WAP-Enabled Banking and Broking: A Case Study

Morgan O'Connor, Macalla Software Limited

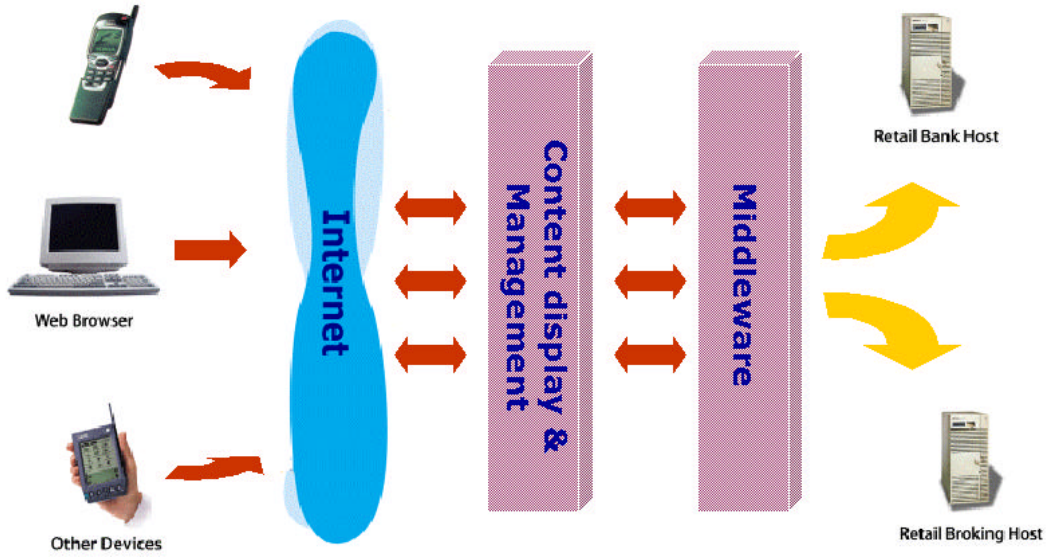
The purpose of this session paper is to discuss the issues associated with the deployment of a secure, WAP-based, transactional banking and broking system, and to build a solution with these issues in mind.

There are many pitfalls and problems that have to be overcome when developing an m-commerce solution, ranging from security to providing support for the multitude of WAP-enabled devices that are currently on the market. The problems of linking into an existing back-end infrastructure also need to be met. This paper will compare and contrast available technologies that could be used to solve each of these issues, and justify the technologies that were finally chosen.

Macalla Software has been used by a number of leading institutions in the financial services sector to give them the capability to provide their services to the wireless market. This paper draws upon the experience of the speaker during the deployment of a number of these WAP applications.

The Project

The goal of this project was to develop a combined retail banking and broking application that was to be delivered to multiple channels, including WAP devices and the Internet. The information had to be retrieved from existing back-end systems on a combination of different platforms. The following diagram illustrates a high-level view of the required solution:



From this diagram, we can see that multiple information sources are to be delivered to multiple distribution channels, with the possibility of adding more on either side at a later stage.

The key feature of designing and implementing this kind of system is its **architecture**. Effective implementation of the architecture gives the system the scalability and flexibility to add further information sources and distribution channels efficiently, as and when they become available.

Firstly, let's go through the typical characteristics of a retail banking/broking solution.

Retail Banking/Broking Characteristics

The attributes that typically classify the deployment of retail, WAP-based, financial applications include:

- Large numbers of end users — the potential user base for retail banks and stockbrokers is large.
- A large number of differing terminals — because the typical retail institution does not directly offer mobile handsets to its customers in order to access the service, it can expect that it will need to support a variety of different devices.
- Relatively non-technical users — this means that the service being offered needs to be easy to operate and access. The operation of WAP banking/broking services is relatively straightforward — assuming good WAP design principles are maintained — and there should be minimum requirements upon the end users to reconfigure their handsets to access the service.
- High volume of transactions with low individual values. As is typical in a retail environment, the amount of capital at risk when conducting an individual transaction is low, but there are large numbers of transactions being executed.
- The ability of a financial institution to reach its customers is enhanced by using the portals of the mobile operators, as it greatly reduces the amount of independent branding required to alert the audience to the service.

Financial Institution Concerns

When a financial institution takes the decision to implement a multi–channel service, there are a number of initial concerns that must be looked at seriously.

Security

Security of information is the primary concern of any financial institution when deploying its applications across the Internet or wireless devices.

Deploying a solution across multiple distribution channels causes its own problems. More often than not, there are different sets of security technologies on the different distribution channels. These available technologies must be investigated and a security strategy drawn up.

The security strategy used by the financial institution is of utmost importance — every service that is delivered over an electronic medium has an associated risk. The key to a successful strategy must be to determine the acceptable level of risk for the service.

Time to Market

As in every industry, time to market is an important concern for financial institutions. Being first to market and early providing additional services can raise the perceived image of the financial institution in the eyes of its customers — not to mention the publicity that can be generated by such a deployment.

Investment in the Future

Financial institutions that recognize the importance of delivering their financial services across multiple distribution channels will need to be confident in the technologies that they are using. By choosing the right technologies, it makes it easier to deploy new services, and to take advantage of new distribution channels as they become available.

Functionally Rich Application

The financial institution will want to deliver as much functionality as possible to its customers. The functionality that will be delivered is dependent upon multiple issues.

All of these concerns must be weighed up, and a balance between them decided upon.

Technical Concerns

When setting out to implement a mobile-based banking and broking application, there are a number of immediate technical problems that need to be overcome.

Multiple Sources of Information

In a typical deployment, there is more than one back-end application to interface with (for example, a banking mainframe and a broking database). These applications usually reside on different platforms, and provide their content in different formats. Because of this, when developing a distribution and transactional system, a piece of middleware is required that will facilitate the consolidation of information into a standard format, thereby allowing for the extraction of that information in a uniform way from multiple sources.

In this paper, I will be showing how we came to choose an XML middleware platform, and how this benefited us in the distribution of the information to multiple channels.

Multiple Distribution Channels

Initially in this project there were two distribution channels: the Internet and WAP. This presented a number of problems:

- **Formatting output** — Because the two channels take the information in different formats (HTML and WML), we had to investigate how we could deliver it in a way that still gave the page designers maximum flexibility.
- **Cross compatibility in WAP devices** — The problem with delivering information to multiple WAP devices is that each device has its own capabilities, such as display, security, configuration, and so on.
- **Security** — How do we create a security policy that can be extended to multiple different distribution channels with the technologies that are currently available?

We felt that we needed to create an architecture that would allow us to add new distribution channels securely and easily, as they became available. In this paper, I will describe technologies that were investigated, and explain the architecture that we decided upon.

Approaching a Solution

There are myriad technologies that could be used. In order to find a solution, we needed to filter through those technologies to come up with what we felt were the most flexible and extensible.

From the overall architecture of the required system, we realized that we would require multiple components in order to build it:

- The information was retrieved from back-end systems, so we would need to interface with those

- The information from those multiple sources would have to be channeled through a common middleware layer
- We would then need a component that handles the display of this information to the customer, based on their device, their preferences, and their access level

The sections below explain the technologies that were used in the final implementation, and look at some of the factors that make these technologies the best choice.

Other issues included the development environment, development tools, the testing environment, testing tools, and the problems concerned with a live deployment of the system.

Middleware

When we were deciding on a middleware solution, there were a number of different areas that were covered:

- Data representation
- Data distribution
- Data manipulation

Data Representation

A generic architecture requires a generic data representation. The solution to this problem is the **Extensible Markup Language** (**XML**). The purpose of XML (and its related technologies) is to provide a means of describing information generically, which facilitates easy machine manipulation and rendering. XML is portable data.

In order to facilitate multi-channel applications and diverse client devices, as much application logic as possible must be factored out into generic layers. XML has rapidly become the universal language for structuring business-critical information, and is defining the rate at which application interaction can be achieved. Factoring behavior into reusable components, and using XML for data representation and communication, provides a truly generic architecture.

Data Distribution

After deciding on XML for data representation, we needed a way to channel this information so that it could be dynamically distributed throughout the enterprise. This is where **DynamiX** from Macalla Software fits in.

Macalla Software's DynamiX is a highly optimized server for handling distribution, caching, filtering, and management of dynamic XML documents in a distributed, Internet-enabled environment.

Data Manipulation

Java provides the best support for portable code, especially with the arrival of the Java Enterprise Edition with its support for JavaBeans and Enterprise JavaBeans (EJB). Java and XML work very well together, and an architecture that utilizes these technologies should provide a mechanism to access XML information through a layer of JavaBeans.

In our architecture, we decided upon JavaBeans as a means by which the underlying XML should be manipulated. This simplifies the way in which the information in an XML document can be set or retrieved. We developed a set of tools that enabled us to create a JavaBean representation of an XML document schema.

Information that needs to be retrieved from the source can be described in an **XML schema**. Using tools that we developed, this schema would be used to create a JavaBean that provides helper properties, so that the information contained in the document can be manipulated.

Example

In this simple example, I will show how a response to a balance request would be satisfied. Firstly, describe your information in an XML schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<Schema name="BalanceResponse">
   <ElementType name="AccountNo">
      <element type="string" minOccurs="1" maxOccurs="1"/>
   </ElementType>
   <ElementType name="Balance">
      <element type="string" minOccurs="1" maxOccurs="1"/>
   </ElementType>
   <ElementType name="Currency">
      <element type="string" minOccurs="1" maxOccurs="1"/>
   </ElementType>
   <ElementType name="SwiftCode">
      <element type="string" minOccurs="1" maxOccurs="1"/>
   </ElementType>
   <ElementType name="BalanceResponse">
      <element type="AccountNo"/>
      <element type="SwiftCode"/>
      <element type="Balance"/>
      <element type="Currency"/>
   </ElementType>
</Schema>
```

A "filled out" version of this schema might look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<BalanceResponse>
   <AccountNo>123456789</AccountNo>
   <SwiftCode>900</SwiftCode>
   <Balance>$200.00</Balance>
   <Currency>USD</Currency>
</BalanceResponse>
```

When you've decided how your XML data should be represented, take the schema, and using a JavaBean generator, create a bean. The bean now contains properties that allow you set and get the values in the underlying XML document. This bean can be used by an application interfacing with an information source to allow it easily to generate the XML representation of the information.

In the code extract below, you can see that the response bean is created and the values are set using the methods that are available in the generated bean:

```
BalanceResponse handleBalance(BalanceRequest request)
{
    // Decode the request parameters here, and
    //  create an appropriate message to send to the host system
    String strAccountID = request.getAccountID();
    ...

    // Send Request to host (e.g. using MQSeries etc.)
    //  and wait for the response
    ...
    Balance hostBalance = m_Host.getBalance(strAccountID, ...);

    // Create a response based on the response from the host
    BalanceResponse xmlResponse = new BalanceResponse();

    responseTrans.setAccountNo(hostBalance.getAccount());
    responseTrans.setSwiftCode(hostBalance.getSwift());
    responseTrans.setBalance(hostBalance.getBalance());
    responseTrans.setCurrency(hostBalance.getCurr());
```

```
    // Return the response
    return xmlResponse;
}
```

When this response bean is returned, the XML middleware component delivers it back to the requesting component.

Summary

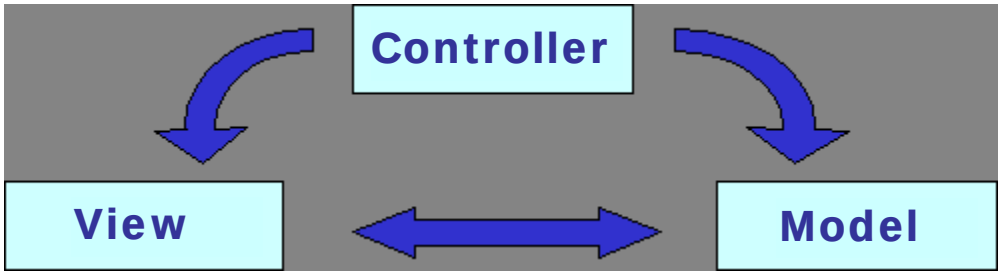
The middleware solution of choice proves to be distributed XML. This enables the data to be easily manipulated by the applications that are interfacing with the information sources. More reasons for using an XML document architecture combined with JavaBeans will be described in the next sections.

Content Display and Management

An important lesson that has been learned from the early years of the Internet is the need to separate content from presentation. Doing so greatly simplifies the distribution of content to multiple channels.

Model View Controller

The **Model View Controller** design pattern is a means by which an application's logic can be decoupled from the view of its information:



As we already have our information contained in JavaBeans, we have solved the "model" part of the equation.

In the architecture that we have deployed, the "controller" is synonymous with a Java servlet. The servlet receives a request from a client, and the request is sent to a bean that generates an XML request document. This request document is sent through the XML middleware to the host interface application, and a response is generated (again with the help of the beans) and returned.

Once we have the information in a bean, it's time to use a "view" to display the information. Built into the controlling servlet is a module that identifies the capabilities of the connecting client. If a WAP-enabled mobile phone is making the request, responsibility for the view will be dispatched to a handler that will return the view of this information in WML format. What technology is this view built on? **JavaServer Pages** (**JSP**).

JSPs combine markup language with snippets of Java code to facilitate the creation of dynamic HTML/WML/XML pages. JSPs have natural support for JavaBean technology, allowing them easily to retrieve information from the beans, and to render that information to the connecting client in the correct format. The following sample JSP file displays the content of the XML document in HTML format:

```
<HTML>

   <HEAD>
      <TITLE>Balance Response</TITLE>
   </HEAD>
```

```
   <jsp:useBean id="BalanceResponse" class="BalanceResponse" scope="session"/>

   <BODY>
      <TABLE BORDER="5" CELLSPACING="5" CELLPADDING="10" BGCOLOR="#C0C0C0">
         <TR CLASS="ListHeader">
            <TD>AccountNo.</TD>
            <TD>Swift Code</TD>
            <TD>Balance</TD>
            <TD>Currency</TD>
         </TR>
         <TR>
            <TD><%=BalanceResponse.getAccountNo()%></TD>
            <TD><%=BalanceResponse.getSwiftCode()%></TD>
            <TD><%=BalanceResponse.getBalance()%></TD>
            <TD><%=BalanceResponse.getCurrency()%></TD>
         </TR>
      </TABLE>
   </BODY>
</HTML>
```

The following JSP will render the same information in WML:

```
<%Response.setContentType("text/vnd.wap.wml");%>
<?xml version="1.0"?>

<jsp:useBean id="BalanceResponse" class="BalanceResponse" scope="session"/>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//Dtd WML 1.1//EN"
                     "http://www.wapforum.org/Dtd/wml_1.1.xml">
<wml>
   <card id="Balance"  title="Account Balance">
      <p mode="nowrap">

         <table columns="4">
            <tr>
               <td><b>Account</b></td>
               <td><b>Swift Code</b></td>
               <td><b>Balance</b></td>
               <td><b>Currency</b></td>
            </tr>

            <tr>
               <td><%=BalanceResponse.getAccountNo()%></td>
               <td><%=BalanceResponse.getSwiftCode()%></td>
               <td>&#163; <%=BalanceResponse.getBalance()%></td>
               <td><%=BalanceResponse.getCurrency()%></td>
            </tr>
         </table>
      </p>
   </card>
</wml>
```

From this, we can see that the JSP files are only concerned with the display of the information, which is what they are best at. The application logic is left up to the controlling servlet.

This degree of separation allows for a very flexible and extensible system. As more distribution channels become available, more views of the information can be created, but this won't affect the logic behind the application.

Testing / Deploying the Solution

We recommend that any financial institution currently looking to deploy a secure financial service over a WAP environment have two environments: Deployment and Test/Backup. The deployment environment is self-evident, but the use of a test/backup environment is prudent in the current climate for a number of reasons:

- Not all mobile operators support WTLS, so they're not able to support secure delivery of banking/broking services using their own WAP systems. Providing an environment that supports banking/broking irrespective of current mobile operator facilities is an important ability.

- Independence from operators. Because use of the operator's WAP gateway is an essential part of the financial institution's m-commerce offering, it is important that an element of independence is possible. An existing application may become non-operational due to a change in the mobile operator's configuration. A separate configuration will allow a backup service.
- New services. With a separate facility, it is possible to try new services with closed user groups, without having to ensure that the service works successfully with *all* the operators in the financial institution's area of operation.

Deployment

For deployment scenarios, Macalla Software recommends the use of the operator's WAP gateway, provided that:

- The use of 128-bit SSL from the WAP gateway to the origin server is enabled.
- The use of 128-bit WTLS for delivery of secure WAP services over the mobile airwaves is enabled.
- Level 2 of the WAP 1.1 WTLS specification is implemented. This will allow authentication of the WAP gateway. The use of the mobile operator's WAP gateway precludes the direct authentication of the bank/broker's service (because authentication is with the mobile operator), requiring users to trust that the mobile operator has securely connected them directly to the correct site.
- Mobile operators give assurances about the configuration and maintenance of the WAP server. Specifically, assurances about the procedures used to ensure that it is not possible for external or internal parties to gain access to financial information about the bank/broker's clients.

The use of the network operator's gateway allows the financial institution to be visible from the operator's portal, which in a retail environment provides great assistance to the service.

Test/Backup

Finally, for the test/backup environment, Macalla Software recommends that the bank/broker use a corporate WAP gateway (for example, Nokia's WAP server or similar). Benefits that accrue include:

- End-to-end security — if a corporate WAP gateway is used, it can be positioned as part of an existing (or proposed) Internet strategy. It would be placed behind the firewall used to deliver traditional web content, and could be configured to accept requests from suitably equipped WAP phones. In this configuration, no information is available to any third party, as the WAP gateway is located upon the facilities of the bank/broker.
- Direct authentication of bank/broker's service — the use of a corporate WAP gateway will allow the handset to authenticate with the service directly.
- Backup capabilities — the corporate WAP gateway allows a backup service to be offered to customers.
- Test facilities — the ability to test the applications with real, live customers without going through the appropriate channels is essential.