The ITB Journal

Volume 3 | Issue 2 Article 2

2002

Virtual Credit Card Processing System

Geraldine Gray

Karen Church

Tony Ayres

Virtual Credit Card Processing System

Geraldine Gray, Karen Church, Tony Ayres

Institute of Technology, Blanchardstown, Dublin 15, Ireland

Abstract:

The virtual credit card processing system is an e-business system we have developed which provides a secure and universal mechanism for making purchases over the Internet. The system uses Remote Method Invocation (RMI), Java Server Pages (JSP), Java Servlets and Java Database Connectivity (JDBC). We also look at the possibility of implementing the system using the Web Services architecture.

1. Introduction:

The Virtual Credit Card Processing System is a web-based multi-tiered distributed application. It is be powered by Servlet/JSP and RMI technology and implements an Oracle database and MySQL database for back-end processing. The system is divided into three separate components, namely an online shop, an on-line bank, and the virtual credit card provider. The online shop comprises of a fully functional shopping cart through which users can purchase goods. The online bank consists of a number of different utilities the user can use to maintain their banking information. The virtual credit card provider generates unique virtual credit card numbers which it transmits to banks using a combination of RMI and Encryption. Users request a virtual credit card number from the online bank. The bank then requests a credit card from the Virtual Credit Card Provider. Users then use this credit card in online shops to make purchases. Shops validate the credit card number used with the bank.

The banking system enables the following operations to be performed: Open/close accounts, Request virtual credit card numbers, Check account balance/details, Change pin number, Transfer money, Pay bills, Use direct debiting functions. The shopping cart system enables the following operations to be performed: Find products, View Products, Buy Products, Register, Login, Post Comments. The virtual credit card component carries out the following operations: Creating the unique credit card numbers, Updating and maintaining transaction logs associated with these credit card numbers, encrypting the credit card numbers and transmitting them using RMI for use by the user.

2. Details of Java Technologies used

2.1 Distributed Technology

Each of the three components runs on its own hardware platform at a remote site. Implementing this distributed architecture requires middleware to provide a programming abstraction between applications running on separate machines We are going to use Java's Remote Method Invocation (RMI) package to provide this programming abstraction. [13,14]

Java RMI is a distributed object model for the Java Platform. It allows communication between objects running on different hosts through a series of message passing. Object methods can be referenced between different computers across a network, and real objects can be passed as arguments and return values during method invocation. Java RMI uses object serialization to convert object graphs to byte-streams for transport. Any Java object type can be passed during invocation, including primitive types, core classes and user-defined classes. The ability of a system to pass actual objects enables clean system design [1,2].

Java RMI provides a fully object-oriented distributed environment. RMI also operates naturally in the Java area, which allows developers to program within a single object model, the Java model. This removes a great deal of complexity. RMI requires no mapping to common interface definition languages. The syntax of remote method invocations is almost identical to local method invocations [1].

To use RMI successfully, you need to create a client, a server, an interface and an implementation of that interface. The interface is simply a list of all the methods accessible remotely by the client. The implementation is the mechanism for implementing the methods listed in the interface. When these objects are created, the server and the implementation run on one machine while the client runs on a separate machine. [2].

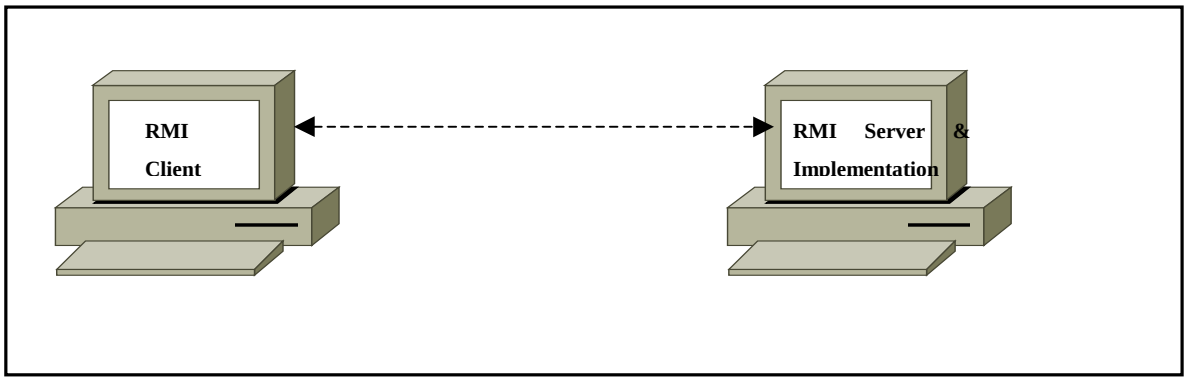

Figure 1: RMI architecture

2.2 Banking System

The banking system comprises of three main components. The first is a web-based application that represents the bank. This application provides the interface with which the user interacts. The second component is a communications medium between the bank and the virtual credit card component; the third component of the bank is a communications medium between the bank and the online shop.

The first requirement of the banking system is a web-based application that encompasses all of the functionality provided by the bank to the user. The user interface was developed in HTML, with the processing implemented using Java Servlets, connecting to an Oracle database using JDBC.

Java Servlets allow the application to be dynamic in nature since the output presented to the user depends on the inputs and requests made by the user.[7] Java Servlets are a Java Based Development tool that enable the generation of dynamic web pages using a request, response mechanism. In this HTTP based request-response paradigm, a client user agent (a web-based interface viewed through a web browser) establishes a connection with a web server and sends a request to the server. The server then issues a response. For web application development, the servlet API provides three primary abstractions: HTTP requests, request processing based on some application logic, and HTTP responses. In addition, the servlet API also provides a mechanism for session tracking and state management.[1] Being a Java technology, Java Servlets inherit all of the advantages that the Java language provides. This includes portability, scalability, robustness and simplicity. Each Servlet created processes a request made by users and then presents a response to the user in HTML format.

The majority of the Servlets we created require a connection to a database holding each users banking information. This connection was implemented using Java Database Connectivity (JDBC). Figure 2 below illustrates the role of Java Servlets and JDBC in this system. As can be seen from the diagram, the banking system is implemented in a three-tier architecture, comprising of a client, a server and a database.

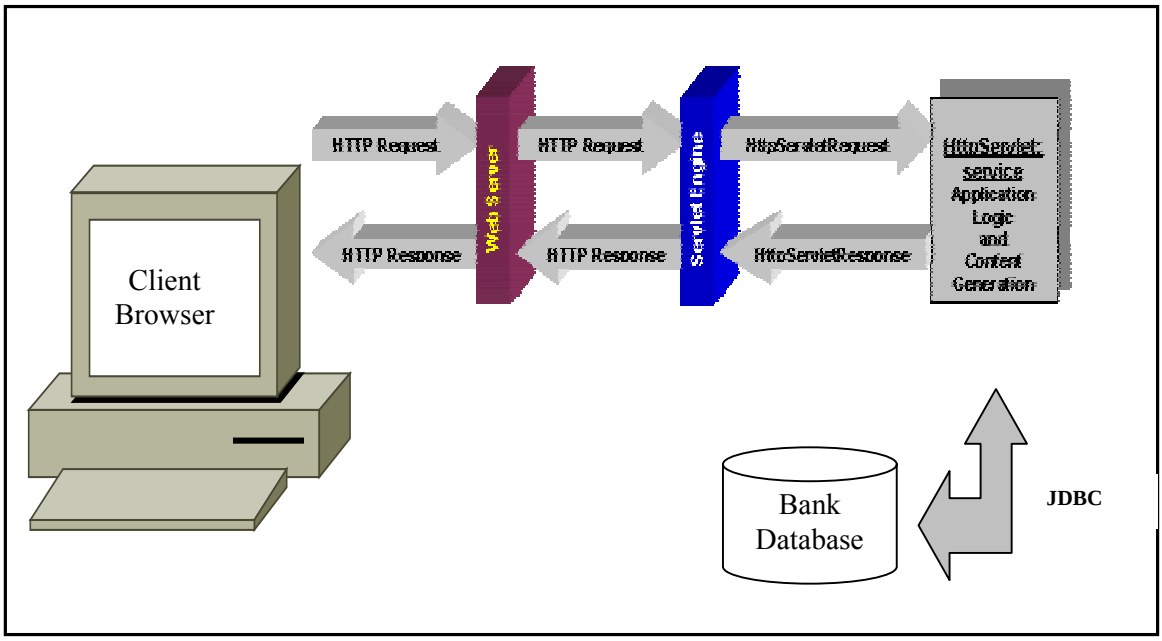

Figure 2: Banking system architecture

The second requirement of the banking system is an RMI component that communicates with the Virtual Credit Card component. The Virtual Credit Card component holds the RMI server and the implementation. The banking system holds the corresponding RMI client to access the methods provided by the RMI Server. The RMI client requests a virtual credit card number for use by the user. The RMI client is in fact a Servlet. This meant importing the java RMI package to facilitate binding the client to the server, i.e. inform the client of the location of the server so that communication can conduct successfully between these objects.

The third requirement of the banking system is a second RMI component. This RMI component allows communication to take place between the online shop and the banking system. When a user purchases goods using a virtual credit card in the online shop, this credit card number is sent back to the bank in a secure manner, so that the users account can be validated and debited for the amount of the purchase made. In this case the bank needs to hold an RMI server and an implementation. The server's primary function is to take the credit card number received from the shop and to carry out some processing and validation on the users account. This server is implemented as a servlet. The Java Servlet package

provides a RemoteHttpServlet class which allows any HTTP servlet to act as an RMI server. This package enables all RMI server operations to be carried out including binding itself to the RMI registry, logging any errors and providing remote methods accessible by the client.[1,7]

Figure 3 below illustrates the programming modules for the banking system:

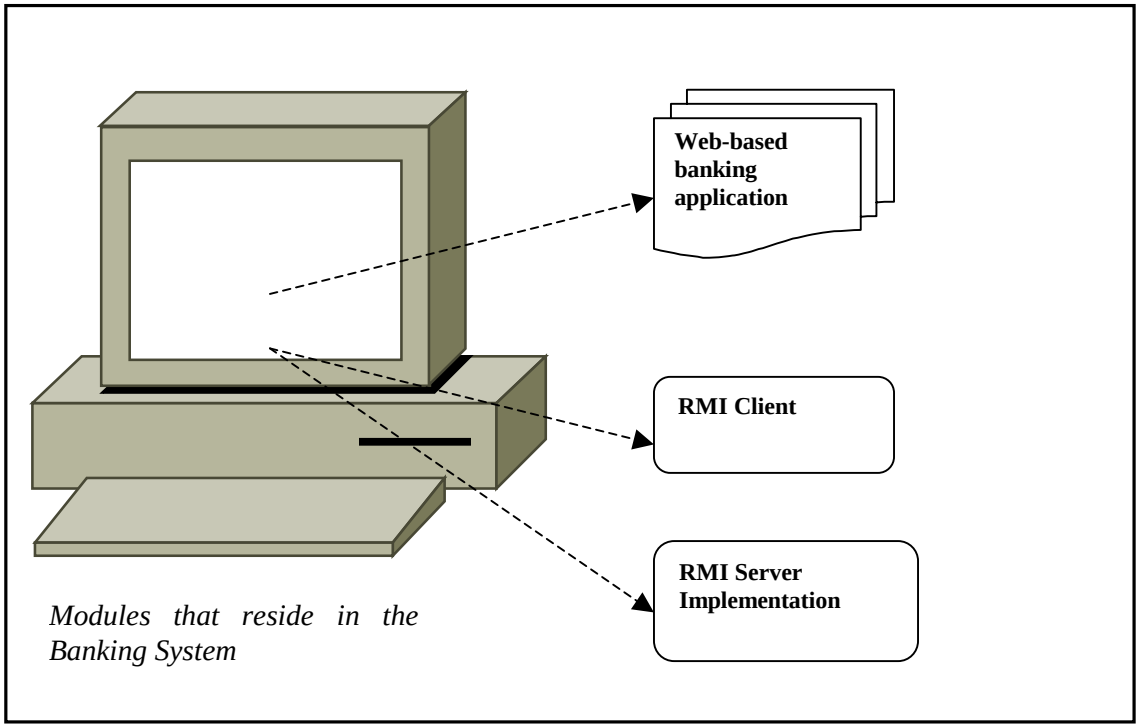

Figure 3: Banking system modules

2.3 Online Shop

The online shop is built using Java Server Pages and MYSQL database. Java Server Pages allow segments of Java code to be included in regular HTML documents. The Java code can be evaluated and the result printed out on the HTML page. Essentially the JSP will be compiled into a Java Servlet by the JSP engine. JSP offers an advantage over Servlets, in that it combines the power of the Java programming language with the simplicity of HTML. [1]

JSP has access to the majority of the API's in the Java language. As with Servlets, JSP also has methods for dealing with HTTP Requests, session tracking and HTTP Responses. Since the Servlet and JSP architecture are implemented and activated differently, JSP has request, response and session objects available to it at all times. This is particularly useful for an

application such as an online shop, as functionality such as tracking items in a shopping cart or identifying if a user is still logged in can be easily employed.

We selected Caucho's Resin 2.0.5 JSP/Servlet engine for the purpose of building the online shop. [15] When dealing with Servlet and JSP technology, Tomcat, the official Sun implementation of the JSP/Servlet specification is often the first Servlet engine that comes to mind. However, we found Tomcat to be slow and unreliable and for this reason we chose Resin.

MYSQL is a popular open source database, widely used for web based applications. Dynamic web applications with MYSQL traditionally used PHP to communicate with the database. However MYSQL-JDBC drivers exist to allow Java application's, or in this instance Java Server Pages, to communicate with MYSQL databases. With this set-up, we have an efficient combination of open source technology and platform independent Java technology. The MYSQL driver we used in this project was the mm.mysql driver, an open source JDBC driver. [16]

All product information on goods for sale in the shop is held in a database; JSP is responsible for retrieving the information based on customer's request. A database connection bean was used to connect to the database. This bean handles connections to the database, executing queries, returning result set objects and closing connections. Session tracking will be used to check the items in the users cart and to test whether or not the user is logged in.

Remote Method Invocation is used at the purchase stage of the shop. The user provides a virtual credit card number, the system will take this number and use RMI to call a method on the bank server to validate that the number is authentic. Based on the response of the bank object, the system will output the success or failure of the purchase attempt. The RMI object will be called from a Java Servlet. This servlet is also used to fulfil the users order, the users purchase information is stored in the ordering table of the database.

2.4 Virtual Credit Card Component

The Virtual Credit Card Component of the system comprises of two separate Java objects. The first java object is responsible for generating random credit card numbers, the second

java object is responsible for communicating with the credit card pool and the banking system in order to process requests from users of the system.

The first requirement of the virtual credit card component is the generation of a pool of unique credit card numbers. The Java API specification includes a java.util.Random package, which allows programmers to generate random numbers. This class called "Random" uses a 48-bit seed, which is modified using a linear formula. When you create a new random number generator with this package, its seed is initialised to a value based on the current time. To ensure that each credit card number is unique, we are going to use the java.util.Date package.[7] The Date package provides a getDate() and a getTime() method, which returns the current date and time in the following format:

- Day, Month, Year, Hour, Minute, Second, Millisecond, etc

As is known, time is constantly changing, so each time we call the getDate() and getTime() methods the numbers returned will be different from the previous. If we take the last 8 digits from the number returned we are guaranteed that they will be different from the previous numbers. This provided us with the unique element we required. To generate the 16-digit credit card number we require we use the java.util.Random package to generate an 8 digit random number which will serve as the first 8 digits of the credit card number. We can then bind these 8 random digits with the 8 unique digits we generated from the getDate(), getTime() methods. An example is shown below, followed by java code snippets to illustrate this process fully: Today's microprocessors can calculate vast quantities of data in nano-seconds. In testing the getDate() methods, we found that the processor ran so fast, that identical dates were returned. We introduced a thread to pause the execution of the getDate method so that a unique number was returned. To guarantee uniqueness, we used the java.util.Random package to generate a subsequent seed which was appended to the number returned from the date method. This ensured a completely random and unique credit card number.

2343 4566 – generated with java.util.Random
1223 5634 – generated with java.util.Date
2343 4566 1223 5634 – resulting credit card number

```
//random component
Random random=new Random(16);
long rand = random.nextLong()*random.nextLong();

//unique component – to be contained in a for loop

java.util.Date d = new java.util.Date();
long mili = d.getTime();
Long longmili = new Long(mili + rand);
String num = longmili.toString();
card_numbers[i] = num;
```

The second objective of the virtual credit card component of the system is to provide a communications medium between the pool of generated credit card numbers and the banking system. This is done using RMI as discussed in section 2.2. above. A series of requests will be carried out by users of the system, which are sent to the virtual credit card component for processing. These requests include requesting a credit card number from the pool and returning the credit card number to the pool.

The deployment diagram in Figure 4 below shows how the 3 systems interact using RMI. The shop system acts as an RMI client for the bank systems RMI Server and the bank system is a client of the Virtual Credit Card RMI server.

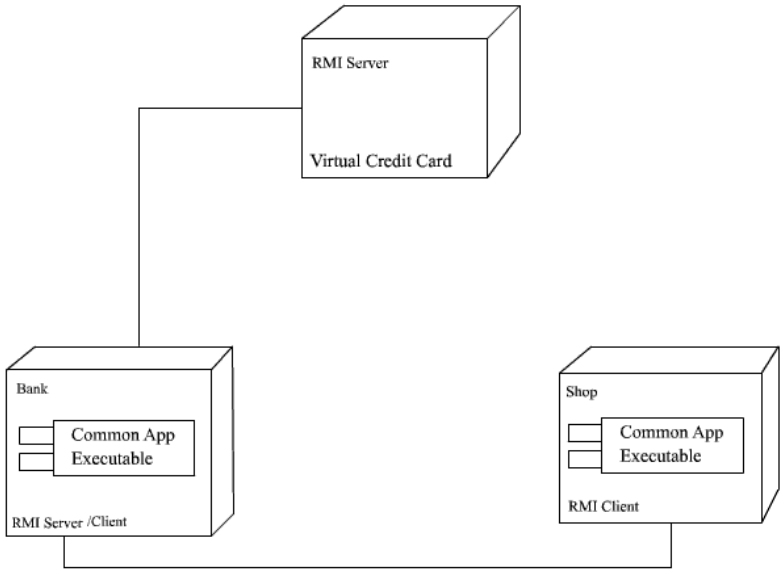

Figure 4: Deployment diagram

3. Future Considerations:

All the technologies as discussed in section 2 above were implemented successfully. However, an inevitable trait of technology is its tendency to change and evolve at a very fast pace. When designing the Virtual Credit Card Processing System using Java as the main programming language, technologies like RMI where considered first and foremost to cater for the distributed nature of the system. Since that time more technologies have emerged that could now be used to cater for this distributed system. These include protocols and languages like WSDL (Web Services Description Language)[10], SOAP (Simple Object Access Protocol)[5,6] and UDDI (Universal Description, Discovery, and Integration)[4] which together can be used to develop platform and language independent distributed web applications.

A web service is a service, available over a network, of components which can be called from a remote application. A WSDL file describes the Web service, enabling service providers and requesting clients to communicate with each other. WSDL defines XML syntax for describing network services as collections of communication endpoints capable of exchanging messages. Documentation for distributed applications and detailing the processes involved in communication between these applications is provided by WSDL service definitions. WSDL allows endpoints and their messages to communicate regardless of the message formats or network protocols being used. Primary uses for WSDL is the design of specifications to invoke and operate Web services on the Internet and to access and invoke remote applications and databases [9].

Simple Object Access Protocol, or SOAP is a protocol for exchanging information in a decentralized, distributed environment. It defines a mechanism for passing information and parameters between clients and servers. An advantage of SOAP is that it is a platform, object and programming language independent protocol and so can be used to create open and compatible systems. This protocol is XML-based and consists of three parts. An *envelope* that defines a structure for describing the contents of a message and how to process it, a set of *encoding rules* for expressing instances of application-defined data types, and a *method* for representing remote procedure calls and responses. A particular advantage of SOAP over a protocol like Java Remote Method Protocol (JRMP) for Java Remote Method Invocation (RMI) is that SOAP uses XML and is therefore text-based. XML is easier to read than a binary stream, making SOAP-based applications easier to debug [5,8].

Universal Description, Discovery, and Integration (UDDI) is like a central point for Web services. A UDDI repository stores descriptions about companies and the services they offer in a common XML format so it is like an information registry of Web services. This service is used by web service brokers to register services that service requesters can then discover and invoke. Web-based applications interact with a UDDI repository using SOAP message. [4,11,12].

For this project, the Virtual Credit Card provider could offer virtual credit cards as a web service. The component could be developed using Enterprise Java Beans. A WSDL XML file would provide information about the services being offered by the EJB's. Banks could then develop their own applications to use this web service.

4. References:

[1] Professional Java Server Programming: with Servlets, JavaServer Pages (JSP), XML, Enterprise JavaBeans (EJB), JNDI, CORBA, Jini and Javaspaces by Andrew Patzer, et al, Wrox Press, 1-861002-77-7, 2000
[2] Distributed Systems, Concepts and Designs, 3rd Edition, Coulouris et al, Addison Wesley, 0-210-61918-0, 2001
[3] Enterprise JavaBeans (3rd Edition), by Richard Monson-Haefel, O Reilly, 0-596-002268, 2001
[4] www.uddi.org/
[5] www.w3.org/TR/SOAP/
[6] www.intelligententerprise.com/010327/feat2_1.shtml
[7] http://www.java.sun.com
[8] http://www.onjava.com/
[9] www.w3.org/TR/wsdl
[10] http://www.capeclear.com/products/capeconnect/terms.shtml
[11] http://uddi.microsoft.com/default.aspx
[12] www.ibm.com/services/uddi/
[13] http://java.sun.com/products/jdk/rmi/
[14] http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html
[15] www.caucho.com
[16] www.mysql.com