Softwired

Communication Middleware for Mobile Applications – A Comparison

Dr. Silvano Maffeis, CTO, Softwired AG.

Silvano.Maffeis@softwired-inc.com
http://www.softwired-inc.com/

In this article we argue that to provide compelling data-driven services to the mobile user, one often needs to deploy a customized client application on the mobile device. This client application usually has to communicate with applications running on a server "back-end". To expedite the development of wireless services, companies are using mobile middleware platforms to connect mobile Java applications to the back-end. Two different middleware standards are compared from the viewpoint of mobile services: CORBA and JMS. We conclude that from the point of view of user experience, scalability, and fit into a Java application server environment (J2EE), a wireless enabled JMS solution is preferred.

This article assumes familiarity with the general concepts of communications middleware, as well as with CORBA and JMS.

1 Challenges of Mobile Applications

To provide compelling data-driven services to the mobile user, wireless service providers seek out applications that take advantage of the features of 2.5 and 3G networks: "always on" communication, push notifications, and higher bandwidth. However, user expectations have already been shaped through exposure to high-end personal computers and high-speed Internet links. In the current (wired) Internet environment, users are accustomed to good Quality of Service, meaning that Web pages and e-mail servers are highly available and accessible at good speed.

In an ideal world, the quickest way to provide good applications for mobile devices is to "port" PC based applications to wireless devices. However, the wireless communication environment is characterized by lower bandwidth than the wired Internet, and by sporadic network disconnects. Therefore, to create a compelling user experience, new applications need to be developed and optimized for the wireless platform. For the sake of vendor neutrality, scalability, user experience, and time to market, those applications will consist of a rich client deployed directly on the mobile device, and of a server application running at the "back-end". We foresee the adoption of Java technology for this type of solution: J2ME (Java-2 Micro Edition) on the mobile device, J2EE (Java-2 Enterprise Edition) application servers at the "back-end", and a 2.5 or 3G network between clients and servers.

J2ME only provides rudimentary communications interfaces (e.g., TCP/IP streams, HTTP streams, and datagram connections) for internetworking wireless clients with servers. In order to cope with sporadic network disconnects, with off-line service usage, as well as with the peculiarities of the various wireless networks, developers need to devote a substantial amount of their time to solving technical networking issues, instead of focusing on the business problem to solve.

Companies are thus adopting mobile middleware platforms aimed at enabling reliable, secure, and scalable services in a 2.5 and 3G world. Mobile services can now be developed and

deployed more quickly, by taking advantage of the offline operation, data synchronization, reliability and security mechanisms provided through the middleware.

In summary, by using Java middleware instead of coding against the raw communications APIs provided by J2ME, time to market is reduced and the user experience is improved.

2 Middleware versus Micro-Browsers

The advantages of deploying a customized Java application on a mobile device, as opposed to using a micro-browser to access pages stored on a server, are manifold:

- The **user experience** can be improved by providing more interactions (e.g., imagine an interactive, "intelligent" map as opposed to downloading an image of a map from a server).
- **Offline operation** can be supported. The user can keep on using the mobile application whilst disconnected from the network. This is typically impossible with a browser.
- **Less data transmission**: A client application can cache data locally, compress data, or perform local computations. This in order to minimize interactions between mobile device and server.
- More interesting types of interaction are possible: **push notifications**, **peer-to-peer sessions**, and so forth.

The disadvantages are:

- **Higher development costs**, because client applications need to be developed.
- **Higher complexity** of the overall solution, because application code is distributed over client and server.

In summary, by using a middleware enabled client application, users get a better usage experience and "more fun", at a lower cost. However, we believe that both models (middleware and micro browsers) are appealing and have their application areas. Actually, the two models can be combined. For example, a micro browser can be used to browse the service offerings of a provider, and to dynamically download a middleware enabled client application.

3 CORBA Middleware

It has been proposed to use CORBA middleware on mobile devices. Low-footprint CORBA implementations are being worked at. Also, the OMG is working on a draft specification titled "Wireless Access and Terminal Mobility in CORBA" (dtc/01-05).

The CORBA specification was first presented back in 1992. Thus, CORBA was not designed for mobile communications and is used almost exclusively in server environments over corporate networks. The standard has gained popularity until about 1998. With the success of the Java platform (and notably J2EE application servers), attention has been moving away from CORBA, towards middleware solutions which are better integrated into the Java platform and are less complex than CORBA: RMI and Enterprise JavaBeans (EJBs) became popular.

As a matter of fact, CORBA was conceived for computing environments in which multiple programming languages are used. However, in homogeneous environments, for example, J2EE or Microsoft .NET, developers tend to prefer the middleware tools integrated into that environment.

These are the points in favor of using CORBA on mobile devices:

- CORBA is a well accepted standard
- CORBA can be used from various programming languages, and not only from Java. CORBA is programming-language "agnostic"
- There is a substantial amount of CORBA experience, books etc. available.
- If a company has invested a lot in developing CORBA based services, mobile clients can connect to those services more easily, by using a CORBA ORB on the device.

These are the points against using CORBA on mobile devices:

- CORBA is inherently based upon a "request/response" synchronous communication model. There is a misfit between this communication model and between the nature of wireless networks: wireless networks are packet based, sporadic network disconnects do occur, transmission speeds and –delays vary a lot, etc.
- The CORBA communication model assumes that communication links are stable, have low error rates, and that network disconnections occur only rarely. This is an acceptable assumption in corporate (wireline) environments, but not in wireless networks.
- Introduction of the Enterprise Java (J2EE) technology has moved attention away from CORBA to other communication mechanisms, notably RMI and JMS.
- Implementing bi-directional communication is awkward, as this normally requires the provision of a CORBA server object on the mobile device.
- CORBA is based on a classical client/server interaction model. However, wireless services often require push notifications, queued communication, and peer-to-peer sessions. One could argue that this is granted through the CORBA Notifications service and through CORBA Time-Independent Invocations (TII). However, we believe that these mechanisms are too "heavy weight" and complex to be deployed on mobile devices.
- ORBs typically support only the TCP/IP protocol, but no packet-based protocols optimized for 2.5G and 3G networks.

In summary, a mobile CORBA ORB is to be considered by companies that have built a substantial amount of CORBA based services, and want to make these accessible to applications running on mobile devices. There is, however, a misfit between the CORBA communications model and the characteristics of wireless networks.

3.1 References

- OMG standards: http://www.omg.org/
- Wireless CORBA specification: http://www.omg.org/techprocess/meetings/schedule/Telecom_Wireless_FTF.html

4 JMS Messaging Middleware

Messaging middleware operates using message queues. These queues are hosted on both the mobile devices and the back-end servers. Communication is fully bi-directional, one-to-one, or one-to-many. Outbound messages are added to a queue and are sent when a network

connection can be established between a mobile device and a server. This enables effective communication between client and server to take place despite sporadic network disconnects or periods of off-line service usage.

JMS is the de-facto standard for messaging middleware in the Java domain. JMS is part of the J2EE platform, and must be offered by application server vendors, in order to claim J2EE compliance.

These are the points in favor of using JMS middleware on mobile devices:

- Messaging middleware has been deployed very successfully in mission critical business systems, since the 1970s.
- Messaging middleware has a large share of the middleware market and is the solution of choice for financial trading floor systems, Enterprise Application Integration (EAI), order processing systems, and other areas.
- There exist very large JMS installations with thousands of concurrent users.
- JMS is an asynchronous transport. Asynchronous transports are ideally suited to packet-oriented networks over which 2.5 and 3G services will operate.
- Although JMS was not explicitly designed for mobile devices, it provides an ideal abstraction layer for developing mobile applications.
- Increased scalability: Many mobile devices can send messages to a server simultaneously. When messages arrive at the server, they are added to an inbound queue and can be dealt with when resources are available, or can be forwarded to other servers for load sharing.
- JMS middleware provides mechanisms for implementing fault-tolerance and load balancing. This is important for mobile services, which are likely to have large numbers of concurrent users.
- JMS middleware is inherently more scalable than CORBA: Communication is mostly asynchronous, and hence throughput is increased. Also, JMS messages can be routed from one data center to another, or can be dispatched to a cluster of servers for load sharing.
- The JMS model is simpler and less complex than CORBA. Therefore developers need less time to get up to speed.
- In practice, JMS allows wireless services to operate more responsively, to recover from sporadic network outages easily, and to allow mobile applications to be operated offline. This makes for a much richer user experience.
- Messaging can be implemented elegantly atop Bluetooth, Wireless LAN, GPRS, UMTS, and Mobitex. etc.
- Messaging middleware typically provide more QoS customization, as well as integrated security.
- JMS implementations for mobile devices are becoming commercially available, for example, iBus//Mobile from Softwired.
- JMS can be integrated with XML and SOAP, as well as with other middleware technologies (CORBA, EJBs, IBM MQSeries, Microsoft .NET, etc.).

These are the points against using JMS middleware on mobile devices:

- JMS is of interest only if applications are written predominantly (but not exclusively) in Java.

- JMS is part of the J2EE platform and is not "officially" part of the J2ME platform yet. However, this is very likely to change in the near future.

4.1 References

- Introduction to JMS (Java Message Service) and Wireless JMS:
  "Professional JMS", Scott Grant et al., Wrox Press Inc; ISBN: 1861004931
- iBus//Mobile wireless JMS product:
  http://www.softwired-inc.com/products/mobile/mobile.html
  http://www.Hayabooza.com/
- JMS Specification: http://java.sun.com/products/jms/docs.html

In summary, JMS provides a compelling interaction model for mobile applications. JMS allows applications to run efficiently atop 2.5 and 3G networks and to support disconnected operation. JMS fits very well into the micro Java end enterprise Java environment. Using JMS, interactive wireless services can be developed more quickly, and user experience is improved. Since JMS is a de-facto standard, companies face less risk of being tied to a single vendor.

5 Vendor Proprietary Middleware Interfaces

In any case, we strongly recommend you pick a mobile middleware platform compatible with either CORBA or JMS. Those are the predominant middleware standards in the Enterprise Java world. However, most of the mobile middleware products offered on the market today are not written in Java, and come with programming interfaces which are vendor proprietary.

By embarking with such a solution, you will need to train your staff on those vendor-proprietary interfaces, and your wireless software will by tied to that vendor's solution. This means you risk redoing your wireless architecture at a later point if you wish to switch to Java or to a standard-based middleware platform.

6 Conclusions

Wireless networks behave differently than wireline networks: devices tend to lose and regain network coverage, sporadic network disconnects do occur, bandwidth is much lower and varies substantially over time. Communications middleware thus needs to addresses these problems and to provide an adequate programming model. In this article we have shown that Java messaging middleware (JMS) is better suited than CORBA for running applications over wireless networks. The main reason is that JMS provides an asynchronous, message based transport. Using message queues hosted on both the client and the server side, JMS applications can be operated in disconnected mode, and data synchronization occurs transparently and immediately, without user intervention.

Dr. Silvano Maffeis is CTO at Softwired, a company specializing in mobile messaging middleware. He holds 6 years of practical experience with CORBA middleware, and 4 years of practical experience in Java messaging middleware and wireless. Silvano Maffeis is the author of various articles and books related to middleware. He can be reached at Silvano.Maffeis@softwired-inc.com, Phone: +41-1-445-2370.