行政院國家科學委員會專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※
※ ※
※ 建置一 EJB Structure ※
※ ※
※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：V 個別型計畫 □整合型計畫
計畫編號：NSC 89 –2213 – E – 009 – 178
執行期間： 89 年 8 月 1 日 至 90 年 7 月 31 日

計畫主持人： 王豐堅
共同主持人：

本成果報告包括以下應繳交之附件：

□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

執行單位：國立交通大學資訊工程系

中 華 民 國 90 年 12 月 31 日

行政院國家科學委員會專題研究計畫成果報告

計畫編號：NSC 89 –2213 – E – 009 – 178
執行期限：89 年 8 月 1 日至 90 年 7 月 31 日

主持人：王豐堅　國立交通大學資訊工程系
計畫參與人員：楊基載等研究生

-. 摘 要

Enterprise JavaBeans（EJB）架構是被設計用來利用現存在伺服器端的原件組裝成一套具有擴充性、安全性、靈活性、分散式和輕薄短小的應用程式。支援 transaction 功能是 EJB 架構的特徵之一。本研究實作一個建構在 EJB 架構的 Transaction 系統。這個 Transaction 系統允許程式開發者利用多個 enterprise beans 在一個 transaction 中同時對多個資料庫作存取。甚至，這些資料庫可能位在不同的伺服器裡。除了管理 transaction 外，這個 Transaction 系統還能自動根據 enterprise beans 的特性來執行 transaction 動作，包括開啟、完成和回返一個 transaction。因此，程式開發者便不需要在程式中明確的區分出 transaction 的範圍，而且 enterprise beans 的撰寫也變成更加簡單。

關鍵字: 企業 Java Bean, 伺服器,交易系統,資料庫,分散式系統.

Abstract

The Enterprise JavaBeans (EJB) [1] architecture is designed to enable building one scalable, secure, flexible, and distributed chin-client applications as reusable and server-side components [2][13]. The EJB architecture was proposed to support for transactions. The research [3] implemented a transaction system of EJB architecture, which allows application developers to invoke enterprise beans to update data in multiple databases in a single transaction. These databases can be distributed in different servers [4]. In addition to manage the transactions, the transaction system could automatically demarcate a transaction on behalf of the enterprise bean, including begin, commit, and rollback it. Hence, the application developer need not explicitly specify transaction demarcation code and the enterprise beans are simpler to develop.

Keywords: EJB, Internet, Client-server, Database, distributed systems, transaction

1. Introduction of EJB

Recently, the development of software design has a great variation. The demands for the software design have tended to provide the capabilities of distributed, thin-client, multi-tier, scalability, reusable and so on. Based on these advancements, in March 1998, Sun Microsystem Inc. proposed the specification of Enterprise JavaBeans (EJB for short)[1], which satisfies all these demands. The EJB technology expects to provide an environment for developers to produce scalable server-based applications using existing components without worrying about the complexity of multi-threaded, transactional and distributed process programming.

The EJB architecture defines a model to simplify the development of distributed enterprise applications, which develops and deploys the reusable components via the network. These components are called *enterprise beans*. The developers only present the invocation of the enterprise beans and need not care the low-level transaction, state management, multi-threading and other complex low-level APIs when developing applications. These complex actions of management will be handled by the EJB architecture automatically. In other words, the developer only concerns about the presentation logic — “thin client”. Usually, a thin-client application is easier to be managed than traditional client/server applications, and both scalability and performance will increase.

"Write Once, Run Anywhere" (WORA) [9] [10] is one of the primary tenets of Java technology. The EJB architecture also follows the philosophy of WORA. Once an enterprise bean is developed, it then can be deployed on multiple platforms without recompilation or modification. In this case, the EJB architecture not only reaches the goal of reuse in object level but also enables portability. Besides, although the EJB architecture is designed with JAVA base, it could be compatible with many applications, such as existing server platform, non-JAVA programming language applications, other JAVA language API and CORBA protocol. [6] [12]

In the EJB architecture, there supports high scalability by using the multi-tier [7][8], distributed application architecture. A multi-tier application is an application that has been partitioned into multiple application components. Multi-tier applications provide a number of significant advantages [13] over traditional client/server architectures, including improvements in scalability, performance, reliability, manageability, reusability, and flexibility.

The EJB architecture is a new and strong infrastructure. [14] It not only connects and manages a wide array database, mainframe applications and customized software solutions; but also provides high availability to handle enterprise-scale computing. The EJB architecture is also a technology for rapid application development, and existing IT investments integration and protection.

2. Transactions in the EJB Architecture

In a client/server system, a process might be divided into several tasks which execute in various sites. During the execution, the working system must guarantee either all these tasks complete, or none of them do at all. These tasks may be treated as in a transaction that owns four properties, including atomicity, consistency, isolation and durability [11]. A transaction system [10] can guarantee the success of the activities in each execution sequence within a distributed environment. The transaction system is an essential component of the EJB architecture and supports flat transactions of two-phase commit protocol.

2.1 The Java Transaction API

In the EJB architecture, the JAVA Transaction API (JTA) [10] is a specification of the interfaces between a transaction manager and the EJB Container, enterprise beans and Java client program. JTA provides a programming interface to start transactions, join existing transactions, commit transactions and rollback transactions. The EJB architecture allows both the Bean Providers and the application developers in EJB Client to use the *javax.transaction.UserTransaction* interface of JTA to control transaction boundaries programmatically. JTA is implemented in EJB Container which provides the *javax.transaction.UserTransaction* interface for the enterprise beans and EJB Client, and the *javax.transaction.TransactionManager* interface to control transaction boundaries on behalf of the EJB Client being managed.

2.2 Transaction approaches of EJB Architecture

The EJB architecture defines three approaches [5] to demarcate transaction boundaries. The first one is a *client-managed* transaction demarcation, where a client makes explicit calls to begin or commit such a transaction. The other two are *bean-managed* and *container-managed* transaction demarcation. An enterprise bean can make a call to begin or commit a *bean-managed* transaction. The EJB Container automatically begins or commits a *container-managed* transaction which is declared by an enterprise bean.

Client-managed transaction

In the EJB architecture, the application developer uses the *javax.transaction.UserTransaction* interface defined in JTA to demarcate

transaction boundaries. The developer could firstly obtain the *javax.transaction.UserTransaction* interface via JNDI (JAVA Naming and Directory Interface). Then, he/she can demarcate the transaction with *begin()* and *commit()* method provided by the *javax.transaction.UserTransaction* interface.

Bean-managed transaction

The enterprise bean can demarcate a transaction programmed by the Bean Provider. The enterprise bean uses the *getUserTransaction()* method from the EJBContext to obtain the *javax.transaction.UserTransaction* interface, which is then used to demarcate a transaction. After getting the transaction status, the enterprise bean can execute the business methods in a transaction between *UserTransaction.begin()* and *UserTransacton.commit()*. Also, it can roll back a transaction using *rollback()* method.

Besides coding enterprise beans with *bean-managed* transaction demarcation, the Bean Provider must also set the attribute "transaction-type element" with *"Bean"* in the deployment descriptor file. The attribute with *"Bean"* specifies that the enterprise bean must be executed with *bean-managed* transaction demarcation in the EJB Container.

Container-managed transaction

The development of enterprise beans with *container-managed* transaction demarcation can be simplified because the developers do not have to explicitly code the transaction's boundaries. When a business method of the enterprise bean is invoked, the EJB Container interposes the method invocation and gains the permission to control the transaction demarcation based on the transaction attribute of the business method.

In the deployment descriptor, each business method has its correspondent transaction attribute, which specifies how the EJB Container should demarcate the transaction. In the EJB architecture, six values for the transaction attribute are defined: *NoSupported, Required, Supports, RequiredNew, Mandatory* and *Never*.

The Bean Provider is responsible to set up the deployment descriptor file as follow: For each session bean, Bean Provider must declare the "transaction-type element" as either *"Bean"* or *"Container"* and set the value of transaction attribute of each business method. For each entity bean, Bean Provider must set the value of transaction attribute of each business method but never set the transaction-type because all entity beans have to use *container-managed* transaction demarcation.

3.Design of Transaction System

3.1 The System Model

Our transaction system is designed and implemented to own the transaction capabilities of the EJB architecture. The transaction system helps EJB Container to select a transaction action when a business method is invoked. The transaction system provides JTA for the EJB applications and the EJB Container to demarcate the transaction. Also, the transaction system manages the transactions based on 2-phase commit protocol. Our transaction system architecture, shown in Figure 1, contains three primary components, including Transaction Code Generator (TCGen), Java Transaction API (JTA) and a Transaction Manager.

In Figure 1, TCGen component, a sub component on the Deployer, generates transaction codes to help the EJB Container to select the transaction action. The implementation of JTA provides the EJB applications and EJB Container a programming interface to demarcate a transaction, eg: start, join, commit and roll back transactions. The Transaction Manager component is used to manage all transactions.

3.2 Design of Transaction Code Generator

When a business method is invoked, the EJB Container selects the transaction action for the method according to the transaction attribute, transaction-type and status of client. The EJB Container can also decide to commit, hold or rollback the

transaction when the business method completes. Hence, there exists a pair of transaction controls in the front and end of the business method correspondingly.

In our design, the transaction controls are coded as two TCGen methods named *PreInvoke()* and *PostInvoke()*. The

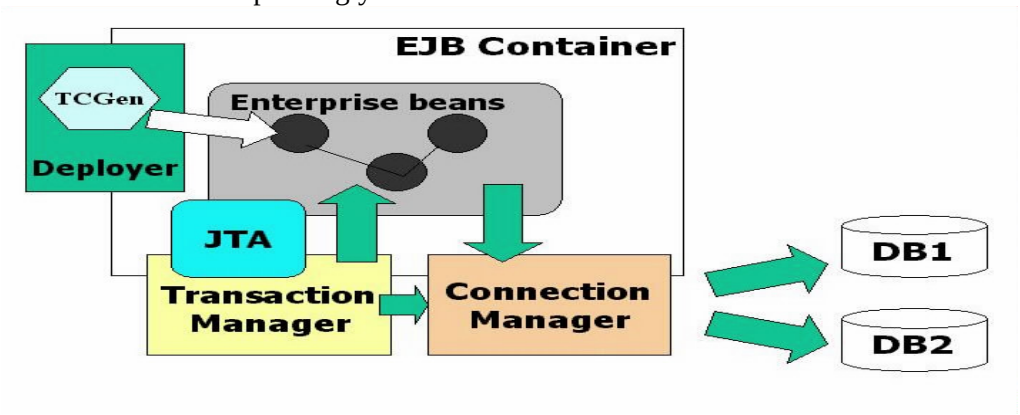

Figure 1 The system architecture of transaction system

PreInvoke() method selects the transaction action. The *PostInvoke()* method is used to commit the transaction started in *PreInvoke()*. These two TCGen methods are coded in the class of *javax.ejb.EJBObject* interface; it is inherited by the class of the enterprise bean's Remote interface. When Deployer generates the class of enterprise bean's Remote interface, it asks TCGen object to add a pair of TCGen methods to each business method. Within the class, the TCGen inserts method invocation of *PreInvoke()* in the front of a business method and *PostInvoke()* at the end.

4. Conclusion and Future Work

In the short report, a transaction system of the EJB architecture, including TCGen, JTA and Transaction Manager, is designed and implemented. The transaction system allows three different transaction approaches defined in EJB architecture to demarcate a transaction in the EJB Container. Especially, the transaction system could help EJB Container to select the transaction action when a business method is invoked. Hence, the application logic requires no transaction codes and the enterprise beans are simpler to write.

The EJB architecture only supports 2-phase commit protocol for flat transactions. However, the 2-phase commit protocol has an uncertainty period because a processor must detect a stable predicate in order to commit or abort after voting *"Ready"*. To avoid the problem, the transaction system may be rebuilt based on 3-phase commit protocol or better one. Besides, nested transactions are not allowed within the business methods. The transaction system also enhances to take advantage of nested transaction, if these vendors provide support for nested transactions in the future.

In the EJB architecture, it is allowed that a business method to perform other enterprise beans which reside in other EJB Container. Hence, the distributed transactions among many EJB Containers are a future topic.

References

[1] Sun Microsystems, *"Enterprise JavaBeans 1.0 Specification"*, in http://java.sun.com/cgi-bin/download3.cgi

[2] Boher. K., Johnson V., Nilsson A. And Rubin B., "*Business process components for distributed object applications*", Communications of the ACM,Vol. 41, No. 6 (June 1998), Pages 43-48

[3] Hapner M. and Finkelstein S., "*Enterprise JavaBeans Technology: Developing and Deploying Business Applications as Components"*, Java ONE conference, 1998. Available in http://java.sun.com/javaone/javaone98/sessions/T401/

[4] Matena V. and Finkelstein S., "*How to Build Enterprise JavaBeans Server Software*", Java ONE conference, 1998. Avaiable in http://java.sun.com/javaone/javaone98/ sessions/T403/index.html

[5] Roth B., "*Enterprise JavaBeans Technology: Tools that Enable Enterprise JavaBeans Technology*", Java ONE conference, 1998. Available in http://java.sun.com/javaone/javaone98/sessions/T402/index.html.

[6] Ed Roman and Rickard Oberg, *"The Technical Benefits of EJB and J2EE Technologies over COM+ and Windows DNA"*, December 1999, in http://java.sun.com/products/ejb/pdf/j2ee_dnabwp.pdf

[7] Sun Microsystems, *"Simplified Guide to the Java 2 Platform Enterprise Edition"*, 1999, pp. 1-1 — 1-3

[8] Secant Technologies, Inc. *"Secant Extreme Enterprise Server for EJB"*, in http://www.secant.com/docs/EJBpaper/ejbpaper.1.html

[9] Sun Microsystems, *"Java Transaction API Specification 1.0.1"*, http://java.sun.com/cgi-bin/download3.cgi

[10] Sun Microsystems, *"Java Transaction Service"*, in http://java.sun.com/products/jts/index.html

[11] Mike Porter and Martin Van Vliet, *"Expand your server-side toolkit with EJB"*, http://www.sunworld.com/swol-04-1999/swol-04-itarchitect.html

[12] Dr. Andreas Vogel and Chief Scientist, *"Building Enterprise Applications for the Net with EJB, CORBA, and XML"*, in http://www.inprise.com/appserver/papers/ejb/part2.html

[13] Sun Microsystems, *"Java 2 Platform Enterprise Edition"*, in http://java.sun.com/j2ee/index.html

[14] Sun Microsystems, *"Java Naming and Directory Interface Application Programming Interface"*, in http://java.sun.com/j2se/1.3/docs/guide/jndi/spec/jndi