Improving Economic Secure Transaction Based on Useful and Advanced Wireless Technology

Khalidn Batiha

Received on July 15, 2004 Accepted for publication on July 18, 2005

Abstract

The main objective of this paper is to develop economic secure transaction by means of enhancing improved session tracking techniques in advanced wireless technology like j2me, which was previously not supported. This has been carried out by modifying some program-coding units applicable for the both most important session tracking techniques. And inheriting some programming classes and code from other technology like j2se. Finally, the results has shown efficiency and secure line between different terminals, this lead to the main objective of my paper which results in produce software provide secure transaction over mobile phones.

Keywords: J2ME, MIDlet, MIDP, RMS, J2SE

1. Introduction

Every e-commerce application must support session tracking. Unfortunately, MIDP (Mobile Information Device Profile), a J2ME (Java 2 Platform, Micro Edition) technology, supports only the standard HTTP protocol, which is stateless [18]. In this paper, I explore ways to add session support into the current MIDP network API framework. I discuss the implementations, usages, and relative merits of two approaches: using cookies, rewriting URLs. In these applications, the server must individually respond to each user according to his/her previous actions. For example, when a user places an item into a shopping cart, the server must remember and associate that item with that particular user. When the same user selects Checkout later, the server must respond with a list of the merchandise he/she previously selected. However, the most widely used Internet e-commerce protocol, HTTP, is stateless. Each HTTP connection is independent and knows nothing about other connections. Standard HTTP connections do not remember client state information. Most developers solve this problem by requiring

© 2005 by Yarmouk University, Irbid, Jordan.
* Business Networking and Systems Management Department, Philadelphia University, Jordan.
the client to embed some unique session identification information (usually a session ID string) in each HTTP connection. E-commerce Web servers can then organize individual HTTP connections into sessions according to the session identification. It usually works like so:

- When a new session's first connection is made, the server generates a new session ID and sends it back to the client.

- The client stores that session ID and attaches it to every subsequent HTTP connection so the server knows all connections belong to the same session that technique requires cooperation between the client and server software. The client and server must exchange session identification information with a previously agreed upon format. The industry has developed two de facto standards for exchanging such session information.

One approach transmits small pieces of text, called cookies, through HTTP connection headers. The other approach attaches a session ID string to the end of each request URL, a technique also known as URL rewriting.

Most Web browsers and desktop HTTP applications support both session-tracking methods. However, the HTTP session support in wireless Java platforms is far from smooth. In the main Java platform designed for cell phones and low-powered Personal Digital Assistant (PDAs)—the Mobile Information Device Profile (MIDP), a Java 2 Platform, Micro Edition (J2ME)-based technology—the HttpConnection object supports neither cookie nor URL rewriting out of the box. Considering the importance of session tracking in e-commerce applications, if I want J2ME/MIDP to be a serious mobile commerce platform, I must equip it with session-aware HTTP connections.

In this paper, I discuss how to implement session tracking in MIDP applications with both cookies and URL rewriting. As we mentioned earlier, session tracking requires a joint effort from both the client and the server. In this paper, I give examples in the context of Java application servers, but you can easily apply the same techniques to other servers.

2. J2ME versus WAP

J2ME apps have much more to offer than those built under the Wireless Application Protocol (WAP), in terms of both features and security. Whereas WAP is a thin-client development protocol, J2ME is a development platform specifically for smart applications [1, 2].

J2ME applications offer the following security advantages over WAP applications:

- Without a WAP gateway in the middle, smart applications can provide scalable end-to-end security from the back end to wireless devices. This will become especially important as the back end evolves into a message-driven Web-services framework.

- Smart applications can store and process data locally, thereby reducing network traffic. Not only does this conserve precious wireless bandwidth and reduce latency,
it reduces the likelihood that crucial information will be intercepted or interrupted (e.g., by denial-of-service attacks).

- Smart applications utilize device-processing power efficiently. Instead of encrypting everything with the same key strength regardless of needs, rich clients can establish a comprehensive differentiating security policy based on the content [2,12,13,14,15,18].

3. MIDlet

A MIDlet is a Java application that conforms to the specifications set out by the Connected Limited Device Configuration (CLDC) and the Mobile Information Device Profile (MIDP). OR simply we can say that an application on a MIDP device is known as MIDlet[1,16].

MIDlets are targeted at mobile devices that provide some level of network connectivity. The devices that will run MIDlets also have several common attributes such as:

- Limited screen size
- Limited memory
- Limited processing power

A MIDlet is similar to a java applet. It does not have a main () method. A MIDlet must extends the javax.microedition.midlet.MIDlet class and implement its following three abstract methods:

- startApp()
- pauseApp()
- destroyApp()

A MIDlet has to define a public no-argument constructor. The MIDlet class defines method that can be invoked by the Application Management Software to start and stop a MIDlet.

- Needs To Write A MIDlet

Before you can write a MIDlet, you need to download the required software which are following:

- The Java Developer's Kit (JDK)
- Connected Limited Device Configuration (CLDC)
- Mobile Information Device Profile (MIDP).

3.1 CLDC objectives, security and area of use.

The objective of CLDC includes:
To define a standard Java platform for small, resource-constrained, connected devices.

To allow dynamic delivery of Java applications and content to those devices.

To enable 3rd party application developers to easily create applications and content that can be deployed to those devices. CLDC Requirements

To run on a wide variety of small devices ranging from wireless communication devices such as cellular telephones and two-way pagers to personal organizers, point-of-sale terminals and even home appliances.

To make minimal assumptions about the native system software available in CLDC devices.

To define a minimum complement or the “lowest common denominator” of Java™ technology applicable to a wide variety of mobile devices.

To guarantee portability and interoperability of profile-level code between the various kinds of mobile (CLDC) devices.

The CLDC configuration addresses the following areas:

- Java™ language and virtual machine features
- Core Java libraries (java.lang.*, java'util.*)
- Input/output
- Networking
- Security
- Internationalization

The CLDC configuration does not address the following areas. These features are addressed by profiles implemented on top of the CLDC:

- Application life-cycle management (application installation, launching, deletion)
- User interface
- Event handling
- High-level application model (the interaction between the user and the application)

The CLDC specification addresses the following topics related to security:

- Low-level virtual machine security is achieved by requiring downloaded Java classes to pass a class file verification step.

- Applications are protected from each other by being run in a closed “sandbox” environment.

- Classes in protected system packages cannot be overridden by applications. Adherence to the Java Language Specification
The general goal for a Java VM supporting CLDC is to be as compliant with the Java2 Language Specification as is feasible within the strict memory limits of the target devices. Except for the following differences, a Java VM supporting CLDC shall be compatible with Chapters 1 through 17 of The Java Language Specification by James Gosling, Bill Joy, and Guy L. Steele. Addison-Wesley, 1996, ISBN 0-201-63451-1:

- No support for floating point data types (float and double).
- No support for finalization of class instances. The method Object. finalize() does not exist.
- Limitations on error handling. Most subclasses of java.lang.Error are not supported. Errors of these types are handled in an implementation-dependent manner appropriate for the device (in contrast, CLDC includes a fairly complete set of exception classes.) Adherence to the Java Virtual Machine Specification

The general goal for a Java VM supporting CLDC is to be as compliant with the Java2 Virtual Machine Specification as is possible within strict memory constraints. Except for the following differences, a Java VM supporting CLDC shall be compatible with the Java Virtual Machine as specified in the Java Virtual Machine Specification (Java Series) by Tim Lindholm and Frank Yellin.


- No support for floating point data types (float and double).
- No support for the Java Native Interface (JNI).
- No user-defined, Java™-level class loaders.
- No reflection features.
- No support for thread groups or daemon threads.
- No support for finalization of class instances.
- No weak references.
- Limitations on error handling.

Apart from floating point support, which has been omitted primarily because the majority of the CLDC target devices do not have hardware support for floating point arithmetic, the features above have been eliminated either because of:

- Strict memory limitations.
- Potential security concerns in the absence of the full J2SE security model.

3.2 The MIDP Libraries

The MIDP libraries provide device-specific functionality. This functionality includes on-device application management, low-level and high-level graphical user interface, persistent storage and extended network capability.
MIDP contains the following packages, the first three of which are core CLDC packages, plus three MIDP-specific packages

- java.lang
- java.io
- java.util
- javax.microedition.io
- javax.microedition.lcdui
- javax.microedition.midlet
- javax.microedition.rms

*Classes Inherited from J2SE

- System Classes From java.lang.
- Data Type Classes From java.lang.
- Collection Classes From java.util.
- I/O Classes From java.io.
- Calendar and Time Classes From java.util.
- Additional Utility Classes.
- Exception Classes From java.lang.
- Error Classes From java.lang.

4. Cookies

Cookies are pieces of NAME=VALUE format text embedded in HTTP headers. Netscape originally proposed the cookie concept as an "HTTP State Management Mechanism;" the specification was published in a request for comments, RFC 2109, in 1997. (Note: RFC 2695 defines the current cookie specification.) Since cookies reside in HTTP headers, they are transparent to applications and users, and thus the most widely used HTTP session-tracking technique.

The server assigns new cookies to a client through the HTTP header set-cookie. One HTTP connection can have multiple set-cookie headers and hence allows the server to set multiple cookies simultaneously. The set-cookie header takes the format:

    set-cookie: NAME=VALUE; expires=DATE; path=PATH;
    domain=DOMAIN_NAME; secure

The first NAME=VALUE is the cookie itself and is required. Each cookie can have many optionally specified attributes, such as expiration time, domain, and path. The domain attribute states the cookie's valid domains. It proves especially important since it protects user privacy and reduces the chances for conflicting cookies. For example, a
cookie set by yahoo.com is not supposed to be sent out when the user visits amazon.com later. If no domain attribute is specified, a cookie's domain defaults to the host name of the server that sets it.

The client program sends out the valid cookies to each URL it connects to. The client sends cookies back to the server in the HTTP header named cookies:

```
cookie: NAME1=VALUE1; NAME1=VALUE2; ...
```

A client can send multiple cookies in one connection header by delimiting them using semicolons.

5. Server side

All major Web server vendors support cookie-based HTTP sessions. In this paper, I discuss examples in the Java Web server context.

The HttpSession interface in any Java 2 Platform, Enterprise Edition (J2EE)-compatible Java servlet/JSP container knows how to handle sessions through cookies. To use cookies in our example, I must first ensure that cookie support is not disabled in the configure files. my example JSP, CookieSession.jsp, uses cookies to track sessions and update visit counters.

As you will see later, an HttpSession object can track sessions using both cookies and URL rewriting, but by default, it uses cookies. So, in our example, if the client's incoming connection sends a valid cookie, the code below returns an existing session object:

```
HttpSession sess = request.getSession(true);
```

Otherwise, the server creates a new session object and sends a new cookie to the client. The server invalidates sessions after a period of inactivity. That expiration time can be set by server administrators in configure files or by servlets at runtime through the sess.setMaxInactiveInterval() method. We can then associate server-side objects containing client state information with the HttpSession object:

```
  sess.setAttribute("Count", count);
```

I can retrieve the above session state information count object later as long as the session remains valid:

```
String count = (String) sess.getAttribute("Count");
```

If no object associates with attribute Count (for example, at the session's beginning), the above statement returns a null value, and we need to start a new counter:

```
if ( count == null ) {
  count = "0";
}
```
6. MIDP cookie support framework

Supporting cookies on the server side is easy; supporting them in MIDP applications proves more challenging.

I try to make cookie support as transparent as possible; so I borrow a technique from Sun's Smart Ticket J2ME/J2EE demo implementation to design cookie-aware connection objects. RMSCookieConnector is a decorator class for the standard MIDP Connector class. It maintains a MIDP record management system (RMS) record store to store all cookies. When RMSCookieConnector.open() is called, that method completes the following steps:

- It calls Connector.open() to create a standard MIDP HttpConnection object:

  ```java
  HttpConnection c = (HttpConnection) Connector.open(url);
  ```

- It iterates through the record-store records and fetches the valid cookies. I discuss this step's details in the next section.

- It assembles the valid cookies into a semicolon-delimited string and sets the string into the HttpConnection object's cookie header:

  ```java
  c.setRequestProperty( "cookie", cookieStr );
  ```

Conclusions

The stateless nature of HTTP requires developers to use their own custom method of managing state through the use of session-specific information. While there are a number of ways of implementing a session management solution, there are benefits and restrictions to each implementation. It is vital that developers understand both the mechanisms available to them, as well as the limitations. For applications requiring an application user to authenticate to access resources, it is imperative the session management process is implemented securely.

While this paper has described first the limitations of other wireless technology as compared to J2ME technology, second developing and enhancing some programming coding units to this technology in order to be in parallel with advancement of secure wireless transaction in order to improve the quality of services provided to the public over mobile phones as well as let developers aware of good session management is only one components of building a secure application.
تطوير السرية في المعاملات الاقتصادية باستخدام التكنولوجيا اللاسلكية

ملخص

إن الهدف الرئيسي من هذه البحث هو تطوير السرية في المعاملات الاقتصادية وذلك بالإضافة وتحسين بعض تقنيات الجلسات في التكنولوجيا اللاسلكية (J2ME) والتي كانت من قبل غير مدعومة وغير مستخدمة في مثل هذه التقنية. وقد تم ذلك من خلال تجريب بعض المقاطع والوحدات البرمجية، والتي تستخدم في تقنية الجلسات وإدخال بعض المقاطع البرمجية من تقنيات أخرى للفئة المجاورة لـ J2SE.

وقد أظهرت نتائج البحث فعالية هذا الفكرة والارتباط بالسرية إلى درجة كبيرة في المراسلات التجارية والاقتصادية.

References


6. ETSI, European telecommunications standards institute GSM recommendations.


