Chapter VII
Engineering Wireless Mobile Applications

Qusay H. Mahmoud
University of Guelph, Canada

Zakaria Maamar
Zayed University, UAE

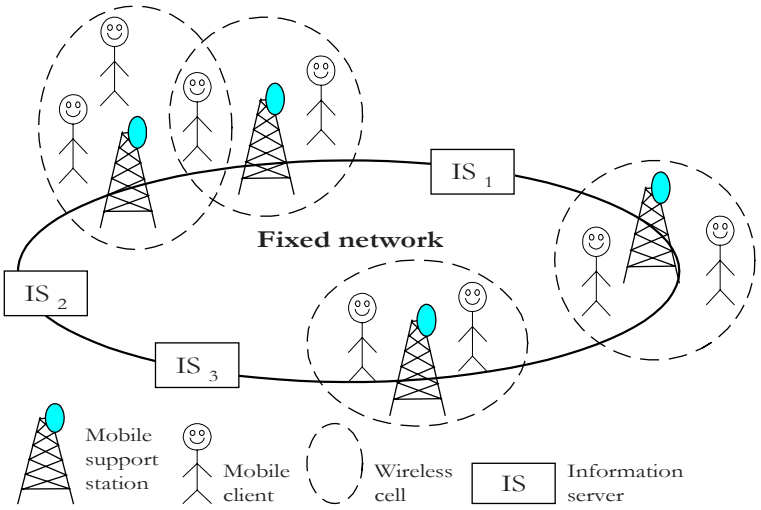
ABSTRACT

Conventional desktop software applications are usually designed, built, and tested on a platform similar to the one on which they will be deployed and run. Wireless mobile application development, on the other hand, is more challenging because applications are developed on one platform (like UNIX or Windows) and deployed on a totally different platform like a cellular phone. While wireless applications can be much smaller than conventional desktop applications, developers should think in the small in terms of the devices on which the applications will run and the environment in which they will operate instead of the amount of code to be written. This paper presents a systematic approach to engineering wireless application and offers practical guidelines for testing them. What is unique about this approach is that it takes into account the special features of the new medium (mobile devices and wireless networks), the operational environment, and the multiplicity of user backgrounds; all of which pose new challenges to wireless application development.

INTRODUCTION

The general mobile computing model in a wireless environment consists of two distinct sets of entities (Figure 1): Mobile Clients (MCs) and fixed hosts. Some of the fixed hosts, called Mobile Support Stations (MSSs), are enhanced with wireless interfaces. A MSS can communicate with the MCs

Figure 1. Mobile Computing Model



within its radio coverage area called wireless cell. A MC can communicate with a fixed host/server via a MSS over a wireless channel. The wireless channel is logically separated into two sub-channels: an uplink channel and a downlink channel. The uplink channel is used by MCs to submit queries to the server via an MSS, whereas the downlink channel is used by MSSs to disseminate information or to forward the responses from the server to a target client. Each cell has an identifier (CID) for identification purposes. A CID is periodically broadcasted to all the MCs residing in a corresponding cell.

A wireless mobile application is defined as a software application, a wireless service or a mobile service that can be either pushed to users' handheld wireless devices or downloaded and installed, over the air, on these devices. Such applications must work within the daunting constraints of the devices themselves:

- *Memory*: Wireless devices such as cellular phones and two-way pagers have limited amounts of memory, obliging developers to consider memory management most carefully when designing application objects.
- *Processing power*: Wireless devices also have limited processing power (16-bit processors are typical).
- *Input*: Input capabilities are limited. Most cell phones provide only a one-hand keypad with twelve buttons: the ten numerals, an asterisk (*), and a pound sign (#).
- *Screen*: The display might be as small as 96 pixels wide by 54 pixels high and 1 bit deep (black and white). The amount of information that can be squeezed into such a tight screen is severely limited.

In addition, the wireless environment imposes further constraints: (i) wireless networks are unreliable and expensive, and bandwidth is low; (ii) they tend to experience more network errors than wired networks; and (iii) the very mobility of wireless devices increases the risk that a connection will be lost or degraded. In order to design and build reliable wireless applications, designers need to keep these constraints in mind and ask themselves what impact do wireless devices with limited resources have on application design?

The motivation for this paper is provided in part by the above characteristics that form

some of the foundations for pervasive computing environments. Such characteristics pose several challenges in designing wireless mobile applications for mobile computing. This paper provides a detailed treatment of the impact of these characteristics on engineering wireless mobile applications and presents a systematic approach for designing them. In addition, it offers practical design techniques for wireless application design and development.

WIRELESS APPLICATIONS

Wireless applications can be classified into two streams (Beaulieu, 2002; Burkhardt et al., 2002):

1. *Browser-based:* Applications developed using a markup language: This is similar to the current desktop browser model where the device is equipped with a browser. The Wireless Application Protocol or WAP (http://www.wapforum.org) follows this approach (Open Mobile Alliance, 2005).
2. *Native applications*: Compiled applications where the device has a runtime environment to execute applications. Highly interactive wireless applications are only possible with the latter model. Interactive applications, such as mobile computer games, are a good example. Such applications can be developed using the fast growing Java 2 Micro Edition (J2ME) platform (http://java.sun.com), and they are known as MIDlets.

Another stream is the hybrid application model that aims at incorporating the best aspects of both streams above. The browser is used to allow the user to enter URLs to download native applications from remote servers, and the runtime environment is used to let these applications run on the device.

WAP Might be Dead, But What Did We Learn?

WAP and J2ME MIDP solve similar problems but each can learn a couple of things from the other. There are special features that are available in WAP but not in MIDP and *vice versa*. These features are summarized as follows:

- MIDP provides a low-level graphics APIs that enable the programmer to have control over every pixel of the device's display. This is important for entertainment applications (such as games) in a wireless environment.
- MIDP is the way to go for games. The nature of MIDlets (they exist on the device until they are explicitly removed) allows users to run them even when the server becomes unavailable (support for disconnected operations).
- WML provides tags and possible presentation attributes, but it doesn't define an interaction model. For example, WML defines a `SELECT` tag for providing a list. Some WAP-enabled devices interpret the `SELECT` tag as a popup menu list while others interpret it as a menu that can be used for navigation. Therefore, there is no standard interaction model defined for this element. If a developer uses it, the application may run well on some devices and poorly on others. MIDlets, on the other hand, provide a clearly defined standard for interaction using commands.

A Micro Browser is Needed

MIDlets combine excellent online and offline capabilities that are useful for the wireless environment, which suffers from low bandwidth and network disconnection. Integrating WAP and MIDP opens up possibilities for new wireless

Figure 2. Combining WAP and J2ME

applications and over the air distribution models. Therefore, WAP and MIDP shouldn't be viewed as competing but rather as complementing technologies. In order to facilitate downloading wireless applications over the air, there is a need for some kind of an environment on the handset that allows the user to enter a URL for a MIDlet Suite, for example. This environment could very well be a WAP browser as shown in Figure 2.

Similar to Java Applets that are integrated into HTML, MIDlets can be integrated into a WML or an XHTML page. Such a page can then be called from a WAP browser and the embedded MIDlet gets downloaded and installed on the device. In order to enable this, a WAP browser is needed on the device. Another alternative approach for over the air provisioning is the use of Short Message Service (SMS) as have been done by Siemens where the installation of MIDlets is accomplished by sending a corresponding SMS. If the SMS contains a URL to a Java Descriptor (JAD) file specifying a MIDlet Suite, then the recipient can install the application simply by confirming the SMS.

DESIGN CHALLENGES AND POSSIBLE SOLUTIONS

In this paper we are more concerned with native interactive applications that can be developed using the J2ME platform or a similar technology. J2M2-based wireless applications can be classified into local (stand-alone) and network applications. Local applications (or stand-alone) perform all their operations on a handheld wireless device and need no access to external data sources through a wireless network. Examples include calculators and single-player games. Network applications, on the other hand, consist of some components running on a wireless device and others running on a network, and thus depend on access to external resources. An example would be an email application, with a client residing on a wireless phone interacting with a Simple Mail Transfer Protocol (SMTP) server to send/receive e-mail messages. A major difference between local and networked applications is in the way they are tested. Local applications are easier to test than network applications. For example, a calculator application can run on a wireless device even when it is not connected to any network, but an email client will not work without a connection to mail servers.

Challenges

The constraints discussed earlier pose several crucial challenges, which must be faced in order for wireless applications to function correctly in the target environment.

- **Transmission Errors:** Messages sent over wireless links are exposed to interference (and varying delays) that can alter the content received by the user, the target device, or the server. Applications must be prepared to handle these problems. Transmission errors may occur at any point in a wireless transaction and at any point during the sending or receiving of a message. They can occur after a request has been initiated, in the middle of the transaction, or after a reply has been sent. While wireless network protocols may be able to detect and correct some errors, error-handling strategies that address all kinds of transmission errors that are likely to occur are still needed.
- **Message Latency:** Message latency, or the time it takes to deliver a message, is primarily affected by the nature of each system that handles the message, and by the processing time needed and delays that may occur at each node from origin to destination. Message latency should be taken into account and users of wireless applications should be kept informed of processing delays. It is especially important to remember that a message may be delivered to a user long after the time it is sent. A long delay might be due to coverage problems or transmission errors, or the user's device might be switched off or have a dead battery. Some systems keep trying, at different times, to transmit the message until it is delivered. Other systems store the message, then deliver it when the device is reconnected to the network. Therefore, it is important to design applications that avoid sending stale information, or at least to make sure that users are aware it is not up to date. Imagine the possible consequences of sending a stock quote that is three days old, without warning the user!
- **Security:** Any information transmitted over wireless links is subject to interception. Some of that information could be sensitive, like credit card numbers and other personal information. The solution needed really depends on the level of sensitivity. To provide a complete end-to-end security solution, you must implement it on both ends, the client and the server, and assure yourself that intermediary systems are secure as well.

Possible Solutions

Here are some practical hints useful to consider when developing mobile applications.

- *Understand the environment*. Do some research up front. As with developing any other software application, we must understand the needs of the potential users and the requirements imposed by all networks and systems the service will rely on.
- *Choose an appropriate architecture*. The architecture of the mobile application is very important. No optimization techniques will make up for an ill-considered architecture. The two most important design goals should be to minimize the amount of data transmitted over the wireless link, and to anticipate errors and handle them intelligently.
- *Partition the application*. Think carefully when deciding which operations should be performed on the server and which on the handheld device. Downloadable wireless applications allow locating much of an application's functionality of the device; it can retrieve data from the server efficiently, then perform calculations and display information locally. This approach can dramatically reduce costly interaction over the wireless link, but it is feasible only if the device can handle the processing the application needs to perform.
- *Use compact data representation*. Data can be represented in many forms, some more

compact than others. Consider the available representations and select the one that requires fewer bits to be transmitted. For example, numbers will usually be much more compact if transmitted in binary rather than string forms.

- *Manage message latency*. In some applications, it may be possible to do other work while a message is being processed. If the delay in appreciable – and especially if the information is likely to go stale – it is important to keep the user informed of progress. Design the user interface of your applications to handle message latency appropriately.
- *Simplify the interface*. Keep the application's interface simple enough that the user seldom needs to refer to a user manual to perform a task. To do so: reduce the amount of information displayed on the device; make input sequences concise so the user can accomplish tasks with the minimum number of button clicks; and offer the user selection lists.

AD-HOC DEVELOPMENT PROCESS

An ad-hoc development process for wireless applications comprises three steps:

1. Write the application: Several Integrated Development Environments (IDEs) are available for developing Java-based wireless applications, e.g. Sun's J2ME Wireless Toolkit, and Metrowerks CodeWarrior.
2. Test the application in an emulation environment: Once the application compiles nicely, it can be tested in an emulator.
3. Download the application to a physical device and test it. Once application's performance is satisfactory on one or more emulators, it can be downloaded to a real device and tested it there. If it is a network application, it is tested on a live wireless network to ensure that its performance is acceptable.

It is clear that many important software engineering activities are missing from this ad-hoc development process. For example, there is no formal requirements analysis phase, and so following an ad-hoc development process may lead to building a product different from the one the customers want, and also testing an application without knowing its requirements is not an easy task. In addition, issues related to the operating environment such as network bandwidth should be considered during the design so that the performance of the application will be satisfactory.

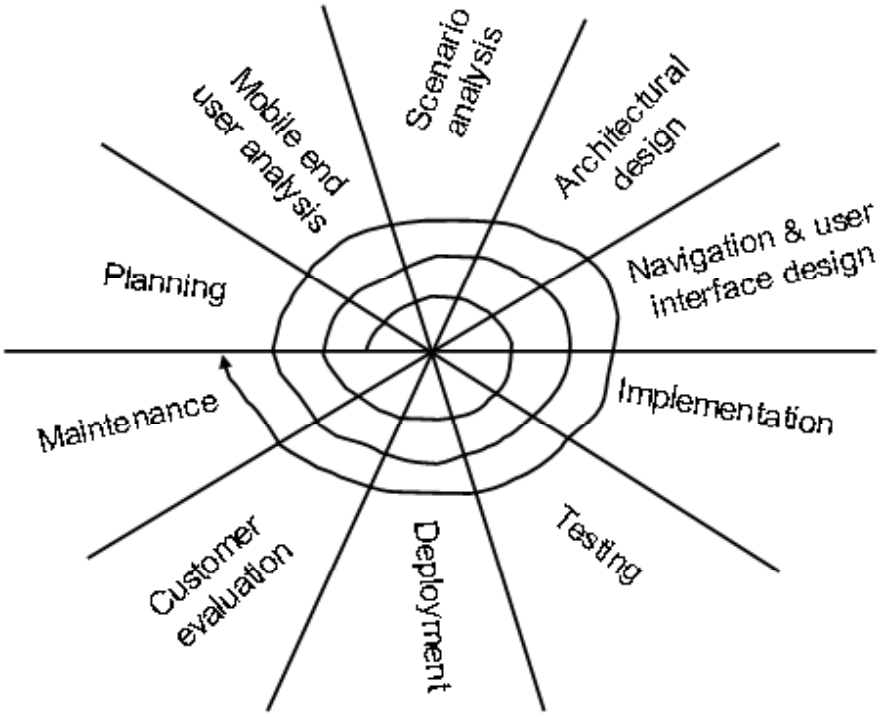
WIRELESS SOFTWARE ENGINEERING

While wireless application development might appear to have less need for the coordination that a process provides, aspects of development, testing, evaluation, deployment, and maintenance of a wireless application have to be integrated in the design process throughout the full development lifecycle. We have put forward a systematic approach to developing wireless applications, which is compatible with the Rational Unified Process or RUP (Jacobsen et al., 2000) in the sense that it is iterative and responsibility-driven. We have developed this systematic approach based on our experience designing and building wireless applications. We recognized that the development of a wireless application is not a one-shot task, and testing wireless applications is more challenging than testing conventional desktop software applications, and therefore an ad hoc development process cannot be used.

Development Activities

Our software engineering approach to wireless application development consists of a set of

Figure 3. Wireless application development activities



manageable activities that if followed properly leads to reliable and maintainable wireless applications. The activities of our approach are shown in Figure 3.

- **Planning**. This iterative process begins with a planning phase, which is an activity that identifies the objectives of the wireless application and specifies the scope of the first increment. In addition, the costs of the overall project are estimated, the risks are evaluated, and a tentative schedule is set.
- **Mobile User Analysis**. First, we must understand the audience of the application and the environment in which it will operate. As an example, if the application is a wireless network-aware application such as a multiplayer game, the study will include the users of the application and how they plan to use it. The output at the end of this phase is a wireless application plan document that serves as the mobile end user requirement.
- **Scenario Analysis**. This phase is similar to conventional software requirements analysis, and therefore concepts and principles of requirements analysis can be applied here (Pressman, 2005). In this phase, the mobile end user, an interaction designer, and a developer sit together to come up with a complete scenario analysis model that takes into account the following types of scenario analysis:
  - **Screen and Interaction Analysis**: The basic unit of interaction between the user and the mobile device is the screen, which is an object that encapsulates device-specific graphic user

input. Therefore, the content to be displayed on the screen is identified. Content may include text fields, menus, lists, and graphics. Interaction analysis specifies how the user interacts with the application. In order to find out how the user will interact with the application, UML (Booch et al., 2000) use cases are developed.

- **Usage Analysis**: The use case model developed during screen and interaction analysis is mainly related to how users interact with the application through the screen. The whole functionality of the application should be described with use cases.
- **Environment Analysis**: The environment in which the application will operate should be described in details. This includes the different wireless networks and backend systems used. In addition, target mobile devices such as cellular phones and PDAs on which the application will run should be described in details.

The output of this phase is an information analysis model document produced by the interaction designer and developer that outlines the functional requirements of the application and the constraints of the environment. This document is reviewed by developers and other stakeholders and modified as required.

- **Architectural Design**. This phase is concerned with the overall architecture (or structure) of the wireless application. Architecture is very important for any application, and no optimization techniques will make up for an ill-considered architecture. Design patterns can be used in this phase to reuse experience in order to come up with an extensible, high-performance architecture. Some of the most important design goals should be to minimize the amount of data transmitted over the wireless link, and to anticipate errors and handle them intelligently. Other design and architecture issues include:

  - *Application partitioning*. Designers need to think carefully when deciding which operations should be performed on the server and which on the wireless device. J2ME allows designers to locate much of an application's functionality on the device; it can retrieve data from the server efficiently, then perform calculations and display information locally. This approach can dramatically reduce costly interaction over the wireless link, but it is feasible only if the device can handle the processing your application needs to perform.
  - *Message latency*. In some applications, it may be possible to do other work while a message is being processed. If the delay is appreciable – and especially if the information is likely to go stale – it is important to keep the user informed of progress.

The outcome of the architectural design phase is a design document that details the system architecture.

Navigation and User Interface Design. Once the application architecture has been established and its components identified, the interaction designer prepares screen mockups and navigation paths that show how the user moves from one screen to another to access services. Figure 4 shows a simple example where the user will have to login before she is able to check her messages.

The user interface is the face of the application to users. A poorly designed user-interface will

Figure 4. Screen Mockups

scare the user away, and a well-designed user interface will give a good first impression and improves the user's perception of the services offered by the application. The user interface must be well-structured and easy to use. Here are some guidelines that can help in designing simple, but yet effective user interfaces for mobile devices with tiny screens.

- Keep the application's interface simple enough that the user seldom needs to refer to a user manual to perform a task.
- Reduce the amount of information displayed on the device.
- Make input sequences concise so the user can accomplish tasks with the minimum number of button clicks.
- Offer the user selection lists.
- Do not depend on any specific screen size.

The output of this phase is a user manual that describes the screen mockups and the navigational paths.

Implementation. In this phase development tools are used to implement the wireless application. There are several tools available for building wireless applications such as Sun's J2ME Wireless Toolkit. We would recommend using a tool that allows installing the application in various emulation environments. Conventional implementation strategies and techniques such as coding standards and code reviews can be used in this phase.

Testing. Software testing is a systemic process to find differences between the expected behavior of the system specified in the software requirements document and its observed behavior. In other words, it is an activity for finding errors in the software system and fixing them so users can be confident that they can depend on the software. Errors in software are generally introduced by people involved in software development (including analysts, architects, designers, programmers, and the testers themselves). Examples of errors include mismatch between requirements and implementation.

Many developers view the subject of software testing as "not fashionable," and as a result too few of them really understand the job software testers do. Testing is an iterative process and should start from the beginning of the project. Software developers need to get used to the idea of designing software with testing in mind. Some of the new software development methodologies such as eXtreme Programming (XP) (Beck, 1999) stress incremental development and testing. XP is ideally suited for some types of applications, depending on their size, scope, and nature. User interface design, for example, benefits highly from rapid prototyping and testing usability with actual users.

Wireless applications, like all other types of software, must be tested to ensure functionality and usability under all working conditions. Testing is even more important in the wireless world

because working conditions vary a lot more than they do for most software. For example, wireless applications are developed on high-end desktop machines but deployed on handheld wireless devices with very different characteristics.

One way to make testing simple is to design applications with testing in mind. Organizing the system in a certain way can make it much easier to test. Another implication is that the system must have enough functionality and enough output information to distinguish among the system's different functional features. In our approach, and similar to many others, the system's functional requirements (features that the system must provide) are described using the Unified Modeling Language (Booch et al., 2000) to create a use case model, then detailing the use cases in a consistent written form. Documenting the various uses of the system in this way simplifies the task of testing the system by allowing the tester to generate test scenarios from the use cases. The scenarios represent all expected paths users will traverse when they use the features that the system must provide.

Deployment. Deploying and running applications in an emulation environment is a very good way to test the logic and flow of your application generally, but you won't be certain it will satisfy users until you test it on a real physical device connected to a wireless network. Your application's performance may be stunning in the emulator, which has all the processing power and memory of your desktop machine at its command, but will it perform well on the handheld device, with its limited memory and processing power, low bandwidth, and other constraints? In this phase, the application is deployed on a live network and evaluated.

Customer Evaluation. Once the application has been deployed, it is ready to be downloaded by users for evaluation and usage. In this phase, users start using the deployed application and report any problems they may experience to the service provider.

Maintenance. Software maintenance encompasses four activities: error correction, adaptation, enhancement, and reengineering (Pressman, 2005). The application will evolve over time as errors are fixed and customers request new features. In this phase, users report errors to and request new features from the service provider, and developers fix errors and enhance the application.

TESTING ISSUES AND TESTING ACTIVITIES

The wide variety of mobile devices such as wireless phones and PDAs results in each device running a different implementation of the J2ME environment. Varying display sizes add to the complexity of the testing process. In addition, some vendors provide proprietary API extensions. As an example, some J2ME vendors may support only the HTTP protocol, which the MIDP 1.0 specification requires, while others support TCP sockets and UDP datagrams, which are optional. Here are some guidelines for testing wireless applications.

Validating the Implementation. Ensuring that the application does what it is supposed to be is an iterative process that you must go through during the implementation phase of the project. Part of the validation process can be done in an emulation environment such as the J2ME Wireless Toolkit (Sun Microsystems, 2005), which provides several phone skins and standard input mechanisms. The toolkit's emulation environment does not support all devices and platform extensions, but it allows making sure that the application looks appealing and offers a user-friendly interface on a wide range of devices. Once the application has been tested on an emulator, you can move on to the next step and test it on a real device, and in a live network.

Usability Testing. In usability testing, the focus is on the external interface and the relation-

ships among the screens of the application. As an example, consider an email application that supports entry and validation of a user name and password, enables the user to read, compose, and send messages, and allows maintenance of related settings, using the screens shown in Figure 3, among others.

In this example, start the test at the Login window. Enter a user name and a password and press the soft button labeled Login. Enter a valid user name and password. The application should display the main menu. Does it? The main menu should display a SignOut button. Does it? Press the SignOut button. Does the application return to the Login screen? Write yourself a note to raise the question, "Why does the user 'log' in but 'sign' out?" Now enter an invalid user name or password. The program should display a meaningful message box with an OK button. Does it? Press the OK button. Does the application return to the Login screen?

You need to test the GUI navigation of the entire system, making notes about usability along the way. If, for example, the user must traverse several screens to perform a function that's likely to be very popular, you may wish to consider moving that particular function up the screen layers. Some of the questions you should ask during usability testing include: is the navigation depth (the number of screens the user must go through) appropriate for each particular function, does the application minimize text entry (painful on a wireless phone) or should it provide more selection menus, can screens of all supported devices display the content without truncating it, and if you expect to deploy the application on foreign devices, does it support international character sets?

Network Performance Testing. The goal of this type of testing is to verify that the application performs well in the hardest of conditions (for example, when the battery is low or the phone is passing through a tunnel). Testing performance in an emulated wireless network is very important. The drawback with testing in a live wireless network is that so many factors affect the performance of the network itself that you can't repeat the exact test scenarios. In an emulated network environment, it is easy to record the result of a test and repeat it later, after you have modified the application, to verify that the performance of the application has improved.

Server-Side Testing. It is very likely that wireless applications communicate with server-side applications. If your application communicates with servers you control, you have a free hand to test both ends of the application. If it communicates with servers beyond your control (such as *quotes.yahoo.com*), you just need to find the prerequisites of use and make the best of them. You can test server-side applications that communicate over HTTP connections using testing frameworks such as HttpUnit (http://httpunit.sourceforge.net)), and measure a Web site's performance using httperf (http://citeseer.nj.nec.com/mosberger98httperf.html), a tool designed for measuring the performance of Web servers.

Testing Checklists

Here we provide checklists that are useful when testing your application, in both emulation and live environments. These checklists include tests that are usually performed by certification programs offered by Nokia and Motorola (Motorola Application Certification Program).

Navigation Checklist. Here are some items to check for when testing the navigational paths of wireless applications:

- *Successful startup and exit*: Verify that your application starts up properly and its entry point is consistent. Also make sure that the application exits properly.
- *Application name*: Make sure your application displays a name in the title bar.
- *Keep the user informed*: If your application does not start up within a few seconds, it should alert the user.

- For large applications, it is a good idea to have a progress bar.
- *Readable text*: Ensure that all kinds of content are readable on both grayscale and color devices. Also make sure the text does not contain any misspelled words.
- *Repainting screens*: Verify that screens are properly painted and that the application does not cause unnecessary screen repaints.
- *Soft buttons*: Verify that the functionality of soft buttons is consistent throughout the application. Verify that the whole layout of screens and buttons is consistent.
- *Screen Navigation*: Verify that the most commonly used screens are easily accessible.
- *Portability*: Verify that the application will have the same friendly user interface on all devices it is likely to be deployed on.

Network Checklist. Some of the items that should be inspected when testing wireless applications are:

- *Sending/Receiving data*: For network-aware applications, verify that the application sends and receives data properly.
- *Name resolution*: Ensure that the application resolves IP addresses correctly, and sends and receives data properly.
- *Sensitive data*: When transmitting sensitive data over the network, verify that the data is being masked or encrypted.
- *Error handling*: Make sure that error messages concerning network error conditions (such as no network coverage) are displayed properly, and that when an error message box is dismissed the application regains control.
- *Interruptions*: Verify that, when the device receives system alerts, Short Message Service (SMS) messages, and so on while the application is running, messages are properly displayed. Also make sure that when the message box is dismissed the application continues to function properly.

PROVISIONING WIRELESS APPLICATIONS

Developers usually build, test, and evaluate an application on a platform similar to the one on which it will be deployed and run. Development of wireless applications is more challenging because they typically are developed on one platform (such as Solaris or MS Windows) but deployed on a totally different one (such as a cell phone or PDA). One consequence is that, while emulators enable developers to do some of their testing on the development platform, ultimately they must test and evaluate the application in the very different environment of a live wireless network.

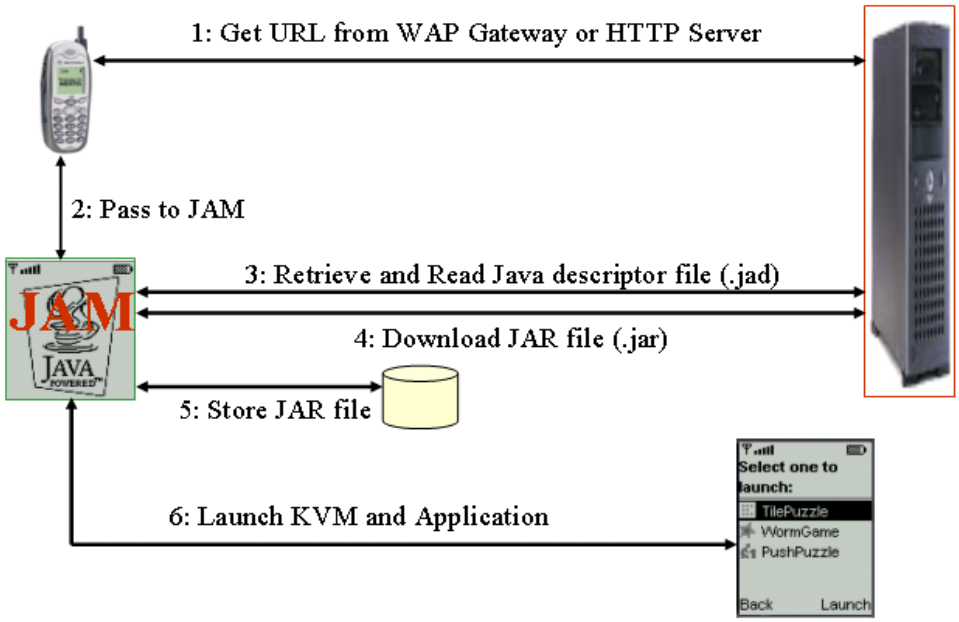
Wireless applications fall into two broad categories:

Local applications (also called *stand-alone applications*) perform all their operations on a handheld wireless device and need no access to external data sources through a wireless network. Examples include calculators and single-player games.

Network applications consist of some components running on a wireless device and others running on a network, and thus depend on access to external resources. An example would be an email application, with a client residing on a wireless phone that interacts with a Simple Mail Transfer Protocol (SMTP) server to send messages.

Although these two types of applications are different, they are deployed in the same way. The big difference shows up later: Local applications are easier to test than network applications. For example, a calculator application can run on a wireless phone even when it is not connected to any network, but an email client won't work without a connection to the SMTP server that actually transmits the messages.

Figure 5. Over the air provisioning



Over the Air Provisioning

For some time, wireless portals in Europe such as Midletcentral have allowed customers to download applications directly to their phones, over the air. Over-the-air provisioning of wireless applications (OTA) is finally available in North America. Nextel customers, for example, can download network-aware wireless applications without an update data cable. OTA is the deployment of wireless Java applications (*MIDlet suites*) from the Internet to wireless devices over a wireless network. Users need not connect their devices to the desktop with a data cable or visit a service center to install or upgrade software. To take advantage of OTA, you must equip your handheld device with a mechanism to discover MIDlet suites available for download, using the device's browser (such as a WAP browser) or a resident application written specifically to identify downloadable MIDlet suites. The process of downloading MIDlets over the air is illustrated in Figure 5.

RELATED WORK

The explosive growth of the wireless mobile application market raises new engineering challenges (Morisio & Oivo, 2003); what is the impact of the wireless Internet on engineering wireless mobile applications for the new wireless infrastructure and wireless handheld devices? Due to the limited experience with wireless technologies and developing wireless applications, little work has been in the area of wireless software engineering. We found a special issue in the IEEE Transactions on Software Engineering on 'Software Engineering for the Wireless Internet' (Morisio & Oivo, 2003). However, out of the six papers accepted in the special issue only two papers deal with the development process. Ocampo et al., (2003) provided

an initial reference process for developing wireless Internet applications, which does not differ significantly from traditional iterative process models but includes domain-specific guidance on the level of engineering processes. Satoh (2003) developed a framework for building and testing networked applications for mobile computing. The framework is aimed to emulate the physical mobility of portable computing devices through the logical mobility of applications designed to run on them; an agent-based emulator is used to perform application-level emulation of its target device.

More recently, Chen (2004) proposed a methodology to help enterprises develop business strategies and architectures for mobile applications. It is an attempt to formulate a life-cycle approach to assisting enterprises in planning and developing enterprise-wide mobile strategies and applications. This methodology is more concerned with business strategies rather than technical details and thus it is targeted at managers rather than developers. And finally, Nikkanen (2004) presented the development work of a browser-agnostic mobile e-mail application. It reports on experiences porting a legacy WAP product to a new XHTML-based browser application, and offers guidelines for developing mobile applications.

Our work is different in the sense that we provide a detailed treatment of the impact of the characteristics of mobile devices and the wireless environment on engineering wireless mobile applications; we discuss the challenges and offer practical solutions for developing mobile applications. We present a systematic approach for designing wireless mobile application. Our approach is iterative just like in (Ocampo et al., 2003), but differs in the sense that our process has more focus on requirements elicitation and more importantly scenario analysis. We do not provide a testing framework, but our testing strategy and checklist is more practical than using just an emulated environment. Finally, unlike the work reported in (Chen, 2004) our methodology is targeted at developers and researchers rather than managers. And, unlike the work in (Nikkanen, 2004), our guidelines and systematic approach is not limited to WAP-based applications, but can be applied to engineering any wireless application.

CONCLUSION AND FUTURE WORK

As the wireless Internet becomes a reality and software developers become comfortable with the methods and processes required to build software, we recognize that the methods developed for conventional systems are not optimal for wireless applications. In particular, wireless application development doesn't always fit into the development model originated to cope with conventional large software systems. Most wireless application systems will be smaller than any medium size project; however, a software development method can be just as critical to a small software project success as it is to that of a large one. In this paper we have presented and discussed a systematic approach to wireless application development, and presented practical guidelines for testing wireless applications. The proposed approach takes into account the special features of the wireless environment. We have successfully used the approach presented to develop various wireless applications ranging from a stock portfolio management application to a mobile agent platform for mobile devices (Mahmoud, 2002). Our future work includes evaluating the effectiveness of the proposed methodology, documenting wireless software design patterns, and building tools to automate the task of testing wireless applications.

There are several interesting research problems in the emerging area of wireless mobile applications and services. Some of these research issues include: Novel mobile services in the area of m-commerce and health care; Security and privacy issues; Mobile agents for mobile services; Discovery and interaction of mobile services;

Enabling roaming of applications and profiles between different wireless standards; Location-aware and context-aware mobile services. We are currently addressing some of these research problems and research results will be presented in future articles.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for the many helpful suggestions for improving this paper. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant No. 045635.

REFERENCES

Beaulieu, M. (2002). Wireless Internet Applications and Architecture, Addison-Wesley.

Beck, K. (1999). Extreme Programming Explained: Embrace Change. Addison-Wesley.

Booch, G., Rumbaugh, J., & Jacobsen, I. (2000). The Unified Modeling Language User Guide, Addison-Wesley.

Burkhardt, J, Henn, H., Hepper, S., Rintdorff, K., & Schack, T. (2002). Pervasive Computing Technology and Architecture of Mobile Internet Applications, Addison-Wesley.

Chen, M. (2004). A methodology for building mobile computing applications. International Journal of Electronic Business, Vol. 2, No. 3, pp. 229-243.

Jacobsen, I., Booch, G., & Rumbaugh, J. (2000). The Unified Software Development Process. Addison-Wesley.

Httperf. citeseer.nj.nec.com/mosberger98httperf.html.

HttpUnit. http://httpunit.sourceforge.net.

Mahmoud, Q. (2002). MobiAgent: An Agent-based Approach to the Wireless Internet. Journal of Internet Computing, special issue on Wireless Internet, pp. 156.162.

Morisio, M., & Oivo, M. (2003). Guest Editors Introduction: Software Engineering for the Wireless Internet. IEEE

Transactions on Software Engineering, Vol. 29, No. 12, pp. 1057-1058.

Motorola Application Certification Program. http://qpqa.com/motorola/iden.

Nikkanen, M. (2004). User-centered development of a browser-agnostic mobile e-mail application. Proceedings of the third Nordic conference on Human-computer interaction, Tampere, Finland, pp. 53-56.

Ocampo, A., Boggio, D., Munch, J., & Palladino, G. (2003): Towards a Reference Process for Developing Wireless Internet Services. IEEE Transactions on Software Engineering, Vol. 29, No. 12, pp. 1122 – 1134.

Open Mobile Alliance. http://www.wapforum.org.

Pressman, R.S. (2005). Software Engineering: A Practitioner's Approach. Sixth Edition, McGraw Hill.

Satoh, I. (2003). A Testing Framework for Mobile Computing Software. IEEE Transactions on Software Engineering, Vol. 29, No. 12, pp. 1112-1121.

Sun Microsystems J2ME: http://java.sun.com/j2me.

Sun Microsystems J2ME Wireless Toolkit: http://java.sun.com/products/j2mewtoolkit.

This work was previously published in Int. Journal of Information Technology and Web Engineering, Vol 1, Issue 1, edited by G. Alkhatib and D. Rine, pp. 59-75, copyright 2006 by IGI Publishing (an imprint of IGI Global).