AN EFFICIENT SYSTEM FOR EJB MOBILIZATION

Liang Zhang, Beihong Jin, Li Lin and Yulin Feng
Institute of Software, Chinese Academy of Sciences, Hai Dian, Beijing, P.R. China

Keywords: Mobile computing, middleware, EJB, MIDP.

Abstract: In these days, conducting business requires more and more employees to be mobile. To be efficient, these mobile workers need to access the enterprise applications with their mobile devices at anytime and anywhere. How to efficiently extend the enterprise applications to the mobile devices becomes a challenging task to the enterprise. In this paper, we present our recently developed system for mobilizing enterprise applications. Considering the characteristics of wireless media, our system can dynamically choose the most appropriate communication method and provide the synchronous exactly-once communication semantic. Security is explored by providing data encryption, two-way authentication and a simple tool for managing the access control list. We also develop a mechanism for supporting priority service. Thread pool and object caching are implemented to increase the efficiency. Lastly, our system offers various tools to enhance the development automatism, while still allowing the programming flexibility by providing a rich set of APIs.

1 INTRODUCTION

We live in a fast-paced world that is easily serviced in an office. But conducting business these days requires more and more employees to be mobile. Executives, consultants and sales people have always travelled as part of their jobs. Many of these mobile workers must return to their offices to do paperwork. This apparently reduces the workers' working hours, increases expenses and worst causes both the enterprise and the workers from getting real-time data, which will put the business in an unfavourable situation under today's keen competition (Dave, 2002).

To keep the competitive edge, more and more enterprises have realized the importance of *mobile computing*, with which the mobile workers can easily access the same set of applications provided at the office while they are away and the enterprise can promptly place information to its mobile workers as needs arise. There is a lot of data showing that mobile computing can create a very high return on investment and bring the enterprise sustainable competition advantages (Douglas, 2004). The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the system architecture. Section 4 describes the communication module. Section 5 deals with the security issue. Section 6 introduces the detail of mobilizing enterprise applications. The final section concludes the paper.

2 RELATED WORK

As more and more workers are mobile now, many IT companies have released their solutions to mobile computing. Basically, there are two types of solutions: **B**rowser/**S**erver (BS)-based solutions and **C**lient/**S**erver (CS)-based solutions. For the former solutions, the mobile worker can use the on-device web browser for accessing the enterprise web-based applications. Microsoft's ASP .Net Mobile and IBM's WebSphere Everyplace Access belong to this type of solutions. Although the BS solution imposes no requirements on the client side, it requires the enterprise to invest new products on the server side for context adaptation and transcoding. The enterprise has to also invest in a portal product and wrap the applications into portals for a portal-based solution. Another problem of a BS solution is it cannot support offline operations (Thierry, 2003).

For the CS-based solutions, the mobile worker can invoke an enterprise application by executing a client program on the mobile device. Typically, the enterprise application is deployed on an application server. As we know, in the world of application middleware, there are two technologies competing with each other. One is the .Net technology by Microsoft and the other is the J2EE technology by SUN and many IT titans. Because J2EE possesses many favourable features such as reliability, scalability and.

Zhang L., Jin B., Lin L. and Feng Y. (2007).
AN EFFICIENT SYSTEM FOR EJB MOBILIZATION.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - SAIC*, pages 173-178
DOI: 10.5220/0002351601730178
Copyright © SciTePress

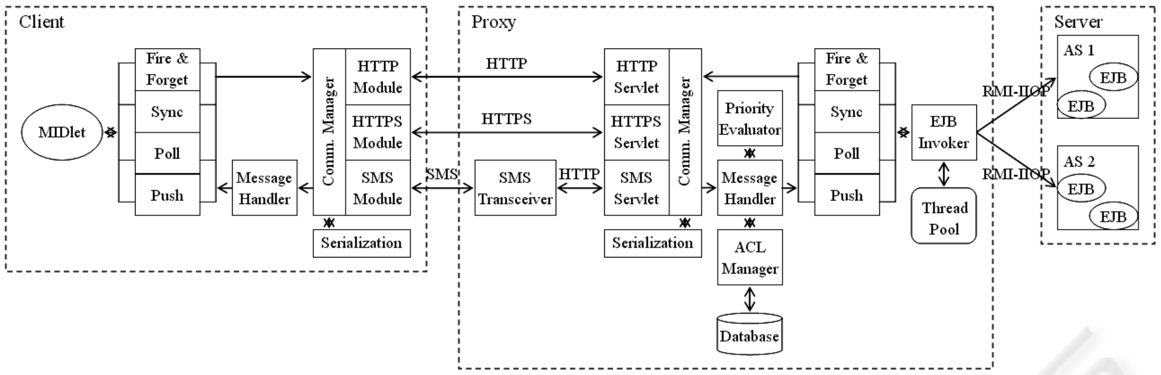

Figure 1: The overall system architecture.

most importantly Java's "Write Once, Run Everywhere" capability, more and more enterprises have adopted the J2EE-certified middleware product for deploying and managing their applications. Therefore, we target our system to the J2EE-certified middleware. In this context, **E**nterprise **J**ava**B**ean (EJB) is widely used because of its features such as easy portability and fast development (Roman, 2004).

On the client side, the client program can be developed with either native programming or platform-based programming. For the native programming, the developer typically has to first install the **S**oftware **D**evelopment **K**it (SDK), then develop the client program with the SDK and finally deploy it to the mobile device. Obviously, this approach makes both the application development and deployment difficult due to the various mobile devices. The platform-based programming provides a device-agnostic solution. Microsoft's .Net Compact Framework and Sun's J2ME technology belong to this type. However, although .Net Compact Framework ensures the client program developed to be run on any Windows CE device, it is not a truly device-agnostic solution as it does not support its program run on other operating systems. The only device-agnostic solution at present is provided by J2ME. J2ME specifies several classes of configurations and profiles for each band of mobile devices, within which MIDP2.0 (Jim, 2002) has the largest popularity. Currently, almost all new mobile devices support MIDP2.0 as their standard configuration. Therefore, at the client side, we choose MIDP2.0 as the target platform.

Our system can be described as EJB component invocation on MIDP2.0 mobile devices. We believe our system will provide the greatest applicability while requiring the lowest investment

3 SYSTEM ARCHITECTURE

The main goal of our system is to provide convenient API support on mobile devices for transparently invoking the existing EJB components at the enterprise. Figure 1 shows the overall architecture of our system, which adopts a three-tier approach. The client-tier is a packaged MIDlet program running on the J2ME platform. Our system provides the following modules in this tier.

- Client side communication module – This module provides the fundamental support for mobile applications as they need to interact with the enterprise. Since wireless communication suffers from the poor transmission media – the air, it is more likely to lose messages. We develop this module to shield the upper layer program from handling message loss and provide the synchronous exactly-once communication semantic.
- Serialization module – Since MIDP2.0 does not support object reflection and serialization, we develop this module to serialize and deserialize objects to and from a byte stream so that they can be transmitted over the wireless.
- Message Handler – A trigger when receiving a new message
- Client side exchange pattern module – Since wireless communication normally causes long transmission delay, it is desirable to provide not only the synchronous EJB component invocation interface but also asynchronous interfaces to the developer.

The developer can develop the MIDlet program by simply invoking the desired interfaces provided by the client side exchange pattern module.

The proxy-tier, except for SMS Transceiver, is a packaged web program that can be run in any servlet container. Our system provides the following modules in this tier.

- Proxy side communication module – This module together with the client side communication module provides the synchronous exactly-once communication semantic.
- Serialization module – The same as the client side
- Message Handler – The same as the client side
- ACL Manager – ACL stands for **A**ccess **C**ontrol **L**ist. This module is responsible for authenticating the client's identity and controls the access authority.
- Priority Evaluator – This module evaluates the priority of each EJB component invocation request and recommends the next request to process with the highest priority.
- Proxy side exchange pattern module – This module together with the client side exchange pattern module provides different EJB component invocation patterns.
- EJB Invoker – This module performs the actual invocation of EJB components.

The server-tier is the multiple EJB components that may span across several **A**pplication **S**ervers (AS) of different brands.

Our system in addition provides several tools for assisting the development and management of mobile applications.

- SMS Transceiver – Since SMS messages cannot directly be sent to or from the proxy web program, we provide a simple program called SMS Transceiver to handle SMS transmission and interact with the proxy through HTTP. The developer can substitute SMS Transceiver by developing a program that bridges the SMS service provider and the proxy the same way as SMS Transceiver does.
- Account Management Tool –This tool is for managing the client's account information such as user name and password.
- ACL Management Tool – This tool is for managing the eligibility for calling a certain method of a certain EJB component.
- EJB Toolkit – This is the most important tool for our system. During the development of a mobile application, the developer needs to modify some parameters of certain modules according to the situation by editing the proper configuration files. He also needs to create some new classes and incorporate them into certain modules. Of course, it is not feasible to require the developer to manually perform these modifications. He should only focus on the development of the MIDlet program. EJB Toolkit automates all of these modifications and packages the proxy side into a **W**eb **Ar**chive (WAR) file ready for deployment. Furthermore, it can generate a sample MIDlet program, based on which the developer can quickly develop the program for the application. After that, the tool can package the client side for deployment. If the developer needs, the tool can even test the program by loading the WAR file into Apache Tomcat and running the MIDlet with SUN's J2ME Simulator. We believe this tool will greatly increase the usability of our system.

4 RELIABLE COMMUNICATION

As described in Section 3, a reliable communication mechanism is the fundamental requirement for any mobile enterprise application that needs interaction with the server. Due to the poor transmission characteristics of the wireless media, messages are frequently lost, duplicated, received in different order, and sometime even transmitted without an upper bound time limit. It is desirable to have a wireless communication mechanism that can shield all these defects and provide to the upper layer program with the synchronous exactly-once communication semantic. Here synchronous means messages are delivered in order with an upper bound time limit, and exactly-once means messages are reliably delivered without duplication.

Table 1: Characteristics of HTTP/HTTPS/SMS.

	Reliability	Duplication	Order	Time Limit
HTTP	Yes	No	Yes	Yes
HTTPS	Yes	No	Yes	Yes
SMS	No	Yes	No	No

The **G**eneric **C**onnection **F**ramework (GCF) in MIDP2.0 defines six connection interfaces, within which only HTTP and HTTPS interfaces are useful because all the others are either for local connections or not able to pass the enterprise's firewall. In addition, we discover that the optional Wireless Messaging API (Marquart, 2002) is widely supported by MIDP2.0 mobile devices. Therefore, it is possible that we use SMS as the alternative path for communication. Let us first look at the characteristics of these three communication methods in Table 1.

HTTP and HTTPS are based on TCP/IP, and therefore in case the link is connected, they can satisfy the desired synchronous exactly-once semantic.

However, it is very likely that the wireless link is disconnected due to various reasons and message loss occurs. SMS sends messages asynchronously, and therefore it cannot guarantee our requirement entirely. So we develop another component above these three methods to guarantee the synchronous exactly-once semantic. We name this component Communication Manager as shown in Figure 1. In fact, Communication Manager performs a simplified TCP's sliding window algorithm (Tanenbaum, 2002) by piggybacking the next expected message id to the messages sent so that the other side can determine whether to need to retransmit some messages. Piggybacking is achieved with the Decorator pattern (Gamma, 1995) by embedding the original message in a decorator class which defines a new NextExpected field and some relevant methods.

Communication Manager maintains three queues: Sending Queue, Receiving Queue and Exception Queue. Sending Queue is further divided into three sub-queues for the three communication methods. Communication Manager determines the proper sub-queue for each new message and transfer messages from one sub-queue to another if the corresponding communication method does not work. In our implementation, if Communication Manager finds HTTP/HTTPS communication fail for twice, it will transfer the messages in the HTTP/HTTPS sub-queue to the SMS sub-queue. If a message sending fails for the predefined maximum retries, it will be reported to Exception Queue to be handled by the upper layer program.

Since wireless communication is either charged by time or data, we provide two different communication modes for HTTP/HTTPS. The user will be prompted a window to select the mode for the first time and the selection will be stored in the persistent **R**ecord **M**anagement **S**ystem (RMS) supported by MIDP2.0. If the communication is charged by time, the client side HTTP/HTTPS will periodically poll its sub-queue and establish the connection when it has some messages to send. Notice that the HTTP/HTTPS connection can only be established from the client side, so the client side has the responsibility to establish the connection even when it has no messages to send but the proxy side may have some invocation return values to deliver. However, if the connection is charged by data, the client side HTTP/HTTPS will simply maintain a long term connection. In this case, both sides can immediately transfer whatever messages in the sub-queue. If the connection fails, the client side will wait a few seconds before trying to establish another connection.

5 SECURITY

In mobile computing, security is a very important topic because data is transmitted over the unprotected air media. Basically, there are three issues to consider for security: encryption, authentication and authorization, which are discussed in the next three paragraphs in detail.

Encryption can prevent attackers from eavesdropping the transmitted data. Of course, it is great to encrypt every transmitted data. However, this will cost a substantial amount of system resources because encryption is a computationally expensive process. It would be much reasonable to only encrypt those critical data such as credit card information and leave the rest in plaintext. Therefore, we provide two sets of interfaces to the developer, one for unencrypted and the other for encrypted. If the developer chooses the unencrypted interface, Communication Manager will select HTTP as the default communication method. If the developer chooses the encrypted interface, Communication Manager will select HTTPS as the default method since HTTPS in MIDP2.0 is carried over any of the following security protocols: **T**ransport **L**ayer **S**ecurity (TLS), **S**ecure **S**ockets **L**ayer (SSL), **W**ireless **T**ransport **L**ayer **S**ecurity (WTLS), and WAP TLS Profile and Tunneling Specification (Jonathan, 2002). SMS is also a secure communication method, and therefore Communication Manager can use SMS as the alternative path for both HTTP and HTTPS in case they do not work.

Authentication is a method to prove the identity of the other side of the communication. With both encryption and authentication, the communication two parties are confident that not a third person can involve the communication. In our system, HTTPS can be used for the client authenticating the proxy. But there is not a ready vehicle for the proxy authenticating the client. Although SMS can support two-way authentication, it cannot guarantee the client is not a stolen device from a legitimate user. There are two methods available for the client authentication. One is password authentication and the other is certificate authentication. In our system, we does not use certificate authentication because it involves the complicated task of distributing the private secret, i.e., for each copy of the client program, it should be packaged with a unique client key and certificate. Also, it cannot prevent a stolen device from accessing the application. We adopt password authentication and implement ACL Manager to authenticate the client with the provided user name and password.

It is common that not every legitimate user can access the same set of EJB components and methods. Based on the client authentication, ACL Manager controls whether an invocation request should be filtered or carried out for processing.

6 EJB INVOCATION

6.1 Exchange Patterns

In our system, we provide not only one synchronous EJB component invocation interface (i.e., Sync) but also three asynchronous interfaces (i.e., Fire-and-Forget, Poll and Push) to the developer. Each interface defines a unique message exchange pattern as follows:

- Sync – The client issues a request message to the proxy and makes the current thread pause until it receives the response.
- Fire-and-Forget – The client issues a request message to the proxy but is immediately returned because it does not expect any response; this interface can be used for invoking those methods without return values.
- Poll – The client issues a request message to the proxy but is immediately returned with a key; the client continues with other business logic; later the client can use the key to poll for the response.
- Push – The client issues a request message to the proxy together with a response handler; the client continues with other business logic; the response handler is triggered immediately after the response is received.

As we can see, the asynchronous exchange patterns do not require the current thread pause for the response, which is extremely useful when wireless communication shall cause long transmission delay. In our implementation, the client side and proxy side exchange pattern modules work together to provide the four invocation patterns. There is one tricky that the Poll pattern in the proxy side module actually performs the same as the Sync pattern and the response is stored in the local memory in the client side module. When later the client requests the response with the key, this process will only involve the client side, which will reduce the wireless transmission cost and increase the response time.

6.2 Prioritized Service

The proxy side Message Handler together with Priority Evaluator can provide prioritized service to EJB component invocation. As shown in Figure 2, each incoming request is first classified according to whether it is a synchronous or asynchronous invocation. For the synchronous invocation, it is inserted into the synchronous queue specific to the client of the invocation. For the asynchronous invocation, it is inserted into the asynchronous queue shared by all clients. The reason is that we desire synchronous invocation to be handled in the same sequence as they are issued, but relax the execution sequence of asynchronous invocation. Message Handler repeatedly invokes Priority Evaluator to evaluate the next invocation request with the highest priority, and passes the winner to EJB Invoker for execution. The candidates for each round of evaluation are mapped in Figure 2 within the dash line that include all the requests in the asynchronous queue and the first request in each synchronous queue. In our implementation, Priority Evaluator considers four factors as follows:

- The period that the request has already stayed in the queue *T1*
- The average invocation time *T2*, which is retrieved from EJB Invoker
- The average round trip time *T3*, which is retrieved from the proxy side Communication Manager
- The request mode, which is either synchronous or asynchronous

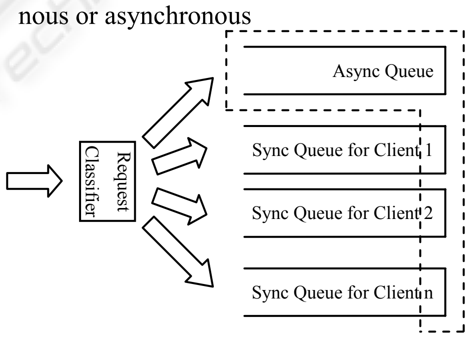

Figure 2: Request classification.

The first three factors add up affecting the response time from the client's point of view. The priority is calculated with the following equation:

$$P = W1 \times Norm(T1 + T2 + T3) + W2 \times (\text{Request Mode} = \text{Synchronous} ~?~ 1 : 0) \quad (1)$$

where *W1* and *W2* respectively mean the weighting factor of the response time and the request mode, and we set both of them to 0.5. *Norm(t)* is a normalization function for time and we define it as follows:

$$Norm(t) = \begin{cases} t/5 & 0 \le t < 5 \\ 1 & t \ge 5 \end{cases} \quad (2)$$

The developer can define his own priority evaluation strategy by replacing our Priority Evaluator.

6.3 Efficient Invocation

EJB Invoker performs the actual invocation of EJB components. It could be the system bottleneck as all the requests are handled in this module. To increase efficiency, we use two strategies: thread pool and object caching.

At the start-up of the proxy, a thread pool with the initial capacity is created. Whenever there is a new request, EJB Invoker will select an available thread from the pool to handle it. As the workload increases, the pool size will also increase. The principle is to always make sure there are enough threads for handling requests unless reaching the predefined maximum. To accomplish this goal, EJB Invoker continuously monitors two variables: the average request incoming rate λ and the average invocation time $1/\mu$. (The average invocation time *T2* defined in Section 6.3 has different meaning from $1/\mu$. *T2* is actually the summation of the average time waiting for an available thread plus $1/\mu$.) If the number of available threads is less than the expected number of new requests (λ/μ) before any working thread can accomplish its task and become available, EJB Invoker will create a new thread to the pool. If the pool size reaches the maximum, EJB Invoker will start to queue requests when all the threads are unavailable. However, if the number of busy threads plus the expected number of new requests is less 2/3 of the pool size, EJB Invoker will destroy one available thread to preserve system resources unless the pool size has reached the initial capacity.

During the invocation of an EJB component, it has to first create a specific context object and then create an EJB home object based on which a reference to a new EJB object can be made. After that, the invocation can be accomplished by calling the appropriate method of the EJB object reference. Creating the whole set of objects during each invocation is apparently a time consuming task. Therefore, we cache these objects in a three-layer tree data structure. The top layer is context objects, the middle layer is EJB home objects and the bottom layer is EJB object references. The upper two layers can actually be created at the start-up of the proxy as the set of EJB components has already been determined during the application development. For the third layer, since each EJB object may be initialized with different parameters, it cannot be created prior to the invocation takes place. However, once created, the EJB object reference shall be cached for later usage.

7 CONCLUSION

In this paper, we present our recent developed system for mobilizing enterprise applications. Our system adopts a three-tier Client/Proxy/Server architecture. As the popular of Java, our system is targeted for providing efficient accessing to EJB components from MIDP2.0 mobile devices. We consider reliability and security issues since the wireless media is both unreliable and unprotected. To deal with the long transmission delay, we provide to the developer not only synchronous invocation API but also asynchronous APIs. We also develop a simple mechanism for supporting priority service. Thread pool and object caching are implemented to increase the system efficiency. Lastly, we develop a set of tools and programs for assisting the development and management of mobile applications, which will greatly increase the usability of our system.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China under Grant No. 60673123 and the National Hi-Tech Research and Development 863 Program of China under Grant No. 2006AA01Z231.

REFERENCES

Dave B. and Linda M. P., "Extending Enterprise Applications to Mobile Users", Technical Report in IBM Pervasive Computing Division, July 2002.

Douglas D., "The Return on Your Mobility Investment: Enterprise Opportunities for Windows Mobile-based Pocket PCs and Smartphones", Technical Report in Microsoft Mobile Devices Division, April 2004.

Jim V. P. and James W., "JSR 118: Mobile Information Device Profile 2.0", November 2002.

Marquart C. F., "JSR 120: Wireless Messaging API", August 2002.

Tanenbaum A., "Computer Networks", Prentice Hall, August 2002.

Gamma E., Helm R., Johnson R. and Vlissides J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, January 1995.

Roman E., Rima P. S. and Gerald B., "Mastering Enterprise JavaBeans", Wiley, December 2004.

Thierry V. and Ray O., "Supporting Disconnected Operation in Wireless Enterprise Applications", Java BluePrints for Wireless White Paper, June 2003.

Jonathan K., "MIDP Application Security", Sun Technical Articles and Tips, December 2002.