer* that practically defines the object representation of the database information. However, those objects do not contain any kind of business application logic. This logic is encapsulated in the next layer, the *business objects layer*. Finally, the top most layer – the *presentation layer* – handles all issues related to how this information is going to be presented to the end user.

Building, setting up and maintaining a business application that is based on the previous model requires extended database and programming knowledge. Furthermore, possible changes in the types of information exchanged in the context of the business application

means having to deal with a large amount of non-trivial and time-consuming tasks. To elaborate on that, a sample list of issues that an administrator of such an application has to deal with follows:

1 How should he structure the information exchanged in the context of the application and how this information model will be made as flexible as possible?

2 In what ways this information is going to be presented to the end user?

3 How will he add access control policies and in what way those policies are going to be linked to the information model?

4 How this information is going to be stored in the database(s) or how the database(s) structure(s) is (are) going to be linked to the information model?

5 How can he link a new information model to an already existing legacy database?

6 In what ways communication and access control related security would be guaranteed?

In this paper we will present an integrated platform that provides an end-to-end solution to those and many other issues that can arise in the design, set-up and maintenance of such a business application. In our approach, the three upper layers (*presentation, business object* and *data objects*) are integrated under a common framework provided by XML structures. So, development, maintenance and security are concisely handled in an implementation neutral way. On the other hand, the actual execution is clearly separated between client, server and database in a well-defined overall three-tier architecture operating in a state of the art security infrastructure environment.

6.2 Platform Overview

The platform design follows contemporary trends. Object-oriented technology and component-based architecture aspects have been followed in order to ensure adaptability, modularity and extensibility. Platform's structural components are categorized into two logical groups:

a) Application execution components (runtime environment)

b) Application set-up components (set-up environment)

Concerning the platform implementation, several technologies have been used each one having its benefits and drawbacks. Summarizing and reasoning the user of each technology:

1 End-to-end use of the Java language and its related technologies (Java Servlets, JDBC) in order to benefit from object-oriented design principles and cross platform functionality.

2 Use of the Extended Mark-up Language (XML) for creating the data structures involved in the context of each business application and as a means of communication between the different platform components.

3 Possible use of smart cards (e.g. Java Card technology) as originator of authentication, role assignment and guarantor of secure operation.

4 Security embedded into the application through appropriate attributes attached to each user interface component. Mapping of roles of authenticated users onto these attributes.

5 Use of the Secure Sockets Layer and use of XML Digital Signing, as standard security services at the communication level, beyond the application level security mentioned above.

6.3 Platform Architecture

6.3.1 Runtime Environment

The platform’s runtime environment consists of four architectural components and three data structures as shown in the following figure:

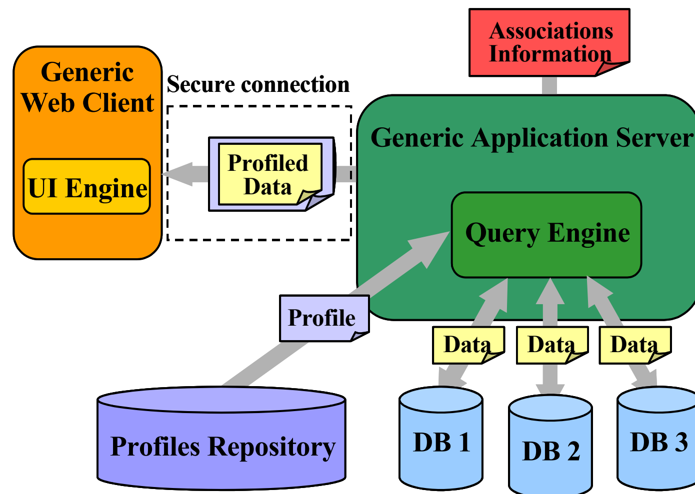


Figure 6-1. HARP's runtime environment.

On the server side, the *Generic Application Server* retrieves and assembles information from one or more data sources - databases depending on the configuration of the platform's instantiation for the specific business application and the respective profile of the logged user. The query engine of the *Generic Application Server* is able to retrieve information from any (and possibly several) database(s) as long as the application administrator configures it appropriately using the *Administration Tool*, presented in the following section. A second tool, the *Policy Tool* (presented in following section as well) enables the creation of the user profiles involved in the context of the business application. Profiles are stored in the *Profiles Repository*.

On the client side, The *Generic Web Client*, using an internal user interface engine visualizes the information collected by the *Generic Application Server* according to the presentation and access control rules entailed in each user profile. The *Generic Web Client* is implemented as a Java applet that can be downloaded on any browser.

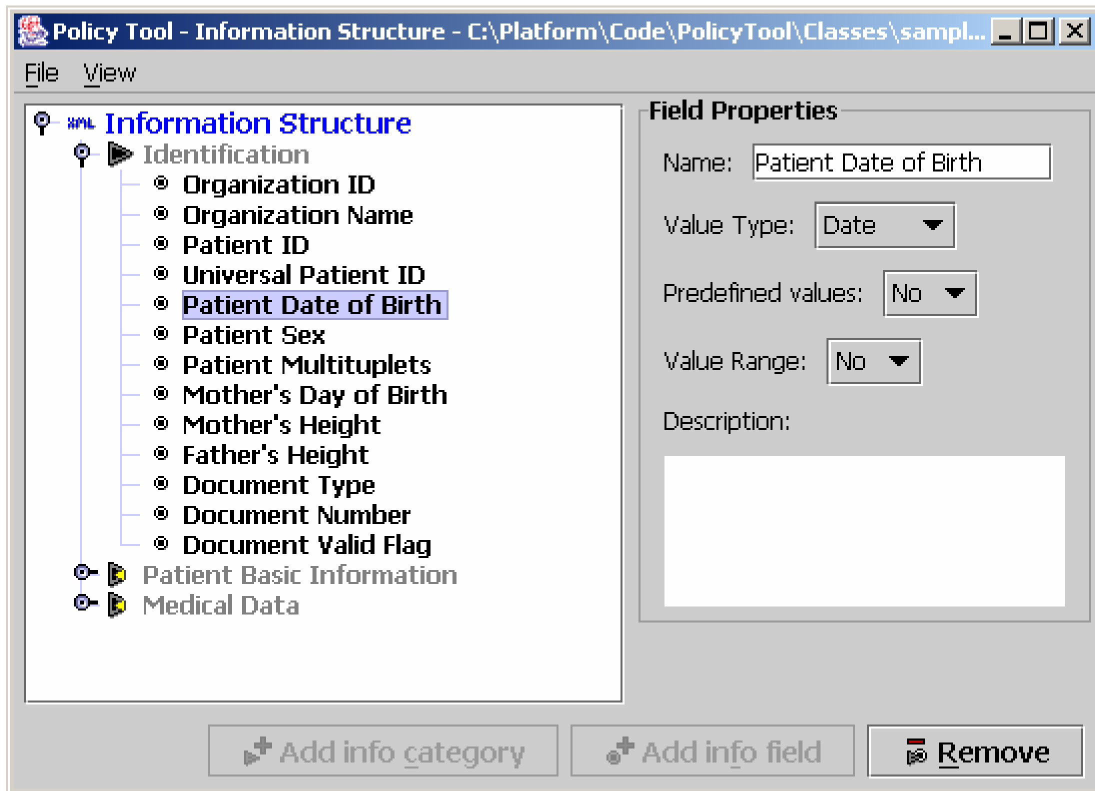
6.3.2 Set-up environment

The set-up environment of the platform consists of two tools, whose names have already been mentioned in the previous section: the *Policy Tool* and the *Administration Tool*. Practically, the instantiation of a business application involves the execution of three steps:

1 Step 1: Information Structure creation

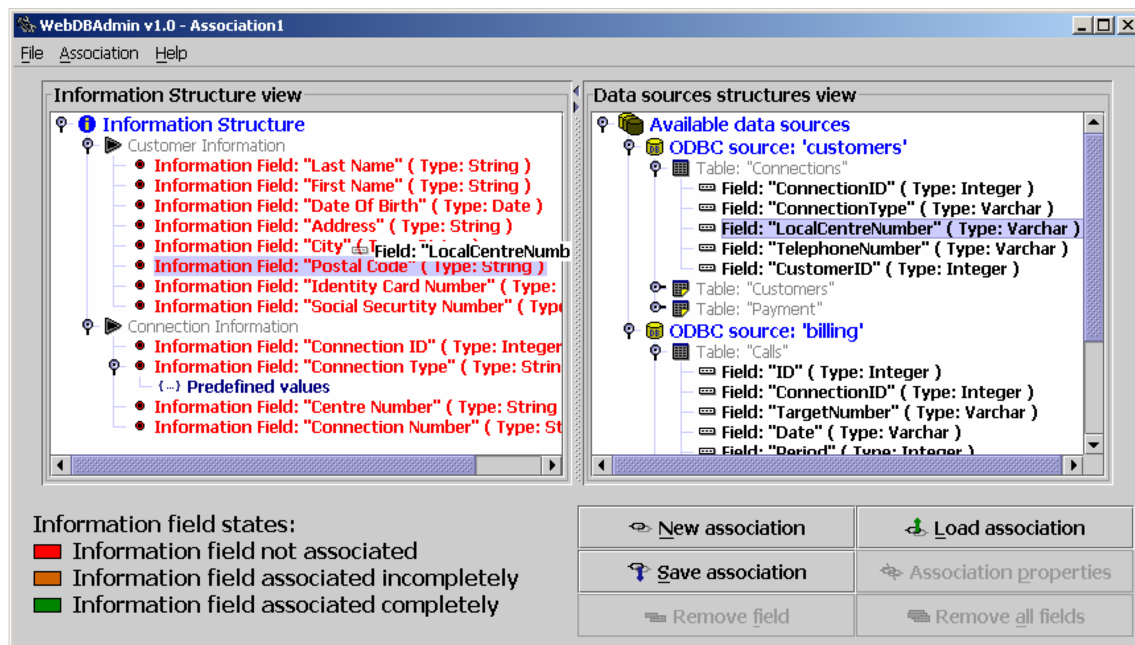
The application administrator defines in a structured way all the information that will be available to the end user in the context of the specific business application. This is

achieved with the use of the *Policy Tool*, which helps the administrator construct trees that contain information categories and information fields and define their respective properties. In this step, the administrator defines how information is going to be presented to end-users, however he does not provide details on how this information is going to be retrieved from the business application data sources.



2 Step 2: Associations definition

The application administrator defines how the *Generic Application Server* is going to navigate the application data sources in order to retrieve the data relative to the information structure defined in the previous step. The *Administration tool* offers an abstraction layer that lets the application administrator associate information fields with database fields and define database navigation rules in a visual way. No SQL coding is required.



3 Step 3: User profiles creation

User profiles are actually sub-structures of the information structure created in Step 1, enhanced with access control and visualization information. The application administrator has to consider the types of users that are going to access the business application, decide what kind of data each type is allowed to see and create the respective views (profiles), using the *Policy Tool*.

After having completed the previous steps the instantiation of the *Generic Application Server* follows automatically according to the specification and information provided in the steps above.

6.3.3 Data Structures

Trying to elaborate more in the nuts and bolts of the platform, we introduce the data structures that control the instantiation and the overall functionality of the platform including its application-level security features.

6.3.3.1 Information structure

Each business application developed in the context of the platform is based on a starting template, which describes all kinds of information that can possibly be presentable and manipulated by all potential users of the application possibly assuming different roles. The information structure is expressed using XML. A sample DTD (Document Type Definition) follows:

```
<!ELEMENT informationstructure (informationcategory +)>
<!ELEMENT informationcategory (informationcategory +|informationfield+)>
<!ELEMENT informationfield EMPTY>
<!ATTLIST informationcategory
    name CDATA #REQUIRED
    abstract (true|false) #REQUIRED>
<!ATTLIST informationfield
    name CDATA #REQUIRED>
```

Listing 1: Information structure DTD

The information structure model is based on the concept of information fields belonging to information categories. For each business application an instantiation of the above structure has to be created. This data structure serves as a general template for the generation of all other data structures, which are involved in the specific application.

6.3.3.2 Profile structure

The *UI Engine* of the *Generic Web Client* uses this XML structure in order to create the user interface that will be presented to each user. This P*rofile structure* contains information about what kind of data a user can view and what kind of actions the user can perform on that data. Furthermore, it contains information about how that data are going to be presented. Thus, the *Profile structure* contains presentation and access control information. All *Profile structures* are subsets of the *Information structure* with the addition of extra information concerning access control and presentation. Here is a relevant DTD:

```
<!ELEMENT informationstructure (informationcategory +)>
<!ELEMENT informationcategory (informationcategory +|informationfield+)>
<!ELEMENT informationfield EMPTY>
<!ATTLIST informationcategory
    name CDATA #REQUIRED
    visualcomponent CDATA #REQUIRED
    abstract (true|false) #REQUIRED>
<!ATTLIST informationfield
    name CDATA #REQUIRED
    visualcomponent CDATA #REQUIRED
    haspredefinedvalues (True|False) #REQUIRED
    editable (True|False) #REQUIRED>
```

Listing 2: Profile structure DTD

6.3.3.3 Data structure

This is the application's 'data carrier'. It contains the actual information that is transferred back and forth between the client and the server during a session. The *Data structure* is also a subset of the *Information structure*. A sample DTD follows:

```
<!ELEMENT informationstructure (informationcategory +)>
<!ELEMENT informationcategory (informationcategory +| informationfield +)>
<!ELEMENT datafield EMPTY>
<!ATTLIST informationcategory
    name CDATA #REQUIRED>
<!ATTLIST informationfield
    name CDATA #REQUIRED
    value CDATA #REQUIRED>
```

Listing 3: Data structure DTD

The *Data structure* passes from the server to the client and vice versa. When the structure comes to the client side, the *Generic Web Client* reads its structure and fills the respective fields of the user interface (that has been previously created with the *Profile structure*). When it comes to the server side, the *Generic Application Server* reads the structure and stores the respective values in the appropriate database fields.

6.3.3.4 Associations information structure

This XML structure entails the adaptability to use the platform with a large family of legacy databases. Specifically, it contains information concerning the association of the data transferred between the client and the server (*Profile structure* fields) and the database fields of the application data sources. The *Associations information structure* contains a description of the database(s) structure(s) and the relative mappings between database tables and fields and *Profile structure* fields. A sample DTD follows:

```
<!ELEMENT databasestructure (associations)>
<!ELEMENT associations (association+)>
<!ELEMENT association (datafield,query)>
<!ATTLIST association
    type CDATA #REQUIRED>
<!ELEMENT datafield EMPTY>
<!ATTLIST datafield
    name CDATA #REQUIRED
    hierarchy CDATA #REQUIRED
    type CDATA #REQUIRED
    haspredefinedvalues (true|false) #REQUIRED
    hasvaluerange (true|false) #REQUIRED>
<!ELEMENT query (step+)>
<!ELEMENT step (selection,condition)>
<!ELEMENT selection (databasefield)>
<!ELEMENT condition (databasefield)>
```

```
<!ATTLIST selection
    type CDATA #REQUIRED
    linkid CDATA #REQUIRED>
<!ATTLIST condition
    linkid CDATA #REQUIRED>
<!ELEMENT databasefield EMPTY>
<!ATTLIST databasefield
    name CDATA #REQUIRED
    hierarchy CDATA #REQUIRED
    type CDATA #REQUIRED>
```

Listing 4: Associations information DTD

The *Associations information structure* is created once for each particular business and loaded by the *Generic Application Server* at start up. Once the latter has loaded the structure, it acquires the necessary knowledge in order to navigate the database(s) and process every requested query.

The following figure provides a graphical presentation of the interrelations between the data structures.

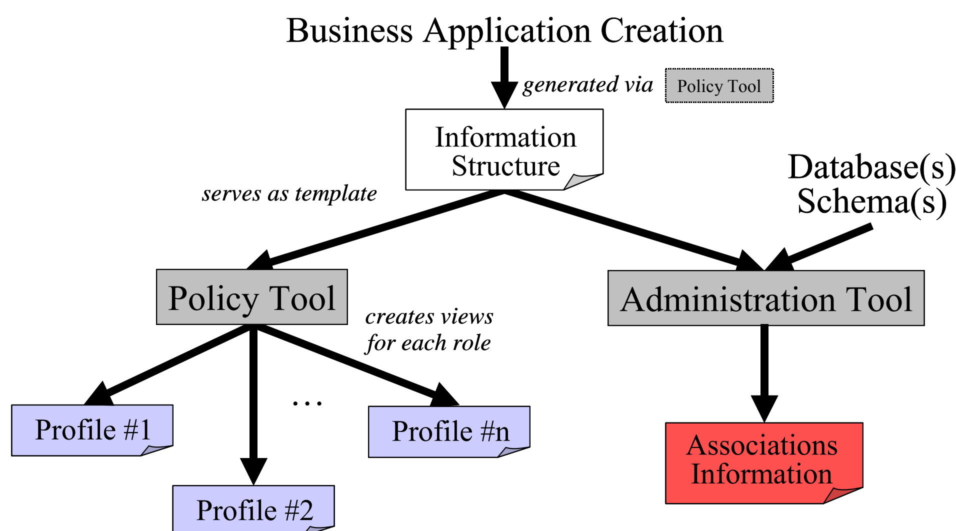


Figure 6-2. Relations between data and metadata structures available in HARP platform.

6.3.3.5 Platform security

The platform's security features are applied in various layers. In the application layer, the platform guarantees security through authentication procedures (username/password pairs, smart cards, certificates) and access control procedures (*Profile structures*). Furthermore, the use of downloadable client functionality (dictated from the server) prevents the disclosure of information concerning the business application logic (e.g. through caching

and disassembling of client Java classes). In the communication layer, information exchange is done over Secure HTTP, thus guaranteeing data integrity and encryption. Information exchange can be enhanced further by the use of XML digital signing, a feature that guarantees non-repudiation of data sent back and forth.

6.3.4 Platform applicability

The concept of the platform is to provide an easy, effective and dynamic way for creating business applications that publish enterprise data to the web in a secure and controlled way. An interesting applicability field is the medical sector and specifically the domain of healthcare records e.g. medical applications that perform remote access to patient records. This example has actually served as a motivation in trying to develop such a solution.

The generic nature of the *Generic Application Server* enables the integration of any legacy database without having to write a single line of SQL code. Even if changes occur in the database layer of a business application, the adaptation of the *Generic Application Server* to the new data sources standards is just some clicks away, thanks to the *Administration Tool*.

On the client side, the platform provides a generic client, whose presentation and access control behaviour is totally dictated by the application server. Therefore, using the *Administration Tool*, application administrators can create profiles and customise the views of the business application end users or user groups.

As a conclusion, the platform presented in this paper can be characterized as a lightweight and fully self-contained solution for opening up enterprise information to the Internet in a flexible, dynamic, and controlled way.

HARP platform is actually an end-to-end solution for developing, setting up and maintaining a database business application with high demands on issues such as security, dynamic configuration, adaptability and flexibility. However, the software market demands for flexible, adaptable and secure business application development environments grow

continuously. In addition to this, new standards and APIs emerge day after day and the old ones become quickly obsolete – especially in the domains of XML and Java technology. These facts serve as a motivation on thinking how this platform can be extended in order to support more features and be able to cover more business and user needs without, however, sacrificing important features such as its dynamic nature and its ease of use.

6.4 The Harp Applet

In figure 6-1 it is shown how the Harp Applet functions. Using Harp's Administration Tool, specialist users can define the structure of the information to be presented to the user, thus defining the structure of the User Interface. The server feeds the client with the appropriate ViewXML (XML bearing the data to be viewed by user) and the UI Engine of the client builds the GUI. Naturally, XML data are all part of the typical for any XML document, tree-like structure. Therefore, the GUI has to represent the relations between information fields. In other words, when it comes to data retrieved from a data base, possible types for information elements can be actual information fields or abstract fields which just categorize actual information fields (Information categories and information fields). Taking a look at the appropriate XML DTD's presented above would help to get the idea.

In J2SE the Swing package is a set of classes, which includes a group of features to help people into building GUI. Tabbed panes, radio buttons, lists, and check boxes were most of the Swing features used to build the Harp Applet. A screenshot of the HARP applet follows (figure 6-3).

Categories appear as tabs on top of GUI panel, while –according to the category selected-fields appear along with their values. According to their range of values and value types, field value elements can be text fields, numeric masks or lists with predefined values. Faded values indicate that the user cannot modify the field's value. All these data and metadata (like the editable attribute for fields) pass to the client via the XML. The server ensures that the ViewXML is filled with the proper data (by proper querying) and this data is presented according to the data type and the role of the user.

After changes, the user can submit the data. At this point, the client builds the appropriate data XML, which returns to the server all changes made by user.

HARP Applet ver. 2.0 - View document

Male specific Clinical Results | Therapy | Treatment Control | Meta

Patient Identification | Patient Basis Information | Medical History | Clinical Results

Field	Value
Universal Patient ID	13
Organisation ID	UHM
Patient ID	12
Document Version	0
Document ID	60
Caryotype Provision	yes
Caryotyping	46, XY
Genetic Deficiency Detection	yes
21-OH-Defect	salt waster
Molecular Findings Explanation	
Further Diagnostic Confidence of Genetic Deficiency	yes
Further Diagnostic Confidence Comment	
Genital Type According to Prader	Prader Stage 1
Prenatal Diagnosis Provision	yes
CAH Newborn Screening	yes
CAH in Family/Relationship	yes
Explanation of Family/Relationship Relation	
Start of Therapy	
Further Clinically Relevant Diseases	no
Clinically Relevant Diseases Comment	
Mother's Date of Birth	1958-03-02
Mother's Height	1.7
Father's Height	1.9

Submit | Back

Figure 6-3. The HARP applet User Interface.

All this procedure holds the dynamic nature that empowers the entire HARP platform, so it was considered a standard for the mobile client as well:

Server sends appropriate XML.

4 Client receives, parses data and UI Engine is initiated.

5 GUI is built.

6 User browses through data and adds/modifies data if he wishes.

7 User submits data.

8 Client builds answer-XML (From GUI → DOM →XML).

9 XML is sent back to server.

The implementation of this procedure by the mobile client is the subject of the next chapter.

Presented here is an example of an XML that the server sends to the client. It bears most of the features mentioned above.


```
 <?xml version='1.0' encoding='utf-8'?>

<xmldata type="DC_Server_Respond_PI_Data">

  <datacategory
        name="Patient Basic Information" abstract="false">
        <datafield
        name="Karyotyping Provision"
        value="Yes"
        type="String"
        haspredefinedvalues="true"
        predefinedvalues="Yes,No,Not Defined"
        editable="true">
      </datafield>
      <datafield
        name="Karyotyping"
        value="46 XX"
        type="String"
        haspredefinedvalues="true"
        predefinedvalues="46 XX,46 YY"
        editable="true">
      </datafield>
      <datafield
        name="Genetic Deficiency Detection"
        value="No"
        type="String"
        haspredefinedvalues="true"
        predefinedvalues="Yes,No,Not Defined"
        editable="true">
      </datafield>
   </datacategory>
  </xmldata>
```

7. XMIDlet, the MIDP Client for HARP

This chapter presents the MIDP client application developed during the elaboration of this diploma thesis, providing instructions and examples of use. It also reviews the design decisions that were taken according to the needs of the system on the one hand and to the limitations imposed by J2ME on the other.

7.1 The MIDP client for HARP platform

During the HARP project - as presented in the previous chapter- a platform was developed where GUI and access to databases were both dynamically created. This was achieved via end-to-end use of XML, along of course with the programming innovations and appropriate tools HARP programmers introduced. HARP developers decided to use the platform for presenting and handling medical data since issues with great importance in the medical world such as distinct and variable user roles, privacy, and security, set a challenging scenario. Using the administration tools and XML, data is retrieved and presented appropriately in a dynamic way. At this point, data is structured and accordingly to this model - structure, the GUI is also structured. At the runtime, the server retrieves data and feeds the client with XML documents bearing the data to be presented along with the

structure of GUI to be created. This was realized so far in a typical applet application using the very popular Java Swing package.

A mobile client is always a valuable extension to any platform. However as presented on chapter 5 “J2ME and XML”, XML, web-centric and multi-tier architecture systems make the integration of a mobile client even more appealing. The cell-phone is evolving into a powerful tool for many professionals and that includes doctors and medical staff. In the medical world the option to obtain and handle patient data anywhere is sometimes invaluable.

All these issues mentioned set the needs for creating the mobile client for HARP, named XMIDlet.

7.1.1 Specifications of XMIDlet

XMIDlet was designed and developed according to the Mobile Information Device Profile version 1.0. It is therefore also compatible with MIDP 2.0 as well, though for compatibility’s sake (since most cell phones implement MIDP 1.0) no further features of version 2.0 were used. Version 1.0 after all provided all features needed regarding GUI and networking.

The size of the XMIDlet JAR file is 47,7 KB. Obviously, size is an important issue in the mobile device environment. In general, most MIDP applications size range from 3KB to 60 KB, with graphics and multimedia boosting the size upwards. In our case, the rather large size of the middlet can easily be explained by another typical reason for a rather large middlet size: XML parsing. As shown on chapter 5, to handle XML parsing an application will have to use on the runtime environment classes of an XML parser application. In our case, the JAR file of the middlet embeds the classes of the Kxml Parser v1.2, a fact that increases the size of the JAR tremendously. More on the parser issue will be presented soon.

XMIDlet was developed in the J2ME Wireless Toolkit Integrated Development Environment (IDE) and tested mostly on the embedded simulator. However, it was also tested and debugged for use in real devices. Specifically, it was tested on the Sony Ericsson T610 cell-phone as well as on NOKIA 6610 phone. Both phones implement the MIDP 1.0 and were GPRS (General Package of Radio Services)-enabled.

7.1.2 XMIDlet networking

As its static counterpart, the XMIDlet is a client application that must have access to the remote server and be able to establish communication in the form of requests and responses (Chapter 3, section 3.3 Networking). The data is transported in the form of XML documents, thus the client receives and sends text through the network connection.
Using the HTTP protocol allows XMIDlet to be portable across all mobile information devices, since all MIDP 1.0 implementations support the HTTP protocol.
Moreover, HTTP can be implemented using both IP protocols (such as TCP/IP) and non-IP protocols (such as WAP and I-mode). In the latter case, the device would have to utilize a gateway that could perform URL naming resolution to access the Internet. The importance of this HTTP benefit is that applications will run on any MIDP device, whether it is a GSM phone with a WAP stack, a phone with I-mode, a Palm VII wireless, a handheld device with Bluetooth, or a GPRS enabled cell-phone.
Naturally, in order to test the XMIDlet application, it was necessary to also develop a server application (a simple HTTP Java Servlet). The HTTP Server (Web Server) our server application run on was the Apache Tomcat 4.0 which was preferred since a. it is absolutely free to download and use and b. it is very reliable and more than adequate for our needs.
The Servlet was developed for testing reasons only, resulting into something very simple with the idea that it should not provide any special handling for the mobile client. On the contrary, the task of adjustment and successfully handling the current XML model was appointed fully on the client side. The idea was after all to develop a client for a preexisting platform, while also examining just how much powerful are MIDP application when it comes to XML parsing. XMIDlet is presented here proudly as the successful result of this research and development.

7.1.3 Parsing XML

7.1.3.1 SAX and DOM

When it comes to parsing XML documents and handling the parsed information there are two basic options regarding the way to manipulate data: the Document Object Model (DOM) and the Simple API for XML (SAX). The DOM specification, released by the World Wide Web Consortium, provides a set of objects, interfaces, methods and properties which are used to create documents and parts of a document, navigate through the document, editing, copying and removing parts as well as adding and modifying attributes. The basic idea of DOM parsing is to create at the beginning an Object Model representing the structure of the XML document – tree. After it is created, this tree model is stored in memory, in order to be available for the application in need of the XML's information.

On the other hand, SAX is used for the same task i.e. parsing, however it works quite differently. The idea in SAX is scanning serially through all the XML elements of the document while on the same time events are raised any time the parser finds something (e.g. the start of an element). This is how SAX works and it in the programming world it is called **event-driven**.

DOM, and the option of having the tree model in memory offer many more options for the application. The application can navigate through the tree structure freely, upwards or downwards and access any data in this structure since it possesses at all times a very detailed map of it. On the other hand with SAX one event follows the other and nothing is kept in memory. However when it comes to extracting a small amount of information from a large XML document SAX is naturally much faster and consumes less resources than DOM. The two approaches should therefore be regarded as complementary.

7.1.3.2 XMIDlet and XML

Since whether using DOM or SAX depends on the kind of data (and metadata) the application needs from the XML, let's examine the data model -provided via XML- and its use as far as the client side is concerned. The ViewXML that the server sends to the client provides all the data to be presented, according to the request of the client. For demonstration and testing purposes during the HARP project, tests were made through an example of a medical electronic health record application. Therefore the ViewXML actually carries such information to be presented on the client. Embedded in it is the structure of the data, which must also be represented graphically since the information is categorized. There

are also certain attributes that define certain properties. Predefined values and whether the information field is editable for this user role are metadata passed on within the XML. Therefore, building an appropriate GUI that successfully brings all this information to the user surely requires for the client to build the Object Model of the information structure and actually find a way of graphically representing this model. To achieve this, the parser should create the DOM from the document and the application should create a graphical representation of the DOM for the user, providing the appropriate navigating and editing capabilities.

However, at the time the development of XMIDlet took place, there was no available free DOM parser. However by using the Vector class inherited by the J2SE, it became possible to create and store all necessary node information available in the XML. Thus a Document Object Model is created for our purposes, as a collection of Vectors.

7.1.4 Resource constraints

Besides the obvious constraint of limited APIs and class libraries, as dictated by CLDC and MIDP, when it comes to creating a client for cell-phones and mobile clients, the developer faces the problems of a small-sized screen and the lack of useful UI devices such as the mouse. Therefore, building a GUI for mobile clients providing -to the most possible extent- the same functionalities as its desktop counterpart was yet another challenge.

As shown on the previous chapter, the desktop client used the Java Swing package for GUI building. Such components (such as the tabbed pane) are not available in the MIDP. Instead, Lists, simple Text boxes and radio buttons were used. Despite the limited selection of GUI elements, the result turned out as a lightweight, easy to understand and handle, Graphical User Interface. The key for this successful result proved to be the tree-like structure stored and reflected onto the representation of data. This advantage comes as a natural consequence of using XML for exchanging data, storing at each time the corresponding Data Model (as a DOM).

As far as memory, storage and processing power constraints are concerned, using proper control structures the XMIDlet application avoids unnecessary use of processing and memory power, while the executable file remains relatively small (around 47 KB). In this file, the significant 78% of the size comes from library files i.e. the kxml parser. At the time of this writing, it is known that if a newer –released after the development of XMIDlet- parser was used, like kxml 2 parser, the total size of the executable file would drop to the very compact size of 20 KB.

7.1.5 The XMIDlet GUI

In the last two chapters the challenges and problems solved during the design and development of XMIDlet were described. Resource constraints on the one hand and requirements set by the HARP platform on the other hand leaded the development to the final solution. The reflection of this final solution, the GUI, is presented here.

Now follows a series of screenshots from the J2ME Wireless Toolkit version 2.3, representing a series of actions a user of XMIDlet would follow.

- **Login – Authorization – Welcome Screen**

To begin with, a user must undergo a certain authorization procedure, the simplest being to provide a valid username and password. Of course, AAA (authentication, authorization and accounting) and security issues were not in the scope of this thesis. However, wishing to embrace possible future work on these crucial matters, the XMIDlet GUI contains the appropriate interface. For our demo, a simple login was executed.

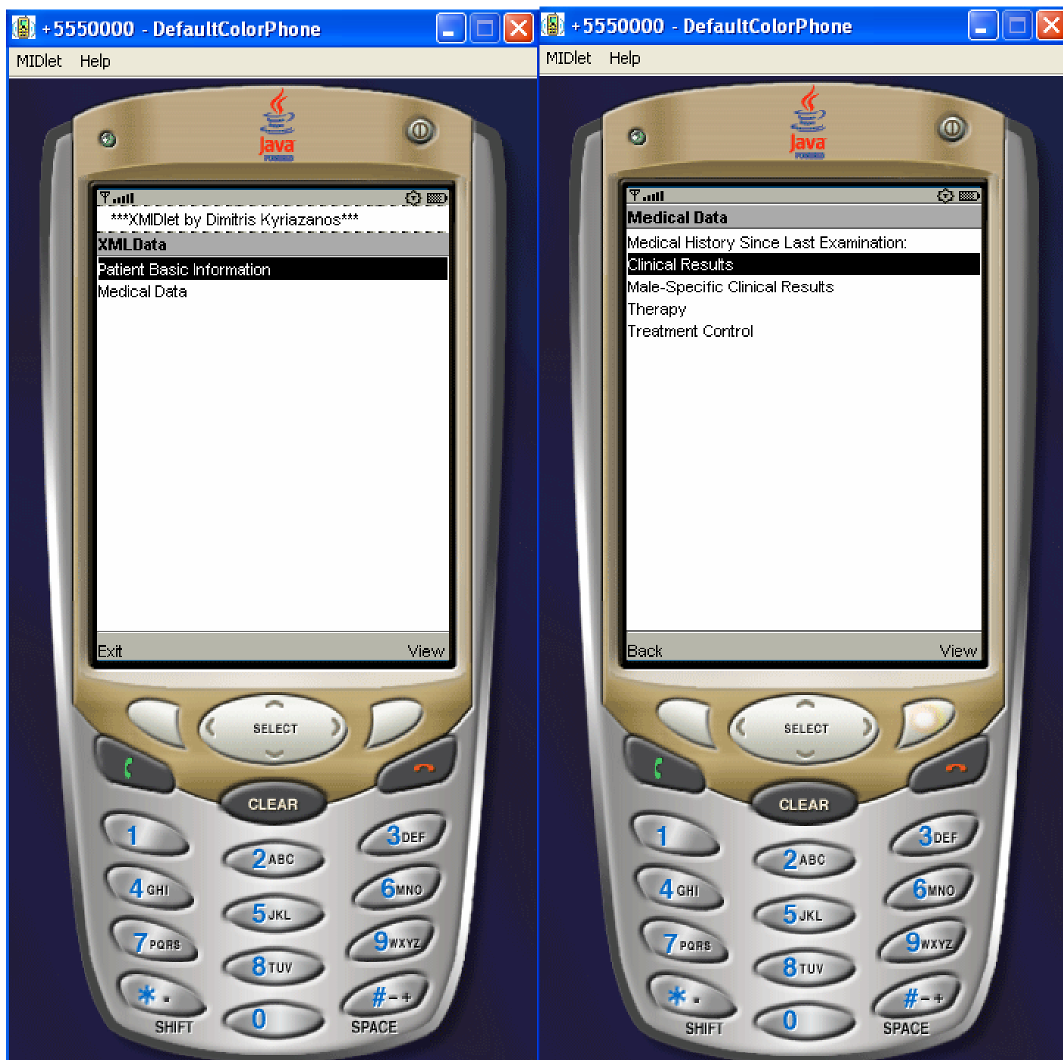
Login GUI & Validation Screenshots

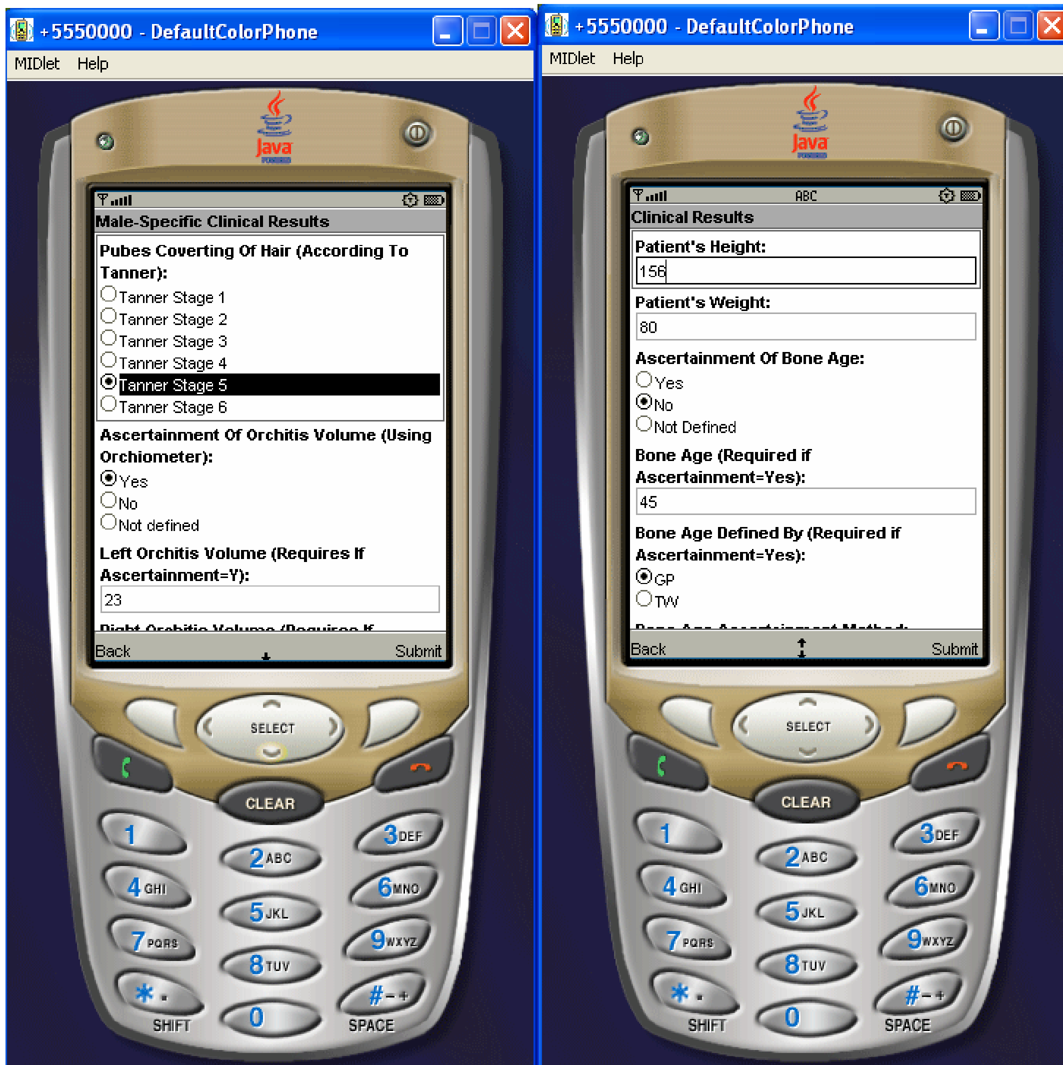
Welcome Screen

- **Main XMIDlet GUI : XML to GUI – Navigation**

Following the welcome screen, the user requests for a certain electronic health record of a patient, stored in a remote web server in XML form. The XMIDlet receives the XML and builds a corresponding interconnected Data Model, i.e. a tree-like structure representing the relations between data. This is then dynamically represented via use of appropriate GUI elements. The user browses through this graphical tree to upper (by selecting **Back**) and lower levels (by selecting **View** for a certain data category). At each time the category – 'father' of the data fields is written on the title bar above all fields, so as to remind the user of his position within the tree. Simply and practically the user can edit the data fields (if permitted), view values of interest to her and submit at any time her changes to the server.

Moreover, in order to ensure compatibility with all PDA's and mobile phones, the GUI is build so as to ensure scalability (up-down scrollbars, use of MIDP GUI elements and not self-drawn elements). Coloring plays no functional role in XMIDlet either, in order to keep black and white screens compatible and functional.



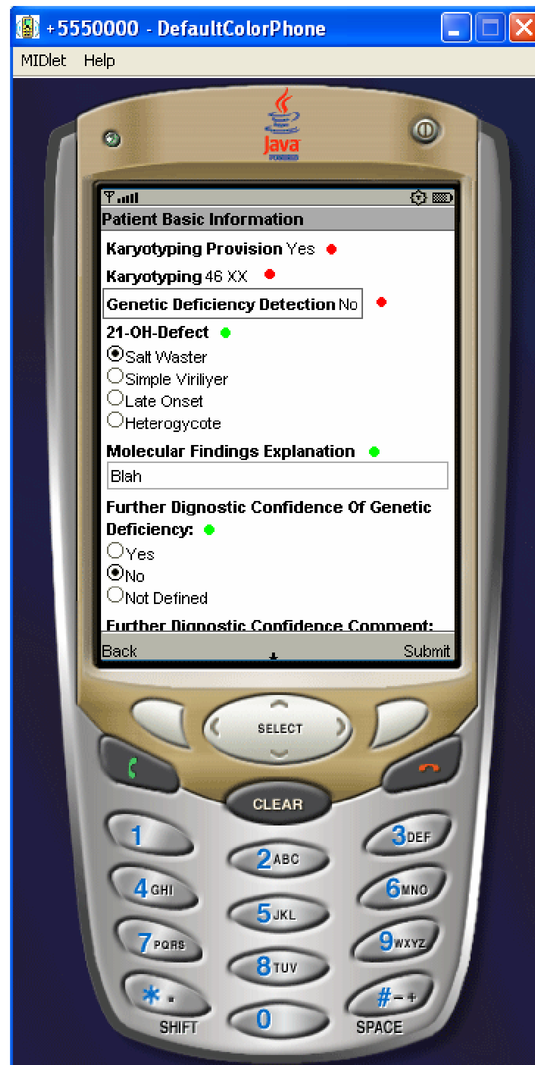


Navigation through the XML-generated GUI
Upper two screenshots: from root to Medical Data (abstract data category containing more data categories)
Lower screenshots: viewing data fields of Male Specific Clinical Results category and Clinical Results Category

Readers should have in mind that any XML (within the boundaries of a certain XML schema – set of guidelines) would be handled by the XMIDlet, by creating the tree structured Data Model and building the appropriate GUI in a dynamic way. Dynamic GUI building with use of XML technologies is a trend which offers many advantages and applications, compared with the traditional static GUI. A static GUI dictates that handling different structured data models is impossible, unless a new static GUI is created from the beginning. Therefore, XMIDlet succeeded to respect to full extent the HARP heritage, by providing to the system a totally generic XML-handling mobile client.

- **Editing rights and permissions**

As mentioned at the beginning of this section, the XMIDlet GUI is designed in a way to embrace future work on AAA and security issues. A most common issue is whether a certain authenticated user has the right to edit the values of a certain data field or just has viewing rights (or perhaps not even this). Such rights are embedded to the incoming XML from the server after a proper authentication-authorization procedure takes place. In XMIDlet these rights and policies result into restricting GUI functionalities to the user accordingly.

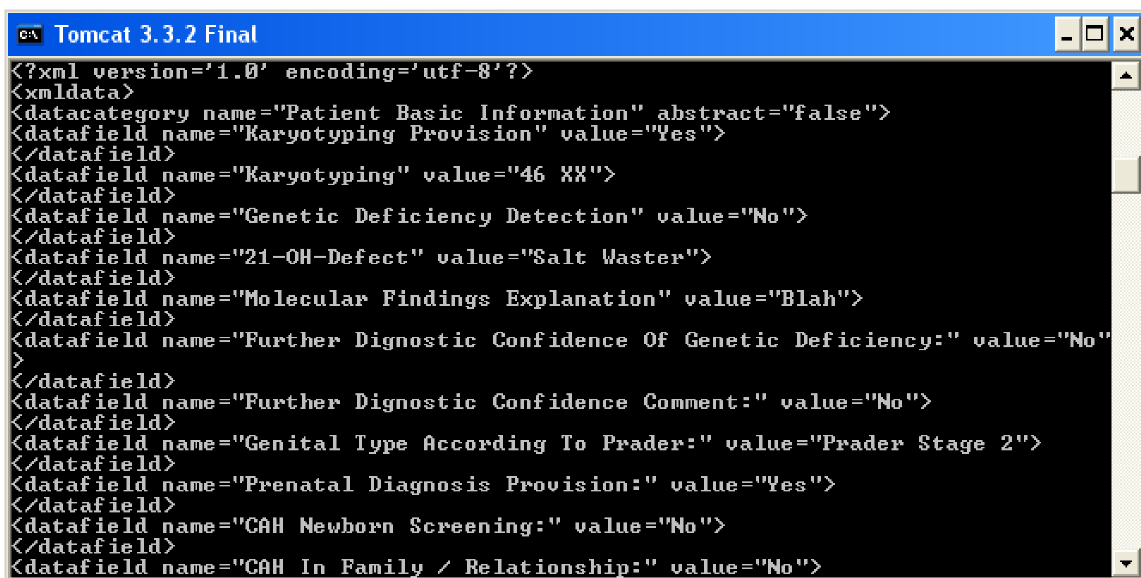


Editable and Non-editable Data Fields
The first three fields from top are non-editable
The rest are editable

➢ **XML generation – the full integration to the HARP platform**

When the submit option is selected, the XMIDlet initiates the procedure of submitting the changed data model back to the server. A reverse procedure is now executed: from the GUI

a Data Model is created and sent back to the server. However, in order to avoid the need for more resources, the Data Model isn't kept in memory. After all, it isn't necessary. Instead, the XML is created and sent line by line serially to the server, which is responsible to properly store all information. Successfully sending the data from the mobile client to the server via the same XML a desktop computer client would send, means that the XMIDlet is fully integrated into the HARP system. The server doesn't handle requests or data from mobile clients any differently than any other exchanges and interactions. And this means, that XMIDlet implementation fully meets the requirements set at the beginning of this thesis.



Apache Tomcat Web Server window: receiving the Data Xml from XMIDlet.

For readers interested in the form of the data transport used for this demo, the XML used in the transport, a part of the (rather large) XML file used is here presented.


```
<?xml version='1.0' encoding='utf-8'?>
<!--     HARP PROJECT- This is the document that the server sends back to the user
                when she is requesting a document
-->
<xmldata type="DC_Server_Respond_PI_Data">

        <datacategory
                name="Patient Basic Information" abstract="false">

                <datafield
```

```
			name="Karyotyping Provision"
			value="Yes"
			type="String"
			haspredefinedvalues="true"
			predefinedvalues="Yes,No,Not Defined"

			editable="false">
		</datafield>
		<datafield
			name="Karyotyping"
			value="46 XX"
			type="String"
			haspredefinedvalues="true"
			predefinedvalues="46 XX,46 YY"

			editable="false">
		</datafield>
		<datafield
			name="Genetic Deficiency Detection"
value="No"
			type="String"
			haspredefinedvalues="true"
			predefinedvalues="Yes,No,Not Defined"

			editable="false">
		</datafield>
			<datafield
			name="21-OH-Defect"
			value="Salt Waster"
			type="String"
			haspredefinedvalues="true"
			predefinedvalues="Salt Waster,Simple Viriliyer,Late Onset,Heterogycote"

			editable="true">
		</datafield>
		<datafield
			name="Molecular Findings Explanation"
			value="Blah"
			type="String"
			haspredefinedvalues="false"

			editable="true">
		</datafield>
			<datafield
			name="Further Dignostic Confidence Of Genetic Deficiency:"
			value="No"
			type="String"
			haspredefinedvalues="true"
			predefinedvalues="Yes,No,Not Defined"

			editable="true">
		</datafield>
...
			<datacategory
				name="Male-Specific Clinical Results" abstract="false">
				<datafield
			name="Pubes Coverting Of Hair (According To Tanner):"
			value="Tanner Stage 3"
			type="String"
			haspredefinedvalues="true"
```

```
        predefinedvalues="Tanner Stage 1,Tanner Stage 2,Tanner Stage 3,Tanner Stage 4,Tanner
Stage 5,Tanner Stage 6,"
        editable="true">
      </datafield>
            <datafield
        name="Ascertainment Of Orchitis Volume (Using Orchiometer):"
        value="Yes"
        type="String"
        haspredefinedvalues="true"
        predefinedvalues="Yes,No,Not defined"
        editable="true">
      </datafield>
      <datafield
        name="Left Orchitis Volume (Requires If Ascertainment=Y):"
        value="23"
        type="String"
        haspredefinedvalues="false"
        editable="true">
      </datafield>
      <datafield
        name="Right Orchitis Volume (Requires If Ascertainment=Y):"
        value="34"
        type="String"
        haspredefinedvalues="false"
        editable="true">
      </datafield>
...
      <datafield
            name="Comments (Required if Control Measures=Yes):"
            value="Yes"
            type="String"
            haspredefinedvalues="false"

            editable="true">
      </datafield>
        </datacategory>

    </datacategory>
</xmldata>
```

7.2 Conclusions – the Future of MIDP and XMIDlet

This thesis concludes here by summarizing the achievements and goals reached:

- In a **dynamic way**, without requiring the slightest change on the pre-existent system, a mobile client for PDAs and cell-phones with MIDP support was **integrated** into the multi-tier architecture. No special handling is required on the server side, as the mobile client communicates to the server in the exact same way the desktop computer client does, via use of XML.
- The MIDP client is designed and implemented as a **generic client**, i.e. the GUI is not static and therefore not bound to a specific application. Instead, the GUI is dynamically created according to the data model contained inside the received XML files. The XMIDlet can therefore be a client for any system which uses XML as data transport with minimum programming effort.
- Resource constraints issues were handled so as to produce a **lightweight yet fully functional** client. The GUI created is compatible even with the most simple black-and-white small-sized screen cell-phones. This compatibility and scalability is ensured via exclusive use of standardized GUI elements available from the Mobile Information Device Profile 1.0.

From the experience gained throughout this thesis, the following facts became clear:

- XML technologies offer a brave new world of capabilities when it comes into designing a multi-tier architecture system. Dynamic integration of new components and functionalities –or even a whole new system configuration- can be achieved by designing systems which benefit from this powerful tool.
- J2ME is a technology for mobile clients not powerful enough to bear the Standard or Personal Java Virtual Machine. The classes offered are but a few however with the magic touch of XML, small wonders can happen! With the rapid evolution of mobile devices however, J2ME could be in danger of extinction, since more and more devices get less and less "micro" when it comes to processing power, memory and

storage capacity. MIDP 2.0 growing support could be the answer for this scenario, along with the fact that J2ME offers a handy, compact platform for developing simple and lightweight mobile applications with networking capabilities.

Future work and expansions of the work done in this thesis could be:

1. Collaboration of the MIDP client with an appropriate Authentication, Authorization and Accounting module. Mobile Applications security is a hot topic concerning many research projects world-wide. Therefore a future work could be having the XMIDlet including or interoperating with an appropriate security component.
2. Cell-phones are evolving rapidly while processing powers, memory and storage capacity also increase. What used to be a mobile phone with messaging function is now a PDA with full multimedia capabilities. MIDP experts are aware of that, so the profile always moves into including more and more powerful classes. What used to be a limited selection of classes and a choice of primitive GUI elements in MIDP 1.0 has already evolved into multimedia support and many other new features (security, , animation graphics, networking), included in MIDP 2.0. As more and more cell-phones support MIDP 2.0 at this time, the XMIDlet could evolve –in the time where MIDP 1.0 becomes obsolete- into having more sophisticated GUI design and functionalities. However, the qualities of being lightweight and simple should always follow the design of such a client.

REFERENCES

[1] WIRELESS JAVA, BY QUSAY H. MAHMOUD, O'REILLY & ASSOCIATES.

[2] OPEN, FLEXIBLE AND PORTABLE SECURE WEB-BASED HEALTH APPLICATIONS, M.VLACHOS AND G. STASSINOPOULOS, PAPER INCLUDED IN THE MAGDEBURG EXPERT SUMMIT TEXTBOOK 'ADVANCED HEALTH TELEMATICS AND TELEMEDICINE', IOS PRESS/OHM.

[3] MOBILE BUSINESS AND INTERNET APPLICATIONS, DEITEL & DEITEL.

[4] JAVA IN A NUTSHELL, , BY DAVID FLANAGAN, O'REILLY & ASSOCIATES.

[5] BEGINNING XML, DAVID HUNTER, WROX PRESS.

[6] JAVA & XML, BRETT MC LAUGHLIN, O'REILLY & ASSOCIATES.

[7] XML ΚΑΙ ΣΧΕΤΙΚΕΣ ΤΕΧΝΟΛΟΓΙΕΣ, ΣΗΜΕΙΩΣΕΙΣ ΜΑΘΗΜΑΤΟΣ «ΔΙΑΔΙΚΤΥΟ & ΕΦΑΡΜΟΓΕΣ». Γ. Ι. ΣΤΑΣΙΝΟΠΟΥΛΟΣ, ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.

[8] JAVA HOW TO PROGRAM, DEITEL & DEITEL.

[9] THE HARP PROJECT WEB-SITE, BY ICCS-NTUA http://www.telecom.ntua.gr/~HARP/HARP/HARP.htm

[10] HARP IST-PROJECT FACT SHEET, http://dbs.cordis.lu/fep-cgi/srchidadb?ACTION=D&CALLER=PROJ_IST&QF_EP_RPG=IST-1999-10923

[11] THE J2ME PLATFORM WEB SITE, http://java.sun.com/j2me/

[12] THE MOBILITY DEVELOPMENT CENTER, SUN DEVELOPMENT NETWORK http://developers.sun.com/techtopics/mobility/

Appendix A:

CLDC classes inherited from J2SE

Appendix A offers detailed API information regarding classes inherited from Java 2 Standard Edition platform to CLDC. There are classes available from the java.io, the java.lang and the java.util packages.

Package java.io

Provides classes for input and output through data streams.

Interface Summary	
DataInput	The `DataInput` interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types.
DataOutput	The `DataOutput` interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream.

Class Summary	
ByteArrayInputStream	A `ByteArrayInputStream` contains an internal buffer that contains bytes that may be read from the stream.
ByteArrayOutputStream	This class implements an output stream in which the data is written into a byte array.
DataInputStream	A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way.
DataOutputStream	A data output stream lets an application write primitive Java data types to an output stream in a portable way.
InputStream	This abstract class is the superclass of all classes representing an input stream of bytes.
InputStreamReader	An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and translates them into characters.

OutputStream	This abstract class is the superclass of all classes representing an output stream of bytes.
OutputStreamWriter	An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are translated into bytes.
PrintStream	A `PrintStream` adds functionality to another output stream, namely the ability to print representations of various data values conveniently.
Reader	Abstract class for reading character streams.
Writer	Abstract class for writing to character streams.

Exception Summary	
EOFException	Signals that an end of file or end of stream has been reached unexpectedly during input.
InterruptedIOException	Signals that an I/O operation has been interrupted.
IOException	Signals that an I/O exception of some sort has occurred.
UnsupportedEncodingException	The Character Encoding is not supported.
UTFDataFormatException	Signals that a malformed UTF-8 string has been read in a data input stream or by any class that implements the data input interface.

Package java.lang

Provides classes that are fundamental to the Java programming language.

Interface Summary	
Runnable	The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread.

Class Summary	
Boolean	The Boolean class wraps a value of the primitive type `boolean` in an object.
Byte	The Byte class is the standard wrapper for byte values.
Character	The Character class wraps a value of the primitive type `char` in an object.
Class	Instances of the class `Class` represent classes and interfaces in a running Java application.
Double	The Double class wraps a value of the primitive type `double` in an object.
Float	The Float class wraps a value of primitive type `float` in an object.
Integer	The Integer class wraps a value of the primitive type `int` in an object.
Long	The Long class wraps a value of the primitive type `long` in an object.
Math	The class `Math` contains methods for performing basic numeric operations.
Object	Class `Object` is the root of the class hierarchy.
Runtime	Every Java application has a single instance of class `Runtime` that allows the application to interface with the environment in which the application is running.
Short	The Short class is the standard wrapper for short values.
String	The `String` class represents character strings.
StringBuffer	A string buffer implements a mutable sequence of characters.
System	The `System` class contains several useful class fields and methods.
Thread	A *thread* is a thread of execution in a program.
Throwable	The `Throwable` class is the superclass of all errors and exceptions in the Java language.

Exception Summary	
ArithmeticException	Thrown when an exceptional arithmetic condition has occurred.

ArrayIndexOutOfBoundsException	Thrown to indicate that an array has been accessed with an illegal index.
ArrayStoreException	Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.
ClassCastException	Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.
ClassNotFoundException	Thrown when an application tries to load in a class through its string name using the `forName` method in class `Class` but no definition for the class with the specified name could be found.
Exception	The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.
IllegalAccessException	Thrown when an application tries to load in a class, but the currently executing method does not have access to the definition of the specified class, because the class is not public and in another package.
IllegalArgumentException	Thrown to indicate that a method has been passed an illegal or inappropriate argument.
IllegalMonitorStateException	Thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.
IllegalThreadStateException	Thrown to indicate that a thread is not in an appropriate state for the requested operation.
IndexOutOfBoundsException	Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.
InstantiationException	Thrown when an application tries to create an instance of a class using the `newInstance` method in class `Class`, but the specified class object cannot be instantiated because it is an interface or is an abstract class.
InterruptedException	Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it.
NegativeArraySizeException	Thrown if an application tries to create an array with negative size.
NullPointerException	Thrown when an application attempts to use `null` in a case where an object is required.

NumberFormatException	Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.
RuntimeException	`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.
SecurityException	Thrown by the system to indicate a security violation.
StringIndexOutOfBoundsException	Thrown by the `charAt` method in class `String` and by other `String` methods to indicate that an index is either negative or greater than or equal to the size of the string.

Error Summary	
Error	An `Error` is a subclass of `Throwable` that indicates serious problems that a reasonable application should not try to catch.
NoClassDefFoundError	Thrown if the Java Virtual Machine tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the `new` expression) and no definition of the class could be found.
OutOfMemoryError	Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.
VirtualMachineError	Thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.

Package java.util

Contains the collection classes, and the date and time facilities.

Interface Summary	
Enumeration	An object that implements the Enumeration interface generates a series of elements, one at a time.

Class Summary	
Calendar	`Calendar` is an abstract base class for converting between a `Date` object and a set of integer fields such as `YEAR`, `MONTH`, `DAY`, `HOUR`, and so on.
Date	The class Date represents a specific instant in time, with millisecond precision.
Hashtable	This class implements a hashtable, which maps keys to values.
Random	An instance of this class is used to generate a stream of pseudorandom numbers.
Stack	The `Stack` class represents a last-in-first-out (LIFO) stack of objects.
TimeZone	`TimeZone` represents a time zone offset, and also figures out daylight savings.
Vector	The `Vector` class implements a growable array of objects.

Exception Summary	
EmptyStackException	Thrown by methods in the `Stack` class to indicate that the stack is empty.
NoSuchElementException	Thrown by the `nextElement` method of an `Enumeration` to indicate that there are no more elements in the enumeration.

Appendix B:

CLDC specific classes

Appendix B offers detailed API information regarding classes inside the CLDC which were purposed for the Generic Connection framework specifically.

Package javax.microedition.io

Classes for the Generic Connection framework.

Interface Summary	
Connection	This is the most basic type of generic connection.
ContentConnection	This interface defines the stream connection over which content is passed.
Datagram	This class defines an abstract interface for datagram packets.
DatagramConnection	This interface defines the capabilities that a datagram connection must have.
InputConnection	This interface defines the capabilities that an input stream connection must have.
OutputConnection	This interface defines the capabilities that an output stream connection must have.
StreamConnection	This interface defines the capabilities that a stream connection must have.
StreamConnectionNotifier	This interface defines the capabilities that a connection notifier must have.

Class Summary	
Connector	This class is factory for creating new Connection objects.

Exception Summary	
ConnectionNotFoundException	This class is used to signal that a connection target cannot be found, or the protocol type is not supported.