การพัฒนาซอฟต์แวร์เชิงคอมโพเนนต์โดยใช้ EJB 2

Component Base Software Development base on EJB 2

นายเชษฐา ถาวรสถิตย์
นายเชี่ยวชาญ เลิศเอกบุรุษ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

การพัฒนาซอฟต์แวร์เชิงคอมโพเนนต์โดยใช้ EJB 2

Component Base Software Development base on EJB 2

โดย

นายเชษฐา ถาวรสถิตย์

นายเชี่ยวชาญ เลิศเอกบุรุษ

อาจารย์ที่ปรึกษา

ดร.วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาซอฟต์แวร์เชิงคอมโพเนนต์โดยใช้ EJB 2

Component Base Software Development base on EJB 2

ผู้จัดทำ

1. นายเชษฐา ถาวรสถิตย์ รหัสประจำตัว 42010090
2. นายเชี่ยวชาญ เลิศเอกบุรุษ รหัสประจำตัว 42010091

อาจารย์ที่ปรึกษา

(ดร.วรวัฒน์ ลิ้มโภคา)

การพัฒนาซอฟต์แวร์เชิงคอมโพเนนต์โดยใช้ EJB 2

นายเชษฐา ถาวรสถิตย์ 42010090
นายเชี่ยวชาญ เลิศเอกบุรุษ 42010091
ดร.วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2545

บทคัดย่อ

การพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์ (Component-based Software Development) เป็นกระบวนการในการสร้างซอฟต์แวร์ โดยนำคอมโพเน็นต์ที่มีอยู่มาใช้ (reuse) เพื่อช่วยลดเวลาและมูลค่าในการพัฒนา

และเนื่องจากการเติบโตของอินเตอร์เน็ต ทำให้มีการพัฒนาสถาปัตยกรรมบนฝั่งเซิร์ฟเวอร์ (server-side architecture) ที่ช่วยในการสร้างเอ็นเตอร์ไพรส์แอพพลิเคชัน (enterprise application) ขึ้นมา หนึ่งในนั้นคือ J2EE ซึ่งเป็นสถาปัตยกรรมของซันไมโครซิสเต็มส์ที่มี Enterprise JavaBeans (EJB) เป็นสถาปัตยกรรมคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์ (server-side component architecture) เป็นส่วนประกอบสำคัญ J2EE จะใช้ภาษา Java ในการพัฒนา จึงสามารถทำงานได้ทุกแพล็ตฟอร์ม (portability) พร้อมทั้งยังมี API อื่นที่ช่วยให้การพัฒนาเอ็นเตอร์ไพรส์แอพพลิเคชัน (enterprise application) ทำได้ง่าย ได้แก่ JSP, Servlet , JMS , JTA และ JDBC เป็นต้น

ต่อมาภายหลังการเกิดของ XML, SOAP และแนวคิดของเว็บเซอร์วิสทำให้สามารถทำงานข้ามแฟลตฟอร์มได้

วิทยานิพนธ์ฉบับนี้ศึกษาถึงวิธีในการพัฒนาเว็บเซอร์วิสเอนเตอร์ไพรส์แอพพลิเคชัน โดยใช้จาวาเทคโนโลยีในการพัฒนา มีการใช้งานข้ามแฟลตฟอร์มร่วมกันระหว่าง .NET เฟรมเวิร์คด้วย และทำการเปรียบเทียบ เทคโนโลยีของทั้งสองค่าย

Component Base Software Development base on EJB 2

Chettha Thawornsathit 42010090

Cheiwchan lertakburut 42010091

Dr. Worawat Limpoka Advisor

ABSTRACT

Component-based software development is a software process that reused software components to reduce time and cost of development.

The growth of Internet encouraged the evolution of the server-side architecture to promote enterprise application. One of the most famous server-side architecture is J2EE from Sun Microsystem proposing Enterprise JavaBeans (EJB) as a server-side component architecture. Due to implementing by Java, J2EE makes uses of portability to be able to working cross platform and, moreover, complementary API easing up many enterprise application development, such as JSP, Servlet, JMS, JTA and JDBC.

Later , after creation of XML, SOAP and Webservice concept, can work cross Platform.

This thesis is to study the webservice enterprise application development by component-based software development methodology using Java technology. Working cross platform with .NET Framework and Compare them.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลที่มีส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์วรวัฒน์ ลิ้มโภคา และ อาจารย์ชุติเมษฎ์ ศรีนิลทา อาจารย์ที่ปรึกษา ที่ให้ความเอาใจใส่ ช่วยแนะนำ สั่งสอน และช่วยเหลือในเรื่อง ๆ ต่าง ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก และขอขอบคุณพี่ ๆ เพื่อน ๆ ที่คอยช่วยเหลือ ให้คำปรึกษา และทำงานด้วยกันมา

สุดท้ายต้องขอขอบพระคุณ บิดา มารดา และญาติมิตร ที่อยู่เบื้องหลังความสำเร็จทั้งหมดนี้ จึงขอกราบขอบพระคุณมา ณ ที่นี้

นายเชษฐา ถาวรสถิตย์

นายเชี่ยวชาญ เลิศเอกบุรุษ

สารบัญ

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

การพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์ (Component-Based Software Development หรือ CBSD) เป็นกระบวนการในการสร้างซอฟต์แวร์ โดยนำคอมโพเน็นต์ที่มีอยู่มาใช้ (reuse) เพื่อลดเวลาและค่าใช้จ่ายในการพัฒนา

และการพัฒนาสถาปัตยกรรมคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์ (Server-side component architecture) ยังช่วยให้การพัฒนาแอพพลิเคชันระดับเอ็นเตอร์ไพรส์ (Enterprise application) ทำได้ง่าย นอกจากนั้นยังมีความน่าเชื่อถือ (Reliable) , มีระบบรักษาความปลอดภัย (Security) ที่ถูกต้องมากขึ้น ทั้งยังช่วยลดความซับซ้อนเนื่องจากโครงสร้างของระบบ ในปัจจุบันได้มีบริษัทต่างๆ ได้พัฒนาสถาปัตยกรรมที่ทำงานบนฝั่งเซิร์ฟเวอร์ (Server-side architecture) 2 สถาปัตยกรรมที่ได้รับการยอมรับมากที่สุดคือ

- Microsoft's Distributed interNet Applications Architecture (DNA) เป็นสถาปัตยกรรมของบริษัทไมโครซอฟท์ (Microsoft) ที่นำเสนอสถาปัตยกรรมคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์คือ COM+ (Component Object Model plus)
- Sun Microsystem's Java 2 Platform, Enterprise Edition (J2EE) เป็นสถาปัตยกรรมของบริษัทซัน ไมโครซิสเต็มส์ (Sun Microsystems) ที่นำเสนอสถาปัตยกรรมคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์คือ EJB (Enterprise JavaBeans)

สถาปัตยกรรมของทั้ง 2 บริษัทมีความแตกต่างที่น่าสนใจคือ J2EE นำเสนอความสามารถในการทำงานข้ามแพล็ตฟอร์ม (portability) และ DNA นำเสนอความสามารถในการทำงานข้ามภาษา (interoperability) เนื่องจากทางบริษัทซันไมโครซิสเต็มส์ได้ออกแบบภาษาจาวา (Java) ให้สามารถทำงานข้ามแพลตฟอร์ม การพัฒนาคอมโพเน็นต์ด้วยภาษาจาวาทำให้คอมโพเน็นต์มีความสามารถนี้อยู่ด้วย ในขณะที่ทางบริษัทไมโครซอฟท์ได้สนับสนุนการสร้างคอมโพเน็นต์จากภาษาใดก็ได้เช่น Visual Basic และ Visual C++ คอมโพเน็นต์ที่สร้างด้วย Visual Basic สามารถนำมาใช้กับ Visual C++ ได้ ความแตกต่างอีกอย่างหนึ่งคือ J2EE เป็น specification เท่านั้น แอพพลิเคชันเซิร์ฟเวอร์ (Application Server) จะถูกพัฒนาโดยผู้ผลิตรายอื่น เช่น BEA WebLogic Server , IBM WebSphere , Oracle 9i Application Server , SilverStream Application Server , iPlanet , Sybase Enterprise Application Server , Borland AppServer เป็นต้น

ทั้ง 2 สถาปัตยกรรมนี้มีข้อจำกัดที่ไม่สามารถนำกลับมาใช้ใหม่ (reuse) ข้ามสถาปัตยกรรมได้ แต่ภายหลังการเกิดของ XML , SOAP และแนวคิดของเว็บเซอร์วิส (Web Service) ทำให้สามารถทำงานร่วมกันได้ผ่านทาง SOAP ซึ่งเป็นโพรโตคอลที่ใช้สำหรับ Remote Procedure Call (RPC) ผ่าน HTTP โพรโตคอล

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 ศึกษาการออกแบบและพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์

1.2.2 ศึกษาสถาปัตยกรรมที่ทำงานบนฝั่งเซิร์ฟเวอร์ ข้อดีและข้อเสียของแต่ละสถาปัตยกรรม

1.2.3 ศึกษา XML และ SOAP

1.2.4 สร้างแอพพลิเคชันโดยใช้สถาปัตยกรรมคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์ และสามารถนำคอมโพเน็นต์มาใช้ใหม่ข้ามสถาปัตยกรรมได้

1.2.5 ศึกษาและพัฒนาเว็บเซอร์วิส

1.3 ขอบเขตของโครงงาน

โครงงานนี้พัฒนาระบบอี-คอมเมิร์ชที่เกี่ยวกับการรับจัดงานแต่งงานโดยใช้ EJB พัฒนาคอมโพเน็นต์, และเวบเซอร์วิสด้วยเวบโลจิกแพลตฟอร์ม7.0 แสดงผลด้วย JSP ที่เรียกใช้คอมโพเน็นต์ผ่านทางเว็บเซอร์วิส ใช้ SOAP (Simple Object Access Protocol) ในการทำหน้าที่เป็นสื่อกลางในเรียกใช้เวบเซอร์วิสของไมโครซอฟท์ (DOT NET FRAMWORK)

1.4 วิธีการดำเนินงาน

การศึกษาในโครงงานเริ่มจากศึกษาทฤษฎีพื้นฐานของการพัฒนาเชิงคอมโพเน็นต์และเวบเซอร์วิส , J2EE ซึ่งเป็นเทคโนโลยีสถาปัตยกรรมบนฝั่งเซิร์ฟเวอร์ที่ใช้ภาษาจาวาในการพัฒนา ประกอบด้วย EJB , JSP , JNDI และ JDBC เป็นต้น โดยใช้แนวคิดของการทำซอฟต์แวร์ให้มีลักษณะที่นำกลับมาใช้ใหม่ได้และขยายเพิ่มเติมได้ง่าย และยังมีส่วนที่ใช้ในการเรียกใช้เว็บเซอร์วิส ข้ามสถาปัตยกรรม ประกอบด้วย XML (eXtensible Markup Language) และ SOAP (Simple Object Access Protocol) โดยจะแบ่งการดำเนินงานเป็น 2 ส่วนคือ ส่วนแอพพลิเคชัน จะศึกษาและออกแบบระบบอี-คอมเมิร์ช ส่วนสถาปัตยกรรม (Architecture) เป็นส่วนของการออกแบบการทำSSL(Seurity Socket Layer)เพื่อเพิ่มความปลอดภัยในการใช้งาน

บทที่ 2
ทฤษฎีคอมโพเนนต์

คอมโพเนนต์คือชิ้นส่วนของซอฟต์แวร์ที่สามารถนำกลับมาใช้ใหม่ได้ (reuse) ในการที่จะนำคอมโพเนนต์ที่มีอยู่ไปใช้จะต้องทำความเข้าใจว่าคอมโพเนนต์นั้นใช้ทำอะไร คอมโพเนนต์จึงเหมือนกล่องดำ (black box) ที่เราไม่จำเป็นต้องรู้ว่ามันมีขั้นตอนการทำงานอย่างไร เรารู้แค่ว่ามันใช้ทำอะไรและจะเรียกใช้งานมันอย่างไร

2.1 ส่วนประกอบของคอมโพเนนต์

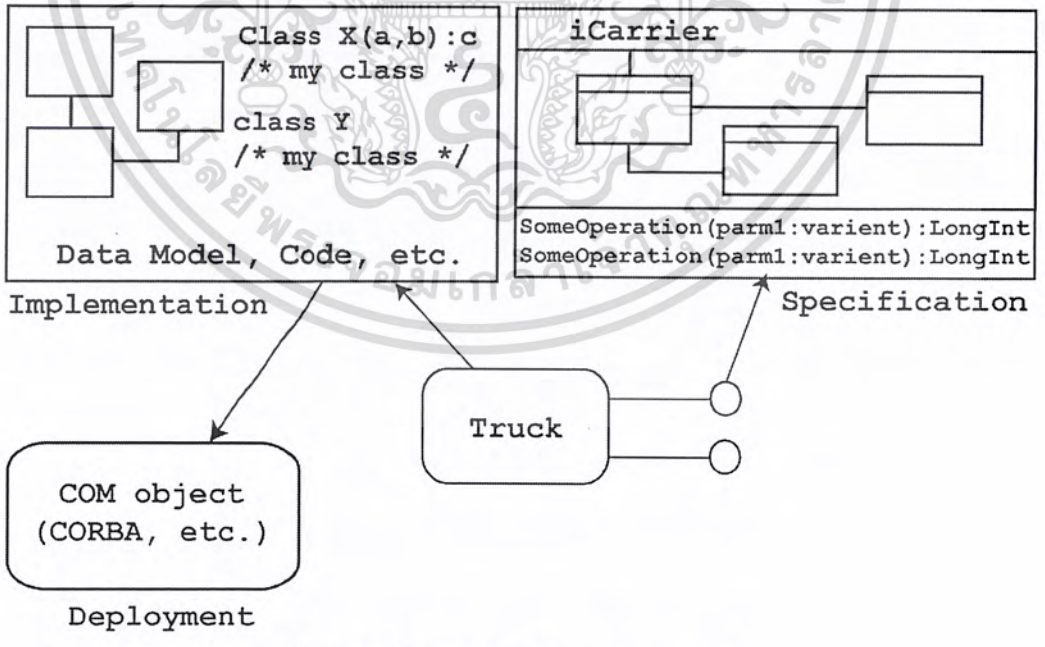
คอมโพเนนต์จะประกอบด้วยส่วนสำคัญ 3 ส่วนดังนี้

2.1.1 อินเทอร์เฟซ (interface) – คอมโพเน็นต์จะถูกเรียกใช้งานผ่านทางอินเทอร์เฟซ

2.1.2 อิมพลีเมนต์เตชัน (implementation) – เป็นโค้ดที่กำหนดการทำงานของคอมโพเนนต์

2.1.3 ดีพลอยเมนต์ (deployment) – เป็นแอ็กซิคิวต์ไฟล์จะใช้ในการทำให้คอมโพเนนต์ทำงานได้ ทำหน้าที่จัดหารันไทม์เอ็นไวรอนเมนต์ในการควบคุมการทำงานของคอมโพเนนต์และจัดหาเซอร์วิสที่จำเป็น

ส่วนประกอบต่าง ๆ ของคอมโพเนนต์ดังรูปที่ 2-1

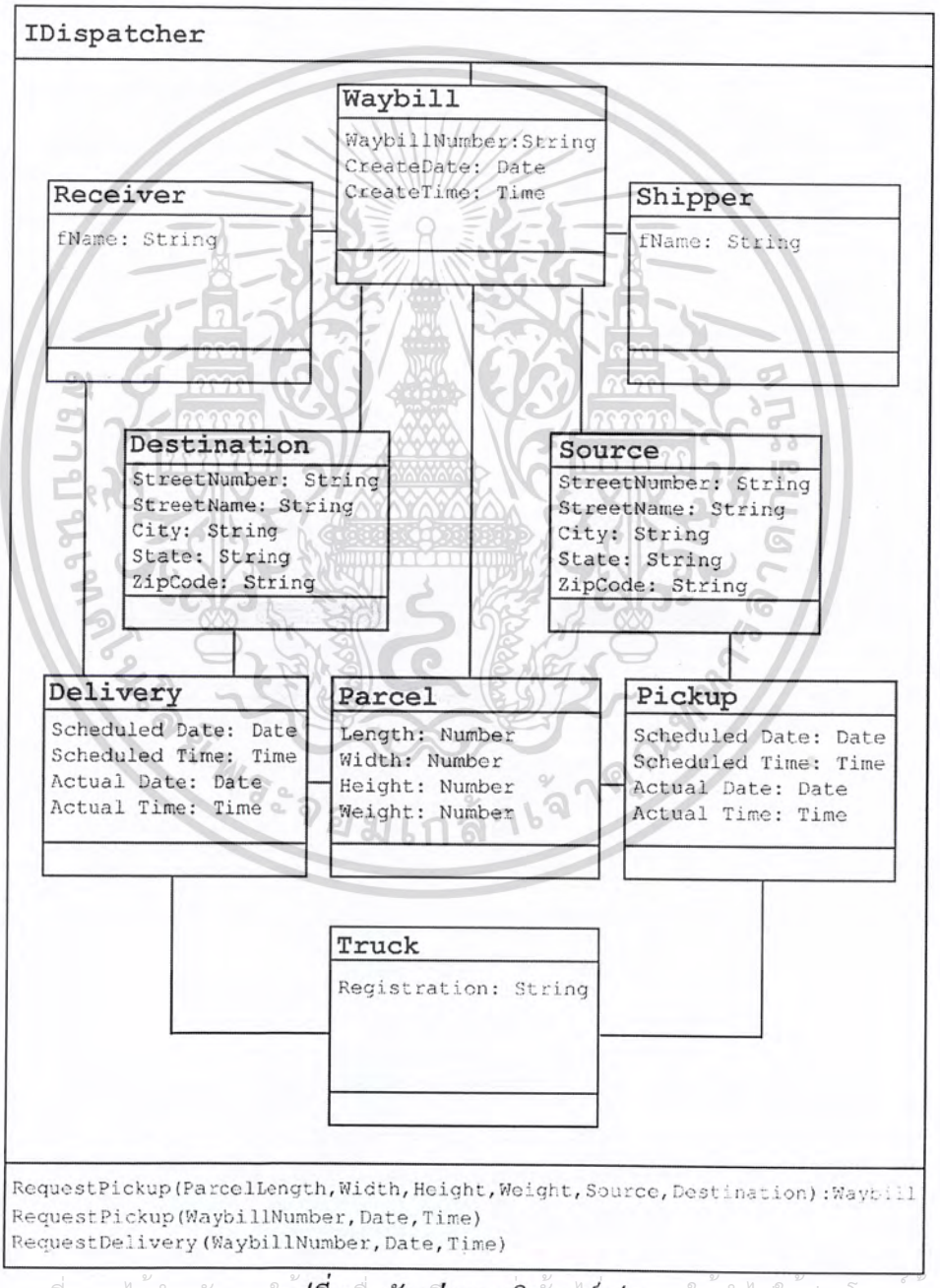


รูปที่ 2-1 คอมโพเนนต์และส่วนประกอบต่าง ๆ

2.2 คุณสมบัติของคอมโพเนนต์

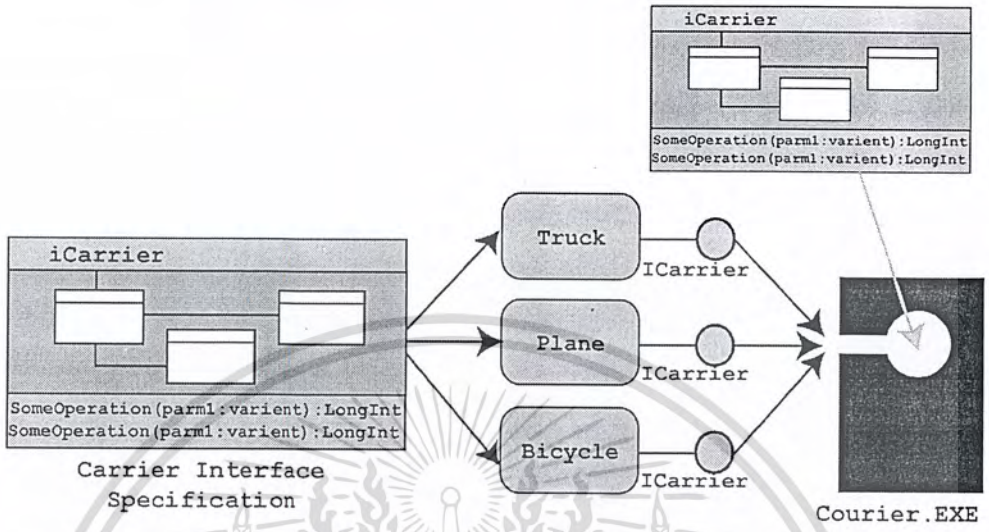
2.2.1 Encapsulate - เป็นกระบวนการในการซ่อนโค้ดหรืออิมพลีเมนต์เตชันของคอมโพเนนต์ ผู้ใช้จะรู้เพียง อินเทอร์เฟซและจะใช้งานผ่านมัน โดยไม่จำเป็นต้องรู้ถึงอิมพลีเมนต์เตชัน ทำให้การเปลี่ยนแปลงอิมพลีเมนต์เตชันไม่มีผลกระทบต่อผู้ใช้ ตัวอย่างอินเทอร์เฟซดังรูปที่ 2-2

2.2.2 Descriptive - เนื่องจากคอมโพเนนต์จะต้องติดต่อผ่านอินเทอร์เฟซเท่านั้น มันจะต้องมีอินฟอร์เมชันของมันเองที่ผู้ใช้สามารถจะเข้าใจได้ โดยอินฟอร์เมชันนั้นจะต้องอธิบายถึงอินเทอร์เฟซ , อิมพลีเมนต์เตชัน และดีพลอยเมนต์



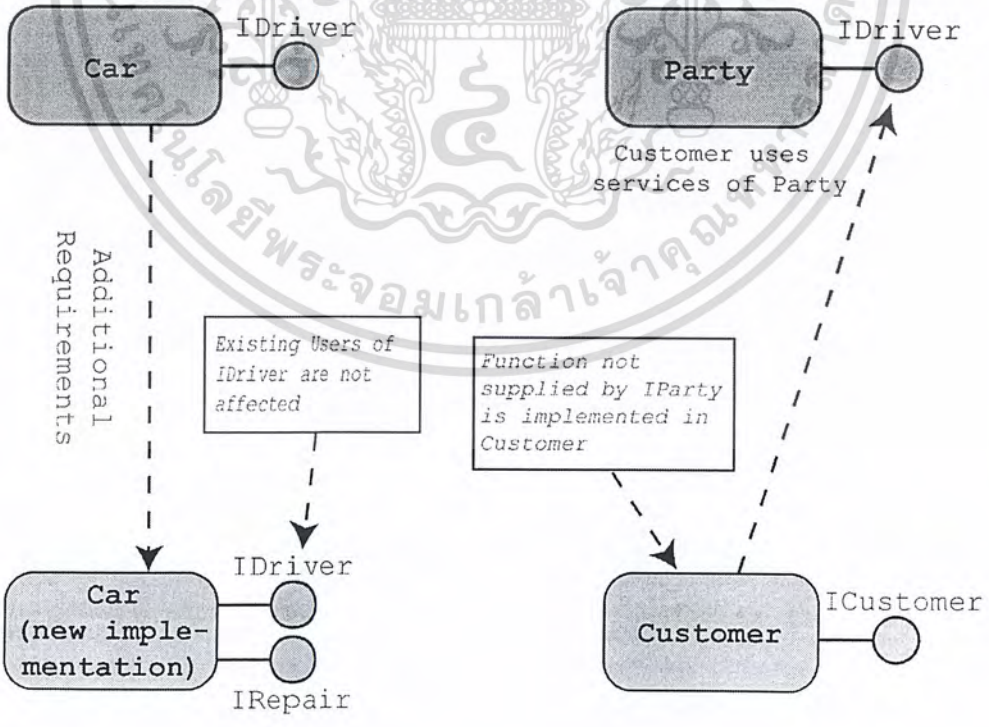
รูปที่ 2-2 ตัวอย่างของอินเทอร์เฟซ

2.2.3 Replaceable - คอมโพเนนต์มีความสามารถในการเปลี่ยนคอมโพเนนต์หนึ่งกับคอมโพเนนต์ใด ๆ ที่มีอินเทอร์เฟซเหมือนกัน ดังรูปที่ 2-3



รูปที่ 2-3 คอมโพเนนต์และการแทนที่

2.2.4 Extensible - สามารถที่จะเพิ่มความสามารถได้ มี 2 วิธีคือ การเพิ่มอินเทอร์เฟซและมอบหมายหน้าที่ (Delegating Responsibility) การสร้างคอมโพเนนต์ใหม่สามารถมอบหมายหน้าที่ให้กับบริการที่มีอยู่ในคอมโพเนนต์ที่มีอยู่แล้ว ดังรูปที่ 2-4

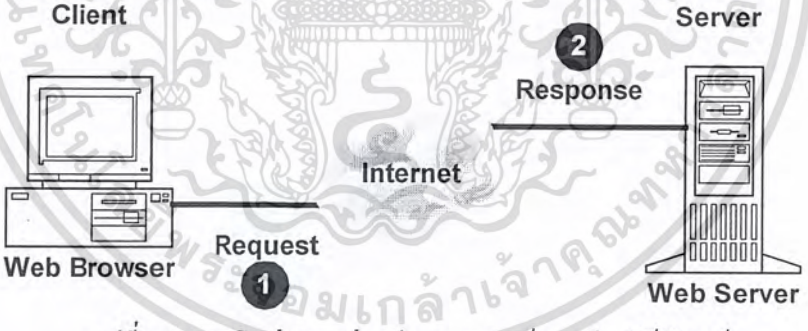


รูปที่ 2-4 การเพิ่มความสามารถคอมโพเนนต์

บทที่ 3
สถาปัตยกรรมบนฝั่งเซิร์ฟเวอร์

เอนเตอร์ไพรส์จาวาบีน (EJB) เป็นสถาปัตยกรรมคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์ (server-side component architecture) ซึ่งช่วยให้กระบวนการพัฒนาแอพพลิเคชันออบเจ็กต์แบบกระจาย (distributed object application) ในระดับเอ็นเตอร์ไพรส์ด้วยภาษาจาวานั้นทำได้ง่ายขึ้น ด้วย EJB เราสามารถสร้างแอพพลิเคชันที่สามารถเพิ่มขยายได้ (scalable) , มีความน่าเชื่อถือ (reliable) และมีความปลอดภัย (secure) โดยไม่ต้องเขียนโค้ดเกี่ยวกับการใช้งานออบเจ็กต์แบบกระจายซึ่งมีความซับซ้อนสูงด้วยตนเอง เราสามารถสร้างคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์ด้วยภาษาจาวาได้อย่างง่ายดายและรวดเร็ว นอกจากนี้ EJB ยังถูกออกแบบขึ้นมาเพื่อสนับสนุนความสามารถในการเคลื่อนย้ายได้ (portability) และความสามารถนำกลับมาใช้ใหม่ได้ (reusability) ของ แอพพลิเคชันในแต่ละมิดเดิลแวร์ของแต่ละผู้ผลิต

แอพพลิเคชันบนฝั่งเซิร์ฟเวอร์ในยุคแรก ๆ นั้น ส่วนประกอบของเว็บยังไม่มีความซับซ้อนเช่นในปัจจุบัน โดยใช้หลักการพื้นฐานของสถาปัตยกรรมแบบไคลเอ็นต์/เซิร์ฟเวอร์ ซึ่งเซิร์ฟเวอร์ในที่นี้ก็คือเว็บเซิร์ฟเวอร์ และไคลเอ็นต์ก็คือ เว็บบราวเซอร์ เช่น Netscape, Mosaic, Lynx , หรือ NCSA โดยเว็บเซิร์ฟเวอร์หนึ่ง ๆ จะให้บริการกับเว็บบราวเซอร์หลาย ๆ ตัวได้พร้อมกัน ดังรูปที่ 3-1



รูปที่ 3-1 การติดต่อระหว่างเว็บบราวเซอร์และเว็บเซิร์ฟเวอร์

แต่ข้อเสียของเว็บไซต์ในตอนนั้นก็คือ หน้าเว็บนั้นเป็น HTML ที่ตายตัว (static) ข้อมูลที่ปรากฏอยู่ไม่สามารถเปลี่ยนแปลงได้เลย เว็บเซิร์ฟเวอร์จะทำหน้าที่แค่เพียงส่งโค้ด HTML กลับมาเท่านั้น เทคโนโลยีที่เกิดขึ้นตามมาก็คือ CGI (Common Gateway Interface) ซึ่งใช้ในการสร้างหน้าเพจแบบไดนามิก เนื่องจากมันมีการประมวลผลก่อนที่จะส่งกลับไป

หลังจากที่ CGI ได้รับความนิยมจนกลายมาเป็นส่วนหนึ่งในมาตรฐานของเว็บแล้ว ก็ได้มีเทคโนโลยีที่ใกล้เคียงกันตามออกมามากมาย เช่น ASP , PHP , JSP , Servlet และอื่น ๆ อย่างไรก็ตามในโครงงานนี้จะกล่าวถึงเฉพาะเทคโนโลยีแอพพลิเคชันทางฝั่งเซิร์ฟเวอร์ของจาวาเท่านั้น

ดีพลอยเมนต์บนฝั่งเซิร์ฟเวอร์ (server-side deployment) คือซอฟต์แวร์ที่ถูกเขียนขึ้นมาให้สามารถรองรับการใช้งานจากผู้ใช้ได้หลาย ๆ คนพร้อมกัน , มีความปลอดภัย , เชื่อถือได้ และมีประสิทธิภาพ ตัวอย่างของดีพลอยเมนต์บนฝั่งเซิร์ฟเวอร์ได้แก่

- ธนาคาร - เครื่องเอทีเอ็มจำนวนมากจะต้องติดต่อเข้าไปยังเครื่องเซิร์ฟเวอร์กลางของธนาคาร
- สาขาร้านค้า - เช่นร้านค้าในเครือข่ายของ Wal-Mart โดยร้านค้าแต่ละร้านจะส่งรายละเอียดการซื้อ สินค้าของลูกค้าแต่ละรายไปยังเซิร์ฟเวอร์กลางของ Wal-Mart
- ฝ่ายสนับสนุนลูกค้า - วิศวกรหลาย ๆ คนสามารถใช้เครื่องเทอร์มินอลเชื่อมต่อเข้ามาเซิร์ฟเวอร์กลางและนำข้อมูลเกี่ยวกับลูกค้าที่ถูกเก็บที่เซิร์ฟเวอร์ไปใช้งานได้
- เว็บไซต์ – ในเวลาใดเวลาหนึ่งจะมีผู้ใช้จำนวนมากทำการติดต่อเข้ามายังเว็บเซิร์ฟเวอร์ และเว็บเซิร์ฟเวอร์เหล่านี้จะต้องติดต่อไปยังเซิร์ฟเวอร์กลาง เพื่อดึงเอาข้อมูลและเรียกใช้การประมวลผลที่จำเป็น

การสร้างดีพลอยเมนต์บนฝั่งเซิร์ฟเวอร์ที่มีความสมบูรณ์นั้นไม่ได้เป็นเรื่องง่าย ปัญหาที่อาจเกิดขึ้นเช่น ปัญหาด้านความสามารถในการเพิ่มขยาย , การจัดการ , ความปลอดภัย , ความน่าเชื่อถือ และอื่น ๆ เนื่องจากว่า การทำงานของไคลเอ็นต์จำนวนมากนั้นจะขึ้นอยู่กับเซิร์ฟเวอร์เป็นสำคัญ การที่เซิร์ฟเวอร์ดาวน์ , ทำงานช้าลง หรือมีผู้ประสงค์ร้ายลักลอบเข้าไปนั้น สามารถก่อให้เกิดความเสียหายได้อย่างมหาศาล ดังนั้นการสร้างดีพลอยเมนต์บนฝั่งเซิร์ฟเวอร์จะต้องค่อย ๆ ถูกสร้างขึ้นมาทีละขั้น และจะต้องมีการทดสอบอย่างรัดกุม

ในการทำดีพลอยเมนต์ที่ดีนั้น จะแบ่งซอฟต์แวร์ออกเป็นเลเยอร์ (layer) โดยในแต่ละเลเยอร์จะมีความรับผิดชอบในส่วนของดีพลอยเมนต์ที่แตกต่างกันออกไป และอาจประกอบด้วยหนึ่งคอมโพเน็นต์หรือมากกว่าก็ได้ เลเยอร์เหล่านี้เป็นเพียงการแบ่งเลเยอร์ในทางความคิด (logical) เท่านั้น ซึ่งอาจแตกต่างจากการแบ่งหน้าที่ความรับผิดชอบของเครื่องคอมพิวเตอร์แต่ละเครื่องในทางกายภาพ (physical) ได้

ระบบที่แบ่งการทำงานเป็นเลเยอร์เป็นระบบที่ถูกออกแบบมาอย่างดี เนื่องจากแต่ละเลเยอร์นั้นจะมีหน้าที่รับผิดชอบที่แตกต่างกันออกไป การแบ่งเลเยอร์ทั่วไปจะแบ่งเลเยอร์เป็นดังนี้

พรีเซนต์เทชัน เลเยอร์ – ประกอบด้วยคอมโพเน็นต์ที่จัดการกับ การติดต่อกับผู้ใช้ (user interface) และการโต้ตอบกับผู้ใช้ (user interaction) พรีเซนเทชันเลเยอร์ของแอพพลิเคชันแบบ stand-alone อาจเขียนด้วยภาษาวิชวลเบสิค ส่วนพรีเซนเทชันเลเยอร์ของดีพลอยเมนต์แบบ Web-base นั้น จะใช้ Java Servlet , JSP , หรือ Java Applet

บิซิเนส ลอจิก เลเยอร์ - ประกอบด้วยคอมโพเน็นต์ที่ใช้ในการแก้ปัญหาทางธุรกิจ คอมโพเน็นต์เหล่านี้อาจเป็น engine ที่มีความสามารถสูง เช่น engine ในการจัดการแคตาล็อก หรือ คำนวณราคาสินค้า โดยปกติแล้วคอมโพเน็นต์เหล่านี้จะต้องถูกเขียนขึ้นโดยภาษาที่มีคุณสมบัติ type-safe เช่น ภาษาจาวา หรือภาษา C++

ดาต้า เลเยอร์ - ถูกเรียกใช้โดยบิซิเนสลอจิกเลเยอร์เพื่อเก็บข้อมูลแบบถาวร โดยทั่วไปดาต้าเลเยอร์จะเป็นระบบฐานข้อมูล

ข้อดีของการแบ่งคอมโพเน็นต์เป็นเลเยอร์คือ ทำให้สามารถเปลี่ยนแปลงแต่ละเลเยอร์ได้โดยจะไม่มีผลกระทบต่อเลเยอร์อื่น ๆ เช่น เราสามารถเปลี่ยนแปลงหน้าจออินเทอร์เฟซในพรีเซนเทชันเลเยอร์ได้ โดยไม่มีผลกระทบต่อเลเยอร์อื่น

แต่การแบ่งลักษณะการทำงานออกเป็นเลเยอร์นี้ เป็นการแบ่งในทางความคิดเท่านั้น ส่วนในทางกายภาพเราจะแบ่งออกเป็นหน่วยที่เรียกว่า "เทียร์" (tier) โดยสถาปัตยกรรมแบบหนึ่งเทียร์ คือการรวมเลเยอร์ทั้งสามเป็นส่วนเดียวทางกายภาพ , สถาปัตยกรรมแบบสองเทียร์ คือการแบ่งเลเยอร์ทั้งสามออกเป็นสองส่วนในทางกายภาพ และสถาปัตยกรรมแบบสามเทียร์ คือการแบ่งเลเยอร์ทั้งสามออกเป็นสามส่วนในทางกายภาพ

3.1 สถาปัตยกรรมแบบหนึ่งเทียร์ (1-Tier Architecture)

ในสถาปัตยกรรมนี้ ทั้งสามเลเยอร์จะรวมกันเป็นคอมโพเน็นต์เดียวซึ่งโดยปกติจะเป็นโปรแกรมทำงานในเครื่อง เช่น เวิร์ดโปรเซสเซอร์ หรือสเปรดชีท ซึ่งหากมีการแก้ไขเปลี่ยนแปลงที่เลเยอร์ใดๆ ก็ต้องแก้ไขโค้ดทั้งหมดทุกครั้ง

3.2 สถาปัตยกรรมแบบสองเทียร์ (2-Tier Architecture)

ในสมัยก่อน ดีพลอยเมนต์ในระดับไฮเอนด์ใช้สถาปัตยกรรมแบบ 2 เทียร์ สถาปัตยกรรมนี้จะรวมบิซิเนสลอจิกเลเยอร์เข้ากับอีกหนึ่งในสองเลเยอร์ที่เหลือ ซึ่งทางเลือกที่เป็นไปได้คือรวมบิซิเนสลอจิกเลเยอร์เข้ากับพรีเซนเทชันเลเยอร์ และรวมบิซิเนสลอจิกเลเยอร์เข้ากับดาต้าเลเยอร์

- การรวมบิซิเนสลอจิกเลเยอร์เข้ากับพรีเซนเทชันเลเยอร์

พรีเซนเทชันเลเยอร์ และบิซิเนสลอจิกเลเยอร์นั้น สามารถนำมารวมกันเป็นเทียร์เดียวได้ โดยแยกดาต้าแอ็กเซสเลเยอร์เป็นอีกหนึ่งเทียร์ต่างหาก ซึ่งเสมือนกับเป็นการสร้างกำแพงกั้นระหว่างบิซิเนสลอจิกเลเยอร์และดาต้าแอกเซสเลเยอร์ ดังรูปที่ 3-2

ถ้าหากเรามองว่าเทียร์แรกเป็นเทียร์ของไคลเอ็นต์ และเทียร์ที่สองเป็นเซิร์ฟเวอร์ สถาปัตยกรรมนี้จะทำให้ทางฝั่งไคลเอ็นต์มีขนาดใหญ่ (fat) ส่วนทางฝั่งเซิร์ฟเวอร์จะมีขนาดเล็ก (thin)

ในสถาปัตยกรรมแบบ 2 เทียร์ ไคลเอ็นต์แอพพลิเคชันจะติดต่อกับดาต้าเลเยอร์โดยผ่านทาง database bridge API เช่น Open Database Connectivity (ODBC) หรือ Java Database Connectivity (JDBC)

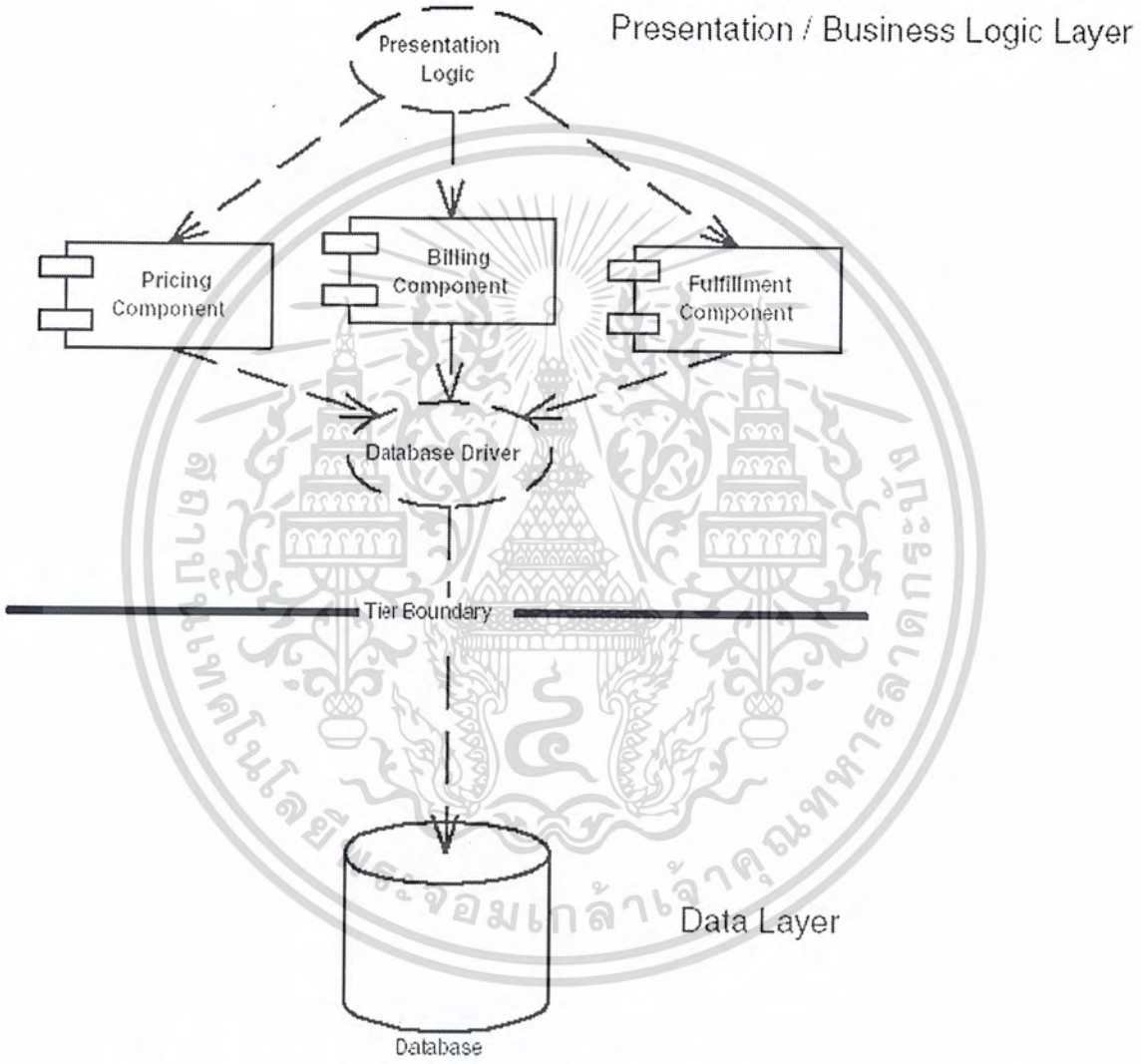
ลักษณะของสถาปัตยกรรมแบบสองเทียร์

1) มีค่าใช้จ่ายในการดีพลอยสูง

ไดรเวอร์ของระบบฐานข้อมูลจะต้องถูกติดตั้งลงในไคลเอ็นต์ทุกเครื่องที่อยู่ในไคลเอ็นต์เทียร์ ซึ่งอาจเป็นเครื่องคอมพิวเตอร์จำนวนหลายร้อย หรือหลายพันเครื่อง

2) มีค่าใช้จ่ายในการเปลี่ยนไดรเวอร์ของระบบฐานข้อมูลสูง

การเปลี่ยนแปลงไดรเวอร์ของระบบฐานข้อมูล จะต้องการการเปลี่ยนแปลงที่เครื่องไคลเอ็นต์ทุกเครื่อง ซึ่งเป็นขั้นตอนที่สิ้นเปลืองมาก และเกือบจะเป็นไปไม่ได้ในความเป็นจริงสำหรับระบบงานที่มีขนาดใหญ่



รูปที่ 3-2 การรวมส่วนพรีเซนเทชันเข้ากับบิซิเนสลอจิกในสถาปัตยกรรมแบบ 2 เทียร์

3) การเปลี่ยนแปลงโครงสร้างของฐานข้อมูลมีค่าใช้จ่ายที่สูง

เนื่องจากไคลเอ็นต์มีขนาดใหญ่ การเข้าถึงฐานข้อมูลเป็นการเข้าถึงโดยตรงโดยผ่านทาง JDBC , SQL/J หรือ ODBC ซึ่งหมายความว่า การเปลี่ยนแปลงโครงสร้างของฐานข้อมูลให้รองรับบิซิเนสลอจิกใหม่ ๆ จะต้องเปลี่ยนแปลงที่ไคลเอ็นต์ทุกเครื่องด้วย

4) การเปลี่ยนแปลงรูปแบบของฐานข้อมูลมีค่าใช้จ่ายที่สูง

ไคลเอ็นต์ที่มีขนาดใหญ่จะยึดติดอยู่กับ API ของระบบฐานข้อมูล เช่น API ของระบบฐานข้อมูล เชิงสัมพันธ์ (relational database) หรือ API ของระบบฐานข้อมูลแบบออบเจ็กต์ (object database) ถ้าหากเราตัดสินใจที่จะเปลี่ยนแปลงรูปแบบของฐานข้อมูล เช่นเปลี่ยนจากการใช้ฐานข้อมูลเชิงสัมพันธ์มาเป็นการใช้ฐานข้อมูลแบบออบเจ็กต์ นอกจากเราจะต้องทำการดีพลอยไคลเอ็นต์ทั้งหมดใหม่แล้ว ยังต้องมีการเปลี่ยนแปลงโค้ดภายในตัวไคลเอ็นต์ให้เข้ากับรูปแบบฐานข้อมูลที่เปลี่ยนแปลงไปอีกด้วย

5) การเปลี่ยนแปลงบิซิเนสลอจิกเลเยอร์มีค่าใช้จ่ายที่สูง

หากมีการเปลี่ยนแปลงที่เกิดขึ้นกับบิซิเนสลอจิกเลเยอร์ ไคลเอ็นต์จะต้องคอมไพล์ใหม่ และทำการดีพลอยใหม่ทั้งหมด

6) การเชื่อมต่อกับฐานข้อมูลมีค่าใช้จ่ายที่สูง

ไคลเอ็นต์แต่ละเครื่องจะติดต่อกับฐานข้อมูลโดยใช้ connection ของมันเอง ในขณะที่ connection มีจำนวนได้จำกัดและมีค่าใช้จ่ายในการสร้างสูง เมื่อไคลเอ็นต์ไม่ได้ใช้ฐานข้อมูล connection จะยังอยู่และ ไคลเอ็นต์อื่นไม่สามารถนำมันไปใช้ได้

7) ประสิทธิภาพของระบบเครือข่ายลดลง

ในแต่ละครั้งที่ไคลเอ็นต์ทำการติดต่อกับฐานข้อมูล จะมีการส่งข้อมูลจำนวนมากไปมา ระหว่างบิซิเนสลอจิกเลเยอร์ และดาต้าเลเยอร์ ซึ่งหากว่าตัวกลางระหว่างเลเยอร์ทั้งสองนี้เป็นระบบเครือข่าย อาจทำให้ข้อมูลในระบบเครือข่ายนั้นมากจนเกิดการติดขัด ซึ่งจะมีผลให้การติดต่อกับฐานข้อมูลนั้นทำได้ช้าลง

- การรวมส่วนบิซิเนสลอจิกเลเยอร์บางส่วน เข้ากับดาต้าเลเยอร์

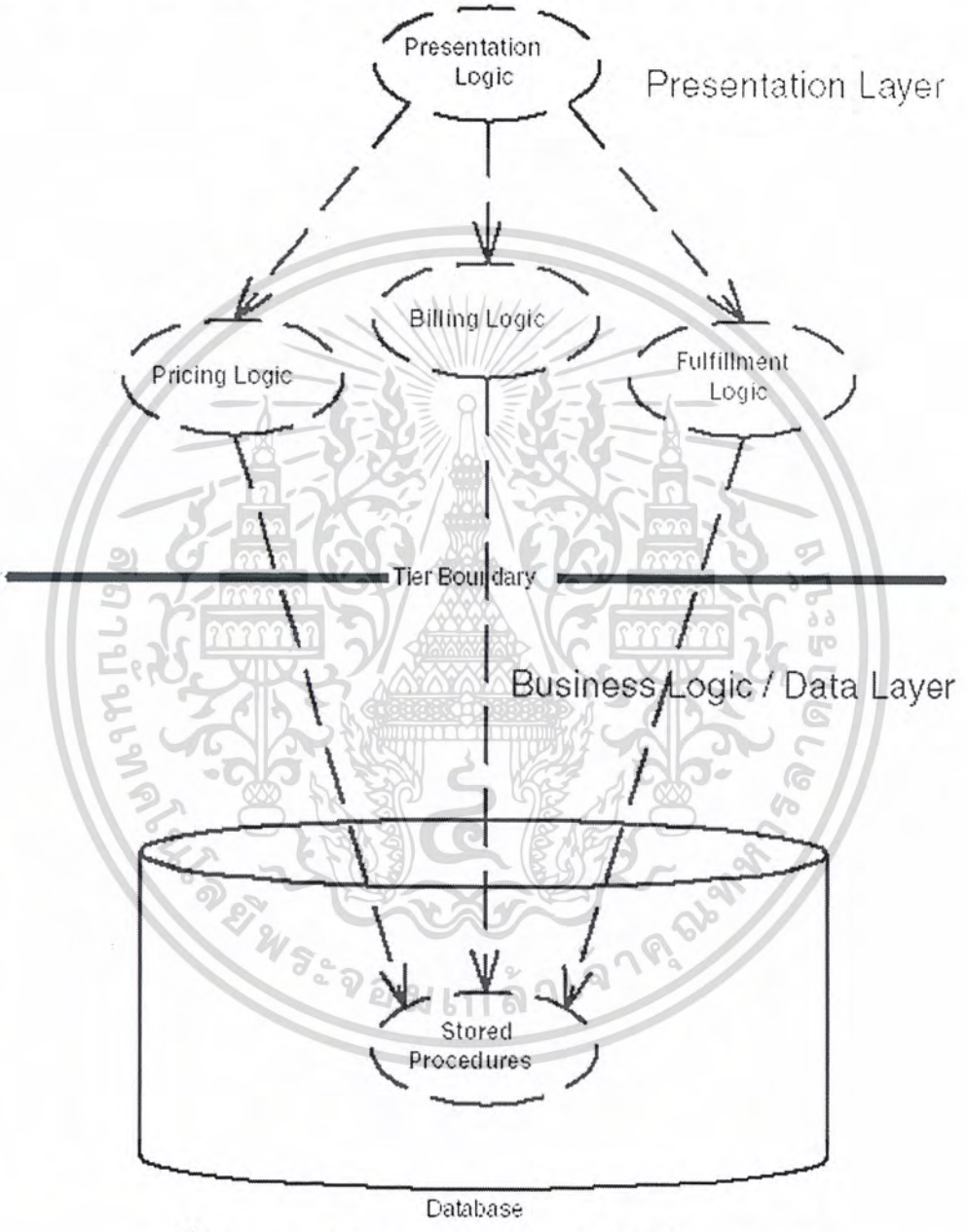
ในระยะหลัง ๆ มานี้ ดีพลอยเมนต์ต่าง ๆ จะนำบิซิเนสลอจิกเลเยอร์มารวมกับดาต้าเลเยอร์ และแยกไว้เป็นเทียร์หนึ่ง ซึ่งแสดงให้เห็นในรูปที่ 3-3 โดยหากมองว่าเทียร์ที่หนึ่งเป็นไคลเอ็นต์ ส่วนเทียร์ที่สองเป็นเซิร์ฟเวอร์ สถาปัตยกรรมแบบนี้จะให้ทำไคลเอ็นต์นั้นมีขนาดเล็ก ส่วนเซิร์ฟเวอร์นั้นจะมีขนาดใหญ่

ในสถาปัตยกรรมดังกล่าว จะนำบิซิเนสลอจิกบางส่วนเข้าไปไว้ในฐานข้อมูล ซึ่งโดยทั่วไปจะเป็นลอจิกที่เกี่ยวข้องกับข้อมูล ฐานข้อมูลจะอนุญาตให้เอ็กซิคิวต์ (execute) ลอจิกภายในฐานข้อมูลได้โดยการเขียนโมดูลที่เรียกว่า store procedure

การรวมลอจิกเข้าไปเป็น store procedure จะเป็นการเพิ่ม scalability และประสิทธิภาพในการทำงาน เนื่องจากว่าลอจิกบางส่วนถูกรวมเข้ากับฐานข้อมูล ดังนั้นการใช้งานเครือข่ายเพื่อติดต่อระหว่างลอจิกกับฐานข้อมูลจะลดลง แทนที่จะทำการ query ข้อมูลในฐานข้อมูลจำนวน n ครั้ง เราสามารถที่จะเรียก store procedure ซึ่งอยู่ภายในฐานข้อมูลให้ทำการ query n ครั้งแทน ซึ่งจะลดปริมาณการใช้เครือข่ายไปได้มากทีเดียว และยังทำให้การประมวลผลฐานข้อมูลเร็วขึ้นอีกด้วย

จะเห็นว่าการรวมเอาลอจิกที่เกี่ยวข้องกับฐานข้อมูลเข้าไปไว้ในฐานข้อมูล ช่วยเพิ่มประสิทธิภาพโดยรวมของดีพลอยเมนต์ขึ้นอย่างมาก แต่บางปัญหาซึ่งเกิดกับสถาปัตยกรรมแบบ 2 เทียร์ก็ยังคงมีอยู่

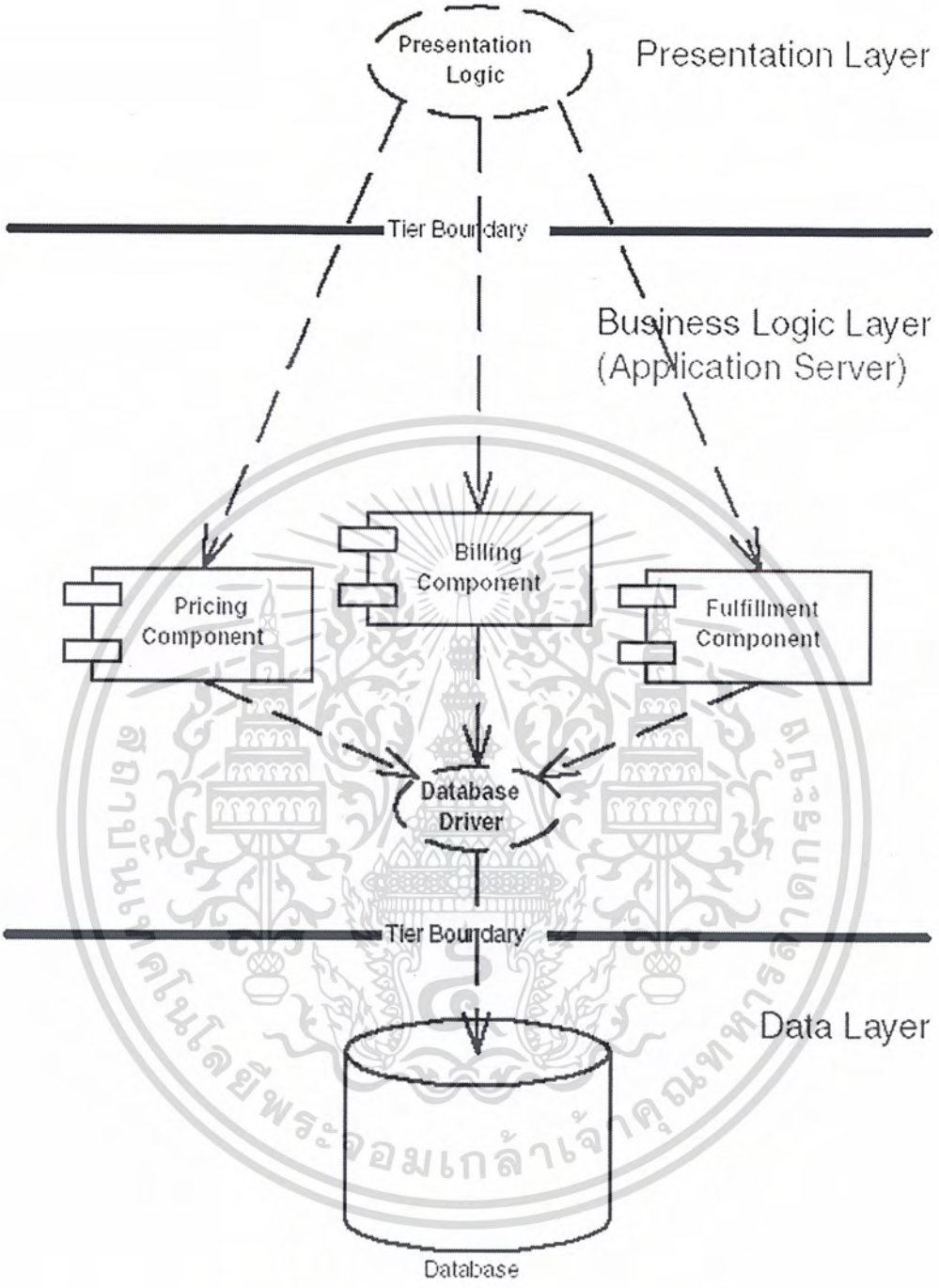
ในขณะที่การสร้าง store procedure นั้นเป็นวิธีการที่สามารถแก้ปัญหาได้หลายอย่าง แต่ตัวมันเองก็ทำให้เกิดปัญหาใหม่ ๆ ขึ้นมาเช่นกัน โดยภาษาที่ใช้ในการเขียน store procedure นั้น จะผูกอยู่กับฐานข้อมูลที่ใช้ ทำให้การเปลี่ยนแปลงระบบฐานข้อมูลที่ใช้อยู่นั้นทำได้ยาก แต่อย่างไรก็ตาม ภาษาจาวาได้ถูกนำมาใช้เขียน store procedure มากขึ้นเรื่อย ๆ ซึ่งจะช่วยเพิ่มความสามารถในการโยกย้าย (portability)



รูปที่ 3-3 การรวมบิซิเนสลอจิกเลเยอร์บางส่วน เข้ากับดาต้าเลเยอร์

3.3 สถาปัตยกรรมแบบสามเทียร์ (3-Tier Architecture)

สถาปัตยกรรมแบบการแยกเลเยอร์ทั้งสามออกจากกันไว้ในแต่ละเทียร์ ตัวอย่างที่ดีของสถาปัตยกรรมแบบสามเทียร์ได้แก่ web-base ดีพลอยเมนต์แบบ 3 เทียร์ ดังแสดงในรูปที่ 3-4 ซึ่งเทียร์ทั้งสามนั้น ถูกแบ่งออกดังนี้



รูปที่ 3-4 Web-base ดีพลอยเมนต์แบบ 3 เทียร์

- พรีเซนเทชันเทียร์ (Presentation Tier)

อาจประกอบไปด้วย Java Servlet , สคริปต์พวก Active Server Page หรือ Java Server Page และลอจิกที่เชื่อมการทำงานของแต่ละส่วนเข้าด้วยกัน

- บิซิเนสลอจิกเทียร์ (Business Logic Tier)

แอพพลิเคชันเซิร์ฟเวอร์เป็นสิ่งจำเป็นในการให้สภาพแวดล้อมในการทำงานที่เหมาะสมกับคอมโพเน็นต์ของบิซิเนสลอจิก แอพพลิเคชันเซิร์ฟเวอร์จะจัดหาเอ็นไวรอนเมนต์ให้กับบิซิเนสลอจิกคอมโพเน็นต์ ทั้งยังทำหน้าที่จัดการคอมโพเน็นต์ และให้บริการที่จำเป็นแก่คอมโพเน็นต์เหล่านี้ ยกตัวอย่างเช่น แอพพลิเคชันเซิร์ฟเวอร์สามารถให้บริการติดต่อฐานข้อมูลกับ บิซิเนสคอมโพเน็นต์ ซึ่งทำให้บิซิเนสคอมโพเน็นต์สามารถเก็บข้อมูลแบบถาวร (persistent) ลงไป และดึงข้อมูลขึ้นมาจากดาต้าเทียร์ได้ แอพพลิเคชันเซิร์ฟเวอร์ยังมีหน้าที่ในการจัดการคอมโพเน็นต์โดยการสร้าง อินสแทนซ์ของคอมโพเนนต์เมื่อจำเป็น

- ดาต้าเทียร์ (Data Tier)

  อาจมีลอจิกเกี่ยวกับข้อมูลซึ่งถูกเก็บไว้ในรูปของ store procedure

ลักษณะของสถาปัตยกรรมแบบ N เทียร์

1) ค่าใช้จ่ายในการดีพลอยต่ำ

เนื่องจากไดรเวอร์ของระบบฐานข้อมูลนั้นถูกติดตั้งอยู่ที่ฝั่งเซิร์ฟเวอร์ แทนที่จะเป็นเครื่องของไคลเอ็นต์ ซึ่งเป็นการสะดวกที่จะทำการติดตั้งซอฟต์แวร์บนฝั่งเซิร์ฟเวอร์ในสภาพแวดล้อมที่ควบคุมได้มากกว่าการติดตั้งบนเครื่องไคลเอ็นต์จำนวนมากมาย

2) ค่าใช้จ่ายในการเปลี่ยนแปลงระบบฐานข้อมูลต่ำ

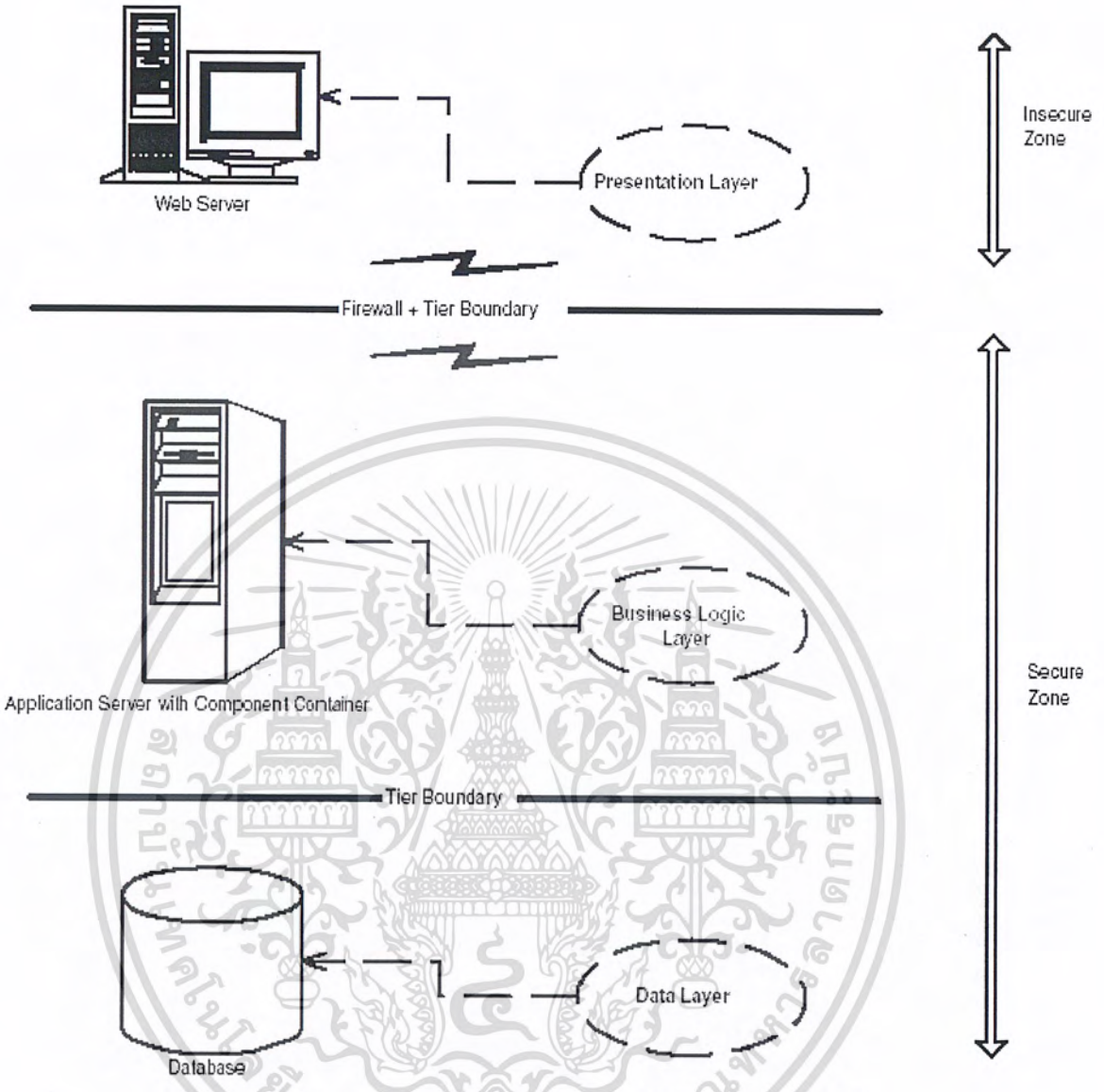
ไคลเอ็นต์จะทำการติดต่อกับฐานข้อมูลโดยผ่านทางเทียร์ซึ่งอยู่ตรงกลาง ทำให้เราสามารถเปลี่ยนแปลงโครงสร้างของฐานข้อมูล , ไดรเวอร์ของฐานข้อมูล แม้กระทั่งฐานข้อมูลโดยไม่จำเป็นต้องดีพลอยไคลเอ็นต์ทั้งหมดใหม่อีกครั้ง

3) ค่าใช้จ่ายในการเปลี่ยนแปลงบิซิเนสลอจิกต่ำ

เพราะการเปลี่ยนแปลงในบิซิเนสลอจิกเลเยอร์ไม่จำเป็นที่จะต้องมีการดีพลอยไคลเอ็นต์เทียร์ใหม่

4) สามารถสร้างเขตความปลอดภัยให้กับส่วนใดส่วนหนึ่งของดีพลอยเมนต์ได้ด้วยไฟล์วอลล์

ใน web-base ดีพลอยเมนต์ เราอาจไม่ต้องการให้บุคคลภายนอกเข้ามาเห็นข้อมูลในบิซิเนสลอจิก เลเยอร์ แต่อนุญาตให้บุคคลภายนอกเข้าถึงพรีเซนเทชันเลเยอร์ได้ ซึ่งสามารถแก้ปัญหานี้ได้โดยการวางไฟร์วอลล์ไว้ระหว่างส่วนพรีเซนเทชัน และบิซิเนสลอจิกเทียร์ ดังแสดงในรูปที่ 3-5

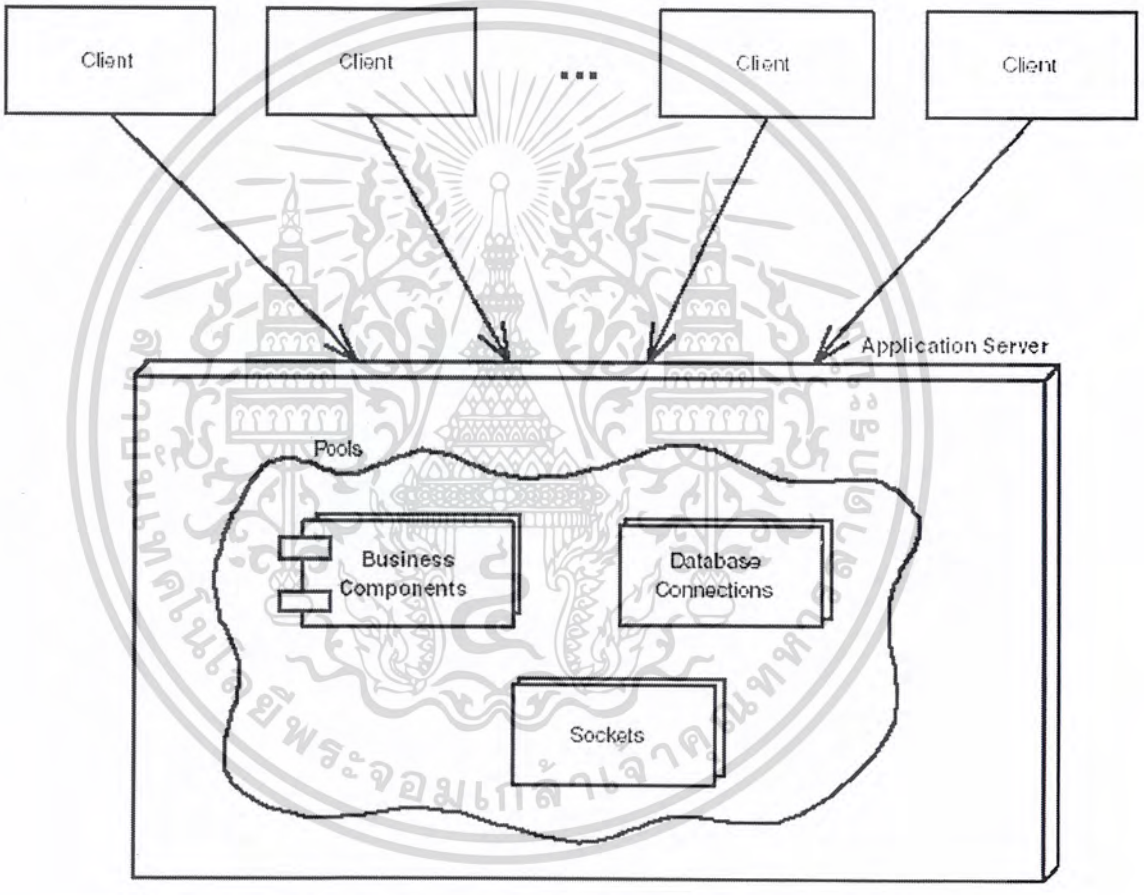


รูปที่ 3-5 การรักษาความปลอดภัยในสถาปัตยกรรมแบบ 3 เทียร์

5) ทรัพยากรสามารถถูกพูล (pool) และนำกลับมาใช้ใหม่ได้อย่างมีประสิทธิภาพ

ในสถาปัตยกรรมแบบสามเทียร์ การเชื่อมต่อไปยังทรัพยากรภายนอกสามารถถูกจัดการได้อย่างมีประสิทธิภาพ การพูลทรัพยากร (resource pooling) นั้นใช้หลักความเป็นจริงที่ว่า ไคลเอ็นต์มักจะทำงานอื่นไปพร้อม ๆ กับการใช้ทรัพยากร เช่น การสร้างส่วนติดต่อกับผู้ใช้ แทนที่บิซิเนสคอมโพเน็นต์จะทำการเชื่อมต่อ และปล่อยการติดต่อกับทรัพยากร (เช่น ฐานข้อมูล) ทรัพยากรนั้นสามารถที่จะพูล และนำไปให้บริการแก่ไคลเอ็นต์หลาย ๆ เครื่องได้ ทำให้จำนวนของการเชื่อมต่อฐานข้อมูล (database connection) นั้น จะมีจำนวนน้อยกว่าจำนวนของคอมโพเน็นต์ทั้งหมดมาก

เนื่องจากการติดต่อกับฐานข้อมูลนั้นเป็นวิธีการที่แพง (expensive) วิธีการนี้จะช่วยเพิ่ม scalability ของดีพลอยเมนต์ นอกจากนี้แล้วการเชื่อมต่อไปยังทรัพยากรนั้นไม่จำเป็นที่จะต้องถูกสร้างขึ้นใหม่เรื่อย ๆ ซึ่งจะช่วยเพิ่มประสิทธิภาพการทำงานของแอพพลิเคชัน วิธีการพูลทรัพยากรนี้สามารถใช้กับทรัพยากรประเภทอื่น ๆ นอกเหนือจากฐานข้อมูลได้อีกด้วย เช่น การเชื่อมต่อซ็อกเก็ต (socket connection) หรือเธรด (thread) จริง ๆ แล้วในสถาปัตยกรรมแบบสามเทียร์นั้น บิซิเนสคอมโพเน็นต์เองสามารถที่จะพูลและให้บริการแก่ไคลเอ็นต์หลาย ๆ เครื่องได้ การพูลคอมโพเน็นต์นั้นหมายความว่าเราไม่จำเป็นที่จะต้องมีคอมโพเน็นต์ส่วนตัวสำหรับไคลเอ็นต์แต่ละเครื่อง การพูลทรัพยากรนั้นถูกแสดงไว้ในรูปที่ 3-6



รูปที่ 3-6 การพูลทรัพยากร (Resource Pooling) ในสถาปัตยกรรมแบบสามเทียร์

6) ประสิทธิภาพถูกแบ่งเป็นส่วน ๆ

หากเทียร์ใดเทียร์หนึ่งเกิดการโอเวอร์โหลด (overload) ขึ้น เทียร์อื่น ๆ จะยังคงทำงานต่อไปได้อย่างปกติ เช่นในกรณีของเว็บดีพลอยเมนต์ ผู้ใช้สามารถที่จะเข้ามายังหน้าแรกของเว็บไซต์มาก ถึงแม้ว่าที่แอพพลิเคชันเซิร์ฟเวอร์จะเกิดการโอเวอร์โหลดก็ตาม

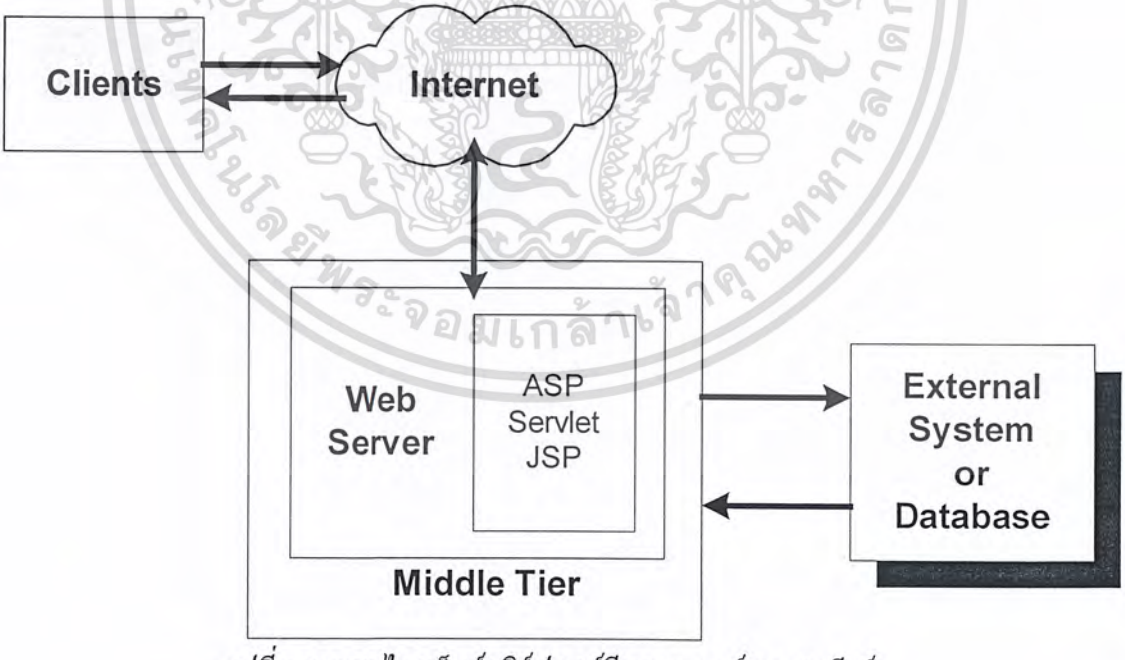
7) ความผิดพลาดในการทำงานถูกแบ่งเป็นส่วน ๆ

หากเกิดความผิดพลาดในการทำงานอย่างรุนแรงขึ้นที่เทียร์ใด ๆ ความผิดพลาดก็จะถูกจำกัดวงอยู่เพียงในเทียร์นั้นโดยไม่มีผลกระทบต่อเทียร์อื่น ๆ ยกตัวอย่างเช่น หากแอพพลิเคชันเซิร์ฟเวอร์เกิดความผิดพลาดในการทำงาน หรือแครช (crash) เว็บเซิร์ฟเวอร์สามารถทำการรายงานความผิดพลาดให้กับผู้ใช้ทราบได้

8) มีค่าใช้จ่ายในการบำรุงรักษาสูง

เนื่องจากเราต้องทำการดีพลอยระบบถึงสามเทียร์ขึ้นไปในทางกายภาพ ทำให้ค่าใช้จ่ายในการติดตั้งซอฟต์แวร์ , การอัพเกรดซอฟต์แวร์ , การดีพลอยใหม่ และค่าใช้จ่ายในการบำรุงรักษาระบบสูงขึ้น

web-base ดีพลอยเมนต์แบบ 3 เทียร์ หากมองแบบไคลเอ็นต์/เซิร์ฟเวอร์ดีพลอยเมนต์แล้วจะกลายเป็น 4 เทียร์ เนื่องจากการแบ่งเป็นเทียร์เป็นการแบ่งทางกายภาพที่ขึ้นอยู่กับว่าแบ่งโดยสิ่งใด ใน web-base ดีพลอยเมนต์จะแบ่งโดยใช้หน้าที่ที่แต่ละส่วนทำในระบบเว็บคือ ส่วนแสดงผล ส่วนประมวลผลบิซิเนสลอจิก และส่วนเก็บข้อมูล แต่แบบไคลเอ็นต์/เซิร์ฟเวอร์ดีพลอยเมนต์จะแบ่งโดยใช้งานที่เครื่องทำเป็นหลัก แบ่งเป็น ไคลเอ็นต์เทียร์ , มิดเดิลเทียร์ และดาต้าเทียร์ โดยมิดเดิลเทียร์อาจจะเป็นเว็บเซิร์ฟเวอร์อย่างเดียว หรือมีแอพพลิเคชันเซิร์ฟเวอร์ด้วยก็ได้ โดยหากมีเว็บเซิร์ฟเวอร์อย่างเดียวก็จะเป็นระบบไคลเอ็นต์/เซิร์ฟเวอร์ดีพลอยเมนต์แบบ 3 เทียร์ ดังรูปที่ 3-7 หากมีแอพพลิเคชันเซิร์ฟเวอร์ด้วยจะเป็นระบบไคลเอ็นต์/เซิร์ฟเวอร์ดีพลอยเมนต์แบบ 4 เทียร์ ดังรูปที่ 3-8

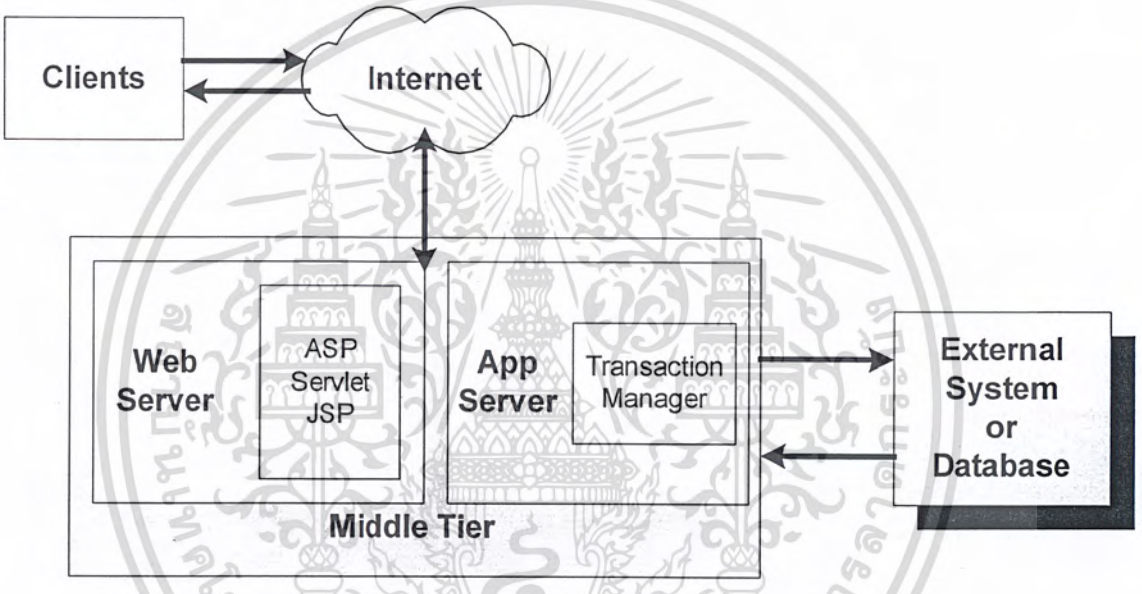


รูปที่ 3-7 ระบบไคลเอ็นต์/เซิร์ฟเวอร์ดีพลอยเมนต์แบบ 3 เทียร์

3.4 สถาปัตยกรรมแบบ N เทียร์ (N-Tier Architecture)

สถาปัตยกรรมแบบเอ็นเทียร์เป็นการเพิ่มเทียร์หนึ่งหรือมากกว่าลงไปในโมเดลแบบ 2 เทียร์ ในดีพลอยเมนต์แบบเอ็นเทียร์ พรีเซนเทชันเลเยอร์ , บิซิเนสลอจิกเลเยอร์ และดาต้าเลเยอร์ จะถูกแบ่งไว้ในแต่ละเทียร์ ในสถาปัตยกรรมแบบ 4 เทียร์หรือมากกว่า จะเป็นการแบ่งเลเยอร์เหล่านี้ลงไปอีก เพื่อทำให้แต่ละส่วนของระบบนั้นสามารถเพิ่มขยายได้โดยอิสระจากกัน

ตัวอย่างที่ดีของสถาปัตยกรรมแบบ N เทียร์ได้แก่ web-base ดีพลอยเมนต์แบบ 3 เทียร์ ดังแสดงในรูปที่ 3-4 ซึ่งเทียร์ทั้งสามนั้น ถูกแบ่งออกดังนี้



รูปที่ 3-8 ระบบไคลเอ็นต์/เซิร์ฟเวอร์ดีพลอยเมนต์แบบ 4 เทียร์

บทที่ 4
เอนเตอร์ไพรส์จาวาบีน

4.1 ส่วนประกอบของ EJB

Enterprise JavaBeans (EJB) ได้นำหลักการในการแบ่งการทำงานเป็นส่วน ๆ (Divide-and-Conquer) มาใช้ในการพัฒนาโปรแกรมที่ประมวลผลบนฝั่งเซิร์ฟเวอร์ EJB จะแบ่งความรับผิดชอบในการพัฒนาระบบทั้งหมดออกเป็นส่วน ๆ โดยในแต่ละส่วนนั้นจะรับผิดชอบโดยผู้ที่มีความเชี่ยวชาญในด้านของตนเอง เวลาที่ใช้ในการดีพลอยระบบ EJB จึงลดลงมากเมื่อเปรียบเทียบกับการพัฒนาระบบงานทั่ว ๆ ไป

หน้าที่ที่เกี่ยวข้องกับการพัฒนาระบบด้วย EJB มีดังนี้

- Bean Provider - เป็นผู้สร้างคอมโพเน็นต์ทางธุรกิจที่สามารถนำกลับมาใช้ใหม่ได้ (Reusable) ซึ่งธุรกิจอื่น ๆ สามารถหาซื้อคอมโพเน็นต์ไปใช้ในธุรกิจของตนเองได้ bean นั้นไม่ใช่แอพพลิเคชันที่สมบูรณ์ในตัวเอง แต่เป็นคอมโพเน็นต์ซึ่งสามารถนำไปดีพลอย และประกอบกันขึ้นมาเป็นแอพพลิเคชันที่สมบูรณ์ได้

- Container Provider – เป็นผู้พัฒนาคอนเทนเนอร์ที่จะทำหน้าที่ในการจัดการกับเอ็นไวรอนเมนต์ในการทำงานให้กับ EJB

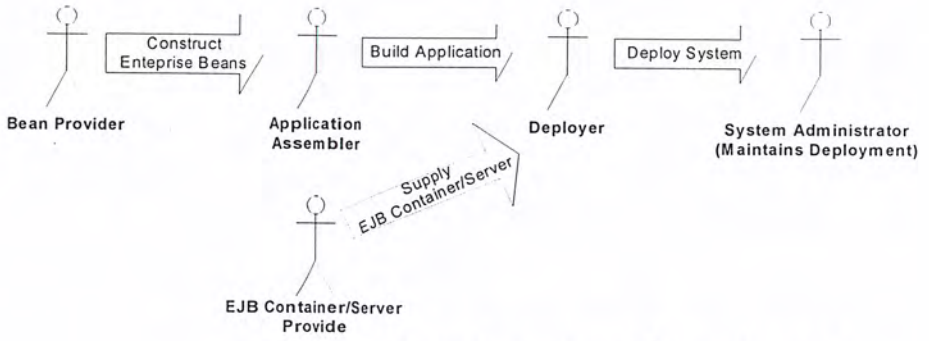
- Server Provider – เป็นผู้พัฒนาแอพพลิเคชันเซิร์ฟเวอร์ แอพพลิเคชันเซิร์ฟเวอร์คือเซิร์ฟเวอร์ในมิดเดิลเทียร์ ทำหน้าที่ในการจัดการและดีพลอยคอมโพเน็นต์ต่าง ๆ โดยในขณะนี้ยังไม่มีข้อแตกต่างที่เห็นได้ชัดเจนระหว่าง Container Provider และ Server Provider

- Application Assembler - เป็นผู้มีหน้าที่รับผิดชอบในการนำคอมโพเน็นต์ต่าง ๆ มาประกอบกันและเขียนแอพพลิเคชันที่ใช้คอมโพเน็นต์เหล่านั้น โดยที่อาจจะต้องเขียนคอมโพเน็นต์บางตัวขึ้นมาเอง Application Assembler อาจเป็นบริษัทภายนอกที่รับทำงานนี้โดยเฉพาะ หรืออาจจะเป็นโปรแกรมเมอร์ภายในบริษัทเองก็ได้

- Deployer – เป็นผู้นำเอาคอมโพเน็นต์ต่าง ๆ ที่ได้สร้างไว้แล้วมาทำการติดตั้งลงในแอพพลิเคชันเซิร์ฟเวอร์เพื่อใช้งาน

- System Administrator - มีหน้าที่ดูแลระบบที่ถูกดีพลอยเรียบร้อยแล้วให้ทำงานเป็นปกติ

ความสัมพันธ์ในการทำงานร่วมกันของแต่ละส่วน แสดงในรูปที่ 4-1



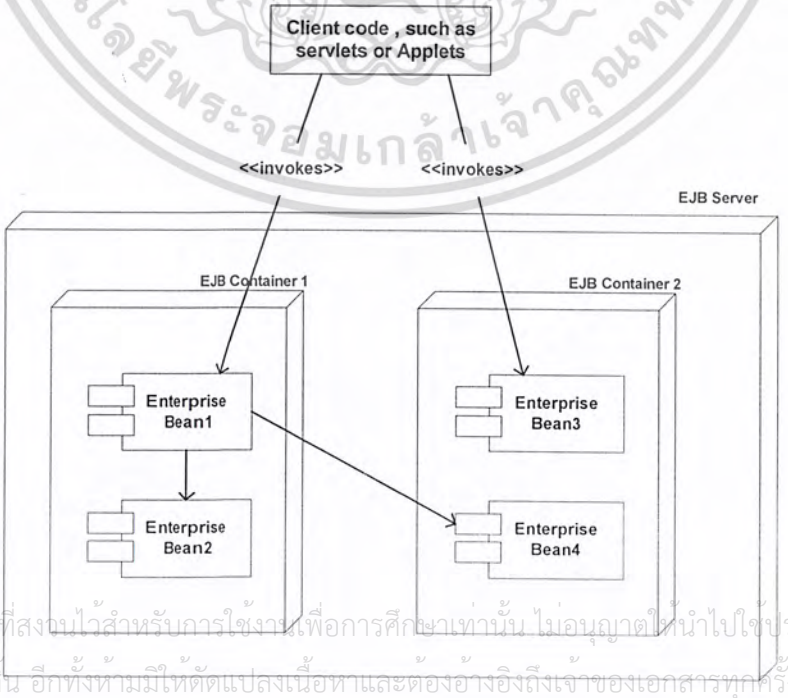
รูปที่ 4-1 ความสัมพันธ์ของแต่ละหน้าที่ในการพัฒนา EJB

4.2 EJB เซิร์ฟเวอร์ และ EJB คอนเทนเนอร์

แอพพลิเคชันเซิร์ฟเวอร์จะจัดหาบริการมิดเดิลแวร์ (middleware service) ให้กับแอพพลิเคชัน เช่น ทรานแซ็กชัน , ความปลอดภัย และอื่น ๆ บริการเหล่านี้จำเป็นสำหรับแอพพลิเคชันที่ต้องการความสามารถในการเพิ่มขยาย , มีประสิทธิภาพ และมีความปลอดภัย ในการให้บริการกับผู้ใช้จำนวนมากพร้อม ๆ กัน แอพพลิเคชันเซิร์ฟเวอร์จะแบ่งเป็น 2 ส่วนคือ

- EJB คอนเทนเนอร์ – เป็นที่อยู่ที่เอนเตอร์ไพรส์บีนสามารถทำงานได้ ภายในคอนเทนเนอร์หนึ่งสามารถมีบีนจำนวนมากทำงานอยู่ คอนเทนเนอร์มีหน้าที่ในการควบคุมบีนที่รันอยู่ภายในตัวมัน

- EJB Server – EJB เซิร์ฟเวอร์จะจัดหารันไทม์เอ็นไวรอนเมนต์ให้กับคอนเทนเนอร์ โดยภายใน EJB เซิร์ฟเวอร์สามารถมีคอนเทนเนอร์ได้มากกว่า 1 คอนเทนเนอร์ EJB เซิร์ฟเวอร์จะควบคุมทรัพยากรระบบในระดับล่าง และจัดสรรทรัพยากรให้กับคอนเทนเนอร์ ความสัมพันธ์ระหว่าง EJB เซิร์ฟเวอร์และคอนเทนเนอร์เป็นดังรูปที่ 4-2



รูปที่ 4-2 ความสัมพันธ์ระหว่าง EJB เซิร์ฟเวอร์และคอนเทนเนอร์

และเนื่องจาก EJB specification ไม่ได้แบ่ง EJB คอนเทนเนอร์และเซิร์ฟเวอร์อย่างชัดเจน ใน EJB specification ได้สมมติว่าทั้ง EJB คอนเทนเนอร์และเซิร์ฟเวอร์มาจากผู้ผลิตเดียวกัน และใช้ในความหมายเดียวกันคือทำหน้าที่ในการควบคุมการทำงานของ enterprise bean ตัวอย่างของแอพพลิเคชันเซิร์ฟเวอร์ที่มี EJB คอนเทนเนอร์/เซิร์ฟเวอร์ดังตารางที่ 4-1

Vendor Product	Edition Version	JDK	EJB	JSP	JMS	J2EE License	J2EE Certif.
Allaire/Macromedia JRun Server	Developer v3.1	1.2 1.3	1.1	1.1	1.0		
	Enterprise v3.1	1.2 1.3	1.1	1.1	1.0	✔	✔
ATG Dynamo	v4.0	1.2.2	1.1	1.1	1.0.2	✔	✔
BEA WebLogic	Server v6.1	1.3.1	1.1 2.0**	1.2	1.0.2	✔	✔
	Enterprise v4.1	1.2	1.1	1.1	1.0.2		✔
Borland (Formerly Inprise) AppServer	v4.5	1.2.2 1.3	1.1	1.1	1.0.2	✔	✔
Fujitsu Software Corp. Interstage **Limited support	Enterprise v4.0	1.2 1.3	1.1 2.0**	1.1 1.2	1.0.2		✔
GemStone GemStone/J	Component v4.1	1.2	1.0	1.1		✔	
	Enterprise v4.1	1.2	1.0	1.1			
	Commerce Automation v4.1	1.2	1.0	1.1			

ตารางที่ 4-1 แอพพลิเคชันเซิร์ฟเวอร์

Vendor Product	Edition Version	JDK	EJB	JSP	JMS	J2EE License	J2EE Certif.
HP Bluestone Total-e-Server	Total-e-Server v7.3	1.2.2 1.3	1.1	1.1	1.0.2	✔	✔
	Total-e-Mobile v1.1	1.2.2 1.3	1.1	1.1	1.0.2		✔
IBM Websphere	Standard v3.4.3	1.2.2		1.1	1.0.2	✔	
	Advanced v4.0	1.3	1.1	1.1	1.0.2 +XA		✔
	Advanced Single Srvr v4.0	1.3	1.1	1.1	1.0.2		✔
	Advanced Developer v4.0	1.3	1.1	1.1	1.0.2		✔
	Enterprise v3.6	1.2.2	1.0	1.0	1.0.2		
	Enterprise v4.0	1.3	1.1	1.1	1.0.2 +XA		✔
In-Q-My Application Server	v4.1	1.2.2 1.3	1.1	1.1	1.0.2	✔	✔
IONA iPortal	Standard v3.0	1.3 1.3.1	1.1	1.2	1.0.2	✔	✔
	Enterprise v1.3	1.3	1.1	1.2	1.0.2		✔
iPlanet	Enterprise v6.0	1.2.2	1.1	1.1	1.0.2	✔	✔
	Enterprise Pro v6.0	1.2.2	1.1	1.1	1.0.2		✔
JBoss **Limited support	v2.4.0	1.2.2 1.3	1.1 2.0**	1.1	1.0.2		

ตารางที่ 4-1 แอพพลิเคชันเซิร์ฟเวอร์ (ต่อ)

Vendor Product	Edition Version	JDK	EJB	JSP	JMS	J2EE License	J2EE Certif.
Lutris	Lutris EAS	1.3	1.1	1.1	1.0.2		
	Enhydra v3.1	1.2.2 1.3		1.1			
	Lutris Enhydra v3.4.2	1.2.2 1.3		1.1			
Oracle 9i AS	Standard v1.0.2	1.1.8 1.2.2	1.1	1.1			
	Enterprise v1.0.2	1.1.8 1.2.2	1.1	1.1		✔	
	Wireless v1.0.2	1.1.8 1.2.2	1.1	1.1			
Persistence Power Tier for J2EE	v7.0	1.2 1.3 1.3.1	1.1	1.1	1.0.2	✔	✔
Pramati Technologies Pramati Server	v2.5	1.2.2, 1.3	1.1	1.1	1.0.2		
**Limited support	v3.0	1.3	2.0**	1.1	1.0.2		
SilverStream Application Server	Developer v3.7.3	1.2 1.3	1.1	1.1	1.0.2		✔
	Enterprise v3.7.3	1.2 1.3	1.1	1.1	1.0.2	✔	✔
Sybase Enterprise Application Server	Small Bus. v3.6	1.2	1.1	1.1	1.0.2		✔
	Advanced v3.6	1.2	1.1	1.1	1.0.2	✔	✔
	Enterprise v3.6	1.2	1.1	1.1	1.0.2		✔

ตารางที่ 4-1 แอพพลิเคชันเซิร์ฟเวอร์ (ต่อ)

4.3 เอนเตอร์ไพรส์บีน

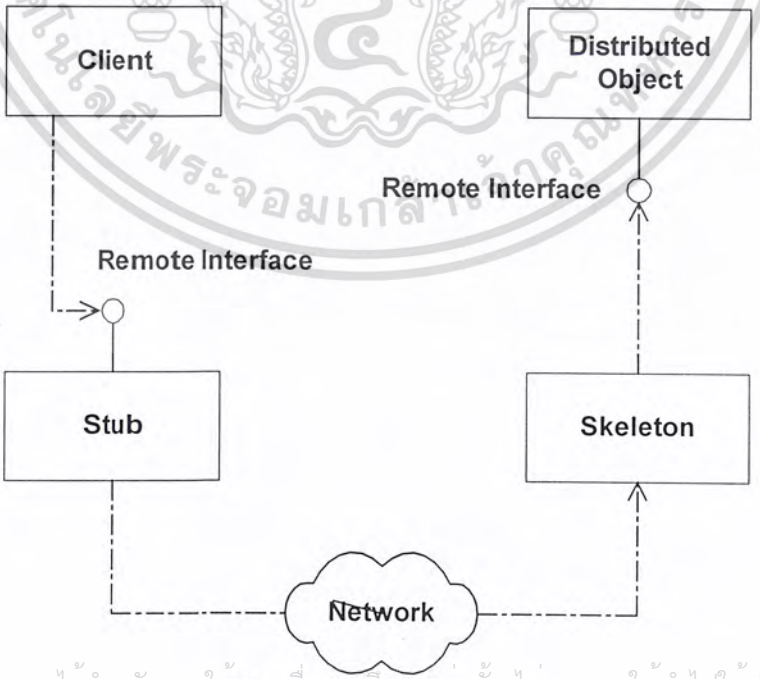
เอนเตอร์ไพรส์บีน เป็นคอมโพเน็นต์ที่ทำงานบนฝั่งเซิร์ฟเวอร์ (Server-side software component) ซึ่งสามารถนำไปดีพลอยในเอ็นไวรอนเมนต์มัลติเทียร์แบบกระจาย (Distributed multi-tier environment) ได้ ภายในหนึ่งเอนเตอร์ไพรส์บีนสามารถประกอบไปด้วยจาวาออบเจ็กต์มากกว่าหนึ่งชิ้นได้ เนื่องจากคอมโพเนนต์นั้นไม่จำเป็นจะต้องเป็นออบเจ็กต์ชิ้นเดียว

ไม่ว่าเอนเตอร์ไพรส์บีนนั้นจะมีองค์ประกอบภายในอย่างไรก็ตาม ไคลเอ็นต์ที่มาเรียกใช้ เอนเตอร์ไพรส์บีนนั้นต้องติดต่อผ่านทางอินเทอร์เฟซของเอนเตอร์ไพรส์บีนเท่านั้น อินเทอร์เฟซเหล่านี้รวมทั้งเอนเตอร์ไพรส์บีนจะต้องเป็นไปตามข้อกำหนดของ Enterprise JavaBeans ข้อกำหนดเหล่านี้จะระบุถึงเมธอดที่เป็นเมธอดบังคับจำนวนหนึ่ง ซึ่งเมธอดเหล่านี้จะทำให้ EJB คอนเทนเนอร์ควบเอนเตอร์ไพรส์บีนได้ในแนวทางเดียวกัน โดยไม่ขึ้นอยู่กับคอนเทนเนอร์ที่เอนเตอร์ไพรส์บีนทำงานอยู่

ไคลเอ็นต์ของเอนเตอร์ไพรส์บีนนั้นจะเป็นอะไรก็ได้ เช่น Servlet , Applet หรือเอนเตอร์ไพรส์บีนอื่น โดยในกรณีหลัง ไคลเอ็นต์ที่ทำการร้องขอไปยังบีนจะทำให้เกิดการร้องขอไปยังบีนตัวอื่น ๆ ต่อไปเรื่อย ๆ ซึ่งเป็นแนวคิดที่ดี เนื่องจากเราสามารถแบ่งบีนที่ทำงานที่ซับซ้อนเป็นส่วนย่อย ๆ ได้

4.3.1 ออบเจ็กต์แบบกระจาย

EJB คอมโพเน็นต์มีพื้นฐานอยู่บน RMI ซึ่งเป็นเทคโนโลยีออบเจ็กต์แบบกระจาย รูปที่ 4-3 จะแสดงการเรียกออบเจ็กต์แบบกระจายของไคลเอ็นต์



รูปที่ 4-3 ออบเจ็กต์แบบกระจาย

- ไคลเอ็นต์เรียก stub ซึ่งเป็น proxy ออบเจ็กต์ทางฝั่งไคลเอ็นต์ (client-side proxy object) stub จะซ่อนการติดต่อผ่านเน็ตเวิร์ก (network communication) จากไคลเอ็นต์ stub จะรู้ว่าจะเรียกผ่านเน็ตเวิร์ก (call over network) อย่างไรผ่านทางซ็อกเก็ต
- stub จะเรียกผ่านเน็ตเวิร์กไปยัง skeleton ซึ่งเป็น proxy ออบเจ็กต์ทางฝั่งเซิร์ฟเวอร์ (server-side proxy object) skeleton จะซ่อนการติดต่อผ่านเน็ตเวิร์กจากออบเจ็กต์แบบกระจาย skeleton จะรู้ว่าจะรับการเรียกอย่างไรผ่านทางซ็อกเก็ต
- skeleton จะเป็นตัวแทนในการเรียกออบเจ็กต์แบบกระจาย ออบเจ็กต์แบบกระจายจะทำงานของมันและจะคืนผลลัพธ์ให้กับ skeleton กลับไปยัง stub ซึ่งจะคืนให้กับไคลเอ็นต์ต่อไป

จุดสำคัญคือทั้ง stub และออบเจ็กต์แบบกระจายจะใช้อินเทอร์เฟซเดียวกัน (ที่เรียกว่ารีโมตอินเทอร์เฟซ) ไคลเอ็นต์ที่เรียกเมธอดผ่าน stub จะคิดว่าเรียกออบเจ็กต์แบบกระจายโดยตรง แต่ในความจริงแล้วไคลเอ็นต์ได้เรียก stub ที่รู้ว่าจะทำมันผ่านเน็ตเวิร์กได้อย่างไร สิ่งนี้เรียกว่า local/remote tranparency

4.3.2 ออบเจ็กต์แบบกระจายและมิดเดิลแวร์

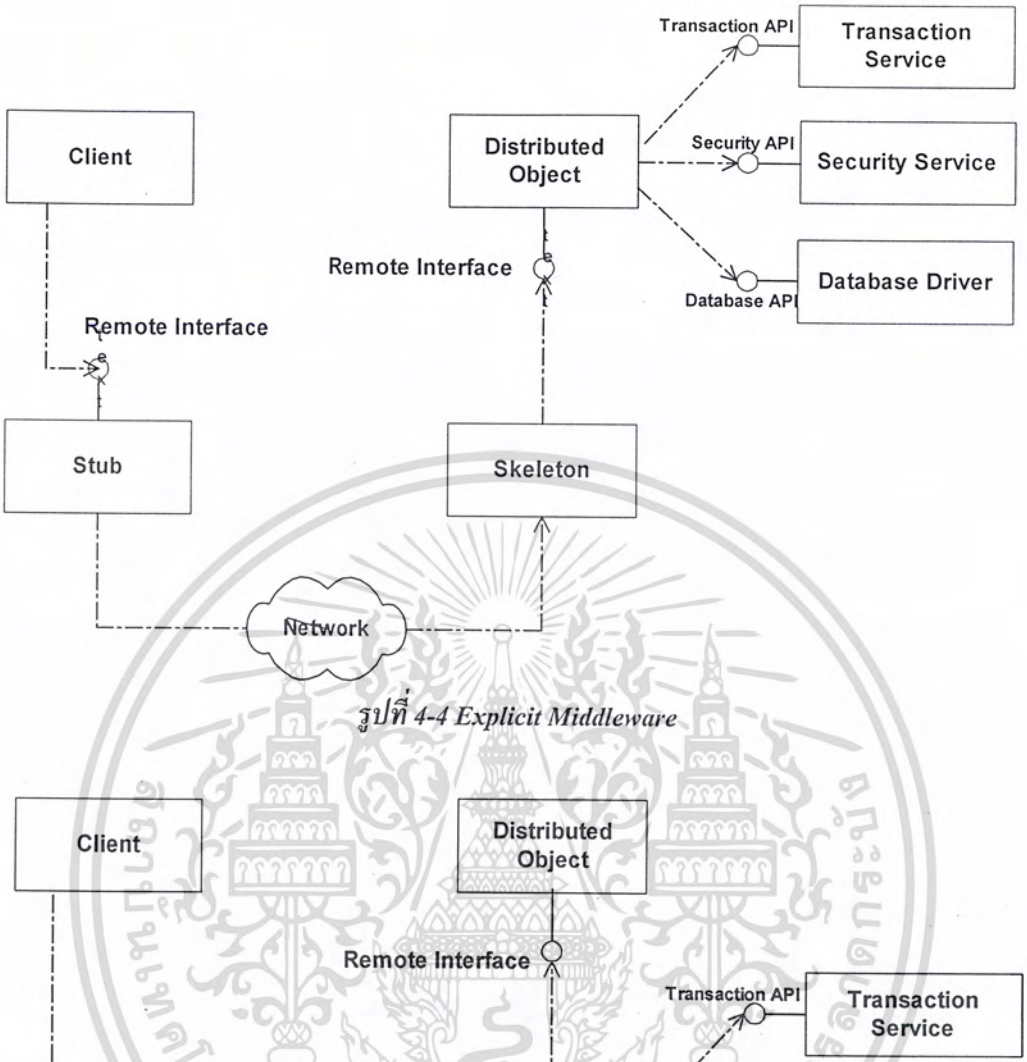
ออบเจ็กต์แบบกระจายนั้นมีประโยชน์มาก ทำให้สามารถเรียกใช้คอมโพเน็นต์บนเน็ตเวิร์กได้ แต่มันก็ยังต้องการบริการจากมิดเดิลแวร์ เช่น ทรานแซ็กชันและความปลอดภัย

- Explicit Middleware

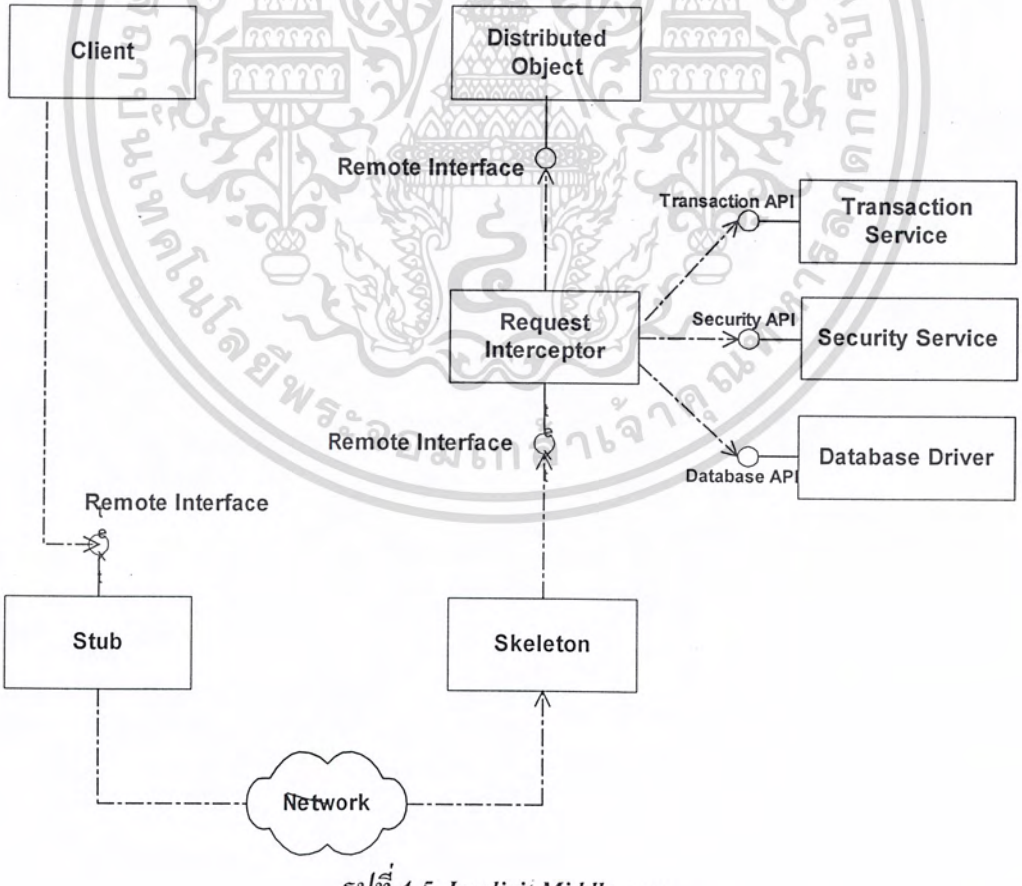
โดยปกติออบเจ็กต์แบบกระจายจะติดต่อกับมิดเดิลแวร์โดยเขียนโปรแกรมติดต่อกับ API ของมิดเดิลแวร์ เช่นหากต้องการการจัดการทางด้านทรานแซ็กชันจะต้องเขียนโค้ดติดต่อกับ transaction API ดังรูปที่ 4-4 มิดเดิลแวร์ประเภทนี้มีข้อเสียคือต้องเขียนโปรแกรมติดต่อกับ API ของมิดเดิลแวร์และหากต้องการเปลี่ยนมิดเดิลแวร์ที่ใช้ จะต้องเขียนโค้ดใหม่

- Implicit Middleware

ความแตกต่างที่สำคัญของระบบในอดีต (transaction processing monitors เช่น TUXEDO หรือ CICS หรือ เทคโนโลยีออบเจ็กต์แบบกระจายแบบเดิม ๆ เช่น CORBA , DCOM และ RMI) และแบบใหม่ที่เป็นเทคโนโลยีที่มีพื้นฐานจากคอมโพเน็นต์ (EJB , CORBA Component Model และ Microsoft .NET) แบบนี้ไม่จำเป็นที่จะต้องเขียนมิดเดิลแวร์ API แต่จะใช้วิธีการประกาศ (declare) เรียกได้อีกอย่างว่า declarative middleware ซึ่งจะง่ายในการเขียนโปรแกรมและดูแลรักษา โดยใน EJB จะถูกประกาศใน deployment descriptor ซึ่งเป็นไฟล์ XML ที่กำหนดแท็กสำหรับการประกาศค่าต่าง ๆ ไว้



รูปที่ 4-4 Explicit Middleware



รูปที่ 4-5 Implicit Middleware

4.4 ชนิดของ bean (Types of Beans)

ตามข้อกำหนดของ EJB 1.0 และ 1.1 นั้น ได้กำหนดประเภทของ Enterprise Bean ไว้สองชนิดด้วยกัน คือ session bean และ entity bean และได้เพิ่ม enterprise bean อีกชนิดหนึ่งเข้าไปใน EJB 2.0 ได้แก่ message-driven bean

4.4.1 Session Bean

Session bean เป็นตัวแทนของงานที่ประมวลผลให้กับไคลเอ็นต์ที่เรียกใช้มัน session bean เป็นออบเจ็กต์ที่ใช้ในการประมวลผลลอจิกทางธุรกิจ (Business Logic) เช่น การแจ้งราคาสินค้า , การใส่ข้อมูลการสั่งสินค้า , การทำธุรกรรมกับธนาคาร , การบีบอัดภาพวีดีโอ , การจัดการกับฐานข้อมูล , การประมวลผลที่ซับซ้อน และอื่น ๆ session bean จึงเป็นคอมโพเน็นต์ที่สามารถนำกลับมาใช้ใหม่ได้ซึ่งบรรจุลอจิกของกระบวนการทางธุรกิจ

เหตุผลที่มันถูกเรียกว่า session bean เนื่องจากว่ามันมีอายุเท่ากับ session ของไคลเอ็นต์ที่เรียกใช้มัน ตัวอย่างเช่น ถ้ามีไคลเอ็นต์ติดต่อเข้ามายัง session bean เพื่อสั่งสินค้า แอพพลิเคชันเซิร์ฟเวอร์จะมีหน้าที่ในการสร้างอินสแทนซ์ของคอมโพเน็นต์ session bean ขึ้นมา เมื่อไคลเอ็นต์ที่เรียกใช้ยกเลิกการติดต่อ (Disconnect) แอพพลิเคชันเซิร์ฟเวอร์สามารถทำลายอินสแทนซ์ของ session bean นั้นได้

ณ เวลาใดเวลาหนึ่ง session bean หนึ่งสามารถถูกเรียกใช้ได้จากไคลเอ็นต์เพียงหนึ่งตัวเท่านั้น นั่นคือจะไม่มีการแบ่งใช้ session bean ร่วมกันระหว่างไคลเอ็นต์ที่เข้ามาขอใช้บริการ ซึ่งเป็นจุดที่แตกต่างจาก entity bean โดย entity bean นั้นจะถูกใช้ร่วมกันหลาย ๆ ไคลเอ็นต์

EJB เซิร์ฟเวอร์มีหน้าที่จัดการและควบคุมวงจรชีวิต (Life Cycle) ของทุก bean นั่นคือ ไคลเอ็นต์จะไม่ได้ทำการสร้างอินสแทนซ์ของ bean ขึ้นมาเองโดยตรง แต่ EJB คอนเทนเนอร์จะเป็นผู้สร้างอินสแทนซ์ของ bean ขึ้นมาให้โดยอัตโนมัติ ในทำนองเดียวกัน EJB คอนเทนเนอร์จะเป็นผู้ทำลายอินสแทนซ์ของ bean ในเวลาที่เหมาะสม ซึ่งทำให้ bean สามารถพูลและนำกลับมาใช้ใหม่กับหลาย ๆ ไคลเอ็นต์ที่ทำการร้องขอเข้ามาได้

session bean มี 2 ชนิดคือ stateful session bean และ stateless session bean

4.4.1.1 Stateful Session Bean

กระบวนการทางธุรกิจ (Business Process) บางอย่าง สามารถทำให้เสร็จสิ้นได้ในการเรียกใช้เมธอดเพียงครั้งเดียว เช่นการคำนวณราคาสินค้า หรือการตรวจสอบยอดบัตรเครดิต แต่กระบวนการทางธุรกิจบางอย่างไม่สามารถทำให้เสร็จสิ้นได้ในการเรียกใช้งานเมธอดเพียงครั้งเดียว

เช่นเว็บไซต์แบบอี-คอมเมิร์ซ ในการใช้งานเว็บไซต์แบบอี-คอมเมิร์ซ ลูกค้าสามารถเพิ่มรายการสินค้าเข้าไปในรถเข็นแบบออนไลน์ ซึ่งการเพิ่มสินค้าเข้าไปในรถเข็นแต่ละครั้งนั้น คอมโพเน็นต์ที่รับผิดชอบจะต้องจดจำสถานะของรถเข็นได้

stateful session bean เป็น session bean ซึ่งถูกออกแบบมาเพื่อให้บริการกับกระบวนการทางธุรกิจที่ต้องมีการเรียกใช้หลายเมธอด หรือต้องมีการใช้ทรานแซ็กชันโดย stateful session bean จะเก็บสถานะต่าง

ๆ ของไคลเอ็นต์ไว้ในตัวมัน ถ้าสถานะของ stateful session bean มีการเปลี่ยนแปลงในระหว่างการเรียกใช้เมธอด ค่าสถานะนั้นจะยังคงอยู่เมื่อไคลเอ็นต์เดิมทำการเรียกใช้เมธอดครั้งถัดไป

4.4.1.2 Stateless Session Bean

กระบวนการทางธุรกิจบางชนิดมีรูปแบบการร้องขอบริการไปยัง bean ด้วยการร้องขอเมธอดเพียงครั้งเดียว เนื่องจากงานของกระบวนการทางธุรกิจประเภทนี้ สามารถถูกทำให้เสร็จสิ้นได้โดยการเรียกใช้เมธอดเพียงครั้งเดียว จึงไม่จำเป็นต้องมีการคงสถานะของไคลเอ็นต์เอาไว้ในการเรียกใช้เมธอดครั้งต่อไป โดย stateless session bean นั้นจะให้บริการกับไคลเอ็นต์ที่ร้องขอเข้ามาโดยไม่แบ่งแยก และจะไม่เก็บสถานะเดิมของไคลเอ็นต์ที่ทำการติดต่อด้วยไว้

ตัวอย่างของ stateless session bean อาจจะเป็น bean ที่รับอินพุตเพื่อใช้ในการคำนวณทางคณิตศาสตร์ที่ซับซ้อน เช่น การบีบอัดข้อมูลเสียงและวีดีโอ ไคลเอ็นต์สามารถส่งข้อมูลที่ยังไม่ได้ถูกบีบอัดพร้อมด้วยคุณสมบัติในการบีบอัดข้อมูลมาให้ bean ซึ่ง bean จะทำการประมวลผลข้อมูลที่รับเข้ามาและส่งข้อมูลที่ถูกบีบอัดแล้วกลับไปยังไคลเอ็นต์ หลังจากนั้น stateless session bean ตัวนี้ ก็สามารถให้บริการไคลเอ็นต์ตัวอื่น ๆ ต่อไปได้ สังเกตว่ากระบวนการนี้ สามารถถูกทำให้เสร็จสิ้นได้ในการเรียกใช้งานเมธอดเพียงครั้งเดียว และ bean จะไม่ต้องเก็บสถานะเดิมของไคลเอ็นต์ใด ๆ เอาไว้

อีกตัวอย่างหนึ่งของ stateless session bean ได้แก่ คอมโพเน็นต์ที่ทำหน้าที่ตรวจสอบความถูกต้องของหมายเลขบัตรเครดิต bean ที่ทำหน้าที่นี้ จะรับข้อมูลหมายเลขบัตรเครดิต , วันหมดอายุ , ชื่อผู้ถือบัตร และจำนวนเงินเข้ามา หลังจากนั้น bean จะส่งข้อมูลตอบกลับไปว่า ข้อมูลนี้ถูกต้องหรือไม่โดยประมวลผลจากข้อมูลที่รับเข้ามา

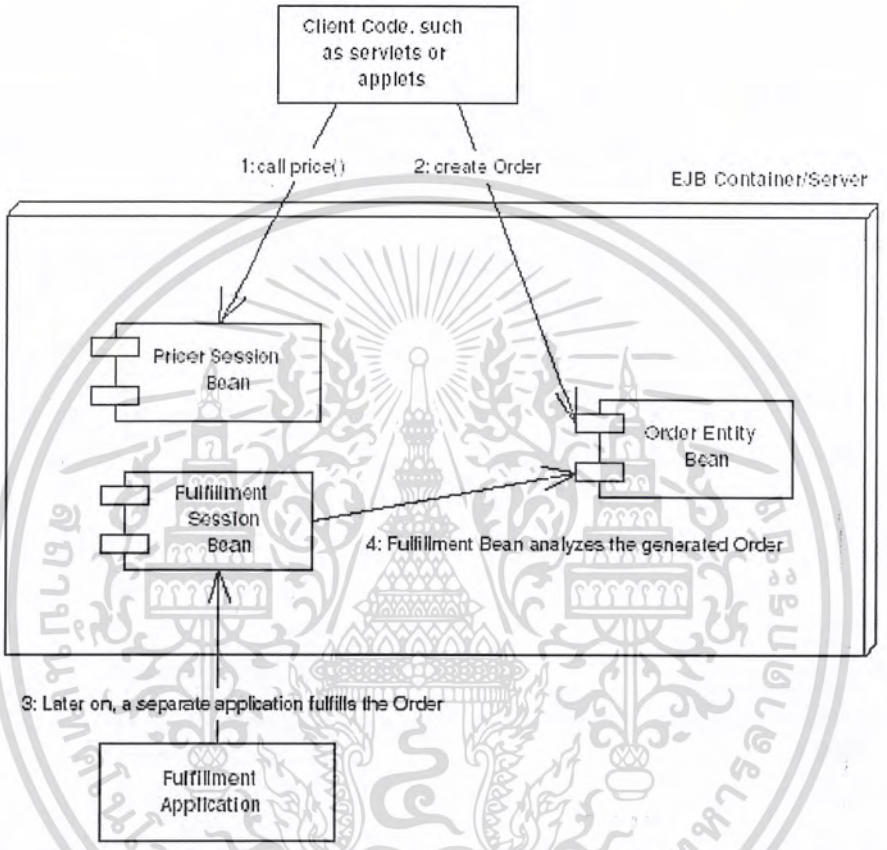
4.4.2 Entity Bean

ส่วนประกอบพื้นฐานของธุรกิจอีกส่วนหนึ่งก็คือ ข้อมูลถาวร (Permanent Data) ซึ่งกระบวนการทางธุรกิจจะต้องนำไปใช้ ดังตัวอย่างเช่น

- คอมโพเน็นต์ที่ทำหน้าที่เป็นพนักงานธนาคาร จะต้องสามารถทำกระบวนการทางธุรกรรมต่าง ๆ ได้ โดยข้อมูลซึ่งใช้ในการทำงานเหล่านี้ ก็คือข้อมูลบัญชีธนาคาร
- คอมโพเน็นต์ซึ่งทำหน้าที่รับรายการสั่งซื้อสินค้า จะต้องทำหน้าที่บันทึกรายการสินค้าที่ลูกค้าสั่งซื้อ ซึ่งข้อมูลที่ถูกสร้างขึ้นโดยคอมโพเน็นต์นี้ ก็คือตัวรายการสั่งสินค้านั่นเอง ซึ่งจะต้องมีรายละเอียดของสินค้าต่าง ๆ ที่ลูกค้าสั่งซื้อ

ในแต่ละสถานการณ์ที่กล่าวมานี้ คอมโพเน็นต์ของกระบวนการทางธุรกิจ ต้องเกี่ยวข้องกับข้อมูลซึ่งถูกเก็บอยู่ในแหล่งเก็บข้อมูลต่าง ๆ เช่นฐานข้อมูลเชิงสัมพันธ์ (Relational Database) เป็นต้น entity bean คือคอมโพเน็นต์ซึ่งเป็นตัวแทนของข้อมูลเหล่านี้ เช่น บัญชีธนาคาร หรือรายการสั่งซื้อสินค้าดังที่ได้ยกตัวอย่างไป นอกจากนี้ entity bean อาจจะแสดงถึงออบเจ็กต์ของข้อมูลจริง ๆ เช่น ลูกค้า , สินค้า หรือพนักงาน

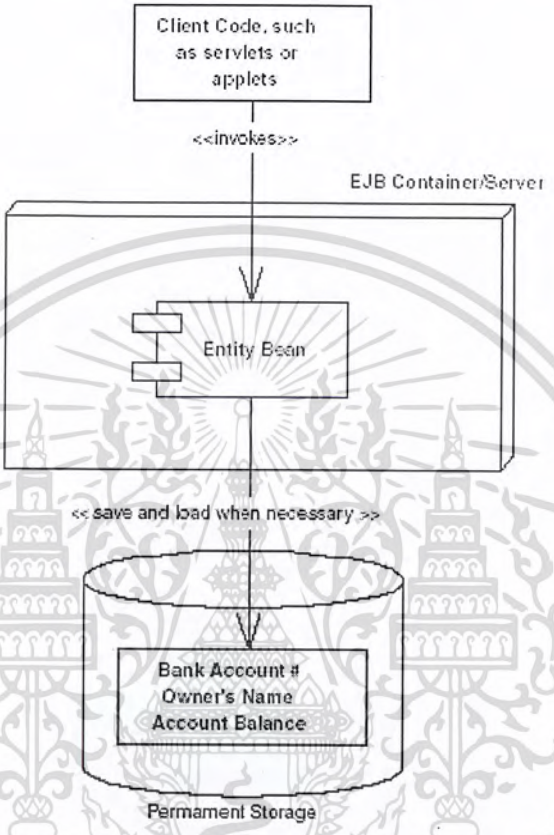
entity bean จะไม่เก็บลอจิกวิธีการประมวลผลไว้ในตัวมัน แต่จะเป็นตัวแทนของข้อมูล session bean จะจัดการกับการประมวลผลทางธุรกิจ และอาจเรียกใช้ entity bean เพื่อเป็นตัวแทนของข้อมูลที่มันทำการติดต่อด้วย ในลักษณะเดียวกับการที่พนักงานธนาคาร ต้องใช้ข้อมูลบัญชีธนาคารในการทำงาน ดังรูปที่ 4-6



รูปที่ 4-6 การนำ Session Bean และ Entity Bean มาใช้ร่วมกันในแอพลิเคชันการคิดราคาของสินค้าที่สั่งซื้อ

ประโยชน์ของ entity bean คือ มันจะเป็นมุมมอง (View) แบบออบเจ็กต์ในหน่วยความจำไปยังแหล่งเก็บข้อมูล ซึ่งถ้าเป็นวิธีการทั่วไป เมื่อแอพพลิเคชันต้องการที่จะใช้ข้อมูล จะต้องลงไปจัดการกับตารางในฐานข้อมูลเชิงสัมพันธ์โดยตรง โดยทำการอ่านและเขียนข้อมูลต่าง ๆ ที่ต้องการ ในขณะที่ entity bean จะเป็นตัวแทนของข้อมูล เราสามารถจัดการกับข้อมูลซึ่งอยู่ในฐานข้อมูลเชิงสัมพันธ์ได้เหมือนกับจัดการกับออบเจ็กต์ เราสามารถอ่านเซตของข้อมูลจากฐานข้อมูลมาไว้ใน entity bean ซึ่งอยู่ในหน่วยความจำ และเราสามารถจัดการกับ entity bean ซึ่งอยู่ในหน่วยความจำนี้ได้โดยการเรียกใช้เมธอดของมัน เช่น เราสามารถเรียกใช้เมธอดซึ่งมีชื่อว่า withdraw() ของ entity bean บัญชีธนาคาร เพื่อทำการลดค่าของตัวแปรแบบ private ในตัว entity bean ซึ่งมีชื่อว่า balance โดยอัตโนมัติ โดยข้อมูลจริง ๆ ซึ่งอยู่ในฐานข้อมูลจะถูกเปลี่ยนแปลงตามไปด้วยโดยอัตโนมัติ

entity bean จะช่วยให้เราใช้ความสามารถของข้อมูลซึ่งมีความถาวร (Persistant) และยังมีคุณสมบัติ encapsulation ในขณะเดียวกัน entity bean จะเป็นเลเยอร์ของการเรียกใช้ข้อมูล (Data Access Logic) ในสถาปัตยกรรมแบบมัลติเทียร์ ดังแสดงในรูปที่ 4-7



รูปที่ 4-7 Entity Bean เป็นมุมมองแบบออบเจ็กต์ไปยังแหล่งเก็บข้อมูล

ความแตกต่างอย่างชัดเจนระหว่าง session bean กับ entity bean อีกอย่างหนึ่งก็คือ ไคลเอ็นต์หลาย ๆ ตัวสามารถเข้ามาใช้บริการจาก entity bean ได้พร้อม ๆ กัน ในลักษณะเดียวกับการที่ไคลเอ็นต์หลาย ๆ ตัวสามารถเข้าถึงข้อมูลในฐานข้อมูลได้พร้อม ๆ กัน และด้วยการใช้ทรานแซ็กชัน ทำให้มั่นใจได้ว่าแต่ละไคลเอ็นต์ที่ทำ operation จะทำงานได้ถูกต้องเป็นอิสระจากไคลเอ็นต์อื่น

entity bean เป็นวิธีการที่มีประสิทธิภาพเมื่อเรามีข้อมูลในฐานข้อมูลอยู่แล้ว ซึ่ง entity bean จะช่วยให้เราสามารถเข้าถึงข้อมูลเหล่านั้นได้ ข้อมูลซึ่งเรานำ entity bean มาใช้แสดงนั้น อาจมีมาก่อนที่เราจะนำ EJB มาใช้ก็ได้ ในความเป็นจริงแล้ว เรคอร์ดในฐานข้อมูลซึ่งแสดงถึงออบเจ็กต์หนึ่ง อาจมีมาก่อนที่ทางหน่วยงานจะตัดสินใจนำภาษาจาวามาใช้ในการพัฒนาแอพพลิเคชัน เนื่องจากว่าโครงสร้างของฐานข้อมูลนั้นไม่ขึ้นกับภาษาใด ๆ (Language-Independent) เรคอร์ดในฐานข้อมูลสามารถที่จะแปลงเป็นออบเจ็กต์ได้ในเกือบจะทุกภาษา ซึ่ง EJB ได้ใช้ประโยชน์ตรงจุดนี้ โดยอนุญาตให้มีการแปลงข้อมูลในฐานข้อมูลให้

เป็นออบเจ็กต์ในภาษาจาวา กระบวนการแปลงข้อมูลนี้ เราอาจจะจัดการด้วย bean หรือให้คอนเทนเนอร์จัดการให้โดยอัตโนมัติก็ได้

entity bean สามารถแบ่งออกได้เป็นอีกสองประเภท ได้แก่ entity bean แบบ bean-managed persistent และแบบ container-managed persistent

4.4.2.1 Bean-managed Persistent Entity Bean

จากที่กล่าวมาแล้ว entity bean เป็นคอมโพเน็นต์ที่มีคุณสมบัติ persistent เพราะว่าสถานะของมันนั้นจะถูกบันทึกลงในแหล่งบันทึกข้อมูล (Secondary Storage) เช่นในฐานข้อมูลเชิงสัมพันธ์ ตัวอย่างเช่น ด้วยเทคโนโลยี Object-relational Mapping เราสามารถแมปออบเจ็กต์ที่อยู่ในหน่วยความจำไปเป็นเรคอร์ดในฐานข้อมูลเชิงสัมพันธ์ซึ่งมีคุณสมบัติ persistent ได้ และเราสามารถนำข้อมูลนี้กลับมาใช้ได้ทุกเมื่อโดยการดึงข้อมูลในฐานข้อมูลกลับมาสร้างเป็นออบเจ็กต์ในหน่วยความจำอีกครั้ง

อีกแนวทางหนึ่ง คือการใช้ฐานข้อมูลแบบออบเจ็กต์ (Object Database) เป็นแหล่งบันทึกข้อมูลแบบ persistent ซึ่งจะบันทึกออบเจ็กต์จริงๆ ลงไปแทนที่จะเป็นเรคอร์ดของความสัมพันธ์

bean-managed persistent entity bean เป็น entity bean ที่ต้องทำการเก็บข้อมูลต่าง ๆ แบบ persistent ด้วยตัวเอง นั่นคือ ผู้พัฒนาคอมโพเน็นต์จะต้องเขียนโค้ดในการแปลงฟิลด์ต่าง ๆ ของ entity bean ซึ่งอยู่ในหน่วยความจำให้กลายเป็นข้อมูลซึ่งเก็บอยู่ในแหล่งบันทึกข้อมูลเช่นฐานข้อมูลเชิงสัมพันธ์ ซึ่งจะต้องเขียนโค้ดสำหรับการเขียน การอ่าน และการค้นหาข้อมูลเข้าไปเป็นส่วนหนึ่งของ entity bean โดยอาจทำผ่านทาง API แบบ persistent ต่าง ๆ เช่น JDBC หรือ SQL/J

4.4.2.2 Container-managed Persistent Entity Bean

Enterprise JavaBeans ช่วยให้นักพัฒนาสามารถพัฒนา entity bean ได้อย่างสะดวกมากขึ้น โดยไม่ต้องเสียเวลายุ่งเกี่ยวกับโค้ดในการทำ persistent ให้กับข้อมูล ตามข้อกำหนดของ EJB 1.1 คอนเทนเนอร์จะต้องสนับสนุนการ persistent ข้อมูลโดยอัตโนมัติสำหรับ entity bean

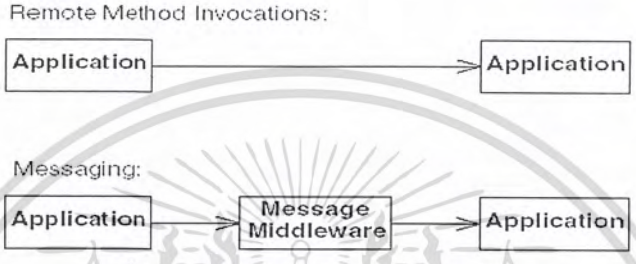
คอนเทนเนอร์จะทำฟังก์ชันในการเข้าถึงข้อมูลโดยอัตโนมัติ รวมถึงการเก็บข้อมูล การอ่านข้อมูล และการค้นหาข้อมูลของคอมโพเน็นต์ โดยที่ผู้พัฒนาไม่ต้องเขียนโค้ดในการติดต่อกับฐานข้อมูลด้วยตัวเองซึ่งจะช่วยประหยัดเวลาในขั้นตอนการพัฒนาลงไปได้มาก สิ่งที่ผู้พัฒนาต้องทำก็คือการระบุว่าต้องการให้ฟิลด์ใดบ้างที่ต้อง persistent หลังจากนั้น คอนเทนเนอร์ก็จะดูแลการ persistent ข้อมูลให้ตามที่กำหนด

ด้วยวิธีการดังกล่าว จะทำให้เราได้โค้ดที่ไม่ขึ้นอยู่กับระบบฐานข้อมูล ทำให้เราสามารถเปลี่ยนแปลงแหล่งเก็บข้อมูลได้อย่างอิสระ เนื่องจากใน Container-managed ไม่มีการเขียนโค้ดติดต่อกับ API ของระบบฐานข้อมูล

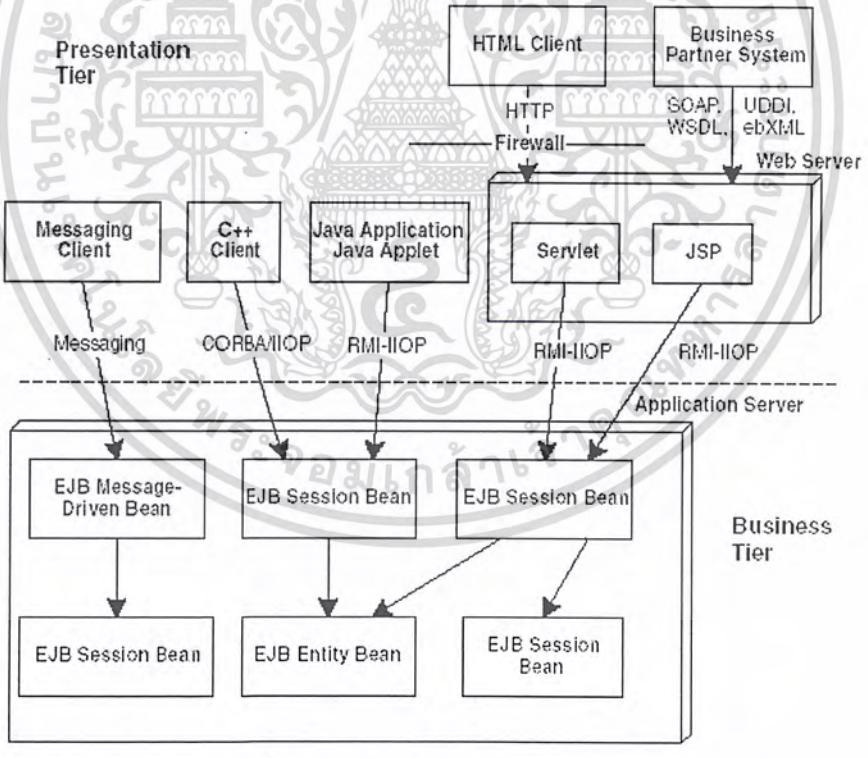
4.4.3 Message-driven bean

เป็นอีกแนวทางของรีโมตเมธอด โดยใช้เมสเซจซึ่งเป็นการติดต่อแบบ asynchronous ภายหลังจากการส่งเมสเซจออกไป ผู้สร้างเมสเซจสามารถที่จะประมวลผลงานอื่น ๆ ได้ ภายหลังที่ผู้รับเมสเซจประมวลผลเสร็จก็อาจกำหนดให้ส่งผลลัพธ์กลับมาให้กับผู้สร้างเมสเซจได้

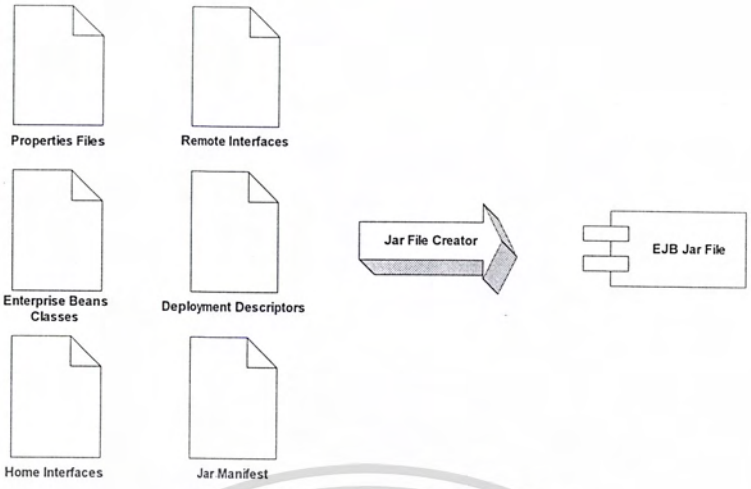
โครงสร้างพื้นฐานที่รองรับการใช้เมสเซจนี้จะเรียกว่า Message-oriented middleware (MOM) มีผลิตภัณฑ์หลายชนิดที่ใช้โครงสร้างพื้นฐานของ MOM เช่น IBM MQSeries , BEA Tuxedo/Q และ Microsoft MSMQ เป็นต้น



รูปที่ 4-8 เปรียบเทียบรีโมตเมธอดกับเมสเซจ



รูปที่ 4-9 การเรียกใช้ enterprise bean



รูปที่ 4-10 ขั้นตอนการสร้างไฟล์ Ejb-jar

หลังจากสร้างไฟล์ ejb-jar แล้ว ถือว่าเอนเตอร์ไพรส์บีนสมบูรณ์ สามารถนำไปดีพลอยในแอพพลิเคชันเซิร์ฟเวอร์ได้ เมื่อนำไปดีพลอย เครื่องมือของผู้พัฒนาคอนเทนเนอร์จะทำหน้าที่ขยายไฟล์ ejb-jar และนำข้อมูลต่าง ๆ ในไฟล์มาใช้งาน

บทที่ 5

Session Bean

session bean เป็น enterprise bean ตัวแทนงานที่ประมวลผลให้กับไคลเอ็นต์หนึ่ง ๆ มีจุดประสงค์เพื่อให้เป็นตัวแทนของบิซิเนสโพรเซส อาจเป็นงานเกี่ยวข้องกับลอจิก , อัลกอริทึม , หรือเวิร์คโฟล์ว ตัวอย่างของบิซิเนสโพรเซส เช่น การตรวจสอบบัตรเครดิต การรับรายการสั่งซื้อสินค้า การคำนวณต่าง ๆ หรือการติดตามยอดสินค้าคงคลัง บิซิเนสโพรเซสเหล่านี้สามารถแทนได้ด้วย session bean

5.1 ลักษณะของ Session Bean

5.1.1 ช่วงชีวิตของ Session Bean (Session Bean Lifetime)

ข้อแตกต่างหลักระหว่าง session bean กับ entity bean คือช่วงชีวิตของมัน session bean เป็นคอมโพเน็นต์ที่มีช่วงชีวิตสั้น โดยมีช่วงชีวิตเท่ากับ session ของไคลเอ็นต์เท่านั้น เป็นที่มาของชื่อ session bean ระยะเวลาของเซสชันของไคลเอ็นต์นั้น อาจยาวนานเท่ากับระยะเวลาที่เว็บบราวเซอร์เปิดอยู่ , ระยะเวลาที่ Applet ทำงาน , ระยะเวลาที่แอพพลิเคชันแบบ stand-alone กำลังทำงาน หรือระยะเวลาที่ bean อื่น ๆ เข้ามาเรียกใช้ bean นั้นก็ได้

โดยทั่วไประยะเวลาของ session ของไคลเอ็นต์จะเป็นตัวกำหนดช่วงชีวิตของ session bean โดย EJB คอนเทนเนอร์จะทำลาย session bean เมื่อไคลเอ็นต์จบการทำงานไป ถ้าหากแอพพลิเคชันเซิร์ฟเวอร์หรือเครื่องคอมพิวเตอร์ที่ session bean อยู่เกิดแครช session bean จะถูกทำลายไปด้วย เพราะ session bean เป็นออบเจ็กต์ที่อยู่ในหน่วยความจำ ต้องอาศัยสภาวะแวดล้อมในการทำงาน

ในทางกลับกัน entity bean มีช่วงชีวิตยาวนาน เนื่องจากเป็นออบเจ็กต์ที่ persistent ซึ่งถือเป็นส่วนหนึ่งของแหล่งบันทึกข้อมูลถาวร เช่น ฐานข้อมูล โดย entity bean ถูกสร้างขึ้นในหน่วยความจำจากข้อมูลในฐานข้อมูล

session bean ไม่มีคุณลักษณะ persistent ไม่มีการบันทึกลงในแหล่งเก็บข้อมูลเหมือนกับ entity bean สามารถทำงานเกี่ยวกับฐานข้อมูลได้ แต่ตัว session bean เองไม่ใช่ออบเจ็กต์ที่ persistent

5.1.2 Session Bean ที่มีการเก็บสถานะและไม่มีการเก็บสถานะ

session bean มี 2 ชนิดคือ stateless session bean และ stateful session bean ข้อแตกต่างระหว่าง bean สองชนิดนี้คือ stateless session bean จะไม่มีการเก็บข้อมูลสถานะของไคลเอ็นต์ไว้ ในขณะที่ stateful session bean จะมีการเก็บสถานะของไคลเอ็นต์ไว้ เช่น ในระบบอี-คอมเมิร์ช รถเข็นแบบออนไลน์จะต้องเก็บสถานะของสินค้าที่สั่งไว้ เป็นต้น

5.1.3 เมธอดทั้งหมดของ Session Bean จะต้องถูก Serialize

เมื่อเรียกเมธอดกับอินสแทนซ์ของ session bean EJB คอนเทนเนอร์จะรับประกันว่าจะไม่มีไคลเอ็นต์อื่นใช้อินสแทนซ์นั้น คอนเทนเนอร์จะป้องกันอินสแทนซ์ของ bean นั้นไว้และให้ไคลเอ็นต์ที่ทำงานอยู่พร้อม ๆ กันไปใช้อินสแทนซ์อื่นหรือคอยจนกว่าอินสแทนซ์ที่มีการใช้งานจะทำเสร็จแล้ว และถ้าหลาย ๆ ไคลเอ็นต์มีการเรียกใช้เมธอดกับ session bean พร้อม ๆ กัน การเรียกนั้นจะถูก serialize หรือทำใน lock-step ซึ่งหมายความว่าคอนเทนเนอร์จะจัดการลำดับการเรียกให้โดยอัตโนมัติ ทำให้อินสแทนซ์หนึ่งจะถูกใช้งานโดยไคลเอ็นต์เดียวเท่านั้น และเนื่องจากการร้องขอของไคลเอ็นต์จะถูก serialize ทำให้เราไม่ต้องเขียนโค้ดให้ bean มีคุณสมบัติ thread-safe จะมีเพียงเธรดเดียวของไคลเอ็นต์ที่จะทำงานกับ bean ในเวลาหนึ่ง ๆ

5.2 Stateless Session Bean

5.2.1 ลักษณะของ Stateless Session Bean

5.2.1.1 ไม่มีการเก็บสถานะ

stateless session bean ไม่เก็บสถานะการทำงานกับไคลเอ็นต์ใด ๆ ไว้ แม้จะสามารถเก็บสถานะภายในได้ แต่ไม่ได้เป็นสถานะเฉพาะสำหรับไคลเอ็นต์ตัวใดตัวหนึ่ง ในมุมมองของไคลเอ็นต์ stateless session bean ทุกตัวเหมือนกันหมด โดยไคลเอ็นต์ไม่สามารถแยกความแตกต่างได้ ในการใช้งานไคลเอ็นต์จะต้องส่งข้อมูลของไคลเอ็นต์ทั้งหมดที่ bean ต้องใช้ในการทำงานไปเป็นพารามิเตอร์ให้กับบิซิเนสเมธอด หรือ bean อาจดึงข้อมูลที่จำเป็นมาจากแหล่งภายนอก เช่นฐานข้อมูลก็ได้

5.2.1.2 สามารถกำหนดค่าเริ่มต้นให้กับ Stateless Session Bean ได้เพียงวิธีเดียว

session bean ถูกสร้างและกำหนดค่าเริ่มต้นจากเมธอด ejbCreate() แต่เนื่องจาก stateless session bean ไม่คงค่าสถานะเอาไว้ในการเรียกเมธอดแต่ละครั้ง ดังนั้นจึงไม่สามารถคงค่าใด ๆ ที่ไคลเอ็นต์ส่งเป็นพารามิเตอร์ให้กับเมธอด ejbCreate() ได้ stateless session bean ไม่จำเป็นที่จะต้องรองรับการเรียกใช้เมธอด ejbCreate() ได้หลายรูปแบบ เพราะในการเรียกใช้อินสแทนซ์ของ bean ครั้งต่อ ๆ ไป bean จะไม่มีข้อมูลของการเรียกใช้เมธอด ejbCreate() ในครั้งก่อนหน้าอยู่เลย

จากเหตุผลดังกล่าว stateless session bean จึงมีเมธอด ejbCreate() ได้เพียงตัวเดียว ที่ไม่รับค่าพารามิเตอร์ใด ๆ และใน home object จะมีเพียงเมธอด create() เข้าคู่กันซึ่งไม่รับพารามิเตอร์ใด ๆ เช่นกัน

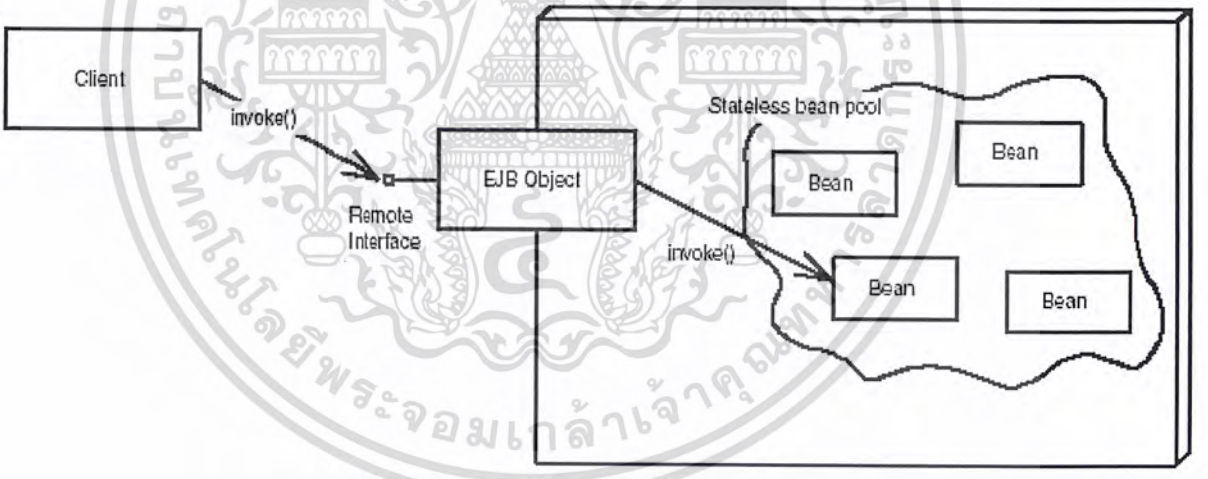
5.2.1.3 คอนเทนเนอร์สามารถพูล และนำ Stateless Session Bean กลับมาใช้ใหม่ได้

เนื่องจากเมธอด ejbCreate() ของ stateless session bean ไม่รับพารามิเตอร์ใด ๆ ไคลเอ็นต์จึงไม่มีการส่งข้อมูลเพื่อเริ่มต้นการทำงานของอินสแทนซ์ของ bean ดังนั้นคอนเทนเนอร์สามารถสร้างอินสแทนซ์ของ stateless session bean ไว้ก่อนที่ไคลเอ็นต์จะติดต่อเข้ามาได้ เมื่อไคลเอ็นต์เรียกใช้งานเมธอดคอนเทนเนอร์ก็ดึงอินสแทนซ์จากในพูลไปให้บริการกับไคลเอ็นต์นั้น แล้วนำอินสแทนซ์นั้นกลับเข้าไปเก็บไว้

ในพูลเช่นเดิมเมื่อไคลเอ็นต์เลิกใช้งาน ทำให้คอนเทนเนอร์สามารถนำอินสแทนซ์ของ bean มาให้บริการกับไคลเอ็นต์ได้อย่างทันที

ผลที่ได้คือในแต่ละครั้งที่มีการร้องขอเข้ามาจากไคลเอ็นต์ อินสแทนซ์ของ session bean ใด ๆ สามารถให้บริการก็ได้ เนื่องจาก stateless session bean เก็บสถานะการทำงานระหว่างการเรียกใช้งานเมธอดในแต่ละครั้งเท่านั้น และไม่เก็บสถานะไคลเอ็นต์เอาไว้อีกหลังจากการเรียกใช้งานเมธอดเสร็จสิ้น stateless session bean แต่ละตัวจะอยู่ในสถานะเดียวกันเสมอหลังจากการเรียกใช้งานเมธอดในแต่ละครั้ง ดังนั้น คอนเทนเนอร์สามารถนำ stateless session bean ใด ๆ ไปให้บริการกับไคลเอ็นต์ได้ในทุก ๆ การเรียกใช้เมธอดแต่ละครั้ง stateless session bean หลาย ๆ ตัวสามารถให้บริการการเรียกใช้เมธอดแต่ละครั้งจากไคลเอ็นต์เดียวกันได้ ซึ่งการอิมพลีเมนต์วิธีการเลือก stateless session bean ให้กับไคลเอ็นต์จะแตกต่างกันไปตามคอนเทนเนอร์แต่ละผลิตภัณฑ์

ประโยชน์ของการพูลอินสแทนซ์คือ พูลของ bean สามารถมีจำนวนน้อยกว่าจำนวนไคลเอ็นต์ที่ติดต่อเข้ามาได้มาก เนื่องจากเวลาที่ใช้ในการคิดของไคลเอ็นต์ อาจเป็นเวลาที่เสียไประหว่างส่งข้อมูลผ่านเครือข่ายหรือเวลาที่มนุษย์ใช้ในการตัดสินใจบนฝั่งไคลเอ็นต์ ในระหว่างนั้นคอนเทนเนอร์อาจนำอินสแทนซ์ของ bean ไปให้บริการกับไคลเอ็นต์ตัวอื่นได้เพื่อประหยัดทรัพยากรของระบบ



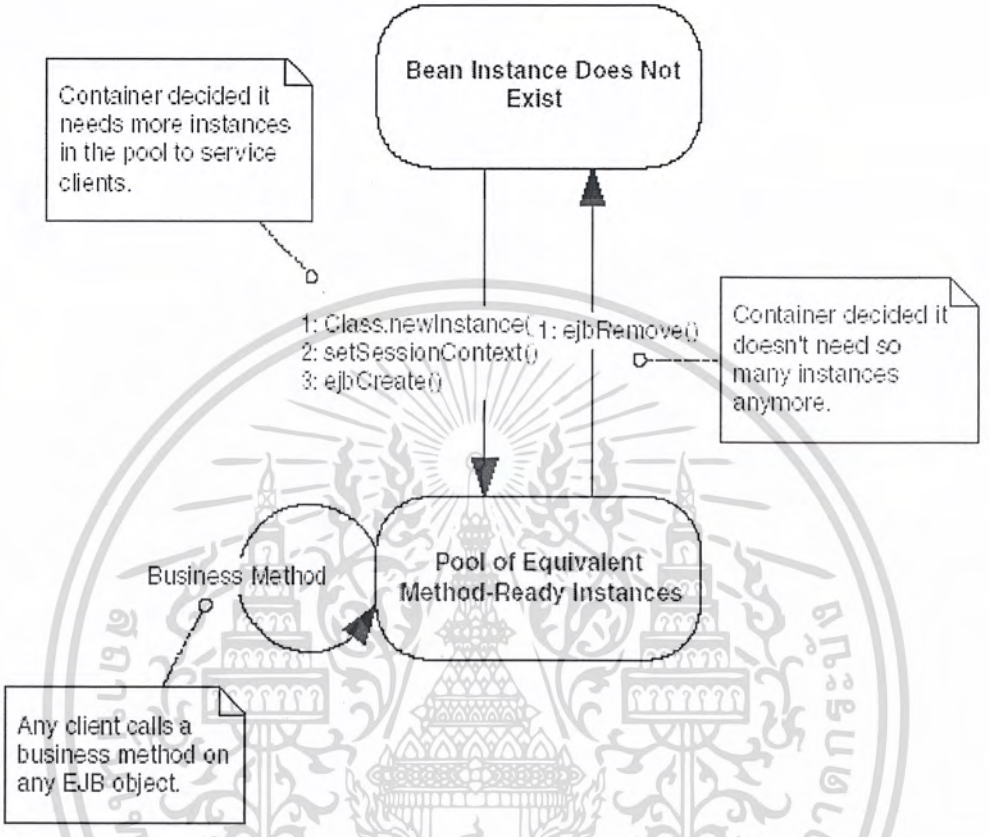
รูปที่ 5-1 การพูล Stateless Session Bean

ขนาดพูลของ bean ไม่จำเป็นต้องกำหนดไว้ตายตัว คอนเทนเนอร์ที่มีความสามารถสูงสามารถปรับขนาดของพูลได้อย่างไดนามิก โดยอัตโนมัติขณะโหลดเปลี่ยนแปลง เช่น ถ้าหากมีไคลเอ็นต์เข้ามาใช้งานตอนกลางวันมากกว่าตอนกลางคืน คอนเทนเนอร์อาจจะสร้างพูลขนาดใหญ่ในตอนกลางวัน และขนาดเล็กกว่าในตอนกลางคืน ช่วยให้ระบบมีทรัพยากรเหลือมากขึ้นสำหรับทำงานอย่างอื่นในช่วงเวลาที่ระบบมีโหลดไม่มาก

การพูลอินสแทนซ์ของ stateless session bean แสดงในรูปที่ 5-1

5.2.1.4 วงจรชีวิตของ Stateless Session Bean

วงจรชีวิตของ stateless session bean ดังรูปที่ 5-2



รูปที่ 5-2 วงจรชีวิตของ Stateless Session Bean

5.2.2 ตัวอย่างโค้ด Stateless Session Bean

ตัวอย่างของ stateless session bean นี้เป็น bean ที่ใช้ในการแปลงค่าเงิน โดยในการโค้ดจะต้องประกอบด้วยส่วนต่าง ๆ ดังนี้

5.2.2.1 Remote Interface

remote interface จะต้องมีบิซิเนสเมธอดทั้งหมดที่มีอยู่ใน bean คอนเทนเนอร์จะอิมพลีเมนต์ remote interface อิมพลีเมนต์นั้นคือ EJB object และ EJB object จะส่ง request ของไคลเอ็นต์ต่อไปยัง bean ที่แท้จริง โค้ดจะเป็นดังนี้

Converter.java

```
package olala;
import javax.ejb.*;
```

```
import java.rmi.RemoteException;
public interface Converter extends EJBObject{
  public double yenToEuro(double yen)     throws RemoteException;
  public double dollarToYen(double dollars)     throws RemoteException;
}
```

remote interface จะต้องขยายมาจาก javax.ejb.EJBObject ซึ่งหมายความว่าคอนเทนเนอร์จะสร้าง EJB object ให้ โดยจะอิมพลีเมนต์จาก remote interface และจะมีทุกเมธอดที่มีใน remote interface โดยตัวอย่างนี้มีบิซิเนสเมธอดคือ yenToEuro(double yen) และ dollarToYen(double dollars) โดยจะต้องอิมพลีเมนต์เมธอดนี้ใน enterprise bean class และเนื่องจาก remote interface ขยายมาจาก java.rmi.Remote มันจะโยน remote exception ด้วย ซึ่งเป็นข้อแตกต่างของบิซิเนสเมธอดในอินเทอร์เฟซ และ enterpise bean class

5.2.2.2 Enterprise Bean Class

bean จะต้องขยายมาจากอินเทอร์เฟซ javax.ejb.SessionBean โค้ดจะเป็นดังนี้

ConverterEJB.java

```
package olala;
import javax.ejb.*;
import java.util.*;
import javax.naming.*;
public class ConverterEJB implements javax.ejb.SessionBean{
  private SessionContext ctx;
  public ConverterEJB()  { }
  public void ejbCreate()  { }
  public void ejbActivate()  { }
  public void ejbPassivate()  { }
  public void ejbRemove()  { }
  public void setSessionContext(SessionContext ctx)  {  this.ctx = ctx; }
  public double yenToEuro(double yen)  {  return yen*0.0077; }
  public double dollarToYen(double dollars)  { return dollars*121.6000; }
}
```

5.2.2.3 Home Interface

home interface จะใช้ในการระบุกลไกในการสร้างและทำลาย EJB object โค้ดดังนี้

ConverterHome.java

```
package olala;
import javax.ejb.*;
import java.rmi.RemoteException;
public interface ConverterHome extends EJBHome{
  public Converter create()     throws CreateException, RemoteException;
}
```

home interface จะต้องขยายมาจาก javax.ejb.EJBHome EJBHome จะกำหนดวิธีในการทำลาย EJB object เอง จึงไม่ต้องเขียนเมธอดนี้ home interface จะต้องมีเมธอดที่ใช้สร้าง EJB object โดยไม่ต้องมีพารามิเตอร์ใด ๆ และเมธอด create() จะต้องโยน java.rmi.RemoteException และ javax.ejb.CreateException

5.2.2.4 Deployment Descriptor

จะต้องประกอบด้วย ejb-xml.jar และ weblogic-ejb-xml.jar โดยภายใน descriptor สามารถมีรายละเอียดของ bean ได้มากกว่า 1 bean

ejb-xml.jar

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <enterprise-beans>
    <session>
      <description></description>
      <ejb-name>Converter</ejb-name>
      <home>olala.ConverterHome</home>
      <remote>olala.Converter</remote>
      <ejb-class>olala.ConverterEJB</ejb-class>
```

```
        <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
      </session>
    </enterprise-beans>
    <assembly-descriptor>
      <container-transaction>
        <method>
          <ejb-name>Converter</ejb-name>
          <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
      </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

weblogic-ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>Converter</ejb-name>
    <jndi-name>Converter</jndi-name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

5.2.2.5 คอมไพล์และแพ็กเก็จ

คอมไพล์จาวาไฟล์ จัดไฟล์ทั้งหมดอยู่ใน path ดังนี้

./Converter/olala/Converter.class

./Converter/olala/ConverterEJB.class

./Converter/olala/ConverterHome.class

./Converter/meta-inf/ejb-jar.xml

./Converter/meta-inf/weblogic-ejb-jar.xml

สร้าง jar ไฟล์โดยใช้คำสั่งดังนี้

./Converter>jar cv0f stdConverter.jar olala meta-inf

และทำคำสั่งต่อไปนี้

./Converter/java weblogic.ejbc –compiler javac stdConverter.jar Converter.jar

แล้วจึงนำ Converter.jar ไปดีพลอยโดยดูวิธีการติดตั้ง WebLogic Server และวิธีดีพลอยได้ในภาคผนวก ก

5.2.2.6 ไคลเอ็นต์

เซต classpath ชี้ไปยัง Converter.jar หรือ path ที่มีอินเทอร์เฟซของ bean โค้ดไคลเอ็นต์ดังนี้

ConverterClient.java

```
import olala.*;
import javax.ejb.*;
import javax.naming.*;
import java.util.*;
public class ConverterClient{
public static void main(String[] args)  {
    try {
      Context ctx = getInitialContext();
      ConverterHome home = (ConverterHome) ctx.lookup("Converter");
      Converter the_ejb = home.create();
      double euro=the_ejb.yenToEuro(100);
      System.out.println("100 yen = "+euro+" euro");
      double yen=the_ejb.dollarToYen(100);
      System.out.println("100 dollar = "+yen+" yen");
       the_ejb.remove();
    } catch (Exception e) {
       e.printStackTrace();
    }
 }
 public static Context getInitialContext() throws NamingException {
   Properties p = new Properties();
   p.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
   p.put(Context.PROVIDER_URL, url);
```

```
    if (user != null) {
      p.put(Context.SECURITY_PRINCIPAL, user);
      if (password == null)  password = "";
      p.put(Context.SECURITY_CREDENTIALS, password);
    }
    return new InitialContext(p);
  }
  static String url = "t3://localhost:7001";
  static String user = null;
  static String password = null;
}
```

คอมไพล์แล้วรันจะได้ผลดังนี้

```
100 yen = 0.77 euro
100 dollar = 12160.0 yen
```

5.3 Stateful Session Bean

stateful session bean เป็น bean ที่เก็บสถานะการทำงานกับไคลเอ็นต์หนึ่ง ๆ ครอบคลุมระยะเวลาการเรียกเมธอดหลายครั้ง

5.3.1 ลักษณะของ Stateful Session Bean

5.3.1.1 การพูล Stateful Session Bean

ในสถานการณ์ที่ไคลเอ็นต์จำนวนมากกำลังทำงานกับ stateful session bean จำนวนหนึ่งในคอนเทนเนอร์ โดยปกติไคลเอ็นต์ต้องมีระยะเวลาในการคิด หรืออยู่ในสถานที่ห่างไกลออกไปมาก และระบบเครือข่ายสามารถส่งข้อมูลได้ช้า แต่หมายความว่าจะต้องมี stateful session bean จำนวนเท่า ๆ กันกับไคลเอ็นต์ในคอนเทนเนอร์ โดย stateful session bean แต่ละตัวจะเก็บสถานะการทำงานของไคลเอ็นต์แต่ละตัวเอาไว้ แต่คอนเทนเนอร์มีทรัพยากรที่จำกัด เช่น หน่วยความจำ ช่องทางติดต่อกับฐานข้อมูล (Socket Connection) หากว่าสถานะของการทำงานที่เก็บนั้นมีขนาดใหญ่ คอนเทนเนอร์ก็จะมีทรัพยากรเหลือไม่เพียงพอแก่การทำงานได้ ซึ่งปัญหานี้จะไม่เกิดกับ stateless session bean เพราะคอนเทนเนอร์สามารถพูล bean จำนวนน้อย เพื่อให้บริการกับไคลเอ็นต์จำนวนมากได้

การพูล stateful session bean ทำได้ไม่ง่าย เมื่อไคลเอ็นต์ติดต่อเข้ามายัง bean ไคลเอ็นต์นั้นจะเริ่มการทำงานกับ bean ทันที และสถานะการทำงานที่เก็บอยู่ใน bean จะต้องคงอยู่ในการเรียกใช้เมธอดครั้ง

ต่อไปของไคลเอ็นต์ตัวเดิมนั้น ดังนั้นคอนเทนเนอร์จะไม่สามารถพูล bean ขึ้นมาและนำไปให้บริการการเรียกใช้งานเมธอดต่าง ๆ แบบไดนามิกได้ เนื่องจาก bean แต่ละตัวนั้น เก็บสถานะโดยเจาะจงกับไคลเอ็นต์แต่ละตัวเอาไว้

ปัญหาดังกล่าว ใกล้เคียงกับปัญหาที่เกิดขึ้นในระบบปฏิบัติการ เมื่อเรียกใช้แอพพลิเคชันบนเครื่องคอมพิวเตอร์ จะมีหน่วยความจำแบบฟิสิคอลที่จำกัดสำหรับให้โปรแกรมทำงาน ระบบปฏิบัติการต้องทำให้แอพพลิเคชันทำงานอยู่ได้ แม้ว่าแอพพลิเคชันจะต้องการปริมาณหน่วยความจำมากกว่าหน่วยความจำจริง ๆ ทั้งหมดที่มีอยู่ โดยระบบปฏิบัติการจะใช้เนื้อที่ในฮาร์ดดิสก์เป็นส่วนเพิ่มขยายของหน่วยความจำ เป็นการเพิ่มหน่วยความจำเสมือน (virtual memory) ของระบบ เมื่อแอพพลิเคชันใด ๆ ที่ยังไม่มีการทำงาน ข้อมูลในหน่วยความจำที่มันใช้งานอยู่ก็จะถูกย้าย (swap) ออกจากหน่วยความจำลงไปยังฮาร์ดดิสก์ เมื่อแอพพลิเคชันนั้นมีการทำงานอีกครั้ง ข้อมูลที่จำเป็นก็จะถูกย้ายกลับเข้ามาในหน่วยความจำ การสลับข้อมูลไปมาจะเกิดขึ้นบ่อยครั้งเมื่อมีการสลับการทำงานไปมาระหว่างแอพพลิเคชัน (context switching)

EJB คอนเทนเนอร์ใช้วิธีแก้ปัญหาที่เกิดขึ้นกับ stateful session bean ในลักษณะเดียวกัน เพื่อจำกัดจำนวนอินสแทนซ์ของ stateful session bean ในหน่วยความจำ คอนเทนเนอร์สามารถ swap stateful session bean ออกไปได้ โดยอาจเก็บสถานะของ bean ไว้ในฮาร์ดดิสก์หรือแหล่งเก็บข้อมูลอื่น การ swap stateful session bean ออกไปเรียกว่าการทำ passivation ทำให้ได้หน่วยความจำและทรัพยากรต่าง ๆ กลับคืนมา เมื่อไคลเอ็นต์ตัวเดิมเรียกใช้เมธอดของ bean อีกครั้ง สถานะการทำงานกับไคลเอ็นต์ที่ถูก passivate ออกไปจะถูก swap กลับเข้ามาใน stateful session bean อีกครั้ง วิธีการนี้เรียกว่า activation ทำให้ bean สามารถทำงานกับไคลเอ็นต์ต่อไปได้

bean ที่ได้รับสถานะโดยการ activate เข้ามาอาจจะไม่ใช่อินสแทนซ์ของ bean ตัวเดิมที่เคยติดต่อกับไคลเอ็นต์ตัวนั้นก็ได้ ซึ่งไม่ทำให้เกิดผลใด ๆ เนื่องจากอินสแทนซ์ของ bean ตัวใหม่นั้น จะทำงานต่อไปจากจุดเดิมที่สถานะของการทำงานได้ถูก passivate เอาไว้

ดังนั้นอินสแทนซ์จำนวนจำกัดเท่านั้นที่จะอยู่ในหน่วยความจำ ขณะไคลเอ็นต์จำนวนมากมายติดต่อเข้ามา แต่การพูลวิธีนี้ก็มีข้อเสีย เพราะการทำ passivation และ activation ต้องมีการใช้ I/O ด้วย อาจทำให้เกิดกรณีคอขวด (bottleneck) ขึ้นได้ ซึ่งแตกต่างจาก stateless session bean ซึ่งสามารถทำพูลได้ง่าย เนื่องจากไม่ต้องเก็บค่าสถานะการทำงานเอาไว้

การทำ passivation อาจเกิดขึ้นได้ทุกขณะ เมื่อ bean นั้นไม่ได้เกี่ยวข้องกับการเรียกใช้งานเมธอดในขณะนั้น การตัดสินใจว่าจะ passivate bean เมื่อไหร่ขึ้นอยู่กับคอนเทนเนอร์ โดยมีข้อยกเว้นอยู่ว่า bean ซึ่งทำงานอยู่ในทรานแซ็กชัน จะไม่สามารถ passivate ได้จนกว่าทรานแซ็กชันนั้นจะเสร็จสมบูรณ์

ส่วนการactivation bean จะเกิดขึ้นเมื่อ bean นั้นได้รับการร้องขอเข้ามา หากไคลเอ็นต์ใด ๆ ทำการร้องขอเข้ามา แต่สถานะการทำงานของมันถูก passivate อยู่ คอนเทนเนอร์จะ activate bean และอ่านข้อมูลที่ถูก passivate อยู่กลับเข้ามาในหน่วยความจำ

โดยทั่วไป การทำ passivation และ activation ไม่มีประโยชน์สำหรับ stateless session bean เนื่องจากไม่มีสถานะที่จะทำการ passivate และ activate และการทำ passivation และ activation นั้นจะถูกนำไปใช้กับ entity bean ด้วย จะได้กล่าวถึงต่อไป

5.3.1.2 ข้อบังคับในการเก็บสถานะของไคลเอ็นต์

สถานะการทำงานของ bean เป็นไปตามกฎที่วางไว้โดย java object Serialization เมื่อคอนเทนเนอร์ passivate bean จะใช้ object serialization ในการแปลงสถานะการทำงานของ bean ไปเป็นบิตข้อมูล หลังจากนั้นจึงเขียนบิตข้อมูลเหล่านี้ลงไปยังแหล่งเก็บข้อมูล เมื่อ bean ถูกเขียนลงไปยังแหล่งเก็บข้อมูลแล้ว หน่วยความจำที่มันใช้อยู่ก็จะถูกปล่อยโดย garbage collector ได้ ส่วนการ activate ก็เป็นขั้นตอนที่ตรงข้ามกัน โดยบิตข้อมูลที่ถูก serialize ที่เก็บอยู่ในแหล่งเก็บข้อมูลจะถูกอ่านกลับขึ้นมาในหน่วยความจำและแปลงไปเป็นข้อมูลของ bean ซึ่งอยู่ในหน่วยความจำ สิ่งที่ทำให้วิธีการเหล่านี้สามารถทำได้ คือ การที่อินเทอร์เฟซ javax.ejb.EnterpriseBean สืบทอดมาจาก java.io.Serializable และทุก ๆ enterprise bean อิมพลีเมนต์อินเทอร์เฟซนี้

สำหรับจาวาออบเจ็กต์ที่เป็นส่วนหนึ่งของสถานะของ bean ก็ใช้หลักการเดียวกันคือออบเจ็กต์นั้นจะต้องอิมพลีเมนต์ java.io.Serializable ด้วย แม้ว่า bean จะต้องเป็นไปตามกฎของ object serialization แต่ EJB คอนเทนเนอร์ไม่จำเป็นต้องใช้โพรโตคอลมาตรฐานในการทำ serialization เสมอไป อาจใช้โพรโตคอลของตัวเองซึ่งสามารถช่วยเพิ่มความยืดหยุ่นและความแตกต่างระหว่างคอนเทนเนอร์ของแต่ละผู้ผลิต

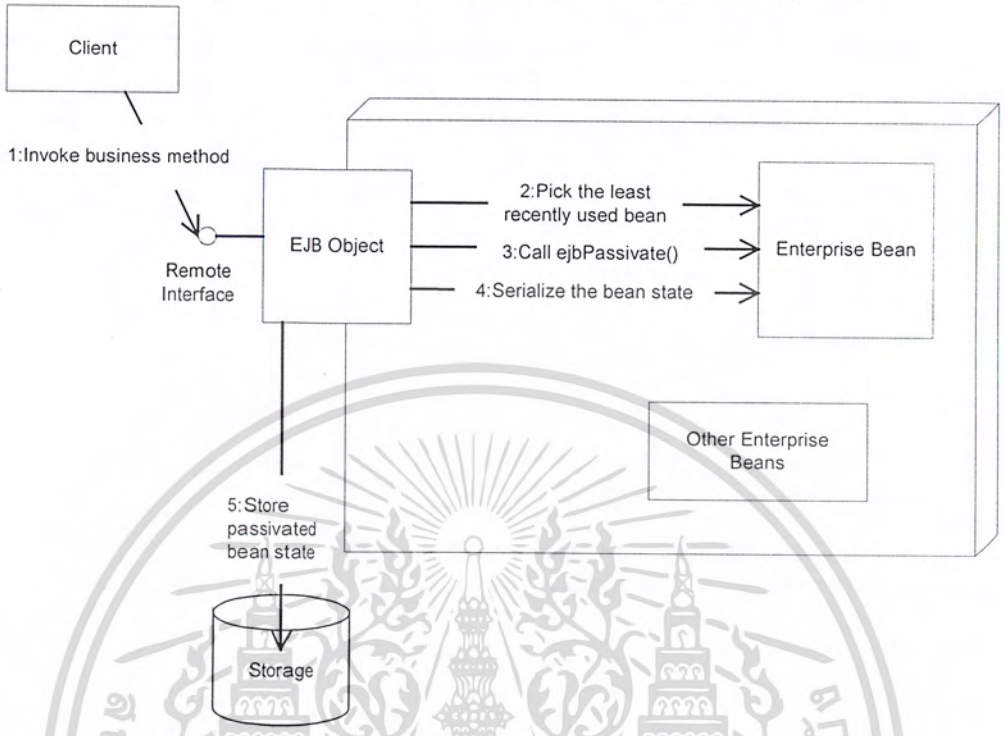
5.3.1.3 ขั้นตอนการทำ Passivation และ Activation

เมื่อ EJB คอนเทนเนอร์ passivate bean คอนเทนเนอร์จะเก็บสถานะการทำงานของ bean นั้นลงไปในแหล่งเก็บข้อมูล เช่นไฟล์ หรือฐานข้อมูล โดยคอนเทนเนอร์จะบอกให้ bean ทราบว่า มันกำลังจะทำการ passivate ด้วยการเรียกใช้เมธอด ejbPassivate() เมธอดนี้เป็นการเตือน bean ว่า สถานะการทำงานของมันกำลังจะถูก swap ออกไป

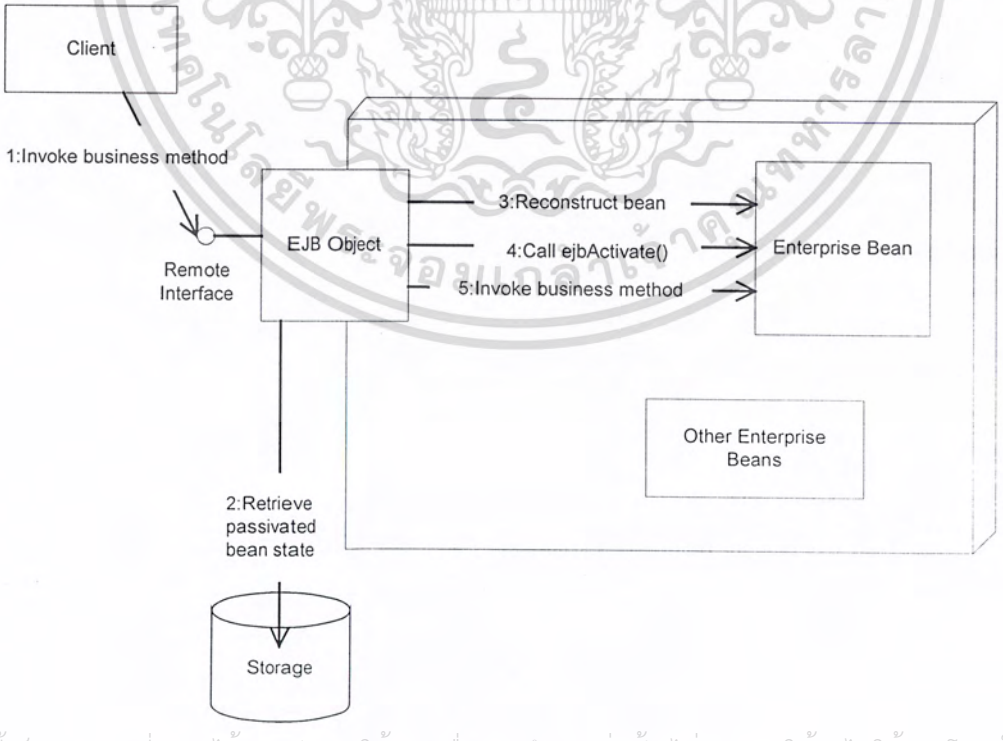
การที่คอนเทนเนอร์บอกให้ bean ทราบด้วยการเรียกใช้เมธอด ejbPassivate() นั้นเป็นสิ่งจำเป็น ทำให้ bean สามารถปล่อยทรัพยากรที่มันกำลังถือครองอยู่ได้ ทรัพยากรในที่นี้อาจเป็นช่องทางการติดต่อฐานข้อมูล , ซ็อกเก็ต , ไฟล์ที่เปิดอยู่ หรือทรัพยากรอื่น ๆ ซึ่งไม่สามารถเก็บลงในดิสก์ได้ หรือไม่มีความจำเป็นที่จะต้องเก็บไว้ เมื่อการเรียกเมธอด ejbPassivate() ของ bean เสร็จสมบูรณ์ bean ก็จะอยู่ในสถานะพร้อมถูก passivate การทำ passivation แสดงในรูปที่ 5-3

ขั้นตอนตรงกันข้ามจะเกิดขึ้นในกรณีของการ activate bean ซึ่งสถานะของการทำงานที่ถูก serialize จะถูกอ่านกลับเข้ามาในหน่วยความจำ และคอนเทนเนอร์จะสร้างสถานะขึ้นมาใหม่ในหน่วยความจำ โดยใช้ Object Serialization หรือสิ่งที่เหมือนกัน (equivalent) ในการแปลง หลังจากนั้นคอนเทนเนอร์จะเรียก

เมธอด ejbActivate() ทำให้ bean สามารถเรียกใช้ทรัพยากรต่าง ๆ ที่จำเป็น ซึ่งถูกปล่อยไปขณะถูก passivate ด้วยเมธอด ejbPassivate() กลับมา ขั้นตอนการทำ activation แสดงในรูปที่ 5-4



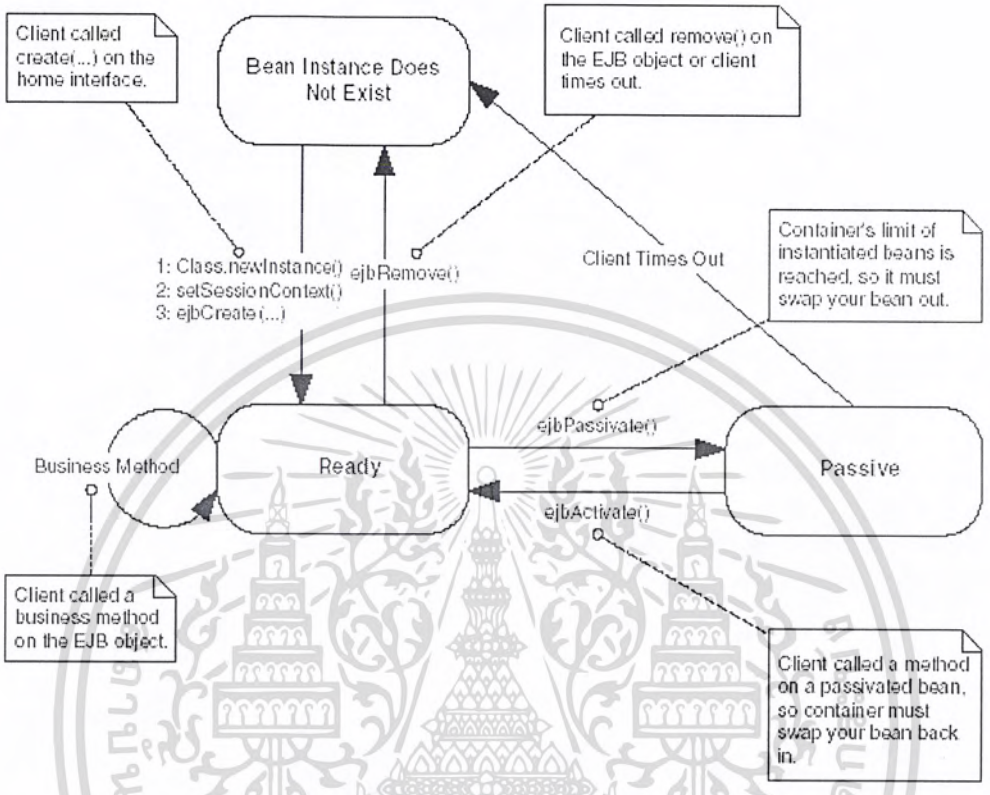
รูปที่ 5-3 การ Passivate Stateful Session Bean



รูปที่ 5-4 การ Activate Stateful Session Bean

5.3.1.4 วงจรชีวิตของ Stateful Session Bean

วงจรชีวิตของ stateful session bean ดังรูปที่ 5-5



รูปที่ 5-5 วงจรชีวิตของ Stateful Session Bean

5.3.2 ตัวอย่างโค้ด Stateful Session Bean

ตัวอย่างของ bean คือ bean ที่ใช้นับ

5.3.2.1 Remote Interface

remote interface จะมีเพียงเมธอดเดียวคือ count() ซึ่งจะต้องอิมพลีเมนต์ใน bean โค้ดดังนี้

Count.java

```
package olala;

import javax.ejb.*;

import java.rmi.RemoteException;

public interface Count extends EJBObject {

   public int count()
      throws RemoteException;
```

```
}
```

5.3.2.2 Enterprise Bean Class

โค้ดดังนี้

CountEJB.java

```
package olala;
import javax.ejb.*;
import java.util.*;
import javax.naming.*;
public class CountEJB implements javax.ejb.SessionBean{
  private SessionContext ctx;
  private int val;
  public CountEJB()  { }
  public void ejbActivate()  { }
  public void ejbPassivate()  { }
  public void ejbRemove()  { }
  public void setSessionContext(SessionContext ctx) {  this.ctx = ctx;  }
  public void ejbCreate(int val) {  this.val=val;  }
  public int count() {  return ++val; }
}
```

bean จะต้องอิมพลีเมนต์ javax.ejb.SessionBean ซึ่งหมายความว่า bean ต้องมีทุกเมธอดที่มีในอินเทอร์เฟซ SessionBean เมธอด ejbCreate() จะมีพารามิเตอร์ที่ใช้ในการกำหนดค่าเริ่มต้นให้กับ bean ซึ่งแตกต่างจาก stateless ที่จะมีเมธอด ejbCreate() โดยไม่มีพารามิเตอร์เมธอดเดียวเท่านั้น

5.3.2.3 Home Interface

โค้ดดังนี้

CountHome.java

```
package olala;
import javax.ejb.*;
import java.rmi.RemoteException;
public interface CountHome extends EJBHome {
  public Count create(int val)      throws CreateException, RemoteException;
```

```
}
```

5.3.2.4 Deployment Descriptor

จะประกอบด้วย descriptor 2 ไฟล์เหมือนกับ stateless session bean ดังนี้

ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <enterprise-beans>
    <session>
      <description></description>
      <ejb-name>Count</ejb-name>
      <home>olala.CountHome</home>
      <remote>olala.Count</remote>
      <ejb-class>olala.CountEJB</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>Count</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

weblogic-ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
      <ejb-name>Count</ejb-name>
      <caching-descriptor>
         <max-beans-in-cache>5</max-beans-in-cache>
         <idle-timeout-seconds>10</idle-timeout-seconds>
      </caching-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>
      <jndi-name>Count</jndi-name>
   </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

5.3.2.5 คอมไพล์และแพ็กเก็จ

คอมไพล์จาวาไฟล์ จัดไฟล์ทั้งหมดอยู่ใน path ดังนี้

./Count/olala/Count.class

./Count/olala/CountEJB.class

./Count/olala/CountHome.class

./Count/meta-inf/ejb-jar.xml

./Count/meta-inf/weblogic-ejb-jar.xml

สร้าง jar ไฟล์โดยใช้คำสั่งดังนี้

/Count>jar cv0f stdCount.jar olala meta-inf

และทำคำสั่งต่อไปนี้

./Count/java weblogic.ejbc –compiler javac stdCount.jar Count.jar

แล้วจึงนำ Count.jar ไปดีพลอย

5.3.2.6 ไคลเอ็นต์

เซต classpath ชี้ไปยัง Count.jar หรือ path ที่มีอินเทอร์เฟซของ bean โค้ดไคลเอ็นต์ดังนี้

CountClient.java

```
import olala.*;
```

```
import javax.ejb.*;
import javax.naming.*;
import java.util.*;
public class CountClient{
  public static void main(String[] args) {
    try {
      Context ctx = getInitialContext();
      CountHome home = (CountHome) ctx.lookup("Count");
      Count count[] = new Count[3];
      int countVal=0;
      System.out.println("Instantiating beans...");
      for (int i=0;i<3;i++) {
        count[i]=home.create(countVal);
        countVal=count[i].count();
       System.out.println(countVal);
        Thread.sleep(500);
      }
      System.out.println("Calling count() on beans...");
      for (int i=0;i<3;i++) {
        countVal=count[i].count();
        System.out.println(countVal);
        Thread.sleep(500);
      }
      for (int i=0;i<3;i++) { count[i].remove(); }
    }
    catch (Exception e) {  e.printStackTrace();  }
  }
  public static Context getInitialContext() throws NamingException{
    Properties p = new Properties();
    p.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
    p.put(Context.PROVIDER_URL, url);
    if (user != null) {
      p.put(Context.SECURITY_PRINCIPAL, user);
```

```
        if (password == null)
            password = "";
        p.put(Context.SECURITY_CREDENTIALS, password);
    }
    return new InitialContext(p);
  }
  static String url = "t3://localhost:7001";
  static String user = null;
  static String password = null;
}
```

คอมไพล์แล้วรันจะได้ผลดังนี้

```
Instantiating beans...
1
2
3
Calling count() on beans...
2
3
4
```

บทที่ 6

Entity Bean

6.1 แนวคิด Persistent concept

6.1.1 Java Object Serialization

ในการทำงานกับออบเจ็กต์ที่เขียนด้วยภาษาจาวา อาจต้องการเก็บสถานะ (state) ของออบเจ็กต์ลงใน permanent storage วิธีหนึ่งที่ทำได้คือใช้วิธีการ object serialization ซึ่งเป็นการแปลงออบเจ็กต์ให้อยู่ในรูป สตรีมของไบต์ข้อมูล (byte stream) หลังจากนั้น สามารถจะทำอะไรกับไบต์สตรีมนี้ก็ได้ เช่น ส่งข้ามระบบเครือข่าย หรือเก็บลง storage เช่น ไฟล์ , ฐานข้อมูล , JNDI ทรี

เมื่อต้องการเรียกกลับขึ้นมาใช้งานใหม่ จะต้องทำในกระบวนการย้อนกลับ คือ อ่านไบต์สตรีมขึ้นมาแล้วนำไปสร้างเป็นออบเจ็กต์ใหม่อีกครั้ง อย่างไรก็ตาม วิธีนี้ไม่เหมาะในงานที่มีการ query บ่อย เนื่องจาก operation ทุกอย่างทำกับออบเจ็กต์ จึงต้องโหลดทุกออบเจ็กต์กลับขึ้นมาเพื่อดูว่าเงื่อนไขตรงกับที่ query หรือไม่

6.1.2 Object-Relational Mapping

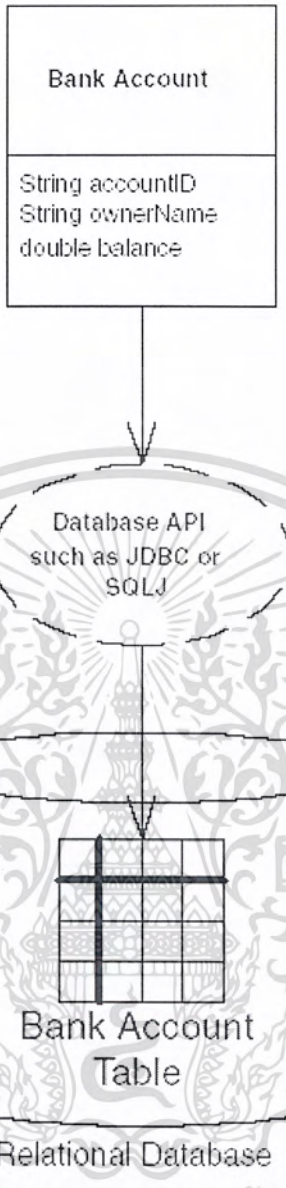
อีกวิธีหนึ่งที่ได้รับความนิยมในการเก็บจาวาออบเจ็กต์ก็คือ การใช้ฐานข้อมูลเชิงสัมพันธ์ (relational database) เช่น Oracle หรือ Microsoft SQL Server แทนที่จะทำการ serialize ออบเจ็กต์ การบันทึกจาวาออบเจ็กต์ใช้ JDBC หรือ SQL/J เพื่อแมปข้อมูลของออบเจ็กต์ไปยังฐานข้อมูลเชิงสัมพันธ์ นอกจากนี้ยังสามารถบันทึกชื่อของจาวาคลาสที่สัมพันธ์กับข้อมูลนี้ เพื่อที่จะสร้าง (instantiate) คลาสที่ถูกต้องตามออบเจ็กต์ที่ต้องการอ่านกลับขึ้นมาในหน่วยความจำ ในการโหลดออบเจ็กต์จากฐานข้อมูล จะต้องสร้างออบเจ็กต์จากคลาสนั้น อ่านข้อมูลขึ้นมาจากฐานข้อมูล และเก็บค่าเหล่านั้นกับฟิลด์ของอินสแทนซ์ของออบเจ็กต์ที่อยู่ในหน่วยความจำ ดังแสดงในรูปที่ 6-1 และ 6-2

6.1.3 ฐานข้อมูลเชิงวัตถุ (Object Database)

Object Database Management System (ODBMS) สามารถเก็บออบเจ็กต์ไว้ในฐานข้อมูลได้โดยไม่ต้องผ่านกระบวนการ O/R mapping และทำให้การเขียนโค้ดเพื่อเข้าถึงข้อมูลทำได้ง่ายขึ้น โดยการเขียนโปรแกรมติดต่อกับ API ของฐานข้อมูลเชิงวัตถุแทนการเขียนโค้ดติดต่อกับ API ของฐานข้อมูลเชิงสัมพันธ์

ฐานข้อมูลเชิงวัตถุส่วนใหญ่ จะให้เครื่องมือในการ query ข้อมูลที่เรียกว่า Object Query Language (OQL) ซึ่งเป็นการ query ในระดับสูงกว่าฐานข้อมูลเชิงสัมพันธ์

นอกจากนี้ ระบบฐานข้อมูลเชิงวัตถุยังมีความถูกต้อง (integrity) และความปลอดภัย (security) สูงกว่าระบบฐานข้อมูลเชิงสัมพันธ์ ในกรณี persistent object มีความซับซ้อนมากๆ เช่น การเก็บฐานข้อมูลของ



รูปที่ 6-1 Object-Relational Mapping

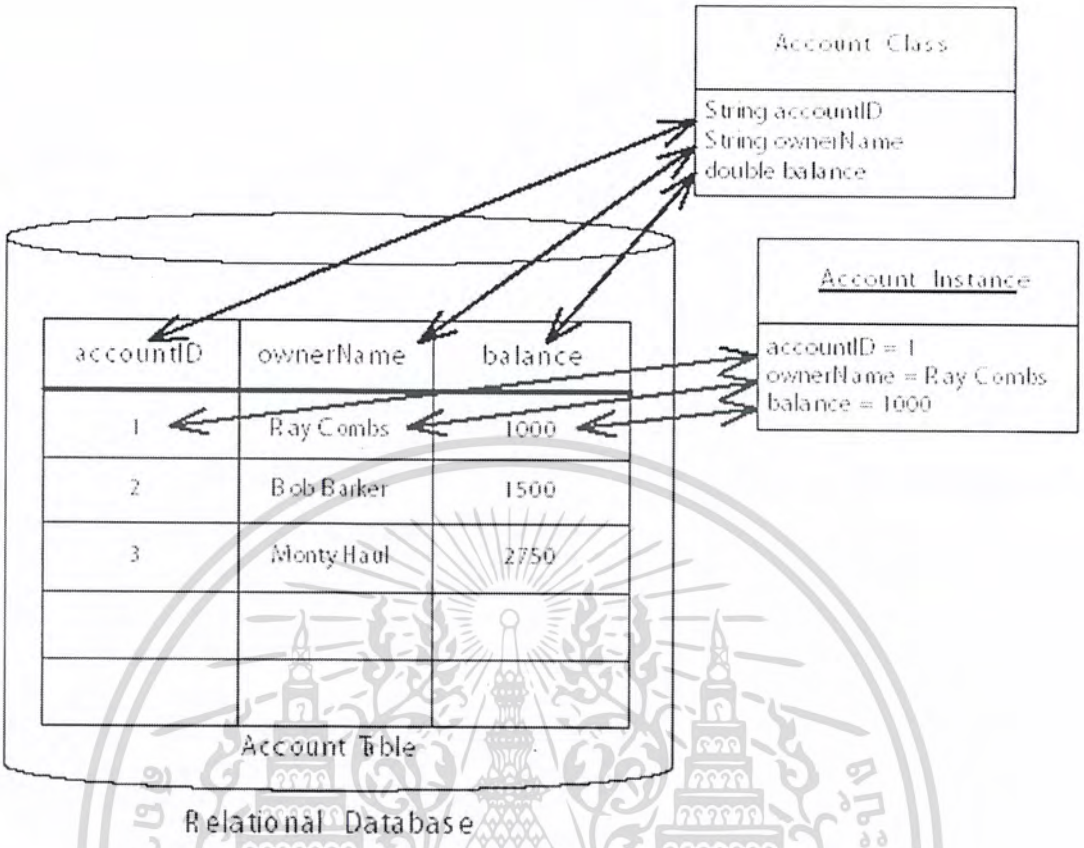
CAD/CAM แต่ถ้าออบเจ็กต์ไม่ซับซ้อนและสามารถแมปไปเป็นฐานข้อมูลเชิงสัมพันธ์ได้ง่าย เช่น แอพพลิเคชันทางธุรกิจต่าง ๆ ซึ่งเน้นที่ปริมาณงานจำนวนมากแล้ว ฐานข้อมูลเชิงสัมพันธ์จะมีประสิทธิภาพที่สูงกว่า

6.2 แนวคิดของ Entity Bean

6.2.1 อินสแทนซ์ของ Entity Bean

คอมโพเน็นต์ที่ถูกดีพลอยแบ่งแยกความแตกต่างได้ดังนี้

Application Logic Components - เป็นคอมโพเน็นต์ที่ให้บริการสำหรับงานทั่ว ๆ ไปเช่น การคำนวณราคาคำสั่งซื้อ



รูปที่ 6-2 ตัวอย่างการแมประหว่างออบเจ็กต์และฐานข้อมูลเชิงสัมพันธ์

การออกบิลบัตรเครดิตของลูกค้า
การหาอินเวิร์สของเมตริก
โดยส่วนใหญ่คอมโพเน็นต์เหล่านี้จะแทนการกระทำ เหมาะกับการใช้กับบิซิเนสโพรเซส

Persistent Data Components - เป็นออบเจ็กต์ที่รู้ถึงวิธีการแปลงตัวมันเองลงใน persistent storage โดยใช้กลไกต่าง ๆ เช่น serialization , O/R mapping ไปเป็นฐานข้อมูลเชิงสัมพันธ์หรือฐานข้อมูลเชิงวัตถุ ออบเจ็กต์ชนิดนี้เป็นตัวแทนของข้อมูล (data) เช่น
ข้อมูลในบัญชีธนาคาร เช่น หมายเลขบัญชี และ ยอดคงเหลือ
ข้อมูลของฝ่ายบุคคล เช่น ชื่อ แผนก เงินเดือน
เรียกได้ว่าคอมโพเน็นต์ชนิดนี้ใช้แสดงถึงสิ่งที่เป็นคำนาม เช่น คน สถานที่ สิ่งของ

การมองข้อมูลเหล่านี้เป็นออบเจ็กต์ แทนที่จะจัดการในระดับข้อมูลดิบ (raw data) นั้น ทำให้จัดการได้ง่ายขึ้น สามารถรวมข้อมูลที่สัมพันธ์กันให้อยู่ในออบเจ็กต์เดียว สามารถใช้เมธอดเพื่อจัดการกับกลุ่มข้อมูลเหล่านี้ และได้รับบริการจากมิดเดิลแวร์ เช่น ทรานแซกชัน , การเข้าถึงระบบเครือข่าย , และการรักษาความปลอดภัย

entity bean เป็นคอมโพเน็นต์ประเภท persistent data component ที่รู้วิธีการแปลงตัวมันเองลงในแหล่งเก็บข้อมูล เช่น ฐานข้อมูล โดยเก็บอยู่ในรูปแบบของฟิลด์ (field) หลาย ๆ ฟิลด์ เช่น ฟิลด์ของหมายเลขบัญชี และฟิลด์ของยอดคงเหลือ ซึ่งจะมีเมธอดที่สัมพันธ์กันอยู่ภายในคอมโพเน็นต์ด้วย เช่น getBankAccountNumber() และ getBankAccountBalance()

entity bean อาจเปรียบเสมือน serializable java object ที่สามารถเก็บตัวเองในรูปกลุ่มของบิตลงในแหล่งบันทึกข้อมูลที่ persistent entity bean สามารถทำ persistent ได้หลายวิธี เช่น serialization , O/R mapping และ object database persistence เนื่องจากข้อกำหนดของ Enterprise JavaBeans ไม่ได้บังคับวิธีการทำ persistent ไว้

ข้อแตกต่างระหว่าง entity bean กับ session bean คือ session bean แสดงถึงกระบวนการทำงานที่เกิดขึ้นโดยผู้ใช้ และจบลงเมื่อผู้ใช้จบการทำงานไป แต่ entity bean จะเก็บตัวข้อมูลจริง ๆ เช่น ข้อมูลของสินค้า , บัญชีธนาคาร , คำสั่งซื้อ หรือ ข้อมูลลูกค้า entity bean ไม่ทำงานที่ซับซ้อนมาก เช่น การออกใบเสร็จให้ลูกค้า แต่ตัวมันจะแทนลูกค้าเองเลย และ entity bean เป็นออบเจ็กต์ที่มีสถานะคงทน (persistent state object) แม้ว่าผู้ใช้จะจบการทำงานไปแล้วก็ตาม

อินสแทนซ์ของ entity bean จะมีลักษณะดังนี้

- เป็นสิ่งที่แทนข้อมูลที่ persistent ในลักษณะของภาษาจาวาที่อยู่ในหน่วยความจำ
- รู้วิธีอ่านค่าตัวเองเข้ามาจากแหล่งบันทึกข้อมูล และสามารถแมปข้อมูลจากในฐานข้อมูลเข้ากับแต่ละฟิลด์ของออบเจ็กต์ได้
- สามารถแก้ไขค่าได้ในหน่วยความจำ เพื่อที่จะเปลี่ยนแปลงค่าข้อมูลจริง
- มีความคงทน สามารถเก็บลงในแหล่งบันทึกข้อมูลอีกครั้งได้เพื่อเป็นการอัพเดตข้อมูล

อินสแทนซ์ของ entity bean คือมุมมองที่มองลงไปยังข้อมูลที่อยู่ในหน่วยความจำเป็นอินสแทนซ์ของ entity bean class

ข้อมูลของ entity bean (entity bean data หรือ data instance) คือกลุ่มของข้อมูลในฐานข้อมูลในระดับกายภาพ เช่น ระเบียนของบัญชีธนาคาร

6.2.2 ส่วนประกอบของ Entity Bean

6.2.2.1 Entity Bean Class

คือคลาสในภาษาจาวาที่จำลองข้อมูลที่ persistent โดย entity bean class จะแมปเข้ากับหนึ่ง entity definition ใน database schema เช่น entity bean class หนึ่งสามารถแมปเข้าได้กับ definition ของตารางเชิงสัมพันธ์ ในกรณีนี้อินสแทนซ์ของ entity bean ของคลาสนั้นจะถูกแมปเข้ากับแถวหนึ่งในตาราง entity bean class สามารถมีเมธอดพื้นฐานเพื่อใช้จัดการหรือเข้าถึงข้อมูล เช่น เมธอดสำหรับลดยอดเงินในบัญชีธนาคาร เป็นต้น เช่นเดียวกับ session bean class EJB กำหนดเมธอดบังคับสำหรับ entity bean class ซึ่งคอนเทนเนอร์จะเรียกใช้เมธอดเหล่านี้โดยอัตโนมัติ

6.2.2.2 Remote Interface

เป็นอินเทอร์เฟซอ้างอิงไปยัง bean ซึ่งไคลเอ็นต์จะเข้ามาเรียกใช้ ภายใน remote interface จะต้องระบุถึงบิซิเนสเมธอดทั้งหมดที่ entity bean มีให้ใช้งาน ผู้พัฒนา EJB คอนเทนเนอร์จะให้เครื่องมือสำหรับอิมพลีเมนท์ remote interface มาให้ โดยส่วนอิมพลีเมนต์ของ remote interface ก็คือ EJB ออบเจ็กต์ ซึ่งทำหน้าที่เป็นชั้นกั้นกลางระหว่างไคลเอ็นต์กับ bean ไคลเอ็นต์จะร้องขอการใช้งาน bean มายัง EJB ออบเจ็กต์แทนที่จะร้องขอกับ bean โดยตรง

เนื่องจาก EJB ออบเจ็กต์เป็นส่วนหนึ่งของคอนเทนเนอร์ มันจะประกอบด้วยลอจิกในการดักจับการเรียกใช้เมธอด และจัดการอินสแทนซ์ของ bean ซึ่งเป็นหลักการเดียวกับ session bean

6.2.2.3 Home Interface

เป็นอินเทอร์เฟซที่ไคลเอ็นต์จะต้องเรียกใช้เพื่อสร้าง , ค้นหา , และทำลาย EJB ออบเจ็กต์ของ entity bean ภายใน home interface ต้องระบุเมธอดบอกถึงวิธีที่มีทั้งหมดในการสร้าง EJB ออบเจ็กต์ใหม่ขึ้นมา , การค้นหาและการทำลาย EJB ออบเจ็กต์เดิมที่มีอยู่แล้ว ผู้พัฒนา EJB คอนเทนเนอร์ จะให้เครื่องมือสำหรับ อิมพลีเมนต์ home interface นี้มาด้วย ส่วนอิมพลีเมนต์ของ home interface ก็คือ home object ซึ่งเปรียบเสมือนเป็นโรงงานที่ใช้ในการสร้าง , ค้นหา และทำลาย EJB ออบเจ็กต์ ในการค้นหา home object จะต้องเรียกใช้บริการ look up ของ JNDI เช่นเดียวกับหลักการของ session bean

6.2.2.4 Primary Key Class

เป็น unique identifier สำหรับ entity bean โดย primary key จะแยกความแตกต่างระหว่างแต่ละ entity bean เช่น ถ้ามี entity bean ประกอบด้วยบัญชีธนาคารทั้งหมดหนึ่งล้านบัญชี แต่ละบัญชีต้องมี unique ID (เช่น หมายเลขบัญชีธนาคาร) ซึ่งจะต้องไม่ซ้ำกันเลยในบัญชีทั้งหมด primary key เป็นออบเจ็กต์ที่สามารถประกอบด้วยหลาย attribute ได้ อาจเป็นข้อมูลที่จำเป็นในการแยกแยะอินสแทนซ์ของ entity bean ในกรณีที่ entity bean เป็นตัวแทนของความสัมพันธ์ที่ซับซ้อน primary key อาจเป็นทั้งออบเจ็กต์เลยก็ได้ EJB ให้ความยืดหยุ่นในการระบุว่าจะใช้อะไรเป็น unique ID โดยการรวม primary key class เข้ากับ entity bean มีกฎอยู่เพียงข้อเดียวว่า primary key class ต้องสามารถ serialize ได้ และเป็นไปตามกฎของ java object serialization

6.2.2.5 Deployment Descriptor

เก็บรายการคุณสมบัติ (properties) ต่าง ๆ ที่คอนเทนเนอร์ใช้ในการดีพลอย deployment descriptor ใช้แจ้งรายละเอียดเกี่ยวกับ bean ให้กับคอนเทนเนอร์ เมื่อจะนำ entity bean ไปใช้งาน ต้องรวมไฟล์เหล่านี้เข้าไว้ในไฟล์ ejb-jar รวมทั้ง manifest file เพื่อใช้ค้นหาตำแหน่งของ bean ในไฟล์ ejb-jar ด้วย

6.2.3 ลักษณะของ Entity Bean

- **Entity Bean มีช่วงชีวิตยาวนาน**

session bean มีอายุเท่ากับ session ของไคลเอ็นต์เท่านั้น เมื่อไคลเอ็นต์จบการติดต่อก็สามารถทำลาย session bean ได้ทันที ส่วน entity bean จะมีช่วงชีวิตยาวนานเท่ากับคอนเทนเนอร์ โดยเริ่มและจบการทำงานพร้อมกับ คอนเทนเนอร์ เช่น ข้อมูลบัญชีธนาคาร

- **Entity Bean มีความคงทนต่อความผิดพลาด**

session bean มีช่วงชีวิตที่สั้น เพียงแค่ session ของไคลเอ็นต์เท่านั้น และเมื่อมีเหตุการณ์ผิดพลาด เช่น JVM แครชเกิดขึ้น session bean จะถูกทำลายทันที แต่ entity bean เป็นส่วนหนึ่งของ persistent storage ดังนั้นการแครชของ JVM หรือฐานข้อมูล จะไม่มีผลกระทบต่อ entity bean เมื่อระบบกลับมาทำงานได้อีกครั้ง อินสแทนซ์ของ entity bean ก็สามารถสร้างขึ้นมาใหม่ได้ โดยการอ่านข้อมูลจากฐานข้อมูล และนำมาสร้างเป็นอินสแทนซ์ของ entity bean เพื่อเป็นตัวแทนของข้อมูลนั้นในหน่วยความจำได้

- **อินสแทนซ์ของ Entity Bean เป็นมุมมองไปยังฐานข้อมูล**

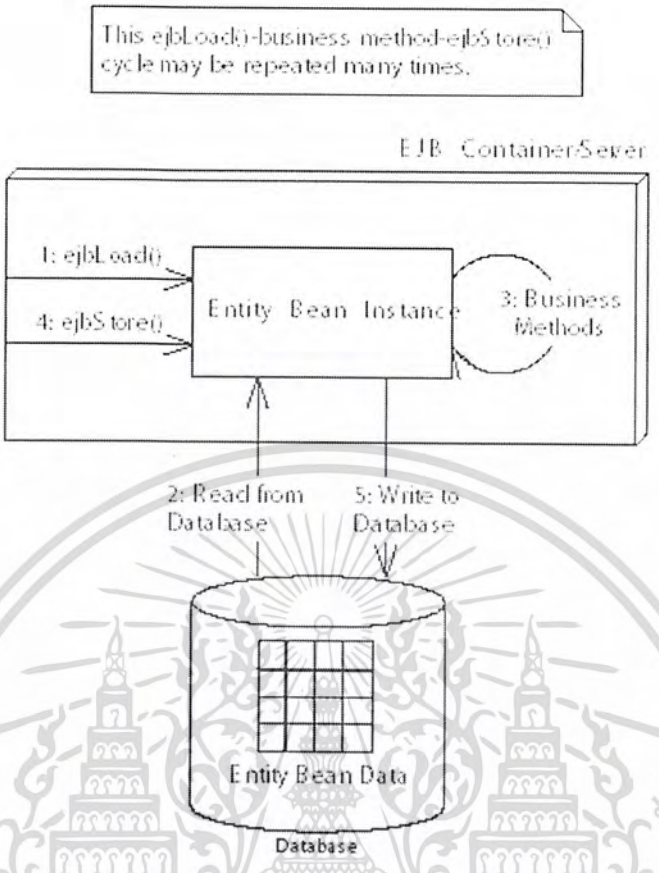
ในการโหลด entity bean เข้ามาเป็นอินสแทนซ์ในหน่วยความจำ จะต้องอ่านข้อมูลที่เก็บอยู่ในฐานข้อมูลแล้วนำมาจัดการใน java virtual machine โดยมองว่าออบเจ็กต์ที่อยู่ในหน่วยความจำและฐานข้อมูลนั้นเป็นสิ่งเดียวกัน หมายความว่า เมื่อแก้ไขอินสแทนซ์ของ entity bean ซึ่งอยู่ในหน่วยความจำ ข้อมูลในฐานข้อมูลจะต้องถูกเปลี่ยนแปลงด้วยโดยอัตโนมัติ entity bean ที่อยู่ในหน่วยความจำนั้น เสมือนเป็นมุมมองไปยังฐานข้อมูล

ในความเป็นจริงแล้ว ข้อมูลหนึ่ง ๆ จะมีอยู่สองชุดด้วยกัน คืออินสแทนซ์ของ entity bean ซึ่งอยู่ในหน่วยความจำ และตัว entity bean จริง ๆ ที่ถูกเก็บอยู่ในฐานข้อมูล ดังนั้นจึงต้องมีวิธีการส่งผ่านข้อมูลไปมาระหว่างจาวาออบเจ็กต์และฐานข้อมูล การส่งผ่านข้อมูลนี้สามารถทำได้โดยเมธอดที่ทุก entity bean class ต้องอิมพลีเมนต์ นั่นคือ ejbLoad() และ ejbStore()

ejbLoad() อ่านข้อมูลจาก persistent storage เข้าไปยังฟิลด์ของ entity bean ในหน่วยความจำ

ejbStore() บันทึกค่าในฟิลด์ปัจจุบันของอินสแทนซ์ของ bean ลงไปยังแหล่งเก็บข้อมูล เป็นการทำงานที่ตรงข้ามกับ ejbLoad()

เมธอดทั้งสองเป็น callback method ที่คอนเทนเนอร์จะเรียกใช้งานโดยอัตโนมัติ เป็นเมธอดการจัดการที่ EJB บังคับให้ต้องมี คอนเทนเนอร์จะพิจารณาเองว่า จะเรียกใช้งานเมธอดเหล่านี้เมื่อใด ขึ้นอยู่กับความสามารถของคอนเทนเนอร์ bean อาจถูกเรียกใช้งานเมธอดเหล่านี้เมื่อใดก็ได้ โดยคอนเทนเนอร์จะพิจารณาจากสถานะของทรานแซ็กชัน และ synchronize ข้อมูลให้โดยอัตโนมัติ โดยปกติแล้วลำดับจะเป็นดังรูปที่ 6-3



รูปที่ 6-3 การโหลดและการเก็บข้อมูลของ Entity Bean

แต่ในบางคอนเทนเนอร์สามารถกำหนดคุณสมบัติให้ enity bean เป็น read-only เท่านั้น ทำให้ไม่จำเป็นต้องเรียก ejbStore()

- **อินสแทนซ์ของ Entity Bean หลายตัวอาจแสดงข้อมูลเดียวกัน**

เมื่อมีไคลเอ็นต์หลายตัวเข้ามาใช้งานข้อมูลต่าง ๆ ในเวลาเดียวกัน จำเป็นต้องออกแบบวิธีการเข้าถึงข้อมูลที่มีประสิทธิภาพสูงให้กับ entity bean โดย entity bean ไม่อนุญาตให้ไคลเอ็นต์หลายตัวใช้อินสแทนซ์ของ entity bean ตัวเดียวพร้อม ๆ กัน เนื่องจากอินสแทนซ์ของ bean ที่ทำเช่นนี้ได้จะต้องเขียนแบบ thread-safe ซึ่งมีความซับซ้อนและผิดพลาดได้ง่าย ตรงข้ามกับวัตถุประสงค์ของ EJB ที่ต้องการให้การพัฒนาแอพพลิเคชันทำได้ง่ายและรวดเร็ว ปัญหาอีกข้อหนึ่งคือ ในขณะที่มีหลาย ๆ เธรดทำงานพร้อมกันนั้น การควบคุมทรานแซกชันโดยระบบจัดการทรานแซกชันจะทำได้ยากมาก

จากปัญหาดังกล่าว EJB จึงอนุญาตให้มีเพียงเธรดเดียวเท่านั้น ที่สามารถทำงานอยู่ในอินสแทนซ์ของ bean ไม่ว่าจะเป็นอินสแทนซ์ของ session bean หรือ entity bean ก็ตาม

เพื่อเพิ่มประสิทธิภาพในการทำงาน คอนเทนเนอร์สามารถสร้างอินสแทนซ์ของ entity bean หนึ่ง ๆ ขึ้นมาได้หลายชุด ทำให้ไคลเอ็นต์หลายตัวสามารถทำงานพร้อมกันได้ โดยแต่ละไคลเอ็นต์จะทำงานกับอินสแทนซ์หนึ่งตัว โดยที่อินสแทนซ์ของ entity bean เหล่านั้นสามารถแทนข้อมูลชุดเดียวกัน

การมีอินสแทนซ์ของ bean หลาย ๆ ชุดแทนข้อมูลชุดเดียวกันนั้น ทำให้เกิดปัญหาใหม่ตามมา คือ ปัญหาเรื่องความถูกต้องของข้อมูล เมื่อไคลเอ็นต์หลายตัวพยายามจะแก้ไขข้อมูลพร้อม ๆ กัน และอาจทำให้มีไคลเอ็นต์ที่ได้ข้อมูลผิดพลาดไป เพื่อแก้ปัญหานี้คอนเทนเนอร์จะทำการ synchronize ข้อมูลของ bean กับข้อมูลที่อยู่ในฐานข้อมูลด้วยการเรียกใช้งานเมธอด ejbLoad() และ ejbStore()

ความถี่ของการ synchronize ข้อมูล คอนเทนเนอร์จะพิจารณาจากทรานแซกชัน ทรานแซกชันทำให้ไคลเอ็นต์รู้สึกเหมือนกับว่าทำงานกับข้อมูลนั้นเพียงคนเดียว ทั้งที่ความเป็นจริงมีไคลเอ็นต์อีกหลายตัวกำลังทำงานกับข้อมูลชุดเดียวกันนั้นอยู่

- **อินสแทนซ์ของ Entity Bean สามารถพูลได้**

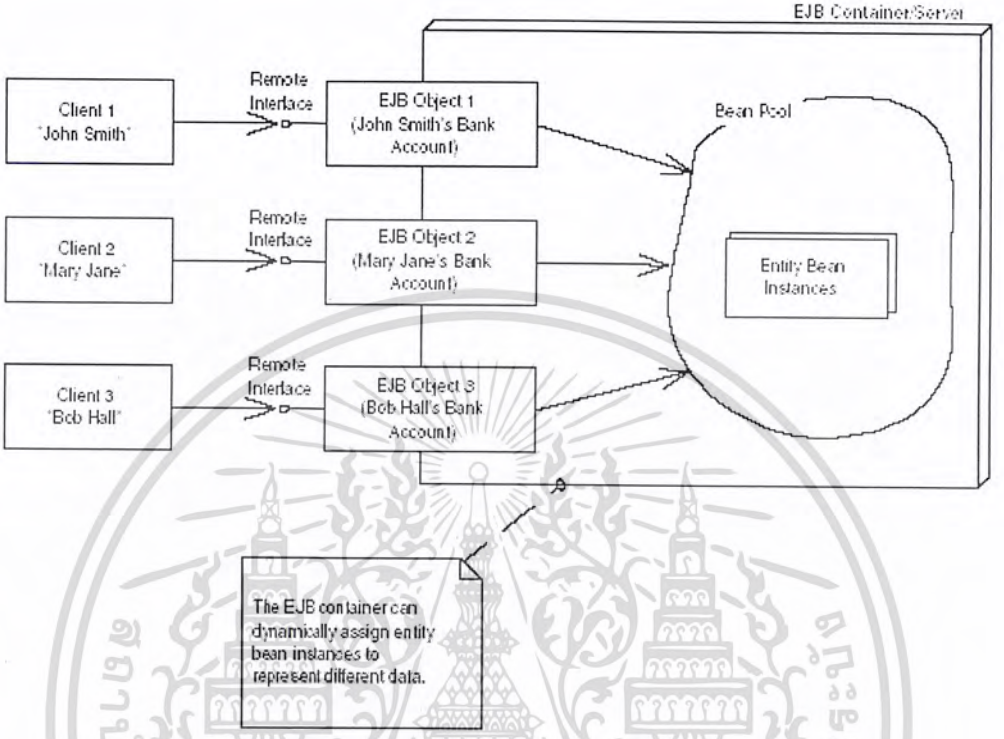
เพื่อประหยัดเวลาในการสร้างและทำลายอินสแทนซ์ของ bean ทุกครั้งที่มีไคลเอ็นต์ติดต่อเข้ามา และเพื่อเพิ่มประสิทธิภาพในการทำงาน อินสแทนซ์ของ entity bean เป็นออบเจ็กต์ที่สามารถนำกลับมาใช้ใหม่และพูลได้ คอนเทนเนอร์สามารถพูลและนำอินสแทนซ์ของ entity bean กลับไปใช้ใหม่เพื่อแทนอินสแทนซ์ของข้อมูลประเภทเดียวกันได้ เช่น คอนเทนเนอร์สามารถใช้อินสแทนซ์ของ entity bean บัญชีธนาคาร เพื่อแทนบัญชีหนึ่ง ๆ ในระบบบัญชีธนาคาร เมื่อใช้งานอินสแทนซ์นี้เสร็จแล้วก็สามารถนำไปให้บริการกับไคลเอ็นต์ตัวอื่น ๆ ได้ต่อไปและอาจแทนข้อมูลคนละชุดกับของเดิม คอนเทนเนอร์จัดการโดยการกำหนดอินสแทนซ์ของ entity bean ให้กับ EJB ออบเจ็กต์ที่แตกต่างกันออกไปในแต่ละไคลเอ็นต์

นอกจากจะช่วยประหยัดเวลาในการสร้างและทำลายอินสแทนซ์ของ bean โดยไม่จำเป็นแล้ว วิธีการนี้ยังช่วยลดทรัพยากรที่ระบบจำเป็นต้องใช้อีกด้วย ดังแสดงในรูปที่ 6-4
ในทำนองเดียวกับ stateful session bean การกำหนดอินสแทนซ์ของ entity bean ให้กับอีก EJB ออบเจ็กต์หนึ่งนั้นมีความซับซ้อน เมื่อ entity bean ถูกกำหนดให้กับ EJB ออบเจ็กต์ใด ๆ มันอาจมีการใช้งานทรัพยากร เช่น socket connection เป็นต้น แต่ในเวลาที่อยู่ในพูล ไม่มีความจำเป็นที่จะต้องใช้ซ็อกเก็ต ดังนั้นเพื่อให้ bean สามารถปล่อย และกลับมาใช้งานทรัพยากรต่าง ๆ entity bean จะต้องอิมพลีเมนต์ callback method ทั้งสองนี้

ejbActivate() เป็นเมธอดที่คอนเทนเนอร์เรียกใช้งานเมื่อจะนำอินสแทนซ์ของ bean ออกจากพูล กระบวนการนี้เรียกว่า activation บอกให้รู้ว่า คอนเทนเนอร์กำลังกำหนด bean ให้กับ EJB ออบเจ็กต์และ primary key หนึ่ง เมธอด ejbActivate() ควรจะร้องขอทรัพยากรที่ต้องใช้งานเมื่อถูกกำหนดให้กับ EJB ออบเจ็กต์แล้ว เช่นซ็อกเก็ต หลักการนี้เป็นหลักการเดียวกับการทำ activation ของ stateful session bean

ejbPassivate() เป็นเมธอดที่คอนเทนเนอร์เรียกใช้งานเมื่อจะนำอินสแทนซ์ของ bean เก็บเข้าไปในพูล กระบวนการนี้เรียกว่า passivation เพื่อบอกให้รู้ว่า คอนเทนเนอร์กำลังถอน bean ออกจาก EJBออบเจ็กต์

เมธอด ejbPassivate() ควรทำการปล่อยทรัพยากรที่ถือครองอยู่ทั้งหมด เช่นซ็อกเก็ตซึ่ง bean ได้ถือครองไว้ขณะ activation หลักการนี้เป็นหลักการเดียวกับการทำ passivation ของ stateful session bean

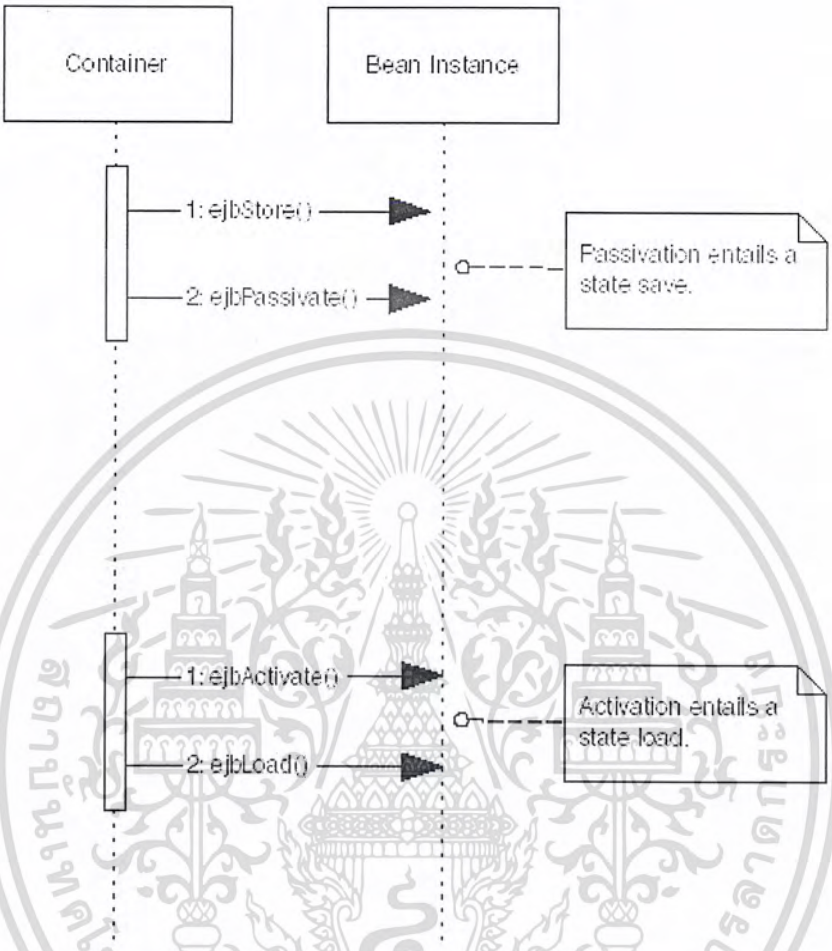


รูปที่ 6-4 การพูลอินสแทนซ์ของ Entity Bean โดยคอนเทนเนอร์

เมื่ออินสแทนซ์ของ entity bean ถูก passivate ไม่เพียงแต่จะต้องปล่อยทรัพยากรที่ถือครองเท่านั้น แต่ต้องบันทึกสถานะของมันลงไปแหล่งเก็บข้อมูลด้วย เพื่อให้ข้อมูลในแหล่งเก็บข้อมูลถูกอัพเดตให้ตรงกับสถานะล่าสุดของอินสแทนซ์ของ entity bean ในการบันทึกฟิลด์ของอินสแทนซ์ลงในฐานข้อมูล คอนเทนเนอร์จะเรียกใช้งานเมธอด ejbStore() ก่อนทำ passivation ทำนองเดียวกัน เมื่ออินสแทนซ์ของ entity bean ถูก activate ไม่เพียงแต่จะต้องร้องขอการใช้งานทรัพยากรที่จำเป็น แต่จะต้องโหลดข้อมูลล่าสุดขึ้นมาจากฐานข้อมูลด้วย ในการโหลดข้อมูลล่าสุดจากฐานข้อมูลนี้ คอนเทนเนอร์จะเรียกใช้เมธอด ejbLoad() หลังจากทำ activation วิธีการนี้แสดงอยู่ในรูปที่ 6-5

entity bean มีความคล้ายคลึงกับ stateful session bean ตรงที่สามารถทำ activation และ passivation ได้ แต่ข้อแตกต่างที่สำคัญคือ entity bean มีเมธอด ejbLoad() เพื่อโหลดค่าสถานะขณะทำ activation และ เมธอด ejbStore() เพื่อบันทึกสถานะขณะทำ passivation entity bean เป็นออบเจ็กต์ที่มีความซับซ้อนกว่า stateful session bean มาก และต้องการวิธีการที่ซับซ้อนมากกว่า object serialization อินสแทนซ์ของ entity bean อาจเก็บข้อมูลของตัวเองลงในฐานข้อมูลแบบออบเจ็กต์โดยใช้ API ของฐานข้อมูลแบบออบเจ็กต์ ทำให้ entity bean ต้องการเมธอด ejbLoad() และ ejbStore() เพื่อจัดการเกี่ยวกับการเก็บค่า

สถานะ ขณะที่ stateful session bean ไม่ต้องการเมธอดเหล่านี้ แต่จะใช้วิธีการ object serialization ในการบันทึกสถานะของ stateful session bean



รูปที่ 6-5 การทำ passivation และ activation ของ Entity Bean

- **Entity Bean สามารถทำ persistent ได้สองวิธี**

entity bean เป็น persistent object ที่รู้ถึงวิธีการแปลงตัวมันเองลงในแหล่งบันทึกข้อมูล เช่น ฐานข้อมูลเชิงสัมพันธ์ โดยใช้หลักการ O/R mapping หรือฐานข้อมูลเชิงวัตถุ มีวิธีการทำ persistent 2 วิธี

1) Bean-Managed Persistent

วิธีนี้ bean เป็นผู้จัดการเกี่ยวกับการติดต่อกับแหล่งบันทึกข้อมูลเอง โดยอิมพลีเมนต์การเรียกใช้ฐานข้อมูลลงไปใน bean เช่น เมื่อทำงานกับฐานข้อมูลเชิงสัมพันธ์ entity bean สามารถทำคำสั่ง SQL INSERT ผ่านทาง JDBC เพื่อบันทึกข้อมูลลงไปในฐานข้อมูลเชิงสัมพันธ์ หรืออาจทำคำสั่ง SQL DELETE ผ่านทาง JDBC เพื่อลบข้อมูลออกจากฐานข้อมูล

2) Container-Managed Persistent

วิธีนี้ให้คอนเทนเนอร์เป็นผู้จัดการเกี่ยวกับ persistent ให้ ทำให้สามารถตัดลอจิกที่เกี่ยวข้องกับการ persistent ข้อมูลออกจาก bean ได้ ไม่จำเป็นต้องเขียนโค้ดลงไปใน bean คลาส เพียงแต่บอก EJB คอน

เทนเนอร์ว่า ฟิลด์ใดเป็น persistent ฟิลด์ โดยระบุไว้ใน deployment descriptor เมื่อคอนเทนเนอร์ทราบแล้วว่า ฟิลด์ใดบ้างที่ต้อง persistent คอนเทนเนอร์จะจัดการลอจิกเกี่ยวกับการเข้าถึงข้อมูลให้โดยอัตโนมัติ

การจัดการเกี่ยวกับ persistent โดยอัตโนมัตินี้ เป็นหน้าที่ของผู้ดีพลอยที่จะระบุว่า entity bean จะแมปลงไปยังแหล่งเก็บข้อมูลอย่างไร เช่น ฟิลด์ของจาวาออบเจ็กต์ชื่อ customerID จะต้องแมปเข้ากับคอลัมน์ชื่อ id ในตาราง customer ของฐานข้อมูล เป็นต้น สิ่งนี้เป็นจุดเด่นที่สำคัญของ entity bean เพราะสามารถเขียนออบเจ็กต์ของข้อมูลที่ไม่ขึ้นกับแหล่งบันทึกข้อมูลได้ และสามารถใช้ได้ในสภาพแวดล้อมที่หลากหลาย

- **Entity Bean สามารถสร้าง,ทำลาย หรือค้นหาได้**

การกำหนดค่าเริ่มต้น และการทำลาย entity bean แตกต่างออกไปจาก session bean เล็กน้อย entity bean เป็นมุมมองลงไปยังฐานข้อมูล และมองอินสแทนซ์ของ entity bean และข้อมูลที่อยู่ในฐานข้อมูลเป็นสิ่งเดียวกัน เพราะข้อมูลจะถูก synchronize กันอยู่เสมอ การสร้างและกำหนดค่าเริ่มต้นให้กับอินสแทนซ์ของ entity bean จึงเป็นการสร้างและกำหนดค่าเริ่มต้นให้กับข้อมูลในฐานข้อมูลด้วย ดังนั้นเมื่อ entity bean ถูกสร้างและกำหนดค่าเริ่มต้นในหน่วยความจำด้วยเมธอด ejbCreate() เมธอดนี้จะสร้างข้อมูลใหม่ในฐานข้อมูลซึ่งสัมพันธ์กับอินสแทนซ์ในหน่วยความจำ ในทำนองเดียวกัน เมื่อเมธอด ejbRemove() ถูกเรียกใช้งาน ข้อมูลในฐานข้อมูลที่สัมพันธ์กันก็จะถูกลบทิ้ง ถ้าเป็นแบบ container-managed persistence คอนเทนเนอร์จะแก้ไขข้อมูลในฐานข้อมูลให้อัตโนมัติ จึงไม่ต้องอิมพลีเมนต์เมธอดทั้งสองนี้

ข้อมูลของ entity bean สามารถระบุชี้เฉพาะตัวได้ จึงสามารถใช้การค้นหา entity bean แทนที่จะต้องสร้างขึ้นมาใหม่ การค้นหา entity bean เหมือนกับการใช้คำสั่ง SQL SELECT การค้นหา entity bean ก็คือการค้นหาข้อมูลในแหล่งบันทึกข้อมูล แตกต่างจาก session bean ที่ไม่สามารถค้นหาได้ เพราะ session bean นั้นไม่ได้เป็นออบเจ็กต์ที่ถาวร

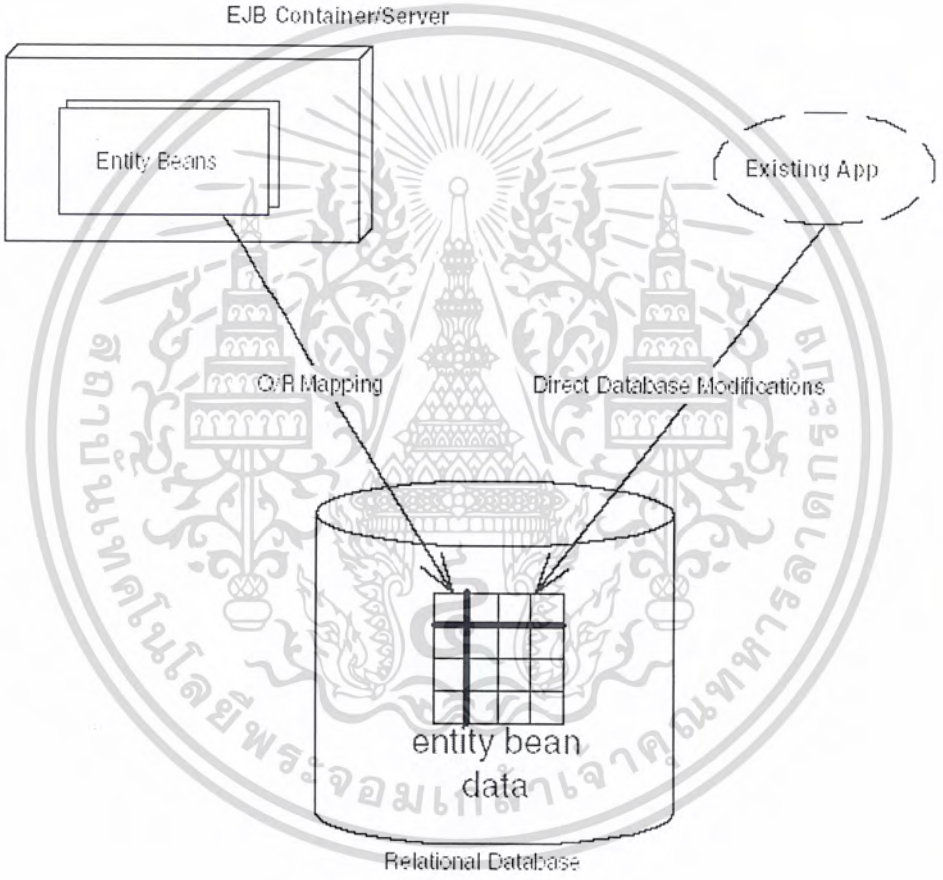
วิธีค้นหา entity bean มีได้หลายวิธีด้วยกัน โดยจะระบุวิธีการเหล่านี้เป็นเมธอดไว้ใน home interface ของ entity bean เรียกว่า finder method นอกเหนือจากเมธอดที่ใช้ในการสร้างและทำลาย entity bean จุดนี้เป็นจุดที่ home interface ของ session bean และ entity bean แตกต่างกัน เพราะใน home interface ของ session bean ไม่มี finder method

- **Entity Bean สามารถใช้เป็นตัวแทน Legacy Data และ Legacy Systems ได้**

entity bean สามารถเกิดก่อนที่จะมีการนำ EJB มาใช้ก็ได้ เพราะ entity bean ก็คือข้อมูล ดังนั้นสามารถนำระบบฐานข้อมูลเก่า (legacy database) หรือระบบเก่า (legacy system) และใช้ entity bean เป็นตัวแทนข้อมูลเหล่านี้ได้

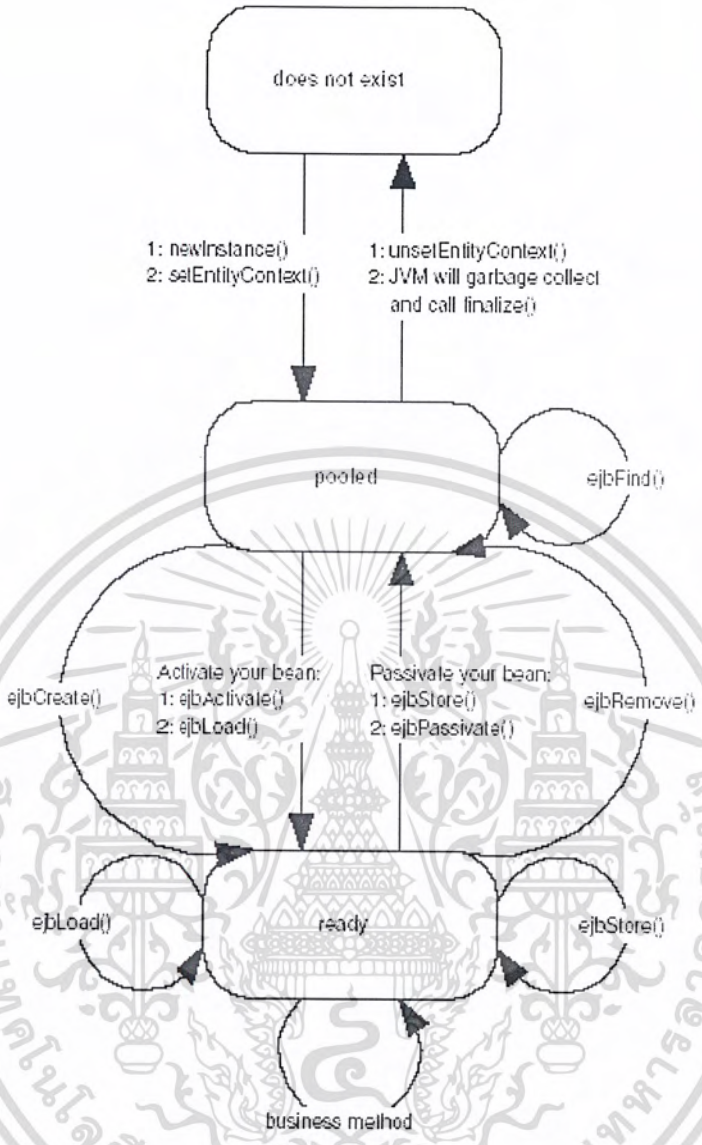
- **Entity Bean สามารถแก้ไขได้โดยไม่ต้องผ่าน EJB**

ปกติ การสร้าง , ทำลาย และค้นหาข้อมูลของ entity bean ทำโดยใช้ home interface แต่สามารถติดต่อกับ entity bean ได้อีกวิธีหนึ่ง คือการแก้ไขฐานข้อมูลซึ่งบันทึกข้อมูลนั้นอยู่โดยตรง เช่น ถ้าอินสแทนซ์ของ bean ถูกแมปเข้ากับฐานข้อมูลเชิงสัมพันธ์ จึงสามารถลบแถวออกจากฐานข้อมูลซึ่งสัมพันธ์กับอินสแทนซ์ของ entity bean ได้โดยตรง (รูป 6-6) , สามารถสร้างข้อมูล entity bean ใหม่ และแก้ไขข้อมูลที่มีอยู่แล้ว โดยการทำงานกับฐานข้อมูลโดยตรง การทำเช่นนี้อาจมีความจำเป็นหากระบบงานส่วนหนึ่งเป็นระบบงานเก่าที่แก้ไขข้อมูลโดยการติดต่อกับฐานข้อมูลโดยตรง



รูปที่ 6-6 การแก้ไขฐานข้อมูลที่สัมพันธ์กับ Entity Bean โดยตรง

วงจรชีวิตของ entity bean ดังรูปที่ 6-7 ซึ่งจะเห็นว่ามีเมธอดที่เพิ่มเติมขึ้นมาคือ ejbFind() , ejbLoad() และ ejbCreate()



รูปที่ 6-7 วงจรชีวิตของ Entity Bean

6.2.4 ตัวอย่างโค้ด Entity Bean

ในการเขียน entity bean class จะต้องอิมพลีเมนต์อินเทอร์เฟซ javax.ejb.EntityBean อินเทอร์เฟซจะกำหนดเมธอดที่ entity bean class จะต้องอิมพลีเมนต์ อินเทอร์เฟซ javax.ejb.EnterpriseBean ไม่ได้กำหนดเมธอดใด ๆ ไว้ ดังนี้

```
public interface javax.ejb.EnterpriseBean implements {
}
```

อินเทอร์เฟซ javax.ejb.EntityBean จะกำหนด callback method ที่ bean ต้องอิมพลีเมนต์ ดังนี้

```
public interface javax.ejb.EntityBean implements javax.ejb.EnterpriseBean{
  public abstract void setEntityContext(javax.ejb.EntityContext);
  public abstract void unsetEntityContext();
  public abstract void ejbRemove();
  public abstract void ejbActivate();
  public abstract void ejbPassivate();
  public abstract void ejbLoad();
  public abstract void ejbStore();
}
```

นอกเหนือจากเมธอดที่เห็น bean จำเป็นที่จะต้องกำหนดเมธอด ejbCreate() ที่ใช้ในการสร้างข้อมูลของ entity bean ใหม่และ ejbFind() ที่ใช้ในการค้นหาข้อมูลของ bean ที่มีอยู่

- ejbCreate()

ejbCreate() ใช้ในการสร้างข้อมูลใหม่ในฐานข้อมูล เมธอดนี้จะมีหรือไม่มีก็ได้ หากไม่ต้องการสร้างข้อมูลใหม่ขึ้นมาก็ไม่จำเป็นที่จะต้องมีเมธอดนี้ ไม่เหมือนกับ session bean ที่จะบังคับให้มีอย่างน้อย 1 เมธอด พารามิเตอร์ของเมธอดนี้มีได้หลากหลายขึ้นอยู่กับข้อมูลในฐานข้อมูลที่ต้องการสร้าง และ ejbCreate() จะต้องมีเมธอด create() ที่มีพารามิเตอร์ตรงกับ ejbCreate() นั้นใน home interface ตัวอย่างเช่น bank account entity bean class ที่ชื่อว่า AccountBean กับ remote interface ที่ชื่อ Account , home interface ชื่อ AccountHome และ primary key class ชื่อ AccountPK โดยให้เมธอด ejbCreate() ใน AccountBean เป็นดังนี้

public AccountPK ejbCreate(String accountId,String owner) …

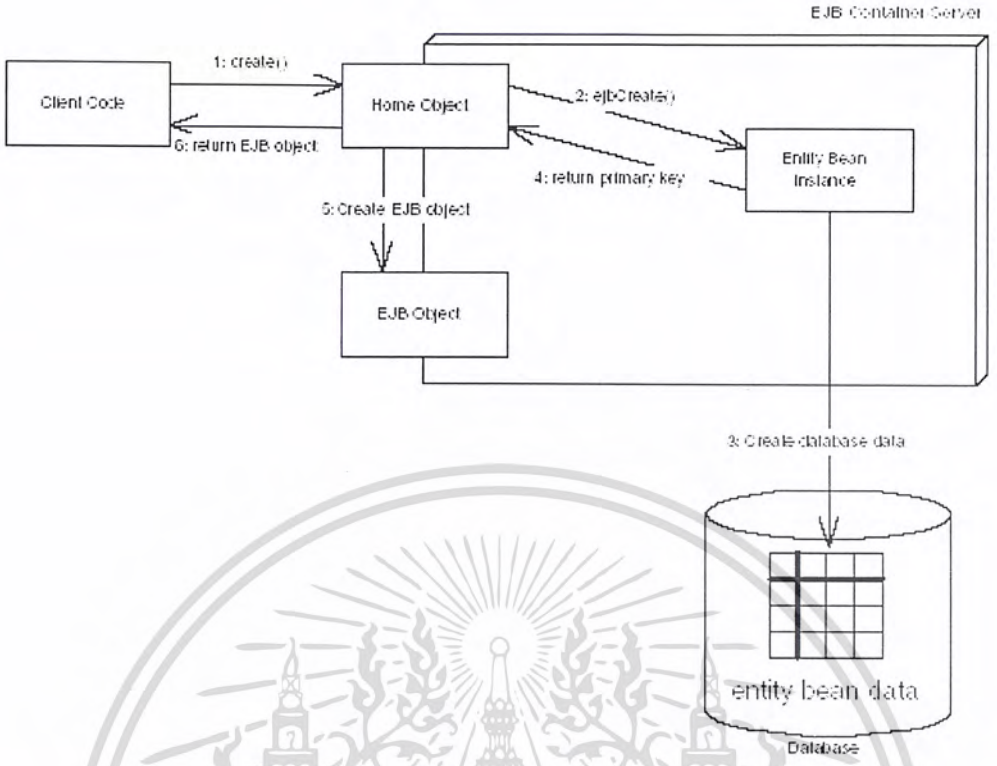
จะต้องมีเมธอด create() ใน home interface ดังนี้

public Account create(String accountId,String owner) …

สังเกตข้อแตกต่างระหว่างค่าที่คืนมาใน 2 เมธอดนี้ จะพบว่าอินสแทนซ์ของ bean จะคืนค่า primary key ขณะที่ home object จะคืนค่า EJB object ที่เป็นเช่นนี้เนื่องจากคอนเทนเนอร์จะคืนค่า primary key ให้กับ home object และ home object จะนำไปสร้าง EJB object แล้วคืนค่า EJB object ให้กับไคลเอ็นต์ ดังรูปที่ 6-8

- ejbFind()

finder method ใช้ในการหา entity bean ที่อยู่ใน storage โดยสามารถมีได้หลายเมธอดดังนี้



รูปที่ 6-8 ความสัมพันธ์ระหว่าง ejbCreate() และ create()

```
public AccountPK ejbFindByPrimaryKey(AccountPK key)
public Enumeration ejbFindAllProducts()
public Enumeration ejbFindBigAccounts(int minimum)
public OrderPK ejbFindMostRecentOrder()
```

finder method มีกฎคือ จะต้องขึ้นด้วย ejbFind , ต้องมีอย่างน้อย 1 เมธอดคือ ejbFindByPrimaryKey , ค่าที่คืนจะต้องเป็น primary key ของ entity bean หรือ enumeration ของ primary key และเหมือนกับ ejbCreate() คือต้องมีเมธอดที่ตอบสนองคู่กับ ejbFind ดังตัวอย่างนี้ finder method ใน AccountBean ดังนี้

```
public Enumeration ejbFindAllProducts() …
public AccountPK ejbFindBigAccounts(int minimum) …
```

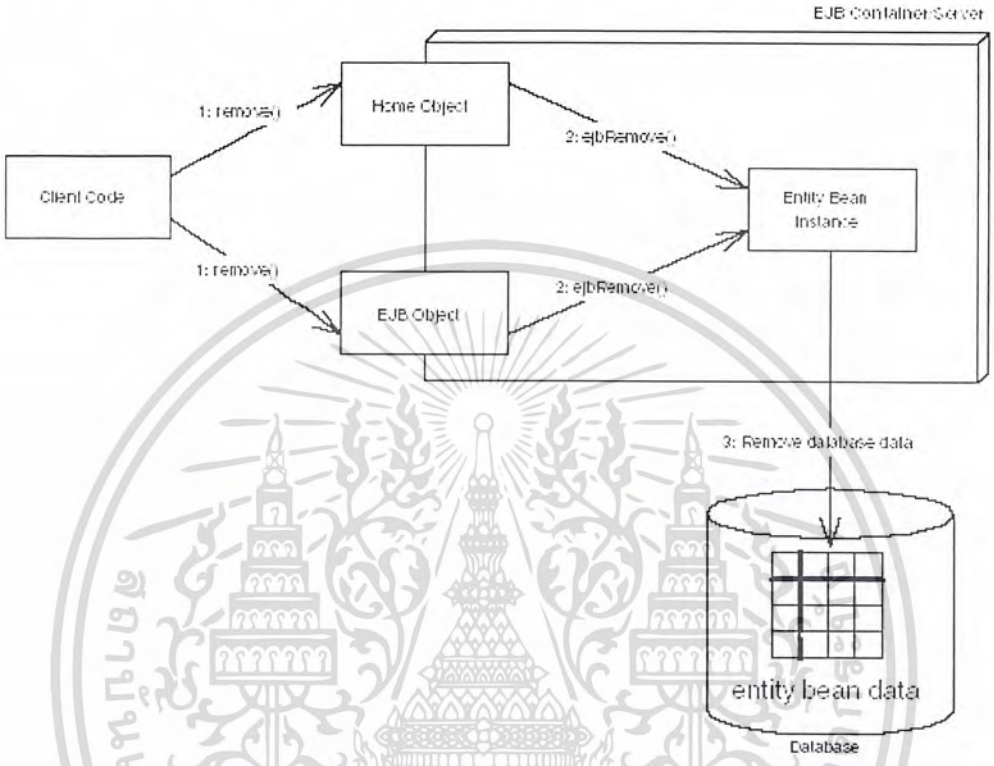
แล้วจะต้องมีเมธอดใน home interface ดังนี้

```
public Enumeration findAllProducts() …
public Account ejbFindBigAccounts(int minimum) …
```

สังเกตว่าในกรณีที่ไม่ใช่ enumeration ค่าที่คืนจะต่างกันด้วยเหตุผลเดียวกับ ejbCreate() และมีกฎพิเศษอีกอย่างหนึ่งคือในกรณีที่เป็น container-managed persistence EJB คอนเทนเนอร์จะอิมพลีเมนต์ finder method ใน EJB object ให้โดยอัตโนมัติ

- ejbRemove()

ไคลเอ็นต์จะใช้เมธอด remove() ในการลบข้อมูลของ entity bean ในฐานข้อมูล โดยเรียกเมธอด remove() กับ EJB object หรือ home object แล้วคอนเทนเนอร์จะเรียกเมธอด ejbRemove() กับ bean ดังรูปที่ 6-9 เมธอดนี้จะไม่ทำลายอินสแทนซ์ของ entity bean แต่จะลบข้อมูลในฐานข้อมูลเท่านั้น



รูปที่ 6-9 การลบข้อมูลของ Entity Bean

6.3 Bean-Managed Persistent Entity Bean

entity bean ทั้ง 2 ชนิดจะต้องอิมพลีเมนต์อินเทอร์เฟซ javax.ejb.EntityBean โดยอินเทอร์เฟซจะกำหนด callback method ที่คอนเทนเนอร์จะเรียกใช้กับ bean เมธอดที่ต้องอิมพลีเมนต์นั้นขึ้นอยู่กับชนิดของ entity bean ตารางที่ 6-1 จะอธิบายถึงเมธอดสำหรับ bean-managed persistent

เมธอด	คำอธิบาย
setEntityContext()	เมื่อคอนเทนเนอร์สร้างอินสแทนซ์ของ bean แล้วจะเรียกเมธอดนี้
ejbFind<...>(<...>)	finder method ใช้ในการค้นหาอินสแทนซ์ของ entity bean ที่มีข้อมูลที่ต้องการ โดย bean จะต้องมีเมธอดนี้อย่างน้อยหนึ่งเมธอดคือ ejbFindByPrimaryKey()
EjbCreate(<...>)	เมื่อไคลเอ็นต์เรียกเมธอด create() กับ home interface คอนเทนเนอร์จะเรียกเมธอด ejbCreate() กับอินสแทนซ์ของ bean ที่พูลอยู่ ejbCreate() จะทำหน้าที่ในการสร้างข้อมูลใหม่ในฐานข้อมูลให้
ejbPostCreate(<...>)	bean class จะต้องกำหนดเมธอด ejbPostCreate() ให้เข้าคู่กับ ejbCreate() คือต้อง

	มีพารามิเตอร์เหมือนกัน คอนเทนเนอร์จะเรียกเมธอดนี้หลังจากเรียก ejbCreate()
ejbActivate()	เมื่อไคลเอ็นต์ต้องการเรียกบิซิเนสเมธอดกับ EJB object แต่ไม่มีอินสแทนซ์ของ entity bean นั้นที่ bind กับ EJB object คอนเทนเนอร์จะนำ bean จากพูลและเปลี่ยนสถานะเป็น ready โดยเมธอดนี้จะไม่ถูกเรียกระหว่างทรานแซ็กชัน
ejbLoad()	คอนเทนเนอร์จะเรียกเมธอดนี้เพื่อโหลดข้อมูลจากฐานข้อมูลไปยังอินสแทนซ์ของ bean
ejbStore()	คอนเทนเนอร์จะเรียกเมธอดนี้เพื่ออัพเดตข้อมูลจากอินสแทนซ์ของ bean ลงฐานข้อมูล
ejbPassivate()	คอนเทนเนอร์จะเรียกเมธอดนี้เมื่อนำ entity bean เก็บลงในพูล
ejbRemove()	จะลบข้อมูลในฐานข้อมูล
unsetEntityContext()	เมธอดนี้ใช้ในการลบ bean ออกจากเอ็นไวรอนเมนต์ คอนเทนเนอร์จะเรียกเมธอดนี้ก่อนที่อินสแทนซ์ของ entity bean จะถูกทำลาย

ตารางที่ 6-1 คำอธิบายเมธอดใน Bean-Managed Persistent Entity Bean

6.3.1 ตัวอย่างโค้ด Bean-Managed Persistent Entity Bean

ตัวอย่าง bean นี้คือ bean ที่เก็บข้อมูลของบัญชีธนาคาร สคริปต์ที่ใช้สร้างเทเบิลดังนี้

```
create table accounts (id varchar(64), ownername varchar(64), balance numeric(18));
```

6.3.1.1 Remote Interface

remote interface ขยายมาจาก javax.ejb.EJBObject เหมือน remote interface ของ bean ชนิดอื่น ๆ สังเกตที่เมธอด withdraw(double amt) จะพบว่ามี exception แบบกำหนดเองคือ AccountException ที่จะโยนออกมาเมื่อจะถอนเงินมากกว่าจำนวนเงินที่มีอยู่ โค้ดดังนี้

Account.java

```
package olala;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Account extends EJBObject{
  public double getBalance()     throws RemoteException;
  public void deposit(double amt)     throws RemoteException;
  public void withdraw(double amt)     throws RemoteException,AccountException;
  public String getId()     throws RemoteException;
  public void setId(String id)     throws RemoteException;
  public String getOwnername()     throws RemoteException;
```

```
  public void setOwnername(String ownername)      throws RemoteException;
}
```

6.3.1.2 Home Interface

โค้ดดังนี้

AccountHome.java

```
package olala;
import javax.ejb.EJBHome;
import javax.ejb.FinderException;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import java.util.Enumeration;
public interface AccountHome extends EJBHome
{
  public Account findByPrimaryKey(AccountPK key)
    throws FinderException, RemoteException;
  public Account create(String id, String ownername)
    throws CreateException, RemoteException;
  public Account create(String id, String ownername, double balance)
    throws CreateException, RemoteException;
  public Enumeration findByOwnerName(String name)
    throws FinderException, RemoteException;
}
```

สังเกตว่าเมธอด create() จะกำหนดให้มีการโยน javax.ejb.CreateException ออกมาได้ และเมธอด find() จะกำหนดให้มีการโยน javax.ejb.FinderException ออกมาได้

6.3.1.3 primary key class

primary key class จะมีหรือไม่ก็ได้ ถ้า primary key มี attribute เดียว (ตัวอย่างนี้คือ id) ตัวอย่างนี้เป็นตัวอย่างที่ใช้ primary key class ถ้าไม่ใช้ในส่วนของ descriptor แทนที่จะระบุคลาสของ primary key เป็น AccountPK ก็จะระบุเป็น java.lang.String แทน เนื่องค่าตัวแปร id เป็นสตริง โค้ดดังนี้

AccountPK.java

```
package olala;
import java.io.Serializable;
```

```
public class AccountPK implements Serializable{
  public String id;
  public int hashCode() { return (id.hashCode()); }
  public boolean equals(Object that)   {
    if (!(that instanceof AccountPK))
      return false;
    AccountPK tmp = (AccountPK)that;
    return (this.id.equals(tmp.id));
  }
  public AccountPK() { }
  public AccountPK(String id) {
    this.id=id;
  }
  public String toString(){
    return id;
  }
}
```

6.3.1.4 Enterprise Bean Class

bean-managed persistent จะต้องแมปกับฐานข้อมูลโดยใช้ JDBC จึงทำให้โค้ดมีขนาดใหญ่และยุ่งยากในการเขียน โดยตัวอย่างโค้ดนี้จะรับ JDBC connection โดยใช้ datasource ซึ่งจะต้องกำหนดโดยดึงจาก resource reference ที่ระบุในไฟล์ XML ทำให้สามารถเปลี่ยนแปลง datasource ได้โดยไม่ต้องยุ่งเกี่ยวกับโค้ด ตัวอย่างนี้ใช้ชื่อ resource reference ว่า jdbc/OlalaDB และค่าของมันเป็นชื่อ JNDI ที่ชื่อ OlalaDB โดยได้กำหนดให้คอนเทนเนอร์ bind datasource กับ JNDI ไว้ วิธี bind datasource อยู่ในภาคผนวก ค โค้ดของ bean ดังนี้

AccountEJB.java

```
package olala;
import javax.ejb.*;
import java.sql.*;
import javax.sql.DataSource;
import javax.naming.*;
import java.util.*;
public class AccountEJB implements EntityBean{
```

```
private String id;
private String ownername;
private double balance;
private EntityContext context;
public AccountEJB()  { }
public void setId(String id){ this.id = id; }
public String getOwnername(){ return(ownername); }
public void setOwnername(String ownername) { this.ownername = ownername; }
public double getBalance(){ return(balance); }
public void setEntityContext(EntityContext ec){ context = ec; }
public void unsetEntityContext(){ this.context = null;}
public void ejbActivate()  { }
public void ejbPassivate()  { }
private Connection getConnection()  throws EJBException  {
  final String DATASOURCE = "java:comp/env/jdbc/OlalaDB";
  try {
    InitialContext ic = new InitialContext();
    DataSource ds = (DataSource)ic.lookup(DATASOURCE);
    return ds.getConnection();
  }
  catch (Exception ex) {
    throw new EJBException(ex.getMessage());
  }
}
private void readData(AccountPK key)      throws FinderException, EJBException    {
  Exception ex = null;
  Connection con = null;
  PreparedStatement ps = null;
  try {
    con = getConnection();
    ps = con.prepareStatement("select ownername,balance from olalaDB.dbo.accounts where id = ?");
    ps.setString(1, key.id);
    ResultSet rs = ps.executeQuery();
```

```
      if (!rs.next())
        ex = new FinderException("Record not found");
      else {
        id = key.id;
        ownername = rs.getString(1);
        balance = rs.getDouble(2);
      }
      rs.close();
    }
    catch (EJBException ee) {
      ex = ee;
    }
    catch (SQLException se) {
      ex = new EJBException(se.getMessage());
    }
    try {
      ps.close();
      con.close();
    }
    catch (Exception e)  { }
    if (ex != null) {
      if (ex instanceof EJBException)
        throw (EJBException)ex;
      if (ex instanceof FinderException)
        throw (FinderException)ex;
    }
  }
  public AccountPK ejbFindByPrimaryKey(AccountPK key) throws FinderException {
    try {
      readData(key);
      return key;
    }
    catch (EJBException ee) {
```

```
        throw new FinderException(ee.getMessage());
      }
    }
    public void ejbLoad() {
      AccountPK key = (AccountPK)context.getPrimaryKey();
      try {
        readData(key);
      }
      catch(Exception ex) {
        throw new RuntimeException(ex.getMessage());
      }
    }
    public void ejbStore() {
      RuntimeException ex = null;
      Connection con = null;
      PreparedStatement ps = null;
      try {
        con = getConnection();
        AccountPK key = (AccountPK)context.getPrimaryKey();
        ps = con.prepareStatement("update olalaDB.dbo.accounts set ownername = ?,balance = ? where id
= ?");
        if (ownername != null)
          ps.setString(1, ownername);
        else
          ps.setNull(1, java.sql.Types.VARCHAR);
        ps.setDouble(2, balance);
        if (id != null)
          ps.setString(3, key.id);
        else
          ps.setNull(3, java.sql.Types.VARCHAR);
        int ret = ps.executeUpdate();
        if (ret == 0)
          ex = new RuntimeException("ejbStore: Execute failed");
```

```
    }
    catch (EJBException ee)  {
      ex = new RuntimeException(ee.getMessage());
    }
    catch (SQLException se) {
      ex = new RuntimeException(se.getMessage());
    }
    try {
      ps.close();
      con.close();
    }
    catch (Exception e)  { }
    if (ex != null)
      throw ex;
  }
  public AccountPK ejbCreate(String id, String ownername, double balance)  throws CreateException {
    this.id = id;
    this.ownername = ownername;
    this.balance = balance;
    CreateException ce = null;
    Connection con = null;
    PreparedStatement ps = null;
    AccountPK key = null;
    try {
      con = getConnection();
      ps = con.prepareStatement("insert into olalaDB.dbo.accounts (id, ownername, balance) values (?,
?, ?)");
      if (id != null)
        ps.setString(1, id);
      else
        ps.setNull(1, java.sql.Types.VARCHAR);
      if (ownername != null)
        ps.setString(2, ownername);
```

```
      else
        ps.setNull(2, java.sql.Types.VARCHAR);
      ps.setDouble(3, balance);
      int ret = ps.executeUpdate();
      if (ret != 1)
        ce = new CreateException("Create failed");
      key = new AccountPK();
      key.id = id;
    }
    catch (EJBException ee) {
      ce = new CreateException(ee.getMessage());
    }
    catch (SQLException se) {
      ce = new CreateException(se.getMessage());
    }
    try {
      ps.close();
      con.close();
    }
    catch (Exception e)  { }
    if (ce != null)
      throw ce;
    return key;
  }
  public void ejbPostCreate(String id, String ownername, double balance)  { }
  public void ejbRemove() throws RemoveException  {
    RemoveException re = null;
    Connection con = null;
    PreparedStatement ps = null;
    try  {
      con = getConnection();
      AccountPK key = (AccountPK)context.getPrimaryKey();
      ps = con.prepareStatement("delete from olalaDB.dbo.accounts where id = ?");
```

```
      ps.setString(1, key.id);
      int result = ps.executeUpdate();
      if (result == 0)
        re = new RemoveException("Remove failed");
    }
    catch (EJBException ee) {
      re = new RemoveException(ee.getMessage());
    }
    catch (SQLException se) {
      re = new RemoveException(se.getMessage());
    }
    try {
      ps.close();
      con.close();
    }
    catch (Exception e)  { }
    if (re != null)
      throw re;
  }
  public AccountPK ejbCreate(String id, String ownername) throws CreateException {
    return ejbCreate(id,ownername,0);
  }
  public void ejbPostCreate(String id, String ownername)  { }
  public Enumeration ejbFindByOwnerName(String name) throws FinderException {
    PreparedStatement pstmt=null;
    Connection con=null;
    Vector v=new Vector();
    try{
      con=getConnection();
      pstmt=con.prepareStatement("select id from accounts where ownerName=?");
      pstmt.setString(1,name);
      ResultSet rs=pstmt.executeQuery();
      while (rs.next()) {
```

```
        String id=rs.getString("id");
        v.addElement(new AccountPK(id));
      }
      return v.elements();
    }
    catch(Exception e) {
      throw new FinderException(e.toString());
    }
    finally{
      try{
        pstmt.close();
      }
      catch (Exception e)  { }
      try {
        con.close();
      }
      catch (Exception e)  { }
    }
  }
  public void deposit(double amt)  {
    balance+=amt;
  }
  public void withdraw(double amt) throws AccountException
  {
    if (amt>balance){
      throw new AccountException("Your balance is "+balance+"!  You cannot withdraw "+amt+"!");
    }
    balance-=amt;
  }
```

6.3.1.5 Exception Class

เป็นคลาส exception ที่สร้างขึ้นเองขยายมาจาก java.lang.Exception โค้ดดังนี้

AccountException.java

```
package olala;
public class AccountException extends java.lang.Exception{
  public AccountException(){
     super();
  }
  public AccountException(Exception e) {
     super(e.toString());
  }
  public AccountException(String s) {
     super(s);
  }
}
```

6.3.1.6 Deployment Descriptor

bean-managed persistent จะมีไฟล์ descriptor 2ไฟล์เหมือนกับ session bean คือ ejb-jar.xml และ weblogic-ejb-jar.xml โค้ดดังนี้

ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <enterprise-beans>
    <entity>
      <description></description>
      <ejb-name>Account</ejb-name>
      <home>olala.AccountHome</home>
      <remote>olala.Account</remote>
      <ejb-class>olala.AccountEJB</ejb-class>
      <persistence-type>Bean</persistence-type>
      <prim-key-class>olala.AccountPK</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-field>
```

```
            <field-name>balance</field-name>
        </cmp-field>
        <cmp-field>
            <field-name>ownername</field-name>
        </cmp-field>
        <cmp-field>
            <field-name>id</field-name>
        </cmp-field>
        <resource-ref>
            <res-ref-name>jdbc/OlalaDB</res-ref-name>
            <res-type>javax.sql.DataSource</res-type>
            <res-auth>Container</res-auth>
        </resource-ref>
    </entity>
  </enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>Account</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

weblogic-ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
    <weblogic-enterprise-bean>
        <ejb-name>Account</ejb-name>
```

```
    <reference-descriptor>
      <resource-description>
        <res-ref-name>jdbc/OlalaDB</res-ref-name>
        <jndi-name>OlalaDB</jndi-name>
      </resource-description>
    </reference-descriptor>
    <jndi-name>Account</jndi-name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

6.3.1.7 คอมไพล์และแพ็กเก็จ

คอมไพล์จาวาไฟล์ จัดไฟล์ทั้งหมดอยู่ใน path ดังนี้

./Account/olala/Account.class

./Account/olala/AccountEJB.class

./Account/olala/AccountHome.class

./Account/olala/AccountException.class

./Account/olala/AccountPK.class

./Account/meta-inf/ejb-jar.xml

./Account/meta-inf/weblogic-ejb-jar.xml

สร้าง jar ไฟล์โดยใช้คำสั่งดังนี้

./Account>jar cv0f stdAccount.jar olala meta-inf

และทำคำสั่งต่อไปนี้

./Account/java weblogic.ejbc –compiler javac stdAccount.jar Account.jar

แล้วจึงนำ Account.jar ไปดีพลอย

6.3.1.8 ไคลเอ็นต์

เซต classpath ชี้ไปยัง Account.jar หรือ path ที่มีอินเทอร์เฟซของ bean โค้ดไคลเอ็นต์ดังนี้

AccountClient.java

```
import olala.*;
import javax.ejb.*;
import javax.naming.*;
import java.util.*;
```

```
public class AccountClient{
  public static void main(String[] args){
    try {
      Context ctx = getInitialContext();
      AccountHome home = (AccountHome) ctx.lookup("Account");
      Account account = null;
      try {
        home.create("123-456.7890","John Smith");
        Enumeration enumeration=home.findByOwnerName("John Smith");
        if (enumeration!=null){
          account=(Account) enumeration.nextElement();
        }
        else{
          throw new Exception("Could not find account");
        }
        System.out.println("Initial Balance = "+account.getBalance());
        account.deposit(100);
        System.out.println("After depositing 100, account balance = "+account.getBalance());
        AccountPK pk=(AccountPK) account.getPrimaryKey();
        account=null;
        account=home.findByPrimaryKey(pk);
        System.out.println("Found account with ID "+pk+".  Balance = "+account.getBalance());
        System.out.println("Now trying to withdraw $150, which is more than is "+"currently available.
This should generate an exception..");
        account.withdraw(150);
      }
      catch (Exception e) {
        e.printStackTrace();
      }
      finally {
        try {
          System.out.println("Destroying account..");
          if (account!=null) {
```

```
            account.remove();
          }
        }
        catch (Exception e) {
          e.printStackTrace();
        }
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  public static Context getInitialContext() throws NamingException{
    Properties p = new Properties();
    p.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
    p.put(Context.PROVIDER_URL, url);
    if (user != null) {
      p.put(Context.SECURITY_PRINCIPAL, user);
      if (password == null)
        password = "";
      p.put(Context.SECURITY_CREDENTIALS, password);
    }
    return new InitialContext(p);
  }
  static String url = "t3://localhost:7001";
  static String user = null;
  static String password = null;
}
```

คอมไพล์แล้วรันจะได้ผลดังนี้

```
Initial Balance = 0.0
After depositing 100, account balance = 100.0
Found account with ID 123-456.7890.  Balance = 100.0
Now trying to withdraw $150, which is more than is currently available.  This should generate an
```

```
exception..
olala.AccountException: Your balance is 100.0!  You cannot withdraw 150.0!
at weblogic.rmi.internal.AbstractOutboundRequest.sendReceive
(AbstractOutboundRequest.java:90)
at olala.AccountEJBEOImpl_WLStub.withdraw(AccountEJBEOImpl_WLStub.java:791)
at AccountClient.main(AccountClient.java:53)
Destroying account..
```

ที่เกิด exception ขึ้นเนื่องจากมีการถอนเงินเกินกว่าที่มี จึงเกิด AccountException ขึ้น

6.4 Container-Managed Persistent Entity Bean

เมธอดที่ต้องอิมพลีเมนต์ของ container-managed persistent ต่างจาก bean-managed persistent ดังตารางที่ 6-2

เมธอด	คำอธิบาย
setEntityContext() (เหมือนกับ bean-managed persistent)	เมื่อคอนเทนเนอร์สร้างอินสแทนซ์ของ bean แล้วจะเรียกเมธอดนี้
ejbFind<...>(<...>)	ไม่จำเป็นต้องอิมพลีเมนต์ finder method คอนเทนเนอร์จะจัดการให้ โดยวิธีในการระบุเพื่อให้คอนเทนเนอร์รู้วิธีแมปนั้นขึ้นอยู่กับผลิตภัณฑ์

ตารางที่ 6-2 คำอธิบายเมธอดใน Container-Managed Persistent Entity Bean

เมธอด	คำอธิบาย
EjbCreate(<...>)	เมื่อไคลเอ็นต์เรียกเมธอด create() กับ home interface คอนเทนเนอร์จะเรียกเมธอด ejbCreate() กับอินสแทนซ์ของ bean ที่พูลอยู่ ejbCreate() จะทำหน้าที่ในการสร้างข้อมูลใหม่ในฐานข้อมูลให้ เมธอดนี้เหมือนกับ bean-managed persistent แต่กับ container-managed persistent ไม่จำเป็นที่จะต้องอิมพลีเมนต์เมธอดนี้
ejbPostCreate(<...>) (เหมือนกับ bean-managed persistent)	bean class จะต้องกำหนดเมธอด ejbPostCreate() ให้เข้าคู่กับ ejbCreate() คือต้องมีพารามิเตอร์เหมือนกัน คอนเทนเนอร์จะเรียกเมธอดนี้หลังจากเรียก ejbCreate()
ejbActivate() (เหมือนกับ bean-managed persistent)	เมื่อไคลเอ็นต์ต้องการเรียกบิซิเนสเมธอดกับ EJB object แต่ไม่มีอินสแทนซ์ของ entity bean นั้นที่ bind กับ EJB object คอนเทนเนอร์จะนำ bean จากพูลและเปลี่ยนสถานะเป็น ready โดยเมธอดนี้จะไม่ถูกเรียก

6.4.1.2 Home Interface

โค้ดดังนี้

ProductHome.java

```
package olala;
import javax.ejb.EJBHome;
import javax.ejb.FinderException;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import java.util.Enumeration;
public interface ProductHome extends EJBHome{
  public Product findByPrimaryKey(String key)     throws FinderException, RemoteException;
  public Product create(String productId, String name, String description, double basePrice)
    throws CreateException, RemoteException;
  public Enumeration findAllProducts()     throws FinderException, RemoteException;
  public Enumeration findCheapProducts(double maxPrice)
    throws FinderException, RemoteException;
  public Enumeration findExpensiveProducts(double minPrice)
    throws FinderException, RemoteException;
  public Enumeration findByBasePrice(double basePrice)
    throws FinderException, RemoteException;
  public Enumeration findByDescription(String description)
    throws FinderException, RemoteException;
  public Enumeration findByName(String name)     throws FinderException, RemoteException;
}
```

6.4.1.3 Enterprise Bean Class

container-managed persistent ช่วยลดการโค้ดติดต่อด้วย JDBC เป็นจำนวนมาก โค้ดดังนี้

ProductEJB.java

```
package olala;
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.CreateException;
```

```
public class ProductEJB implements EntityBean{
  public String productId;
  public String name;
  public double basePrice;
  public String description;
  private EntityContext context;
  public ProductEJB()  { }
  public String getProductId() {  return(productId); }
  public String getName() {   return(name); }
  public void setName(String name) {   this.name = name; }
  public double getBasePrice() {   return(basePrice); }
  public void setBasePrice(double basePrice) {   this.basePrice = basePrice;  }
  public String getDescription() {  return(description); }
  public void setDescription(String description) {  this.description = description; }
  public void setEntityContext(EntityContext ec) {  this.context = ec;  }
  public void unsetEntityContext() {   this.context = null;  }
  public void ejbActivate()  { }
  public void ejbPassivate()  { }
  public void ejbLoad()  { }
  public void ejbStore()  { }
  public void ejbRemove()  { }
  public String ejbCreate(String productId, String name, String description, double basePrice)
    throws CreateException  {
    this.productId = productId;
    this.name = name;
    this.basePrice = basePrice;
    this.description = description;
    return null;
  }
  public void ejbPostCreate(String productId, String name, String description, double basePrice) { }
}
```

6.4.1.4 Deployment Descriptor

container-managed persistent จะต้องมีไฟล์ descriptor เพิ่มจากเดิม โดยไฟล์นั้นจะเป็นไฟล์ที่ใช้แมปข้อมูลจากออบเจ็กต์ไปเป็นข้อมูลในฐานข้อมูล โดยอาจจะมีไฟล์ที่ใช้แมปนั้นมากกว่าหนึ่งไฟล์ก็ได้

ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
  <enterprise-beans>
    <entity>
      <description></description>
      <ejb-name>Product</ejb-name>
      <home>olala.ProductHome</home>
      <remote>olala.Product</remote>
      <ejb-class>olala.ProductEJB</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-field>
        <field-name>description</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>basePrice</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>name</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>productId</field-name>
      </cmp-field>
      <primkey-field>productId</primkey-field>
    </entity>
  </enterprise-beans>
```

```
  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>Product</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

weblogic-ejb-jar.xml

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN'
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>Product</ejb-name>
    <persistence-descriptor>
      <persistence-type>
        <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
        <type-version>5.1.0</type-version>
        <type-storage>META-INF/weblogic-cmp-rdbms-jar-Product.xml</type-storage>
      </persistence-type>
      <persistence-use>
        <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
        <type-version>5.1.0</type-version>
      </persistence-use>
    </persistence-descriptor>
    <jndi-name>Product</jndi-name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

weblogic-ejb-jar.xml จะแมปข้อมูลโดยใช้ข้อมูลในไฟล์ตามที่ระบุในแท็ก type-storage

weblogic-cmp-rdbms-jar-Product.xml

```
<?xml version="1.0"?>
<!DOCTYPE weblogic-rdbms-bean PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB
RDBMS Persistence//EN' 'http://www.bea.com/servers/wls510/dtd/weblogic-rdbms-persistence.dtd'>
<weblogic-rdbms-bean>
  <pool-name>OlalaPool</pool-name>
  <table-name>products</table-name>
  <attribute-map>
    <object-link>
      <bean-field>description</bean-field>
      <dbms-column>description</dbms-column>
    </object-link>
    <object-link>
      <bean-field>basePrice</bean-field>
      <dbms-column>basePrice</dbms-column>
    </object-link>
    <object-link>
      <bean-field>name</bean-field>
      <dbms-column>name</dbms-column>
    </object-link>
    <object-link>
      <bean-field>productId</bean-field>
      <dbms-column>productId</dbms-column>
    </object-link>
  </attribute-map>
  <finder-list>
    <finder>
      <method-name>findAllProducts</method-name>
      <finder-query>
        <![CDATA[(= 1 1)]]>
      </finder-query>
```

```
</finder>
<finder>
  <method-name>findCheapProducts</method-name>
  <method-params>
    <method-param>double</method-param>
  </method-params>
  <finder-query>
    <![CDATA[(< basePrice $0)]]>
  </finder-query>
</finder>
<finder>
  <method-name>findExpensiveProducts</method-name>
  <method-params>
    <method-param>double</method-param>
  </method-params>
  <finder-query>
    <![CDATA[(> basePrice $0)]]>
  </finder-query>
</finder>
<finder>
  <method-name>findByBasePrice</method-name>
  <method-params>
    <method-param>double</method-param>
  </method-params>
  <finder-query>
    <![CDATA[(= basePrice $0)]]>
  </finder-query>
</finder>
<finder>
  <method-name>findByDescription</method-name>
  <method-params>
    <method-param>java.lang.String</method-param>
  </method-params>
```

```
        <finder-query>
          <![CDATA[(= description $0)]]>
        </finder-query>
      </finder>
      <finder>
        <method-name>findByName</method-name>
        <method-params>
          <method-param>java.lang.String</method-param>
        </method-params>
        <finder-query>
          <![CDATA[(= name $0)]]>
        </finder-query>
      </finder>
    </finder-list>
</weblogic-rdbms-bean>
```

ไฟล์นี้จะใช้พูลของฐานข้อมูลในการสร้าง connection โดยมีชื่อว่า OlalaPool ที่ระบุในแท็ก pool-name และชื่อเทเบิล products ที่ระบุในแท็ก table-name รวมถึงการแมปข้อมูลจากชื่อของคอลัมน์ไปเป็นชื่อของตัวแปร ส่วน find method จะใช้สคริปต์ในการค้นหา โดยสคริปต์ (= name $0) หมายถึง where name = พารามิเตอร์ตัวแรก และ (= 1 1) หมายถึงค้นหาทั้งหมด โดยวิธีการนี้จะแตกต่างกันไปตามแต่ละผลิตภัณฑ์

6.4.1.5 คอมไพล์และแพ็กเก็จ

คอมไพล์จาวาไฟล์ จัดไฟล์ทั้งหมดอยู่ใน path ดังนี้

./Produc/olala/Product.class

./Product/olala/ProductEJB.class

./Product/olala/ProductHome.class

./Product/meta-inf/ejb-jar.xml

./Product/meta-inf/weblogic-ejb-jar.xml

./Product/meta-inf/ weblogic-cmp-rdbms-jar-Product.xml

สร้าง jar ไฟล์โดยใช้คำสั่งดังนี้

./Product>jar cv0f stdProduct.jar olala meta-inf

และทำคำสั่งต่อไปนี้

./Product/java weblogic.ejbc –compiler javac stdProduct.jar Product.jar

แล้วจึงนำ Product.jar ไปดีพลอย

6.4.1.6 ไคลเอ็นต์

เซต classpath ชี้ไปยัง Product.jar หรือ path ที่มีอินเทอร์เฟซของ bean โค้ดไคลเอ็นต์ดังนี้

ProductClient.java

```
import olala.*;


import javax.ejb.*;
import javax.naming.*;
import java.util.*;
public class ProductClient {
  public static void main(String[] args){
    ProductHome home=null;
    try {
      Context ctx = getInitialContext();
      home = (ProductHome) ctx.lookup("Product");
      home.create("123-456.7890","P6.250","350 Mhz Pentium",200);
      home.create("123-456.7891","P5-400","400 Mhz Pentium",300);
      home.create("123-456.7892","P5-450","450 Mhz Pentium",400);
      home.create("123-456.7893","SD-64","64 MB SDRAM",50);
      home.create("123-456.7894","SD-128","128 MB SDRAM",100);
      home.create("123-456.7895","SD-256","256 MB SDRAM",200);
      Enumeration enum=home.findByName("SD-64");
      System.out.println("The following product descriptions match the product name SD-64:");
      while (enum.hasMoreElements()){
        Product prod=(Product) enum.nextElement();
        System.out.println(prod.getDescription());
      }
      System.out.println("Calling finder to find all products that cost $200");
      enum=home.findByBasePrice(200);
      while (enum.hasMoreElements()){
        Product prod=(Product) enum.nextElement();
        System.out.println(prod.getDescription());
```

```
    }
  }
  catch (Exception e) {
    e.printStackTrace();
  }
  finally{
    if (home!=null){
      try{
        System.out.println("Destroying products..");
        Enumeration enum=home.findAllProducts();
        while (enum.hasMoreElements()){
          try{
            Product prod=(Product)enum.nextElement();
            prod.remove();
          }
          catch (Exception e){
            e.printStackTrace();
          }
        }
      }
      catch (Exception e){
        e.printStackTrace();
      }
    }
  }
}
public static Context getInitialContext()  throws NamingException
{
  Properties p = new Properties();
  p.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
  p.put(Context.PROVIDER_URL, url);
  if (user != null) {
    p.put(Context.SECURITY_PRINCIPAL, user);
```

```
        if (password == null)
          password = "";
        p.put(Context.SECURITY_CREDENTIALS, password);
      }
      return new InitialContext(p);
    }
    static String url = "t3://localhost:7001";
    static String user = null;
    static String password = null;
}
```

คอมไพล์แล้วรันได้ผลดังนี้

```
The following product descriptions match the product name SD-64:
64 MB SDRAM
Calling finder to find all products that cost $200
350 Mhz Pentium
256 MB SDRAM
Destroying products..
```

6.5 เปรียบเทียบ BMP กับ CMP Entity Bean

การเลือกระหว่าง entity bean 2 ชนิดนี้ขึ้นอยู่กับงานที่จะทำ เนื่องจากทั้ง 2 ชนิดนี้มีข้อดีและข้อเสียต่างกัน โดยจะเปรียบเทียบในหัวข้อต่าง ๆ ดังนี้

1) จำนวนโค้ดและเวลาในการพัฒนา

container-managed persistent มีจำนวนโค้ดน้อยกว่าและพัฒนาได้เร็วกว่า เนื่องจากไม่ต้องเขียนโค้ดแมปข้อมูลด้วย JDBC

2) Bugs

การดีบัก container-managed persistent ยากกว่า bean-managed persistent เนื่องจาก bean-managed persistent เป็นการควบคุมโค้ด JDBC ทั้งหมดทำให้ง่ายต่อการตรวจสอบโค้ดเหล่านั้น แต่ container-managed persistent จะช่วยลดการเกิดบักได้มากกว่า bean-managed persistent ที่ต้องเขียนโค้ดเองทั้งหมด

3) การควบคุม

bean-managed persistent ควบคุมผ่านทาง JDBC จึงมีความยืดหยุ่นในการแมปออบเจ็กต์ไปยังฐานข้อมูล สำหรับ container-managed persistent ขึ้นอยู่กับว่าคอนเทนเนอร์รองรับการแมปที่ซับซ้อนแค่ไหน

4) ความเป็นอิสระจากแอพพลิเคชันเซิร์ฟเวอร์และฐานข้อมูล

container-managed persistent เป็นอิสระจากฐานข้อมูล เนื่องจากไม่มีการโค้ด JDBC สามารถจะเปลี่ยนแปลงได้ง่าย โดยอาจเปลี่ยนจากฐานข้อมูลเชิงสัมพันธ์เป็นฐานข้อมูลเชิงวัตถุโดยไม่ต้องแก้ไขโค้ดใด ๆ แต่การแมปจากออบเจ็กต์ไปยังฐานข้อมูลไม่มีมาตรฐานที่แน่นอน จึงไม่ portable กับแอพพลิเคชันเซิร์ฟเวอร์ใด ๆ ส่วน bean-managed persistent นั้น portable กับแอพพลิเคชันเซิร์ฟเวอร์และฐานข้อมูลที่เป็นกลางคือรองรับ SQL มาตรฐาน

5) การเรียนรู้และค่าใช้จ่าย

นักพัฒนาส่วนใหญ่จะมีความเข้าใจในการทำงานกับฐานข้อมูลเชิงสัมพันธ์ ดังนั้นการใช้ bean-managed persistent จึงไม่ต้องการการเรียนรู้ แต่กับ container-managed persistent จะต้องมีการเรียนรู้วิธีการแมปของผลิตภัณฑ์ใด ๆ ที่ใช้ รวมถึงค่าใช้จ่ายในการเรียนรู้วิธีการแมปในแต่ละผลิตภัณฑ์ของแอพพลิเคชันเซิร์ฟเวอร์

บทที่ 7

Security

Security หมายถึง เทคนิคในการที่จะมั่นใจได้ว่าข้อมูลที่ถูกเก็บอยู่ในคอมพิวเตอร์ หรือข้อมูลที่ถูกส่งผ่านระหว่างเครื่องคอมพิวเตอร์จะไม่ถูกเปลี่ยนแปลง โดยหลักการวัด security โดยทั่วไปนั้นส่วนใหญ่จะเกี่ยวกับการ พิสูจน์ข้อความ และการเข้ารหัส

การส่งข้อมูลใน e-comerce นั้นจะมี access point หลายจุดซึ่งผู้ประสงค์ร้ายจะสามารถดักจับ และเปลี่ยนแปลงข้อมูลได้ โดยเฉพาะยิ่งธุรกิจมีการกระจายตัวมากเท่าไรก็จะยิ่งเพิ่มโอกาสที่จะโดนโจมตีได้มากขึ้นจึงมีความต้องการ software ที่จะมาช่วยจัดการในด้านนี้

โดยในตอนต้นของบทนี้จะพูดถึงทฤษฎีที่เกี่ยวข้องกับการทำ Security ก่อน

7.1 Cipher Suite

Cipher Suite คือ กระบวนการ encryption ของ SSL ที่รวม key exchange algorithm,symmetric encryption และ secure hash algorithm เข้าด้วยกัน ตัวอย่างเช่น Cipher Suite ที่เรียกว่า RSA_WITH_RC4_128MD5 ใช้ RSA สำหรับ key exchange , RC4 ที่ใช้ 128-bit key ในการ encryption และ MD5 สำหรับการทำ Message digest

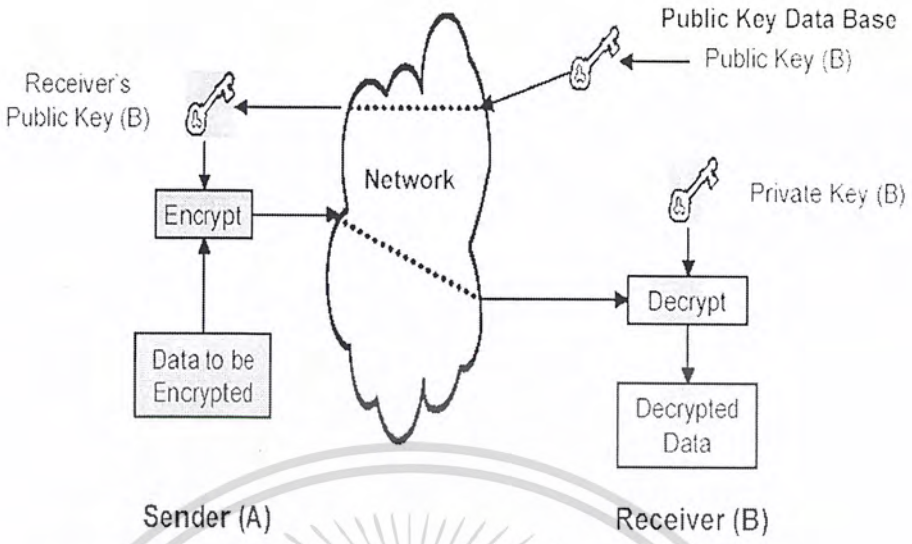
7.2 Public Key Algorithms

Public key(or asymmetric key)algorithms เป็นการใช้ key 2 key ที่แตกต่างกัน แต่มีความสัมพันธ์กันทางคณิตศาสตร์

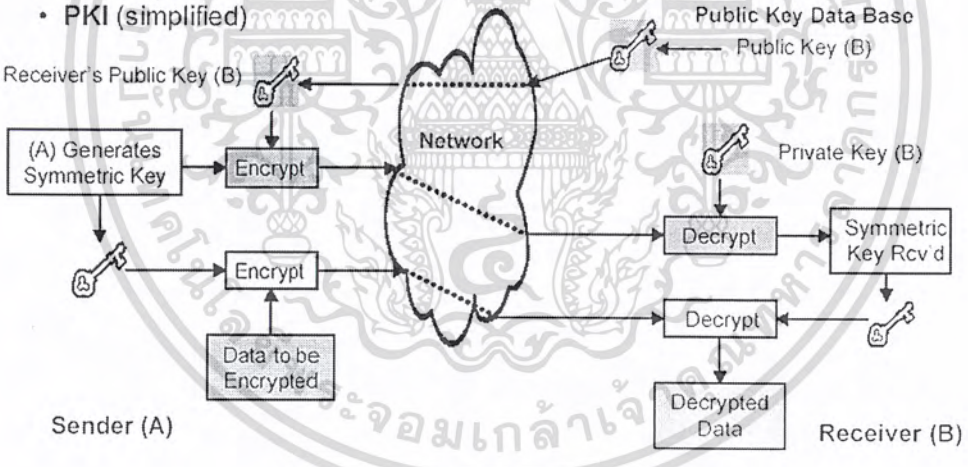
- A public key(ที่ซึ่งถูกแจกจ่ายโดยทั่วไป) ใช้สำหรับการยืนยัน digital signatureว่าเป็นของผู้ส่งจริง หรือ ใช้ในการเปลี่ยนข้อมูลให้อยู่ในรูปที่ไม่สามารถเข้าใจได้
- A private key(ที่ซึ่งถูกเก็บเป็นความลับ) ใช้ในการสร้าง digital signature หรือ เปลี่ยนข้อมูลที่ถูกส่งมาให้กลับอยู่ในรูปเดิม

7.3 Symmetric Key Algorithms

ใน symmetric key algorithms จะใช้ key เดียวกันทั้ง encrypt และ decrypt message โดยการใช้วิธีนี้จะเร็วกว่าแบบ public key cryptography ประมาณ 1,000 เท่า



รูปที่ 7-1 แสดงการใช้คีย์คู่ในการติดต่อ

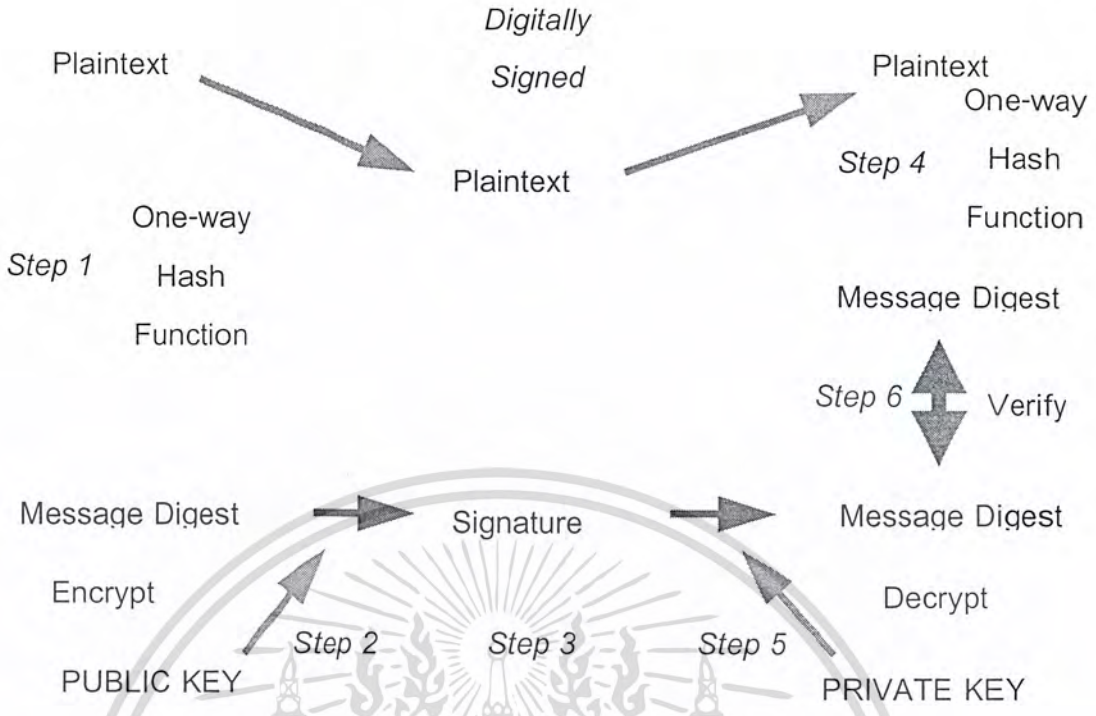


รูปที่ 7-2 แสดงการได้มาซึ่งคีย์เดี่ยวในการติดต่อ

7.4 Message Digest Algorithms

สนับสนุน message digest algorithm แบบ MD5 และ SHA(Secure Hash Algorithm) ซึ่งเป็น one-way hash algorithm โดย one-way hash algorithm จะ convert message ให้อยู่ในรูปแบบ string ที่ fix length ซึ่งเรียกว่า message digest หรือ hash value

MD5 จะเป็น high-speed 128-bit hash ส่วน SHA จะมีความปลอดภัยมากกว่าโดยใช้ 160-bit hash แต่จะช้ากว่า MD5



รูปที่ 7-3 แสดงการนำ Message Digest ไปใช้ใน public-private key encyption

7.5 ลายมือชื่อดิจิตอล

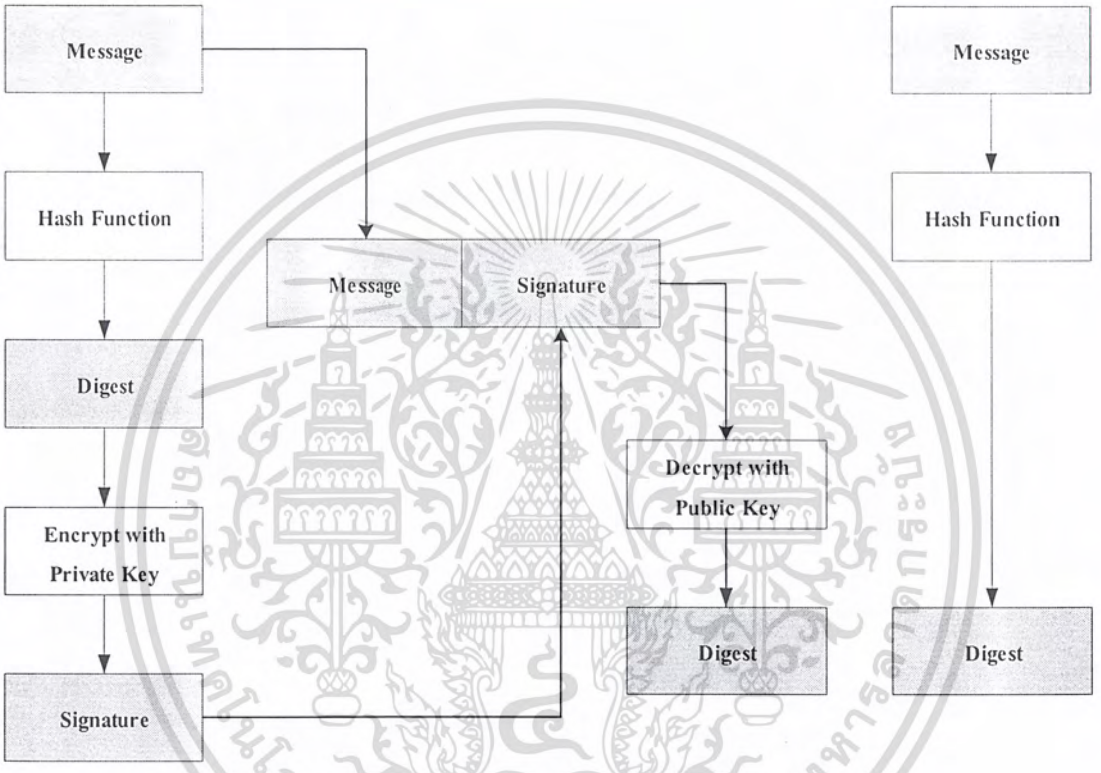
ลายมือชื่อดิจิตอลใช้เมื่อต้องการความมั่นใจในแหล่งที่มาของเอกสาร เปรียบเสมือนลายมือชื่อซึ่งเฉพาะเจ้าของจริงที่สามารถคำนวณขึ้นได้ แต่ลายมือชื่อนี้สามารถพิสูจน์ได้ กล่าวคือบุคคลอื่นสามารถตรวจสอบได้ว่าลายมือชื่อนั้นมาจากผู้สร้างจริงๆ วิธีที่ใช้ในการทำลายมือชื่อดิจิตอลที่มีความปลอดภัยค่อนข้างสูงก็คือ การใช้ public-private key โดยเข้ารหัสค่าลายมือชื่อด้วยคีย์ส่วนตัว และคนอื่นสามารถใช้คีย์สาธารณะพิสูจน์ได้ว่าลายมือชื่อมาจากคีย์ส่วนตัวที่ตรงกัน

คุณสมบัติที่สำคัญของลายมือชื่อดิจิตอลที่สำคัญนั้นจะต้องประกอบด้วย 2 ประการคือ

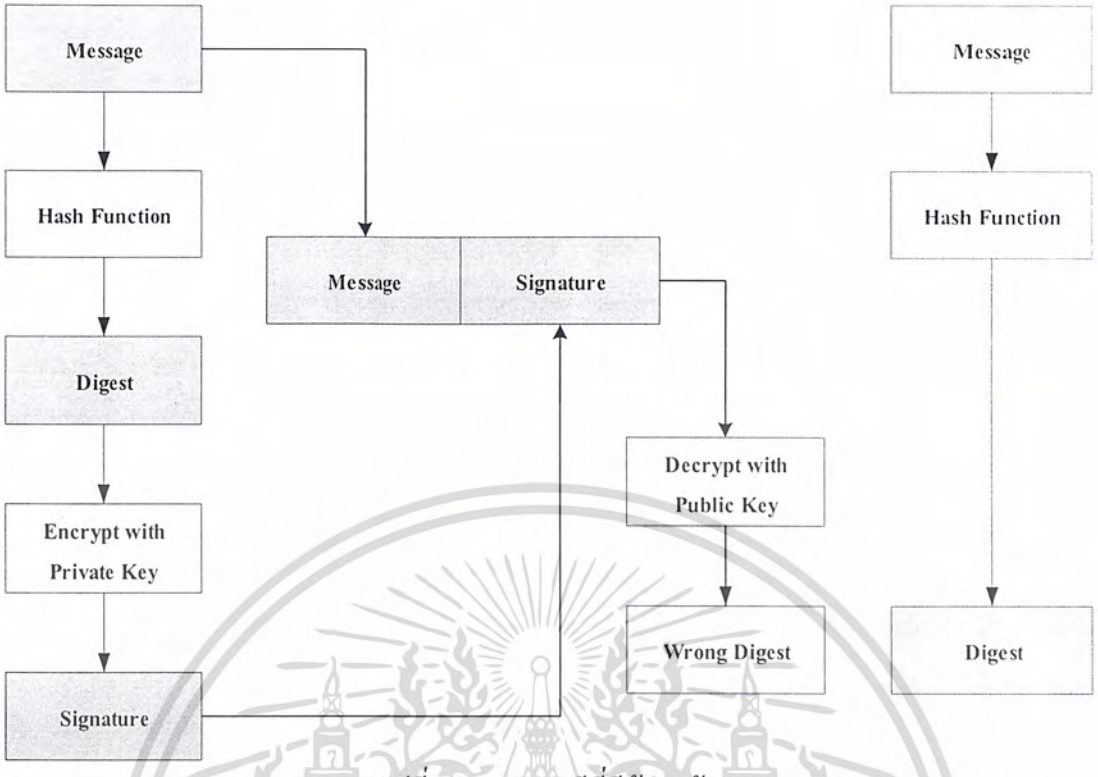
- สามารถยืนยันได้ว่าข้อมูลที่รับมานั้นไม่มีการเปลี่ยนแปลงระหว่างการส่ง
- สามารถยืนยันได้ว่าข้อมูลนั้นได้รับการยืนยันจากผู้ลงลายมือชื่อจริงๆ

การทำลายมือชื่อดิจิตอลนั้นทำได้โดยนำข้อความแรกเริ่มไปเข้าแฮชฟังก์ชันจะได้เป็นเมสเซจไดเจสต์เพื่อเป็นการยืนยันความถูกต้องของข้อมูลก่อน จากนั้นจะนำเมสเซจไดเจสต์ที่ได้ส่งไปเข้ารหัสกับคีย์ส่วนตัวของผู้ส่งเพื่อเป็นการยืนยันว่าเป็นเจ้าของเอกสารนั้นจริง และจะได้สิ่งที่เรียกว่าลายมือชื่อดิจิตอลออกมาส่งไปพร้อมกับข้อมูลด้วย ทางด้านรับนั้นจะเป็นวิธีการที่คล้ายกับด้านส่งโดยจะต้องเริ่มจากการหาเมสเซจไดเจสต์ของข้อมูลที่ได้รับมาเก็บไว้ก่อน จากนั้นจะนำลายมือชื่อดิจิตอลที่ได้รับมาด้วย มาทำการถอดรหัสด้วยคีย์สาธารณะของผู้ส่ง ในขั้นตอนนี้เองที่จะสามารถยืนยันบุคคลผู้ส่งได้ เพราะถ้าหากผู้ที่ลงลายมือชื่อไม่ได้เป็นบุคคลที่เราคาดว่าเป็นเจ้าของ ก็จะไม่สามารถถอดลายมือดิจิตอลได้ ถ้าหากการยืนยัน

บุคคลผู้ส่งสำเร็จแล้วจะได้รับเมสเซจไดเจสต์ของข้อมูลก่อนจะส่งออกมา ด้านรับจะต้องทำการตรวจสอบความถูกต้องของข้อมูลโดยการนำเมสเซจไดเจสต์ที่คำนวณได้ในตอนรับข้อมูล กับเมสเซจไดเจสต์ที่ได้หลังจากการถอดรหัสลายมือชื่อดิจิตอลออกมา นำมาเปรียบเทียบกัน ถ้าหากเมสเซจไดเจสต์ที่ได้ออกมามีค่าเท่ากันนั่นแสดงว่าข้อมูลที่ได้รับนั้นเป็นข้อมูลเดียวกันจากทางด้านส่ง และไม่มีการเปลี่ยนแปลงข้อมูลระหว่างการส่ง ขั้นตอนการส่ง และการรับของวิธีการนี้ แสดงไว้ในรูปที่ 7-4 โดยในกรณีที่มีการปลอมแปลงจะเป็นดังรูปที่ 7-5



รูปที่ 7-4 แสดงวิธีการทำลายมือชื่อโดยใช้แฮชฟังก์ชัน และคีย์ต่างๆ



รูปที่ 7-5 แสดงกรณีที่มีผู้แอบอ้าง

เมื่อมีผู้แอบอ้าง ผู้แอบอ้างไม่รู้คีย์ส่วนตัวของผู้ส่งจริงๆ จึงใช้คีย์ของตัวเอง ซึ่งไม่เข้าคู่กับคีย์สาธารณะทางฝั่งผู้รับ ทำให้ทางฝั่งผู้รับไม่สามารถถอดรหัสลายมือชื่อดิจิตอลได้ จึงรู้ว่าไม่ใช่ผู้ส่งตัวจริงเป็นคนส่ง

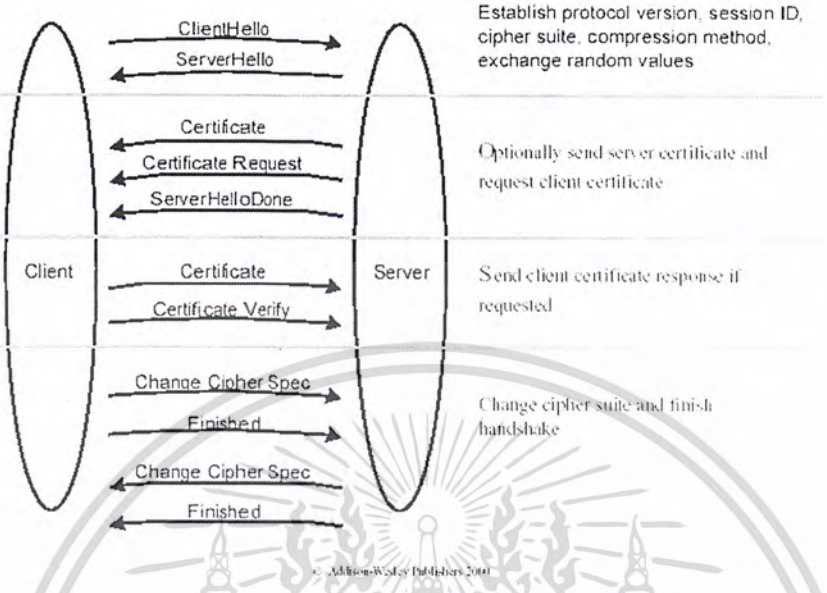
7.6 SSL Protocol

SSL Protocol สร้างความปลอดภัยให้กับ application ที่ทำการติดต่อผ่าน network ซึ่งมีรายละเอียดดังนี้

- กลไกในการที่แต่ละ application สามารถใช้ยืนยันหลักฐานของ application อื่นได้
- เข้ารหัสข้อมูลที่ใช้ในการแลกเปลี่ยนระหว่าง application

เมื่อ SSL Protocol ถูกใช้ target จะทำการยืนยันตัวเองกับ initiator ถ้า target request initiator จะสามารถยืนยันตัวเองกับ target ได้ SSL connection จะเริ่มด้วยการ handshake ระหว่าง application จะ แลกเปลี่ยน didital certificate กัน ตกลง encryption algorithm ที่จะใช้ และ สร้าง encryption key ที่จะใช้ตลอดช่วงการติดต่อ

- SSL Protocol Handshake Phases



รูปที่ 7-6 แสดงทรีเวย์แฮนด์เชค

7.7 เอกสารสิทธิ์(Digital Certificates)

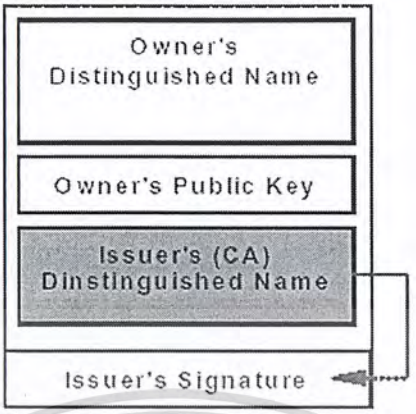
ลักษณะของเอกสารสิทธิ์ดิจิตอล

เอกสารสิทธิ์เป็นเสมือนกับบัตรประจำตัวประชาชน ซึ่งจะบ่งบอกรายละเอียดของบุคคล เพื่อใช่ในการยืนยันตัวตน เราสามารถส่งเอกสารสิทธิ์ไปควบคู่กับลายมือชื่อดิจิตอลเพื่อใช่ในการยืนยันว่าเอกสารสิทธิ์นั้นไม่มีการเปลี่ยนแปลง การสร้างเอกสารสิทธิ์ทำโดยผู้ใช้ทุกคนทำการขอเอกสารสิทธิ์กับองค์กรพิสูจน์สิทธิ์

(Certificate Authority : CA) โดยการส่งคีย์สาธารณะและข้อมูลตามที่องค์กรพิสูจน์สิทธิ์นั้นกำหนดไปและทำการขอเอกสารสิทธิ์มา การติดต่อต้องทำเป็นการส่วนตัวหรือติดต่อผ่านระบบการพิสูจน์ตัวตนที่ปลอดภัย เราสามารถกำหนดสิ่งที่จำเป็นในการสื่อสารได้ดังต่อไปนี้

1. ผู้ใช้ทุกคนสามารถคำนวณหาชื่อและคีย์สาธารณะของเจ้าของเอกสารสิทธิ์ได้
2. ผู้ใช้ทุกคนสามารถตรวจสอบได้ว่าเอกสารสิทธิ์มาจากองค์กรพิสูจน์สิทธิ์จริงๆไม่ได้ถูกปลอมแปลงมา
3. ผู้ใช้สามารถตรวจสอบได้ว่าเอกสารสิทธิ์นั้นหมดอายุหรือไม่
4. ผู้ที่สามารถอ้างและอัพเดต(update)เอกสารสิทธิ์ได้มีเพียงองค์กรมีอำนาจในการรับรองสิทธิ์เท่านั้น

5. ผู้ใช้ทุกคนสามารถตรวจสอบเอกสารสิทธิ์ได้เป็นประจำ



รูปที่ 7-7 แสดงส่วนประกอบต่างๆของเอกสารสิทธิ์

ความสำคัญของเอกสารสิทธิ์

ในการเข้ารหัสแบบคีย์ต่างๆนั้นเราจะต้องมีการสร้างทั้งคีย์ส่วนตัวและคีย์สาธารณะ ซึ่งโดยทั่วไปการสร้างคีย์จะทำโดยโปรแกรมที่จะใช้คีย์นั้น เช่น โปรแกรมเว็บบราวเซอร์หรือโปรแกรมติดต่ออิเล็กทรอนิกส์เมล์ หลังจากที่สร้างคีย์ทั้งสองเรียบร้อยแล้วการเก็บรักษาคีย์เป็นเรื่องสำคัญ เราจะต้องเก็บรักษาคีย์ส่วนตัวไว้ให้ดีอย่าให้ใครแอบมาเห็นหรือขโมยไปได้ จากนั้นจะเป็นการตัดสินใจว่าจะทำการแจกจ่ายคีย์สาธารณะของเราไปสู่ผู้อื่นด้วยวิธีใด เช่นการแจกคีย์โดยส่งอิเล็กทรอนิกส์เมล์ไปให้เพื่อนหรือบุคคลที่ติดต่อกับเรา แต่วิธีนี้เราอาจส่งคีย์ไปให้ไม่ได้ครบทุกคน และยังต้องเป็นภาระการจัดส่งคีย์ให้กับบุคคลใหม่ๆที่ต้องการติดต่อด้วย นอกจากนี้ยังไม่สามารถทำให้ผู้รับมั่นใจได้ว่าคีย์ที่ส่งไปนั้นเป็นคีย์ของเราจริงๆ เนื่องจากอาจมีผู้อื่นแอบสร้างคีย์โดยใช้ชื่อเราและแอบอ้างส่งคีย์ดังกล่าวให้กับผู้อื่นเพื่อให้เข้าใจว่าเป็นคีย์ของเราได้

วิธีที่ดีกว่าและใช้อยู่ในปัจจุบันก็คือการใช้ระบบแจกจ่ายคีย์ที่เชื่อถือได้โดยจะมีองค์กรที่ทำหน้าที่เฉพาะเป็นองค์กรที่สาม(Third Party) ในการรับรองและระงับการรับรองคีย์ที่เรียกว่าองค์กรพิสูจน์สิทธิ์(Certificate Authority - CA) โดยองค์กรพิสูจน์สิทธิ์นี้จะตรวจสอบคีย์สาธารณะของเราด้วยหลักฐานว่าคีย์ของเรานั้นๆเป็นของจริง พร้อมทั้งตรวจสอบข้อมูลส่วนตัวของเรา(ข้อมูลที่ตรวจสอบจะมากน้อยแค่ไหนขึ้นอยู่กับระดับชั้นของการรับรอง) เมื่อผู้อื่นได้รับคีย์ของเราก็สามารถที่จะตรวจสอบกับผู้ออกเอกสารสิทธิ์นี้ว่าคีย์ที่ได้รับเป็นของจริงหรือไม่ ซึ่งตัวเอกสารสิทธิ์นี้จะเปรียบเสมือนบัตรประชาชนดิจิตอลของเราที่ใช้บอกได้ว่าเราเป็นบุคคลที่อ้างจริงๆในระบบเครือข่ายหรือการส่งข้อมูลทางอิเล็กทรอนิกส์

ในปัจจุบันบริษัทที่ออกเอกสารสิทธิ์ดิจิตอล(Digital Certificate) คือบริษัท Verisign , Cyvertrust , Global Sign และ Nortel โดยในเอกสารสิทธิ์ดิจิตอลจะประกอบด้วยข้อมูลต่างๆ ได้แก่ ชื่อของผู้ถือเอกสารสิทธิ์ ชื่อของบริษัทผู้ออกเอกสารสิทธิ์ คีย์สาธารณะของผู้ถือเอกสารสิทธิ์ วันหมดอายุของผู้ถือ

เอกสารสิทธิ์(โดยทั่วไปจะกำหนดระยะเวลา 6 เดือนหรือ 1 ปี) ระดับชั้นของเอกสารสิทธิ์ และเลขหมายของตัวเอกสารสิทธิ์ดิจิตอล

เอกสารสิทธิ์ดิจิตอลแบ่งออกได้เป็นสี่ระดับชั้นตามระดับการตรวจสอบข้อมูลของเจ้าของเอกสารสิทธิ์

1. ระดับที่หนึ่ง เป็นชั้นที่ออกเอกสารสิทธิ์ได้ง่ายที่สุดเนื่องจากมีการตรวจสอบน้อยที่สุด โดยจะตรวจแค่ชื่อผู้ถือเอกสารสิทธิ์ และที่อยู่อิเล็กทรอนิกส์เมล์(e-mail address) ว่าถูกต้องจริงเท่านั้น
2. ระดับชั้นที่สอง จะตรวจสอบเลขประจำตัวประชาชน เลขประจำตัวของระบบสวัสดิการหรือประกันสังคม(social security number) และวันเดือนปีเกิด
3. ระดับชั้นที่สาม จะมีการตรวจสอบเพิ่มเติมเกี่ยวกับประวัติการใช้เครดิต และการชำระเงิน
4. ระดับชั้นที่สี่ ยังไม่มีการออกมาเป็นมาตรฐานอย่างแน่ชัด แต่จะเป็นการตรวจสอบข้อมูลเพิ่มเติมเกี่ยวกับตำแหน่งงานในองค์กรด้วย

ในการขอเอกสารสิทธิ์นั้นจะต้องกระทำบนกระบวนการที่ปลอดภัย ซึ่งในปัจจุบันมี 2 วิธีคือ

1. การขอเอกสารสิทธิ์ผ่านโปรแกรมประเภทบราวเซอร์ เช่น อินเตอร์เน็ตเอ็กซ์พลอเรอร์, เนสเคปท์ เป็นต้น โดยผู้ใช้เข้าไปยังโฮสต์ที่ให้บริการการขอเอกสารสิทธิ์หรือเว็ปไซต์ที่เปิดให้บริการดังกล่าวอยู่ ตัวโปรแกรมประเภทบราวเซอร์จะทำการสร้างคู่คีย์ คีย์ส่วนตัวจะเก็บไว้ที่เครื่องของเรา จากนั้นจะส่งคีย์สาธารณะและข้อมูลส่วนตัวของเราไปยังโฮสต์หรือไซต์ที่เป็นขององค์กรพิสูจน์ จากนั้นองค์กรพิสูจน์สิทธิ์จะทำการรับรองข้อมูลแล้วจึงส่งเอกสารสิทธิ์กลับมาที่เครื่องของเรา
2. การขอเอกสารสิทธิ์อีกรูปแบบหนึ่งคือ การส่งเอกสารสิทธิ์ที่ไม่ผ่านระบบเครือข่าย กล่าวคือ ผู้ขอต้องไปทำการขอที่สำนักงานขององค์กรพิสูจน์สิทธิ์โดยตรง โดยทำเรื่องของเอกสารสิทธิ์และกรอกข้อมูลส่วนตัว องค์กรพิสูจน์สิทธิ์จะทำการรับรองและบันทึกข้อมูลของผู้ขอแล้วทำการส่งเอกสารสิทธิ์พร้อมทั้งคีย์ส่วนตัวให้กับผู้ขอในรูปแบบของไฟล์ในแผ่นดิสก์ให้ผู้ขอนำไปบันทึกในเครื่องของตนเองต่อไป

บริการพิสูจน์สิทธิ์แบบ X.509 (X.509 Authentication Service)

ระบบ X.509 เป็นระบบพิสูจน์สิทธิ์ที่สำคัญมากในระบบเครือข่าย โดย X.509 เป็นอนุกรมย่อยของ X.500 ซึ่งกำหนดมาตรฐาน ITU-T โดยขณะที่ X.500 เป็นตัวกำหนดโครงสร้างในลักษณะที่เป็นไดเรกทอรีหรือโครงสร้างนั้น X.509 จะทำหน้าที่ในการพิสูจน์สิทธิ์ให้กับส่วนต่าง ๆ ของไดเรกทอรีนั้น สำหรับรูปแบบการใช้งานจะเน้นไปที่การพิสูจน์บุคคล เพื่อยืนยันการติดต่อเป็นสำคัญ

การทำงานของ X.509 จะมีโครงสร้างการทำงานที่เป็นไดเรกทอรี โดยในที่นี้ไดเรกทอรี

จะทำหน้าที่เป็นที่เก็บข้อมูลที่ใช้ในการยืนยัน ซึ่งโดยทั่วไปจะอยู่ในรูปของเอกสารสิทธิ์ซึ่งในเอกสารสิทธิ์จะบรรจุคีย์สาธารณะของผู้ใช้ที่เข้ารหัสโดยคีย์ส่วนตัวขององค์กรที่จ่ายใบเอกสารสิทธิ์มาให้ สำหรับการทำงานของ X.509 นั้นจะมีขอบเขตการนำไปใช้งานที่กว้างขวางมาก เช่น ใช้ในการทำ Mail Security ใช้ในการทำ IP Security ใช้ในการทำ Web Security หรือหากจะกล่าวว่า เมื่อใดที่ต้องการพิสูจน์บุคคลหรือยืนยันเครื่องคอมพิวเตอร์แล้ว ก็มักจะอยู่ในขอบข่ายการทำงานของ X.509 เสมอ

X.509 ได้ถูกนำเสนอเมื่อปี 1988 จากนั้นได้ผ่านการปรับปรุงเป็นลำดับขั้น ในประเด็นต่าง ๆ รวมทั้งเรื่องความปลอดภัยด้วย จากนั้นก็ได้ออกมาเป็นข้อเสนอที่ปรับปรุงแล้วในปี 1993 และปรับปรุงอีกครั้งในปี 1995 โดยการทำงานของ X.509 จะใช้การเข้ารหัสแบบคีย์สาธารณะแล้วใช้มาตรฐานลายมือชื่อดิจิตอลในการรับรองข้อมูล สำหรับอัลกอริธึมนั้น ไม่ได้ระบุแน่นอนโดยสามารถเลือกใช้ได้หลายตัวแต่ที่แนะนำคืออาร์เอสเอ

รูปแบบทั่วไปของเอกสารสิทธิ์

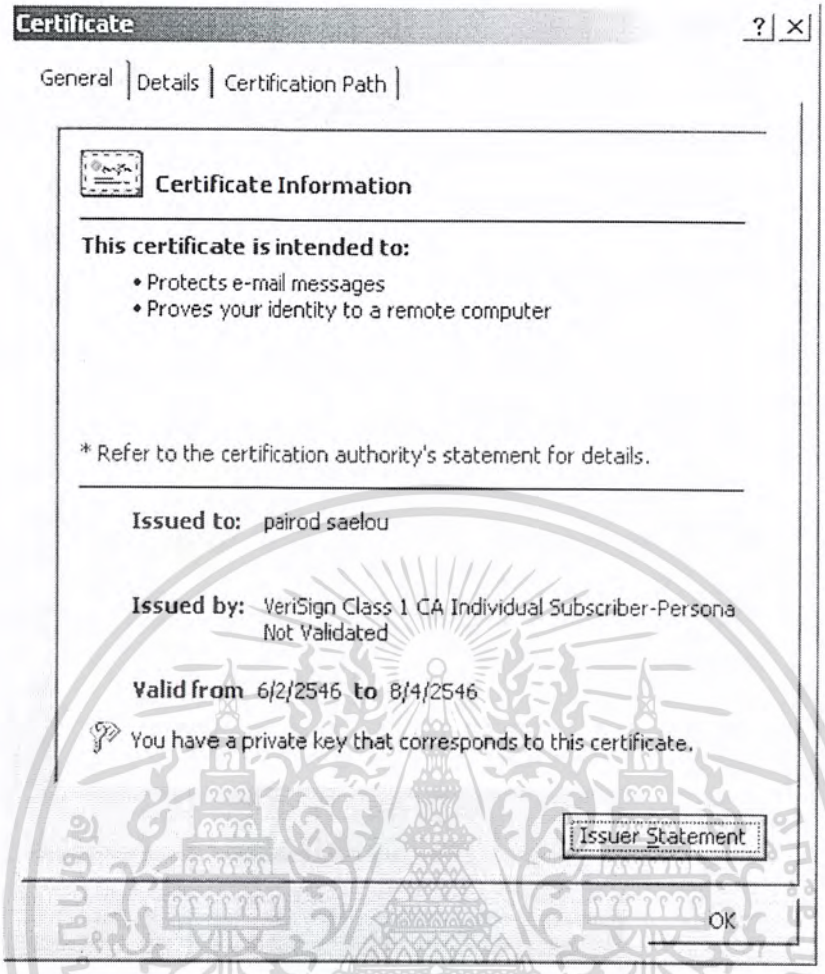
- เวอร์ชัน (Version) แสดงหมายเลขเวอร์ชัน เพราะในแต่ละเวอร์ชันจะมีรูปแบบของข้อมูลที่ไม่เหมือนกันก็ได้ โดยปกติจะเป็นเวอร์ชัน 1 แต่หากในเอกสารสิทธิ์มีการใช้
- หมายเลขลำดับ (Serial Number) เป็นเลขจำนวนเต็ม โดยจะต้องไม่ซ้ำกันในองค์กรที่ออกเอกสารสิทธิ์ โดยเลขนี้จะเป็นเลขที่จะใช้อ้างถึงแต่ละเอกสารสิทธิ์ที่สร้างขึ้นมา
- อัลกอริธึมที่ใช้สร้าง (Signature Algorithm Identifier) เป็นฟิลด์ที่ระบุอัลกอริธึมที่ใช้ในการสร้างเอกสารสิทธิ์
- ชื่อผู้ออกเอกสารสิทธิ์ (Issue Name) เป็นชื่อขององค์กรที่ออกเอกสารสิทธิ์
- ช่วงเวลาที่รับรองเอกสารสิทธิ์ (Period of Validity) เป็นตัวบอกว่า ให้ใช้เอกสารสิทธิ์นี้ตั้งแต่วันที่เท่าไหร่และสิ้นสุดวันที่เท่าไหร่
- ชื่อเจ้าของเอกสารสิทธิ์ (Subject Name) เป็นชื่อของบุคคลที่เอกสารสิทธิ์ใบนี้อ้างถึงหรือแทนตัวบุคคลนั้น
- ข้อมูลของคีย์สาธารณะ (Subject's Public Key Information) เป็นฟิลด์ที่เก็บคีย์สาธารณะและระบุถึงอัลกอริธึมที่ใช้กับคีย์นี้ขึ้นมา รวมถึงพารามิเตอร์อื่น ๆ ด้วย
- ตัวระบุผู้ออกเอกสารสิทธิ์ (Issuer Unique Identifier) เป็นฟิลด์ออปชันที่ใช้ในการระบุถึงองค์กรที่ออกเกอสารสิทธิ์ ในกรณีที่ชื่อ X.500 มีการนำไปใช้กับส่วนอื่น ๆ
- ตัวระบุผู้ขอเอกสารสิทธิ์ (Subject Unique Identifier) เป็นฟิลด์ที่ใช้ในการระบุถึงตัวบุคคลที่เป็นเจ้าของเอกสารสิทธิ์ ในกรณีที่ชื่อ X.500 มีการนำไปใช้กับส่วนอื่น ๆ
- ส่วนขยาย (Extension) เป็นกลุ่มของฟิลด์ที่เพิ่มเติมข้อมูลอื่น ๆ เข้ามาด้วย
- ลายมือชื่อ (Signature) จะบรรจุเมสเซจไดเจสต์ของข้อมูลในทุกฟิลด์ที่เข้ารหัสด้วยคีย์ส่วนตัวขององค์กรพิสูจน์สิทธิ์เพื่อเป็นการยืนยันว่า เอกสารสิทธิ์นี้สร้างมาจากองค์กรดังกล่าวจริง ๆ โดยจะมีข้อมูลที่ระบุวิธีการแฮชและวิธีการเข้ารหัสด้วย

ในการใช้งานเอกสารสิทธิ์จะมีช่วงเวลาใช้งานที่จำกัดแน่นอน ดังนั้นหากผู้ใช้ต้องการใช้เอกสาร
สิทธิ์ต่อไป ก็ต้องขอต่ออายุเอกสารสิทธิ์ก่อนที่จะหมดอายุ แต่หากจะมีการหมดอายุโดยที่ไม่ขอต่อหรือมีการเลิกใช้เอกสารสิทธิ์อาจจะเนื่องมาจากพนักงานลาออก หรืออาจจะเนื่องจากเอกสารสิทธินี้ไม่ปลอดภัยแล้วก็ต้องทำการเรียกคืน (Revoke) และองค์กรพิสูจน์สิทธิ์จะต้องมีการจัดทำรายการเอกสารสิทธิ์ที่ถูกเรียกคืน (Certificate Revocation List - CRL) ซึ่งจะเก็บไว้ในไดเรกทอรีและรับรองโดยองค์กรพิสูจน์สิทธิ์ ซึ่งผู้ที่ต้องการตรวจสอบเอกสารสิทธิ์ว่าเป็นเอกสารสิทธิ์ที่ไม่ใช้งานแล้วหรือไม่ ก็ต้องขอรายการเอกสารสิทธิ์ที่ถูกเรียกคืนไปตรวจสอบ

และเนื่องจากเอกสารสิทธิ์ไม่สามารถปลอมได้ ดังนั้นการเก็บเอกสารสิทธิ์ไว้ที่องค์กรพิสูจน์สิทธิ์จึงไม่ต้องมีกลไลพิเศษมาป้องกันแต่อย่างใด กล่าวคือผู้ใช้คนใดที่เป็นสมาชิกขององค์กรพิสูจน์สิทธิ์ก็สามารถเข้าถึงเอกสารสิทธิ์ของผู้ใช้คนอื่น ๆ ได้ทุกคน โดยเอกสารสิทธิจะเก็บอยู่ในไฟล์เพียงไฟล์เดียวที่มีขนาดเล็ก นอกจากจะสามารถขอเอกสารสิทธิ์จากองค์กรพิสูจน์สิทธิ์แล้วผู้ใช้ยังสามารถส่งเอกสารสิทธิ์ไปให้กันเองได้อีกด้วย โดยผ่านทางสื่อต่าง ๆ เช่น จดหมายอิเล็กทรอนิกส์, ส่งผ่านแผ่นดิสก์เก็ต เป็นต้น

อย่างไรก็ตาม เนื่องจากระบบเครือข่ายในปัจจุบันมีขนาดใหญ่โตกว้างขวางมาก และการติดต่อสื่อสารก็ไม่ได้มีลักษณะเฉพาะกลุ่มอีกแล้ว ดังนั้นการที่จะให้ผู้ใช้ทุกคนมาใช้เอกสารสิทธิ์ที่รับรองโดยองค์กรพิสูจน์สิทธิ์เดียวกันทั้งหมดก็อาจเป็นเรื่องยาก หากผู้ใช้ 2 คนใช้เอกสารสิทธิ์ที่รับรองจากองค์กรพิสูจน์สิทธิ์คนละแห่งก็จะไม่สามารถตรวจสอบเอกสารสิทธิ์ของอีกฝ่ายได้ว่าเป็นฉบับจริงหรือไม่ ซึ่งในกรณีเช่นนี้ ก็อาจจะใช้วิธีสำเนาคีย์สาธารณะขององค์กรพิสูจน์สิทธิ์ของผู้ใช้อีกคนหนึ่งมาทำการตรวจสอบเองก็สามารถทำได้ เช่น กำหนดให้มี CA-A และ CA-B โดยให้บริการกับผู้ใช้ A และ B แต่เนื่องจากในครั้งแรกที่ A สำเนาคีย์สาธารณะของ CA-B มานั้นอาจเกิดการปลอมแปลงได้ เพราะสิ่งที่เรามีอยู่ก็คือคีย์สาธารณะของ CA-A ของเรา แต่เอกสารสิทธิ์ของ CA-B ทำให้เราไม่สามารถตรวจสอบว่าเอกสารสิทธิ์ที่ได้รับมานั้นเป็นฉบับที่ถูกต้องหรือไม่ ดังนั้นวิธีดังกล่าวจึงถือว่า ไม่มีความปลอดภัยเพียงพอ

สำหรับอีกวิธีการอีกแบบ คือให้ CA-A เก็บเอกสารสิทธิ์ของ CA-B เอาไว้ด้วยและ CA-B ก็เก็บเอกสารสิทธิ์ของ CA-A เอาไว้เช่นกัน ด้วยวิธีนี้เราก็สามารถให้องค์กรพิสูจน์สิทธิ์ตรวจสอบเอกสารสิทธิ์ได้ไม่ว่าเอกสารสิทธิ์นั้นจะรับรองจาก CA-A หรือ CA-B ก็ตาม เช่น ผู้ใช้ A ต้องการตรวจสอบเอกสารสิทธิ์ที่รับรองจาก CA-A ก็สามารถทำได้เลยเพราะรู้คีย์สาธารณะของ CA-A อยู่แล้วเนื่องจากเป็นสมาชิก CA-A และหากผู้ใช้ A ต้องการตรวจสอบเอกสารสิทธิ์ที่รับรองโดย CA-B ผู้ใช้ A ก็ขอเอกสารสิทธิ์ของ CA-B จาก CA-A โดยเอกสารสิทธิ์ดังกล่าวจะรับรองโดย CA-A ดังนั้นจึงแน่ใจได้ว่าเอกสารสิทธิ์ของ CA-B ที่ได้รับนั้นเป็นของจริงและคีย์สาธารณะของ CA-B ก็เป็นของจริง



รูปที่ 7-8 แสดงเอกสารสิทธิ์ที่ตรวจสอบโดยบราวเซอร์

จากนั้นจึงนำเอาคีย์สาธารณะของ CA-B ไปตรวจสอบเอกสารสิทธิ์อีกทีก็จะทราบได้ว่าเอกสารสิทธิ์นั้นเป็นของจริงหรือไม่ และนอกเหนือจาก CA-A และ CA-B แล้วการเชื่อถือ(Trust)กันเช่นนี้ยังสามารถกระทำกับองค์กรพิสูจน์สิทธิ์อื่นๆไปเรื่อยๆ อย่างไรก็ตามหากองค์กรพิสูจน์สิทธิ์มีเป็นจำนวนมากแล้ว ก็มีความจำเป็นที่จะต้องจัดโครงสร้างการเชื่อถือกันขององค์กรพิสูจน์สิทธิ์ให้เป็นระบบ ไม่เช่นนั้นก็อาจมีการเชื่อถือกันแบบยุ่งเหยิงและทำให้การทำงานเป็นไปอย่างไม่มีประสิทธิภาพได้ ซึ่งมาตรฐาน X.509 ก็ได้แนะนำให้มีการใช้ระบบเชื่อถือกันในรูปของความสัมพันธ์แบบระดับชั้น(Hierachial)

Certificate Authority

Digital certificate จะถูกออกโดย certificate authority ที่ซึ่งเป็นองค์กร(บุคคลที่สาม)ที่เชื่อถือได้ เมื่อ certificate authority สร้าง digital certificate แล้ว จะทำการเข้ารหัสด้วย private key ของcertificate authority นั้น แล้วจะทำการส่งไปให้กับผู้ที่ขอเอกสารสิทธ์นั้น

โดยผู้ขอเอกสารสิทธิ์จะยืน ยัน ตัวของ certificate authority นั้น ด้วย public key ซึ่งถูกออกด้วย high-level certificate authority ซึ่งมีความน่าเชื่อถือกว่า

7.8 การอิมพลีเมนต์ Security ใน Web Logic

Weblogic จะมีส่วนของ authentication,authorization และ encryption ให้บริการเพื่อที่จะป้องกันความปลอดภัยในทรัพยากรของคุณ แต่อย่างไรก็ดีวิธีการเหล่านี้ก็ยังไม่สามารถแก้ปัญหาในกรณีที่มีคนค้นพบช่องโหว่ หรือ จุดอ่อน ใน deployment environment ของคุณได้

ทางที่ดีคือ หลังจากที่คุณ deploy weblogic server แล้ว คุณควรจะให้ผู้ที่เชี่ยวชานในด้านความปลอดภัยมาตรวจสอบ และแนะนำในการปรับปรุงให้มีความปลอดภัยยิ่งขึ้น นอกจากนี้คุณอาจจะติดตามวารสารต่างๆที่เกี่ยวกับความปลอดภัย ,ตรวจสอบเรื่องของ bug และคอย update service pack ต่างๆ

7.8.1 Security Realms

Security Realms ประกอบไปด้วยกลไกในการป้องกัน resource ของ Weblogic server โดย resorce ของ Weblogic server จะถูกป้องกันภายใต้ Security realms เพียงอันเดียวเท่านั้น และโดยนโยบายหนึ่งเดียวของ Security realms เมื่อ user ต้องการที่จะเข้าถึง resource ใดๆที่เป็นของ realms นั้นๆ Server จะพยายามที่จะ authorize user คนนั้น

7.8.2 User

คือ entities ที่มาเรียกใช้ Weblogic server เช่น application endusers หรือ Weblogic server ด้วยกันเอง

เมื่อ user ต้องการที่จะ access Weblogic server ,user คนนั้น จะต้องแสดงหลักฐาน(เช่น password หรือ digital certificate) ผ่าน JAAS เพื่อที่จะ authentication ต่อ security realm เมื่อ Weblogic server สามารถยืนยัน user นั้นได้จะทำการเข้าไปมีส่วนร่วมกับ user ด้วย thead ที่ทำการ execute code ของ user แต่ก่อนที่ thread จะเริ่มทำการ execute code server จะทำการตรวจสอบบทบาท และสิทธิ ของ user อีกทีเพื่อยืนยันว่า user สามารถทำต่อได้

การระบุ user ใน Weblogic นั้น คุณสามารถใช้การระบุ password สำรับ user คนนั้นๆ โดย weblogic จะทำการ hash ทุก password หลังจากได้รับ request password จาก client จะถูก hash และ webserver จะทำการเปรียบเทียบกับ password ที่ถูก hash แล้ว

7.8.3 Group

Group คือ กลุ่มของ user การจัดการกับ group นั้นมีประสิทธิภาพมากกว่าการจัดการเป็นทีละ user ยกตัวอย่างเช่น administrator สามารถกำหนด permissions สำรับ 50 user ได้ในครั้งเดียวโดยการกำหนดบทบาท และสิทธิให้กับ group

7.8.4 Access Control Mechanisms

กลไก 2 แบบที่มีอยู่ใน Weblogic server ซึ่งทำการด้วยกันเพื่อทีจะควบคุม และป้องกันการเข้าถึง resource คือ Security Policies และ Roles

SSL Protocol กำหนด feature ของ security ดังนี้

- Server authentication – Weblogic server ใช้ digital certificate ของมันเพื่อที่จะใช้ในการยืนยันกับ client
- Client Identify Verification – Client ก็จะทำการยืนยันตัวเองกับ Weblogic server โดยใช้ digital certificate ของมันเอง โดย Server จะทำการตรวจสอบ digital certificate ของ client และ เริ่ม SSL connection ถ้า Digital certificate ได้รับการยืนยันแล้วว่าถูกต้อง วิธีนี้เรียกว่า two-way SSL เนื่องจาก user มีได้หลายคนดังนั้น client ต้องใช้ทั้ง username และ password(or digital certificate) ในการ authentication และ authorization
- Confidentiality – ทั้ง client request และ server response จะถูก encrypt เพื่อที่จะทำให้ การแลกเปลี่ยนข้อมูลเป็นส่วนตัวมากขึ้นตลอดเส้นทางใน network
- Data Integrity – data ที่ flow ระหว่าง client กับ server จะถูกป้องกันจากกลุ่มบุคคลที่3

7.8.5 Authentication Mechanisms

User จะต้องทำการยืนยันตัวเองทุกครั้งที่ request ของ user ต้องการที่จะ เข้าถึง resource ที่ถูก protect ไว้โดย Weblogic Server ด้วยเหตุนี้เอง user จะต้องมีหนังสือแนะนำตัว(username/password หรือ digital certificate) ต่อ Weblogic server ซึ่ง Weblogic server จะมีกลไกในการยืนยันตัวตน 2 แบบ คือ

- Username/Password authentication – user ID และ password จาก client จะถูกส่งไปยัง server เพื่อที่จะทำการ check กับข้อมูลที่มีและ ถ้าถูกต้องก็จะอนุญาตให้เข้าถึง resource ได้ SSL(หรือ HTTPS) protocol จะสามารถถูกใช้เพื่อเพิ่มระดับของความปลอดภัยในการส่ง username และ password เนื่อง จาก protocol นี้ จะทำการ encrypt ข้อมูลที่ถูกส่งระหว่าง client และ server
- Certificate authentification – เมื่อ SSL หรือ HTTPS client request ถูก initiate Weblogic sever จะตอบกลับโดยส่ง digital certificate ของมันไปให้ client โดย client จะทำการตรวจสอบ แล้วส่ง digital certificate ของมัน Server จะทำการตรวจสอบ digital certificate ของ client โดยใช้ CertAuthhenticator class ของมันทำการ ดึงเอาข้อมูลออกมาจาก digital certificate ของ client เพื่อตรวจสอบว่าเป็นของ user คนไหนแล้วทำการยืนยัน user จาก security realm การทำ mutual authebtication นี้มีประโยชน์ในกรณีที่ เราต้องการจะจำกัด user ให้อยู่ในวง user ที่เราเชื่อถือได้เท่านั้น

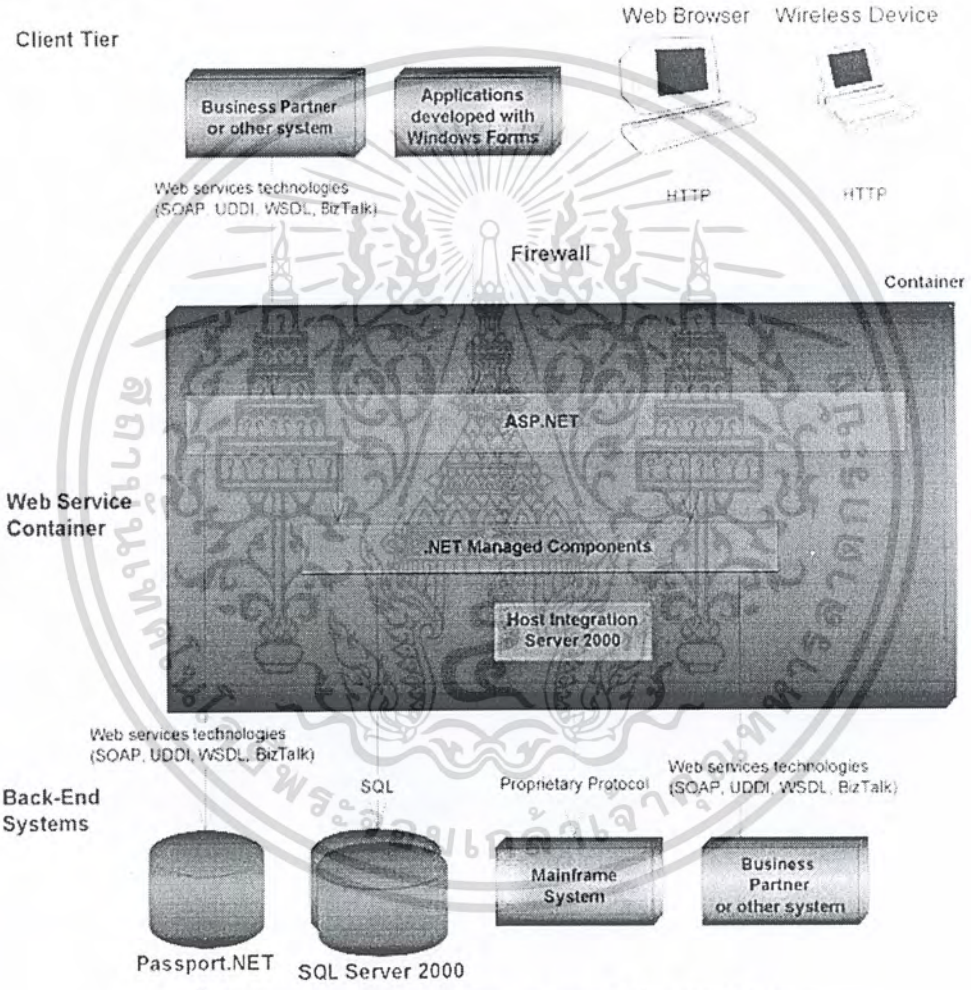
บทที่ 8

เปรียบเทียบ .NET เว็บเซอร์วิส และจาวาเว็บเซอร์วิส

ในบทนี้เราจะกล่าวถึงเทคโนโลยีในการพัฒนาโซลูชั่นจากทั้งสองค่าย ตั้งแต่แพล็ตฟอร์มในการพัฒนา และในส่วนของการสร้างเว็บเซอร์วิส

8.1 แพล็ตฟอร์มการพัฒนา

8.1.1 แพล็ตฟอร์มของ .NET



รูปที่ 8 - 1 แพล็ตฟอร์มการพัฒนาระบบภายใต้เทคโนโลยี .NET

<u>.NET แอพพลิเคชั่น</u>

ถูกดีพลอยอยู่ภายในคอนเทนเนอร์ ที่มีเซอร์วิสที่จำเป็นให้บริการสำหรับแอนเตอร์ไพรส์ แอพพลิเคชั่น เช่น ทรานแซกชัน ความปลอดภัย และ เมสเสจจิ้งเซอร์วิส

<u>ชั้นของธุรกิจ</u>

ของ.NET แอพพลิเคชั่นถูกสร้างโดยใช้ .NET เมเนจคอมโพเนนต์(managed component) ชั้นนี้ทำในส่วนของการประมวลผลทางธุรกิจและการจัดการข้อมูล การติดต่อไปยังฐานข้อมูลโดยผ่าน ADO.NET มีการติดต่อกับระบบเก่า(legacy system) โดยผ่าน Microsoft Host Integration Server 2000 เช่นติดต่อกับ COM Transaction Integrator (COM TI) หรือติดต่อคู่ค้าทางธุรกิจผ่านเว็บเซอร์วิส

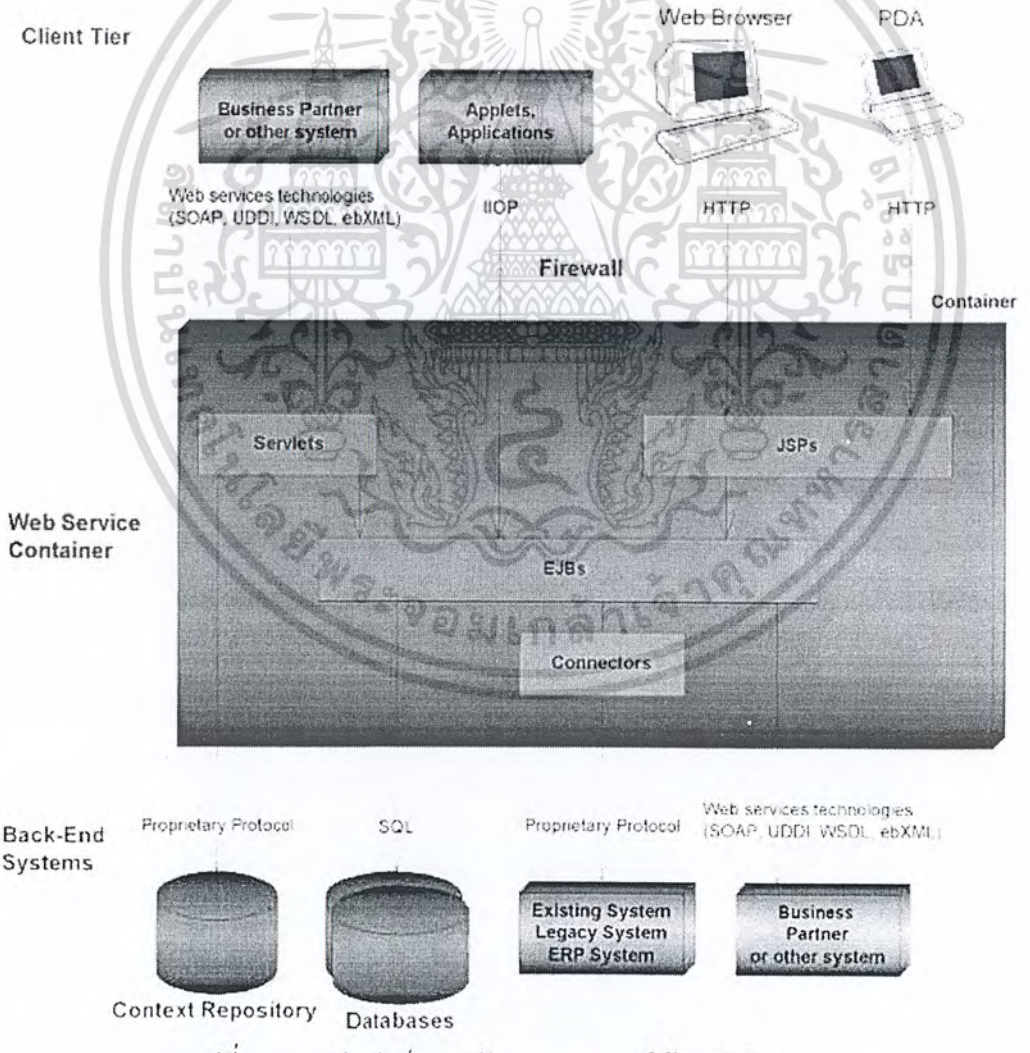
<u>คู่ค้าทางธุรกิจ</u>

สามารถใช้ .NET แอพพลิเคชั่นติดต่อโดยผ่านเว็บเซอร์วิส

<u>ไคลเอนต์ที่เรียกใช้งาน</u>

ก็เป็นได้ทั้งผ่านเว็บ ASP.NET ซึ่งให้ผลได้ในรูปของ HTML,XHTML,WML และรูปแบบของวินโดส์ฟอร์ม

8.1.2 แพล็ตฟอร์มของจาวา



รูปที่ 8 - 2 แพล็ตฟอร์มการพัฒนาระบบภายใต้เทคโนโลยี J2EE

J2EE แอพพลิเคชั่น

ถูกดีพลอยอยู่ภายในคอนเทนเนอร์ ที่มีเซอร์วิสที่จำเป็นให้บริการสำหรับเอนเตอร์ไพรส์ แอพพลิเคชั่น เช่น ทรานแซกชัน ความปลอดภัย

ชั้นของธุรกิจ

ในแอพพลิเคชั่นขนาดใหญ่ถูกสร้างโดยใช้ Enterprise JavaBeans (EJB) คอมโพเน็นต์ ติดต่อฐานข้อมูลโดย JDBC หรือ SQL/J หรือติดต่อกับระบบเดิมผ่าน Java Connector Architecture (JCA) ติดต่อผ่านเว็บเซอร์วิสโดยใช้ จาวา API สำหรับXML (JAX APIs)

คู่ค้าทางธุรกิจ

ติดต่อผ่านเว็บเซอร์วิส ตัวเซิร์ฟเลท(servlet) เป็นตัวคอยรับการร้องขอที่เข้ามาละส่งผลลัพธ์กลับไปโดยใช้ JAX APIs เป็นตัวจัดการเกี่ยวกับเว็บเซอร์วิส

ไคลเอนต์ที่เรียกใช้งาน

นอกเหนือจากการผ่านเว็บ และเว็บเซอร์วิสแล้ว ยังมีแอพเพลตที่สามารถจะติดต่อกับจาวาบีนผ่านยังโปรโตคอล Internet Inter-ORB Protocol (IIOP) ได้ด้วย

8.2 เปรียบเทียบเทคโนโลยีของทั้งสองค่าย

เริ่มจากแสดงให้เห็นถึงความคล้ายคลึงกันของทั้งสองแพล็ตฟอร์มดังตารางข้างล่าง และจะทำการพิจารณาเปรียบเทียบให้เห็นในแต่ละหัวข้อ

Feature	J2EE	.NET
Type of technology	Standard	Product
Middleware Vendors	30+	Microsoft
Interpreter	JRE	CLR
Dynamic Web Pages	JSP	ASP.NET
Middle-Tier Components	EJB	.NET Managed Components
Database access	JDBC SQL/J	ADO.NET
SOAP, WSDL, UDDI	Yes	Yes
Implicit middleware (load-balancing, etc)	Yes	Yes

ตารางที่ 8 – 1 ตารางแสดงการสนับสนุนการทำงานด้านต่างๆของ 2 แพล็ตฟอร์ม

8.2.1 Vendor Solution

ทางด้านของ J2EE นั้นเป็นเสปกที่ออกมาโดยมีเวนเดอร์ (vendor) หลายเจ้าที่นำเอามาตรฐานเหล่านั้นไปพัฒนาผลิตภัณฑ์ (product) ออกมารองรับ รวมทั้งมีเครื่องมือให้ใช้ค่อนข้างหลากหลาย ข้อดีคือมีตัวเลือกในการพัฒนาโซลูชั่นค่อนข้างหลากหลาย แต่ข้อเสียก็มาจากข้อดีของมันนั่นเองคือ ความไม่เข้ากันของแต่ละผลิตภัณฑ์ ไม่สามารถที่จะย้ายโค้ดโปรแกรมที่พัฒนาข้ามผลิตภัณฑ์ได้อย่างครบถ้วนซึ่งการพัฒนาระบบจริงนั้นเหมาะกับการเลือกใช้ผลิตภัณฑ์จากค่ายเดียวกันมากกว่า

ทางฝั่ง .NET ออกผลิตภัณฑ์ออกมานำ เนื่องจากเป็นผู้ออกเสปกและพัฒนาเองทั้งหมด รวมทั้งระบบที่ทำการสนับสนุนการทำงานส่วนมากก็ปเป็นของไมโครซอฟท์เอง

8.2.2 การสนับสนุนระบบเดิม

ในระบบเดิมที่มีการใช้งานโดยหลากหลายภาษาหลยระบบในส่วนของจาวานั้นก็มีหลากหลายทางในการทำให้สามารถใช้งานร่วมกันกับระบบเดิมเหล่านั้นได้

- ใช้ Java Message Service (JMS) เชื่อมกับระบบเก่า
- ใช้เว็บเซอร์วิสในการอินทริเกรตกับระบบต่างๆ
- ติดต่อผ่าน CORBA

รวมทั้งยังมี J2EE Connector Architecture (JCA) ทำหน้าที่เป็น ตัวติดต่อกับทรัพยากรอื่นๆ เช่น SAP R/3 ,CICS/COBOL และอื่นๆ

ฝั่ง .NET มีตัว Host Integration Server 2000 เป็นตัวช่วยจัดการเกี่ยวกับทรานแซกชันไปยังระบบเมนเฟรม หรือตัว Microsoft Message Queue (MSMQ) ในการทำงานร่วมกันกับระบบเดิมที่ใช้ IBM MQSeries และสุดท้ายคือตัว BizTalk Server2000 ทำการอินทริเกรตระบบบนโปรโตคอลของ B2B

8.2.3 การยอมรับทางการตลาด

ทางด้านจาวามีมากกว่า 50 บริษัทที่ให้การยอมรับในแพล็ตฟอร์ม

อีกด้านนั้นทีมการตลาดของไมโครซอฟท์ทำงานด้านนี้ได้ค่อนข้างดี รวมทั้งวินโดวส์ก็มีผู้ใช้อยู่มากมาย รวมทั้งทางด้านไมโครซอฟท์ประกาศตัวเกี่ยวกับเรื่องเว็บเซอร์วิสมาก่อนจาวา ในข้อนี้สรุปได้ว่าไมโครซอฟท์มีการทำการตลาดที่ดีกว่า

8.2.4 ภาษาที่รองรับ

จาวามีจุดเด่นที่เป็นภาษากลางที่สามารถนำไปใช้งานได้หลากหลายแพล็ตฟอร์ม

.NET มีจุดเด่นตรงที่รองรับได้หลายภาษาที่เป็นไปตามข้อกำหนดของ .NET เฟรมเวิร์ก โดยข้อดีของมันคือ ภายใต้โปรเจคเดียวกันสามารถพัฒนาได้จากหลายภาษา แต่ข้อเสียก็เยอะเช่นกัน เช่น ความเสี่ยงของระบบใหม่นี้ที่ทางไมโครซอฟท์พัฒนาขึ้นมา ของใหม่ย่อมมีความเสี่ยงเกิดขึ้น

- การดูแลรักษาระบบ เนื่องจากเขียนด้วยหลากหลายภาษา ถ้าบุคคลผู้เชี่ยวชาญในบางภาษาลาออกไปทำให้คนอื่นๆที่ดูแลลำบากในการทำความเข้าใจโค้ดภาษานั้นๆ

- การสร้างองค์วามรู้ การใช้งานหลากหลายภาษาส่งผลให้โปรแกรมเมอร์ไม่สามารถแชร์ประสบการณ์หลายๆอย่างในการพัฒนาแก่กันได้

ในการพัฒนาระบบนั้นก็ต้องพิจารณาให้เหมาะสม รวมถึงคำนึงถึงผลกระทบของการตัดสินใจด้วย

8.2.5 การสนับสนุนแพล็ตฟอร์มเดิม

J2EE ไม่มีปัญหาเรื่องการเชื่อมต่อกับระบบเดิมมากนัก แต่ตัว JCA และ เว็บเซอร์วิสก็ยังต้องการการเขียนโค้ดใหม่เพิ่มเติม ถึงแม้ว่าไมโครซอฟท์ .NET ได้รับการพัฒนามาจาก MTS และ COM+ แต่อย่างไรก็ตามรูปแบบการพัฒนาก็เปลี่ยนไปโดยสิ้นเชิง มีคอมมอนรันไทม์(CLR)เข้ามาจัดการ มีการจัดการเกี่ยวกับคอมโพเน็นต์ที่เปลี่ยนไป ทำให้อาจต้องมีการเขียนโค้ดใหม่เกือบ 60% เพื่อให้สามารถใช้ประโยชน์จากรันไทม์เอนเวอรอนเมนต์ใหม่นี้ได้อย่างเต็มที่

8.2.6 การทำงานข้ามแพล็ตฟอร์ม (Portability)

J2EE เป็นมาตรฐานสำหรับหลายๆแพล็ตฟอร์ม และรองรับการอิมพลีเมนต์ได้หลากหลาย แต่ถึงอย่างไรก็ตาม ยังมีปัญหาเรื่องของการนำโค้ดไปใช้ใหม่กับต่างผลิตภัณฑ์ เนื่องจากแต่ละผลิตภัณฑ์ก็ยังมีการเพิ่มความสามารถในผลิตภัณฑ์ของตัวเองด้วยที่ไม่เป็นไปตามมาตรฐาน J2EE ทางซันเองก็มีซอฟแวร์ที่ให้การทดสอบว่าผลิตภัณฑ์เป็นไปตามมาตรฐาน J2EE ไหม

ถึงวันนี้ ตัวผลิตภัณฑ์ที่รองรับการใช้งาน .NET ก็มาจากไมโครซอฟท์แห่งเดียวจึงไม่มีปัญหาส่วนนี้ แต่มีการพัฒนาเฟรมเวิร์กบนลินุกซ์ด้วยแต่ยังไม่สมบูรณ์นักการพัฒนาคลาสไลบารี่ มีการทดสอบการรันโค้ดโปรแกรมง่ายข้ามจาก วินโดส์ไปยังลีนุกซ์ซึ่งก็ได้ผลถูกต้อง

ถึงแม้ว่าไมโครซอฟท์จะรับปากถึงการทำให้ .NET สามารถใช้งานข้ามระบบได้ก็ตาม แต่จากที่ผ่านมาเช่น COM คอมโพเน็นต์ ไมโครซอฟท์ก็สามารถทำให้ใช้ได้บนระบบปฏิบัติการอื่นเช่นกัน แต่ไม่มีการพัฒนาเซอร์วิสอื่นๆขึ้นมารองรับการใช้งานเลย ดังนั้นในส่วนของ .NET เราก็ต้องติดตามกันต่อไปว่าจะเป็นอย่างไร

8.2.7 เครื่องมือที่ใช้ในการพัฒนา

จาวามีเครื่องมือให้เลือกใช้ค่อนข้างเยอะ เช่น Forte, Visual Cafe, Visaul Age, JBulider

ไมโครซอฟท์ ใช้เครื่องมือ Visual Studio.NET ในการพัฒนาซึ่งมีเอนเวอร์รอนเมนต์การใช้งานคล้ายกับ VisualStudio ตัวเดิมซึ่งโปรแกรมเมอร์คุ้นเคยอยู่แล้ว รวมทั้งเครื่องมือของไมโครซอฟท์เองก็มีประสิทธิภาพในการพัฒนาค่อนข้างมาก

8.2.8 การสนับสนุนการใช้งานเว็บเซอร์วิส

J2EE สนับสนุนการใช่งานเว็บเซอร์วิสโดยใช้ Java APIs สำหรับการแปลภาษา XML แต่อย่างไรก็ตามในส่วนนี้ยังอยู่ระหว่างการพัฒนา ยังต้องใช้ปรับแต่งเขียนโค้ดเพิ่มเติมด้วยตัวเองเพื่อให้เป็นเว็บเซอร์วิสได้ ผลิตภัณฑ์ของแต่ละแห่งก็มีการพัฒนาไลบรารี่ในส่วนนี้เพิ่มเติมขึ้นเองเพื่อให้การใช้งานเป็นไปอย่างอัตโนมัติและรวดเร็ว ซึ่งส่งผลต่อการผูกติดซอร์ซโค้ดกับระบบ

เฟรมเวิร์กของ .NET ถูกพัฒนามาเพื่อสนับสนุนเว็บเซอร์วิสอย่างเต็มตัวตั้งแต่แรกเริ่มอยู่แล้ว

ไม่ว่าอย่างไรก็ตามมาตรฐานของเว็บเซอร์วิสยังต้องพัฒนาอีกยาวไกล เช่นเรื่องของการจัดการ ทรานแซกชัน เป็นต้นดังนั้นทั้งสองค่ายก็จะต้องเร่งออกผลิตภัฑ์มารองรับในส่วนนี้ด้วย

8.2.9 การขยายการรองรับการใช้งานของระบบ(Scalability)

การเพิ่มประสิทธิภาพของระบบโดยทั่วไปเราสามารถที่จะเพิ่มได้การเลือกใช้ฮาร์ดแวร์ใหม่ๆที่มีประสิทธิภาพ ทาง J2EE สามารถใช้งานได้หลากหลายแพล็ตฟอร์มซึ่งแพล็ตฟอร์มนั้นก็สามารถเลือกใช้โพรเซสเซอร์ได้หลายตัวด้วย

ทาง .NET นั้นใช้งานบนวินโดส์เท่านั้นจึงมีข้อจำกัดทางด้านการสนับสนุนโพรเซสเซอร์

8.2.10 สรุปการเลือกใช้งาน

ทั้งสองแฟล็ตฟอร์มสามารถนำไปสู่จุดมุ่งหมายที่เดียวกันได้ การที่จะเลือกใช้ผลิตภัณฑ์จากค่ายใดก็ต้องพิจารณาจากหลากหลายองค์ประกอบได้แก่ นักพัฒนาในองค์กรของคุณมีความถนัดระบบใด , ระบบเดิมที่ใช้งาน, ความสัมพันธ์กับบริษัทเวนเดอร์ , และก็ลูกค้าของบริษัทคุณ

8.3 เปรียบเทียบจาวาและ .NET

มีความพยายามเอาจาว่าและ .NET มาเปรียบเทียบกัน โดยมีการเขียนโปรแกรมเลียนแบบแซมเปิลแอพพลิเคชั่น (Sample Application) ตัวหนึ่ง ชื่อ เพ็ทสโตร (Petstore) ของจาว่าใช้ชื่อว่า เพ็ทช็อป (Petshop) แล้วนำมาวัดประสิทธิภาพกัน ผลออกมาว่า .NET ให้ประสิทธิภาพในการทำงานรวดเร็วกว่าเพ็ทสโตรที่พัฒนาด้วยจาว่าถึง 17 เท่า ซึ่งคุณสามารถไปดาวน์โหลดโค้ดได้ฟรี การเปรียบเทียบครั้งนี้ได้รับการประชาสัมพันธ์กันขนานใหญ่ แต่ก็มีโปรแกรมเมอร์ออกมาวิจารณ์กันมาก ถึงความเป็นไปได้ที่อาจจะมีการลำเอียงในการเปรียบเทียบ โดยประเด็นที่มีการโต้แย้งหลักๆ ดังนี้

1.เพ็ทสโตร (Petstore) ถูกออกแบบมาบนพื้นฐานของการแสดงคุณลักษณะของจาว่าเป็นหลัก ไม่ได้เน้นเรื่องของความเร็ว การที่เพ็ทช็อปจะมาโจมตีในส่วนของความเร็วและเหมารวมไปหมดว่า .NET เหนือกว่านั้น ไม่น่าจะถูกต้องนัก

2. เพ็ทช็อป (Pet Shop) ใช้สโตรโพรซิเดอร์ในฐานข้อมูลในการทำงาน ซึ่งแน่นอน ย่อมให้ความเร็วในการทำงานเหนือกว่าเพ็ทสโตรอยู่แล้ว และจะไม่สามารถนำไปทำงานบนฐานข้อมูลระบบอื่น ได้ทันทีเหมือนเพ็ทสโตร เพราะสโตรโพรซิเดอร์ไม่เหมือนกัน ถ้าจะเปรียบเทียบกันอย่างเป็นธรรมก็ต้องเขียนโปรแกรมติดต่อฐานข้อมูลโดยไม่ใช้สโตรโพรซิเดอร์

3. จำนวนบรรทัดของโปรแกรมนั้นเพ็ทช็อปใช้จำนวนบรรทัดเพียง 1 ใน 3 ของเพ็ทสโตร ส่วนหนึ่งเป็นเพราะโครงสร้างภาษา C# ที่พัฒนามาให้กระชับกว่าอยู่แล้ว แต่อีกส่วนหนึ่งก็เกิดจากการเขียนโปรแกรม โดยมีจำนวนบรรทัดของ Error Exception น้อยกว่า

4. การทดสอบนั้น ทำบนแพล็ตฟอร์มของวินโดวส์ ซึ่งแน่นอนว่า .NET ย่อมจะทำงานได้เป็นอย่างดีกับผลิตภัณฑ์จากบริษัทเดียวกัน แต่ถ้านำเอาไปทดสอบบนระบบอื่นๆ เช่นยูนิกซ์ (UNIX) อาจจะไม่ให้ผลแตกต่างกันมากถึงขนาดนี้

5. มีการใช้แคช (Cache) จำนวนมากกับเพจ (Page) ที่พัฒนาบนเพ็ทช็อป ซึ่งแน่นอน ย่อมทำให้เพ็ทช็อปทำงานได้เร็วขึ้นอยู่แล้ว ขณะที่เพ็ทสโตรไม่ได้เน้นในเรื่องนี้

6. ประเด็นอื่นๆ มีอีกมากมาย เช่น การวางโมเดล (Model) ที่ไม่เหมือนกัน ความเสถียร

8.4 การพัฒนาเว็บเซอร์วิสจากทั้งสองแพล็ตฟอร์ม

8.4.1 การเรียกใช้งานข้ามแพล็ตฟอร์ม

จากโปรโตคอล SOAP ที่ใช้ในการติดต่อสื่อสารเว็บเซอร์วิสมีการอธิบายการทำงานแค่เพียงสามารถให้มีการเรียกใช้งานข้ามระบบได้เท่านั้น โดยแลกเปลี่ยนบริการข้อมูลต่างๆในรูปของ XML จากการเรียกใช้งานจริงก็สามารถทำการเรียกใช้เมธอดได้อย่างถูกต้อง

8.4.2 ปัญหาที่พบ

ดังที่กล่าวไว้ในหัวข้อที่แล้วว่า สามรถเรียกเปลี่ยนเซอร์วิสข้อมูลได้เพียงเบื้องต้นเท่านั้น จากการพยายามที่จะให้ทั้งสองค่าย มีการใช้เซสชันระหว่างกัน ไม่สามรรถทำได้ เพราะเป็นเพียงการพัฒนาในระบบของตัวเองเท่านั้น

การใช้งานในธุรกิจ ปัญหาเรื่องของทรานแซกชัน ที่ต้องมีการทำ two phrase commit การใช้งานเว็บเซอร์วิสไม่สามารถเรียกสนับสนุนการทำงานในส่วนนี้ได้ เนื่องจากโปโตคอล,ภาษา XML ที่ใช้ ไม่ได้รองรับการจัดการทรานแซกชันเลย ยังต้องรอมาตรฐานที่คาดว่าจะออกมาในประมาณปี 2547

ความปลอดภัยในการรับส่งข้อมูลข้ามเครือข่ายก็เป็นปัญหาสำคัญในทางธุรกิจเช่นกัน โดยปัจจุบันการใช้งานที่มีให้ใช้คือ การใช้การพิสูจน์ตัว(Authentication) ผ่านเว็บเซิร์ฟเวอร์ของระบบตนเอง รวมด้วยการปรับแต่งระบบให้ใช้โปรโตคอล SSL แต่ก็ทำให้ประสิทธิภาพของระบบลดลงไปด้วย มาตรฐานของความปลอดภัยในเว็บเซอร์วิสเพิ่งจะออกมาได้ไม่นาน ออกโดยองค์กร W3C คือมาตรฐาน WS-Security ทางด้าน .NET นั้นออกปลั๊กอินของ ASP.NET เพื่อให้ใช้งานตามมาตรฐานได้ ส่วนฝั่ง J2EE นั้น BEA Web Logic ในเวอร์ชัน 2 ถึงจะมีการสนับสนุนซึ่งอยู่ระหว่างการพัฒนา

ทางด้านจาวายังมีปัญหาที่เกิดขึ้นอีกอย่างคือ การพอร์ตหรือนำโค้ดที่เขียนเพื่อดีพลอยบนเว็บเซิร์ฟเวอร์ตัวหนึ่งไปใช้บนเว็บเซิร์ฟเวอร์ของบริษัทอื่นนั้นไม่สามรรถทำได้ เนื่องจากมการผูกติดกับไลบรารี่ของเว็บเซิร์ฟเวอร์ของแต่ละบริษัทด้วย

ข้อมูลเปรียบเทียบ	.NET	Java

1. โครงสร้างภาษา การจัดการหน่วยความจำ JIT เครื่องเสมือน(Virtual Machine) มีหลักการคล้ายๆกัน	แนวคิดแบบ OOP ลอกแบบมาจากจาว่า C++ เพิ่มเติมแนวคิดบางประการจากจาว่า เช่น เมตาดาต้า (Meta Data) มี Intermediate Language(IL) เป็นภาษากลาง ทำงานบน Common Language Runtime (CLR)	ต้นแบบมาจาก C/C++ มี CORBA IDL และ ORB ช่วยทำให้โค้ดหลายๆภาษาใช้ออบเจกต์ (Object) ร่วมกันได้ ทำงานบน J2EE แต่ไม่ใกล้ชิดกันเหมือนกับ IL ทำงานกับ CLR บน .NET
2. ภาษาที่สนับสนุน	C#,VB.NET เป็นหลัก ภาษาอื่นๆที่สามารถสนับสนุน แต่อาจไม่สะดวกในการใช้งาน เช่น C++,COBOL,Python,Delphi,J#	จาวาเป็นหลัก สามารถใช้ภาษาอื่นได้บ้าง แต่ไม่ดี ต้องการเครื่องมืออื่นเพิ่มเติม เช่น C++,C
3. ราคา	Visual Studio .NET ซึ่งมีราคาแพง และฟรี สำหรับ SDK และ Web Matrix ซึ่งเป็นเครื่องมือพื้นฐานในการพัฒนาที่ไม่เก่งนัก	มีเครื่องมือฟรีมากมายแจกจ่าย เครื่องมือที่เก่งกว่าได้แก่ J-Builder ราคาถูกกว่า .NET
4. ความเร็วในการพัฒนา ในงานขนาดเท่ากัน	เร็วกว่าจาวา (ประมาณ 2-5 เท่ากับงานที่เท่ากัน โดยเฉพาะงานเว็บหรือฐานข้อมูล) เพราะเครื่องมือที่มีมาให้ฉลาด และอำนวยความสะดวกได้มาก	
5. ความเร็วในการทำงานของแอพพลิเคชัน	เร็วกว่า 17 เท่าตามการเปรียบเทียบแอพพลิเคชันที่คล้ายกัน Petstore/Petshop	น่าจะช้ากว่า แต่ไม่น่าจะถึงกับที่ถูกกล่าวอ้างในการเปรียบเทียบระหว่าง PetStore/PetShop
6. การทำงานข้ามแพล็ทฟอร์ม	- ทำงานได้เฉพาะระบบวินโดวส์ แต่ในอนาคต จะสนับสนุนระบบอื่น เช่น ลีนุกซ์ (Linux) - สามารถทำงานข้ามแพล็ตฟอร์มได้ ถ้าเขียนโปรแกรมในลักษณะของเว็บแอพพลิเคชัน (Web Application) ASP.NET แต่การแสดงผลและการทำงานบางอย่าง เช่น Validation Control ก็ไม่สมบูรณ์ 100% บนบราวเซอร์อื่น	ทำงานได้หลากหลายแทบทุกแพลิ็ตฟอร์ม

7. คลาสที่สนับสนุน	.NET Framework SDK- มีหลากหลายกว่า 1400 คลาส ที่พัฒนาโดยไมโครซอฟท์ ระบบค่อนข้างผูกขาดบนไมโครซอฟท์	จาวาคอร์เอพีไอ (Java core API) เป็นระบบเปิด มีพัฒนาโค้ดเปิด (Open source) และแจกฟรีมากมาย
8. ความเข้ากันได้กับ Application/Solution อื่นๆ	เข้ากันได้ดีกับผลิตภัณฑ์ของไมโครซอฟท์เอง เช่น ไมโครซอฟท์ออฟฟิศ (MS Office) Exchange, MS SQL	ได้รับการยอมรับจากระบบที่หลากหลายกว่า เช่น IBM , Notes , SAP
9. ขนาดที่เหมาะสมของโครงการและการลงทุน	เหมาะสำหรับโครงการระยะสั้น-กลาง ขนาด ระยะเวลาสั้น-กลาง เช่นเดียวกัน เหมาะกับโครงการขนาดเล็กที่ต้องแข่งขันกันด้านราคา หรือทำเป็นแอพพลิเคชันสำเร็จรูป	เหมาะสมกับโครงการขนาดกลาง-ใหญ่ ระยะเวลาโครงการยาวนานกว่า และมีรายละเอียดข้อจำกัดทางเทคนิค และระบบค่อนข้างมาก
10. การเปิดกว้างทางเทคโนโลยี	เทคโนโลยีค่อนข้างอยู่ในวงจำกัด ข้อมูลเชิงลึกไม่เปิดเผย โครงการในลักษณะโค้ดเปิดค่อนข้างน้อย โค้ดตัวอย่างฟรี หาดีๆได้ยากกว่า	ค่อนข้างเปิดเผยอย่างกว้างขวาง มีโครงการโค้ดเปิด จำนวนมาก แจกจ่ายให้โค้ดดีๆที่ค่อนข้างสมบูรณ์แบบให้ดาวน์โหลดฟรี
8. การสนับสนุนเกี่ยวกับเว็บ	เก่งมากสำหรับงานบนเว็บ มีเทคโนโลยี ASP.NET เขียนเว็บแอพพลิเคชันได้ง่าย และคล้ายคลึงกับการเขียนโปรแกรมบนวินโดวส์ทั่วไป มีเครื่องมือที่เก่งด้านเว็บเซอร์วิส (Webservices) ,SOAP มีคลาสสำเร็จที่สร้างเอกสาร XML ให้เลย โดยใช้เพียง ADO.NET	จาวาเซิร์ฟเวอร์เพจ (Java Server Pages) เครื่องมือในการพัฒนายังค่อนข้างพื้นๆ ต้องเข้าใจ HTML Tag ต่างๆเอง เครื่องมือยังไม่เก่งมากนักเกี่ยวกับงานเว็บเซอร์วิส ต้องเขียนโปรแกรมเพื่อจัดการกับเว็บเซอร์วิสเองเป็นส่วนใหญ่ โดยใช้ JDBC, EJB, JMS, Java XML Libraries
12. สนับสนุนกับวินโดวส์ฟอร์ม	ใช้เครื่องมือพัฒนา Visual Studio.NET ในการพัฒนา	ใช้จาวาสวิงส์ (Java Swing) ซึ่งได้รับการสนับสนุนจากเครื่องมือพัฒนาจาวาที่หลากหลาย
13. Inter-Operatibility	เน้นการทำงานข้ามภาษา	เน้นการทำงานแพล็ตฟอร์ม

ตารางที่ 8 – 2 สรุปข้อเปรียบเทียบระหว่างสองแพล็ตฟอร์ม

บทที่ 9
แอพพลิเคชันของระบบทัวร์

9.1 Overview OLALA Wedding Planner

ระบบ OLALA Wedding Planner เป็นระบบจัดการการแต่งงาน ที่เป็นเว็บเซอร์วิสให้บริการจัดหาโรงแรม , ของที่ระลึก , ถ่ายรูปแต่งงาน , การจองแพ็คเกจทัวร์ , การจองวงดนตรีหรือการแสดง ที่เป็นส่วนหนึ่งในงานแต่งงาน โดยผู้ใช้บริการสามารถค้นหาข้อมูลต่างๆได้ตามต้องการ โดยกรอกรายละเอียดที่จำเป็นในการค้นหาลงไป เช่น ต้องการจองห้องจัดเลี้ยง ข้อมูลที่ต้องใส่ในการค้นหาอย่างน้อย คือ สไตล์การตกแต่งสถานที่ห้องจัดเลี้ยงแบบไหน , การจองการถ่ายรูปแต่งงาน ข้อมูลที่ต้องใส่อย่างน้อย คือ ประเภทของการถ่ายรูป (ในสถานที่ , นอกสถานที่) และช่วงราคา เป็นต้น จากนั้นทางระบบจะเรียกใช้เซอร์วิสต่างๆ ที่ เว็บเซอร์วิสมีให้เรียกใช้

เมื่อผู้ใช้บริการทำการจองเรียบร้อยแล้ว จะสามารถเข้าตรวจสอบดูรายละเอียด , เปลี่ยนแปลง และยกเลิกการจองครั้งนั้นๆได้ ส่วนเรื่องการชำระค่าบริการจะเป็นการชำระผ่านบัตรเครดิต เมื่อชำระค่าบริการเรียบร้อยแล้ว หากผู้ใช้บริการต้องการยกเลิกรายการที่จองไว้ ผู้ใช้บริการจะไม่ได้ค่าบริการคืน ซึ่งเป็นกฎของทางระบบ OLALA Wedding Planner

ในปริญญานิพนธ์ฉบับนี้จะแบ่งการออกแบบบริการต่างๆเป็น 2 เทคโนโลยี คือ เทคโนโลยีของ .NET เฟรมเวิร์ค และ เทคโนโลยีของจาวา ดังรูปที่ 11-1 เพื่อทำการศึกษาการเรียกใช้คอมโพเนนต์ข้ามสถาปัตยกรรม โดยทำการติดต่อสื่อสารกันผ่านโปรโตคอล SOAP

เทคโนโลยีของจาวาจะประกอบด้วย

ธนาคาร

ระบบทัวร์

ระบบเอ็นเตอร์เทนเมนต์

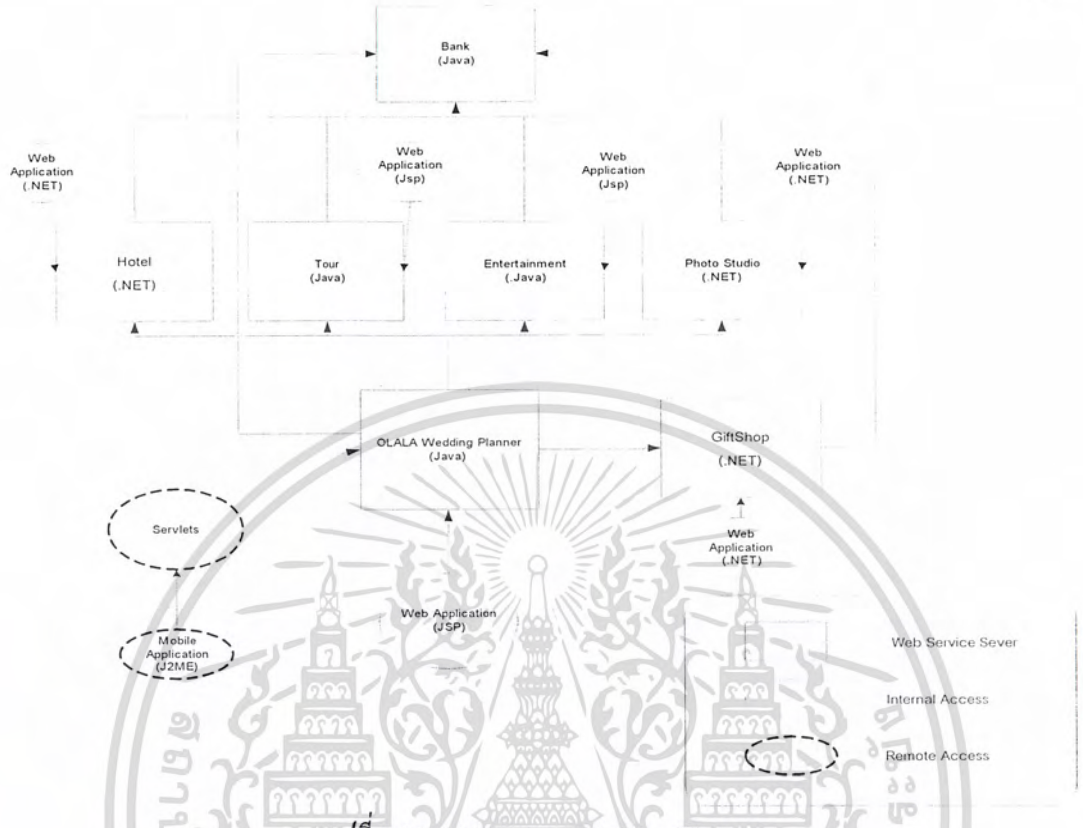
ระบบ OLALA Wedding Planner

เทคโนโลยีของ .NET เฟรมเวิร์คประกอบด้วย

ระบบโรงแรม

ระบบสตูดิโอถ่ายภาพ

ระบบขายของชำร่วย



รูปที่ 9-1 ระบบ OLALA Wedding Planner

9.2 รายละเอียดของแอพพลิเคชัน

ระบบ Olala Tour เป็น ระบบจัดการแพ็กเกจทัวร์ ที่เป็นเว็บเซอร์วิส ให้บริการจอง , ค้นหาแพ็กเกจทัวร์ โดยทางผู้ใช้บริการจะต้องสมัครเป็นสมาชิกก่อน

9.3 ฟังก์ชันการทำงาน

ระบบทัวร์จะประกอบด้วย 4 stateless session bean ดังนี้

9.3.1 BankingServiceBean

ให้บริการเกี่ยวกับ การจ่ายเงิน มีเมธอดดังนี้

9.3.1.1 Paydown ใช้แก้ไขการจ่ายเงินของลูกค้า

9.3.2 CustomerServiceBean

ให้บริการเกี่ยวกับการจัดการรายละเอียดลูกค้า มีเมธอดดังนี้

9.3.2.1 AddCustomer ใช้ในการสมัครเป็นลูกค้าใหม่

9.3.2.2 EditCustomer ใช้แก้ไขรายละเอียดลูกค้า

9.3.2.3 GetCustomerDetail ใช้ดูรายละเอียดของลูกค้า

9.3.2.4 Login ใช้ตรวจสอบสิทธิการเข้าสู่ระบบ

9.3.3 PackageServiceBean

ให้บริการเกี่ยวกับรายละเอียดของแพ็กเกจ มีเมธอดดังนี้

9.3.3.1 Search ใช้สำหรับค้นหาที่นั่งตามความต้องการของลูกค้า โดยถ้าหากสามารถค้นหาแพ็กเกจการท่องเที่ยวที่มีลักษณะตรงตามความต้องการของลูกค้าและมีจำนวนที่นั่งเพียงพอ ก็จะส่งกลับมา เวลาต้องการจองก็นำข้อมูลที่ได้จากการค้นหานี้ไปจอง

9.3.3.2 GetPackagelist ใช้เรียกดูรายการของแพ็กเกจทั้งหมด

9.3.3.3 GetPackageDetail ใช้เรียกดูรายละเอียดของแพ็คเกจหนึ่ง

9.3.4 ReserveServiceBean

ให้บริการเกี่ยวกับการจอง การยกเลิก การดูข้อมูลของรายการที่จอง มีเมธอดดังนี้

9.3.4.1 CreateContactID ใช้สร้างหมายเลขการติดต่อของลูกค้า

9.3.4.2 CancelContactID ใช้ยกเลิกหมายเลขการติดต่อของลูกค้า

9.3.4.3 ReserveBooking ใช้จองแพ็คเกจ

9.3.4.4 ChangeBooking ใช้เปลี่ยนแปลงรายละเอียด

9.3.4.5 ConfirmBooking ใช้ยืนยันการจอง

9.3.4.6 CancelBooking ใช้ยกเลิกการจอง โดยใส่ หมายเลขการติดต่อของลูกค้า, หมายเลขการจอง

9.3.4.7 GetCustomerContact ใช้เรียกดูรายละเอียดการติดต่อของลูกค้าคนหนึ่ง

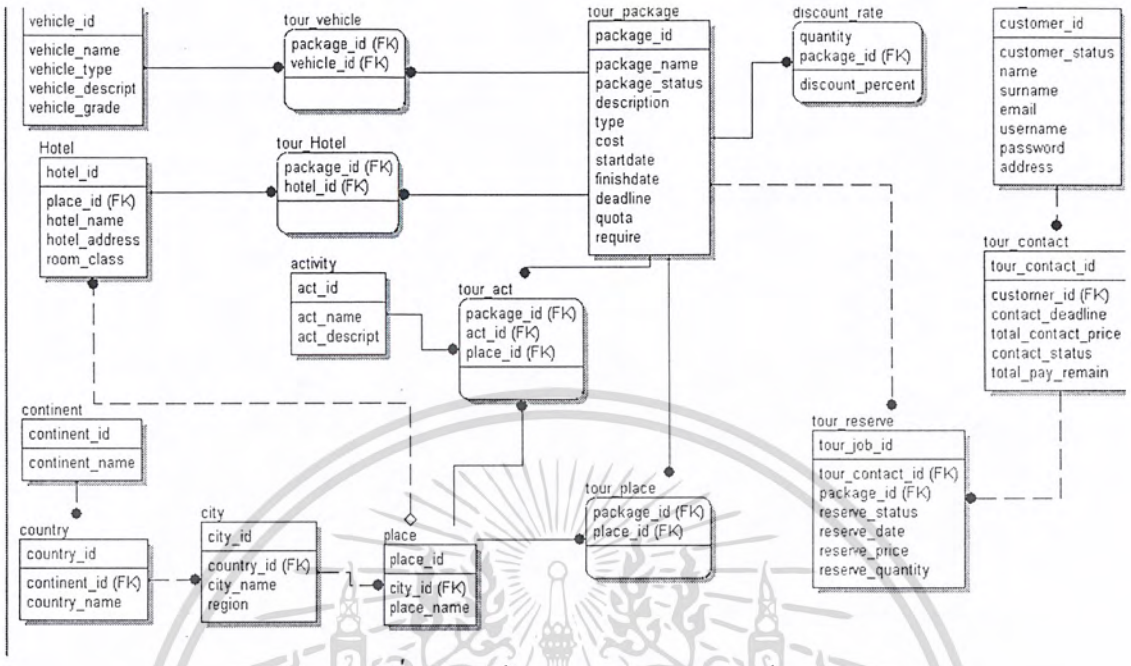
9.3.4.8 GetBooking ใช้เรียกดูรายละเอียดการจองของแพ็คเกจหนึ่ง

9.3.4.9 GetContactJob ใช้เรียกดูรายละเอียดการจองของการติดต่อของลูกค้า

9.3.4.10 GetJobDetail ใช้เรียกดูรายละเอียดการจองของลูกค้าคนหนึ่ง

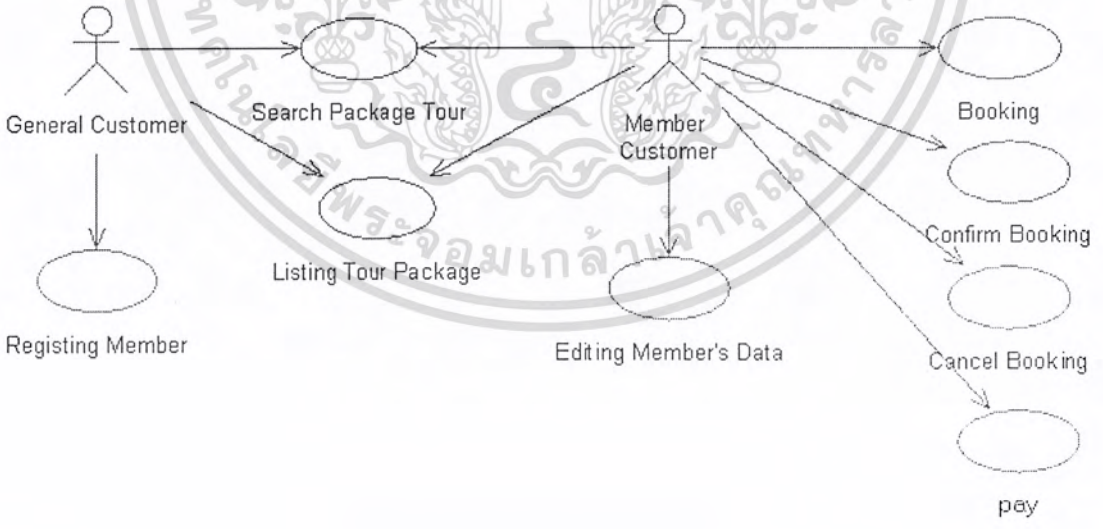
9.4 อีอาร์ไดอะแกรม

ระบบทัวร์จะมี ER ดังรูป เพื่อเก็บรายละเอียดของ แพ็กเกจทัวร์และข้อมูลการจองของลูกค้า



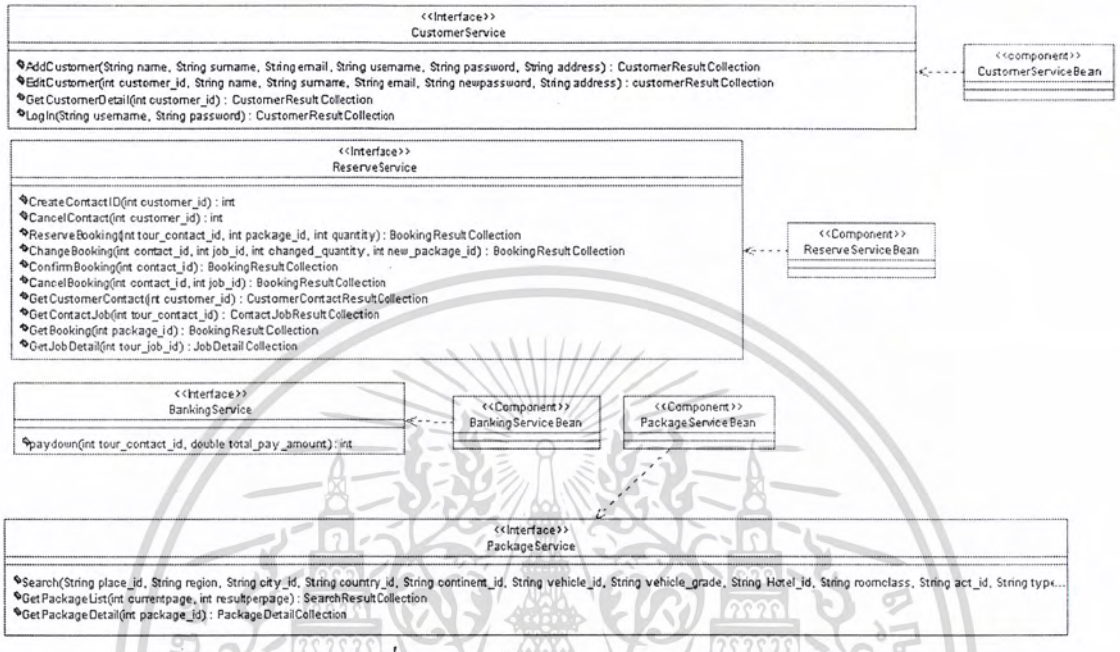
รูปที่ 9-2 อีอาร์ไดอะแกรมของระบบทัวร์

9.5 ยูสเคสไดอะแกรม



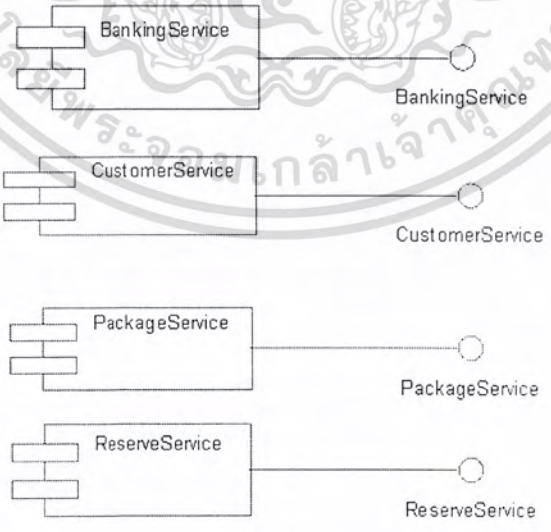
รูปที่ 9-3 ยูสเคสไดอะแกรมของระบบทัวร์

9.6 คลาสไดอะแกรม



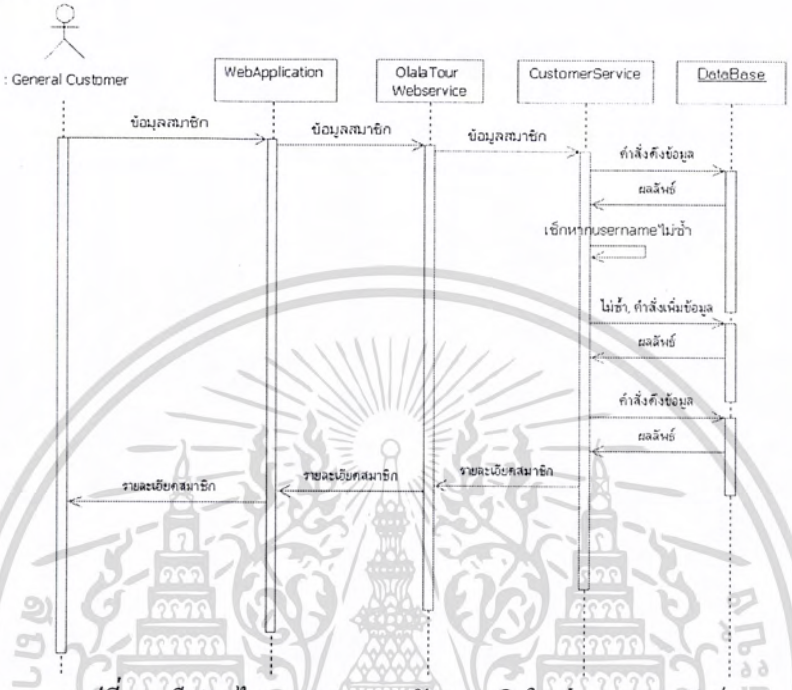
รูปที่ 9-4 คลาสไดอะแกรมของระบบทัวร์

9.7 คอมโพเนนต์ไดอะแกรม

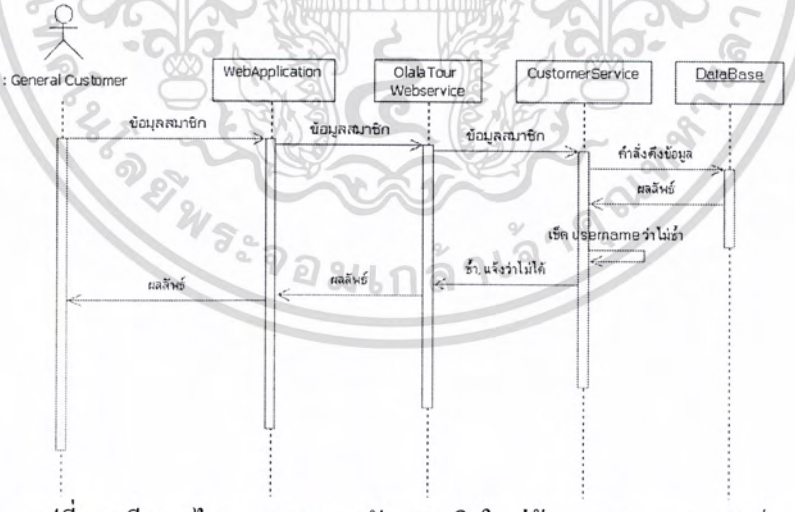


รูปที่ 9-5 คอมโพเนนต์ไดอะแกรมของระบบทัวร์

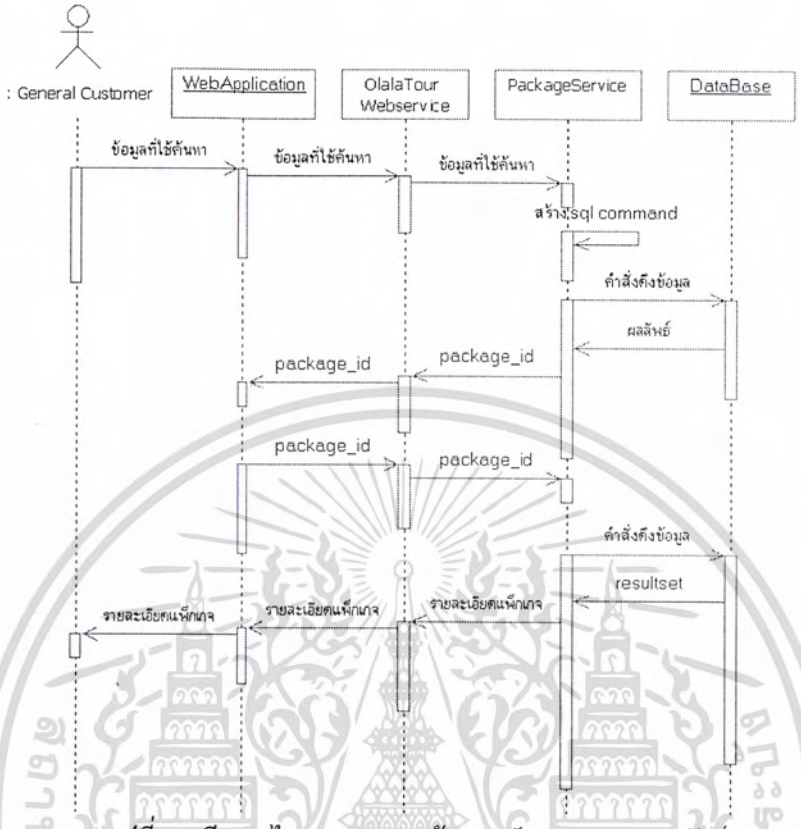
9.8 ซีเควนไดอะแกรม



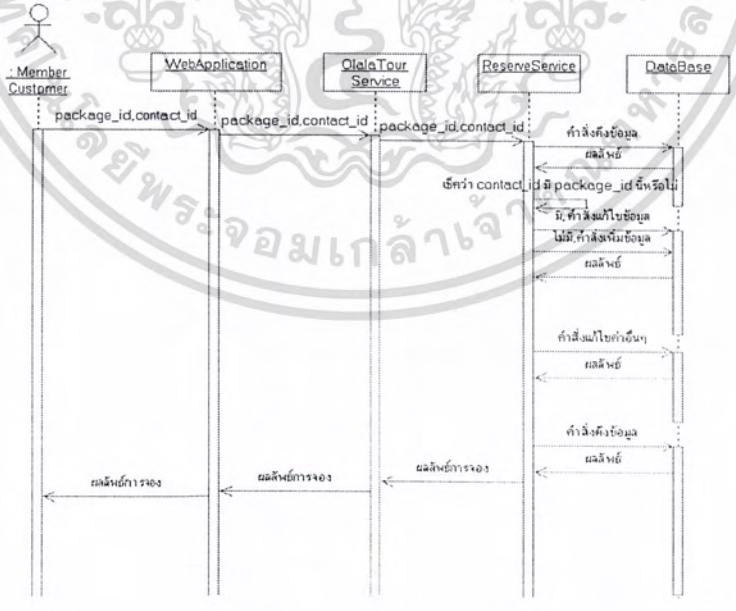
รูปที่ 9-6 ซีเควนไดอะแกรมการสมัครสมาชิกใหม่ของระบบทัวร์



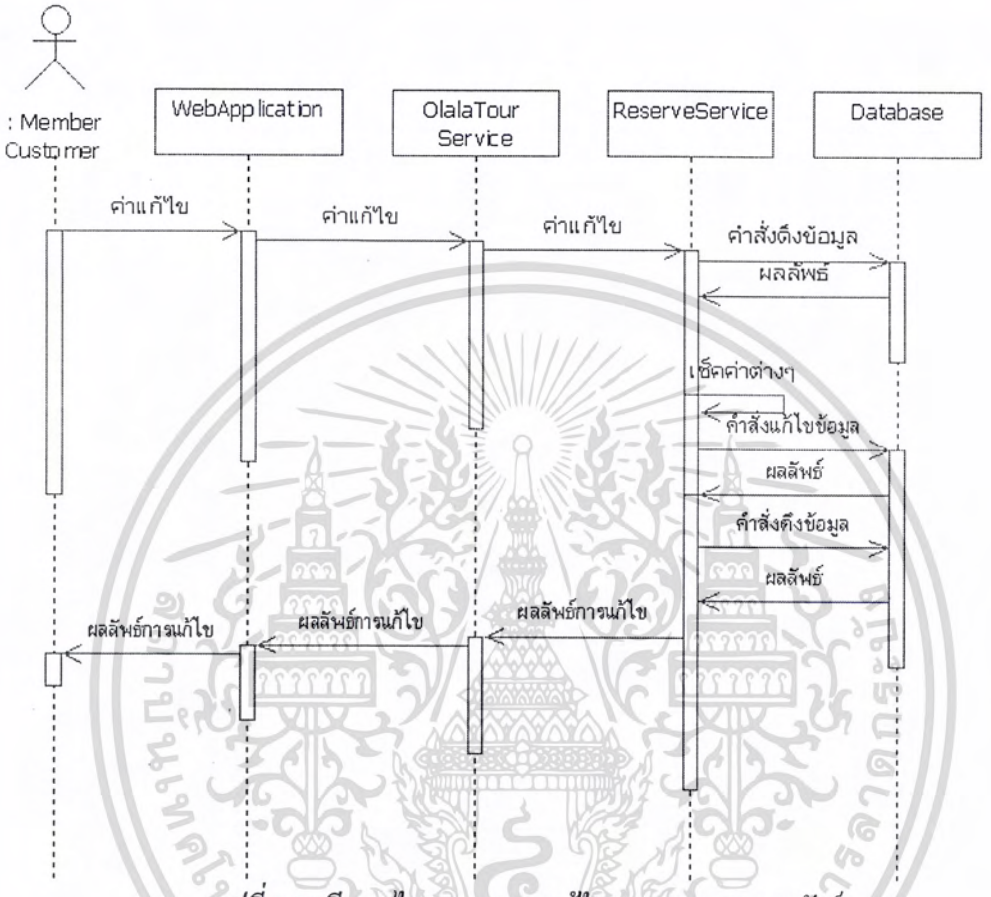
รูปที่ 9-7 ซีเควนไดอะแกรมการสมัครสมาชิกใหม่ล้มเหลวของระบบทัวร์



รูปที่ 9-8 ซีเควนไดอะแกรมการค้นหาแพ็กเกจของระบบทัวร์



รูปที่ 9-9 ซีเควนไดอะแกรมการจองแพ็กเกจใหม่ของระบบทัวร์

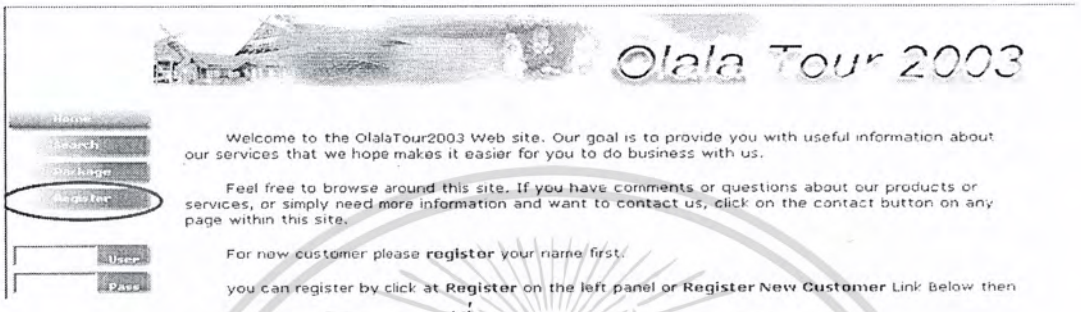


รูปที่ 9-10 ซีเควนไดอะแกรมการแก้ไขการจองของระบบทัวร์

9.9 คู่มือการใช้งาน

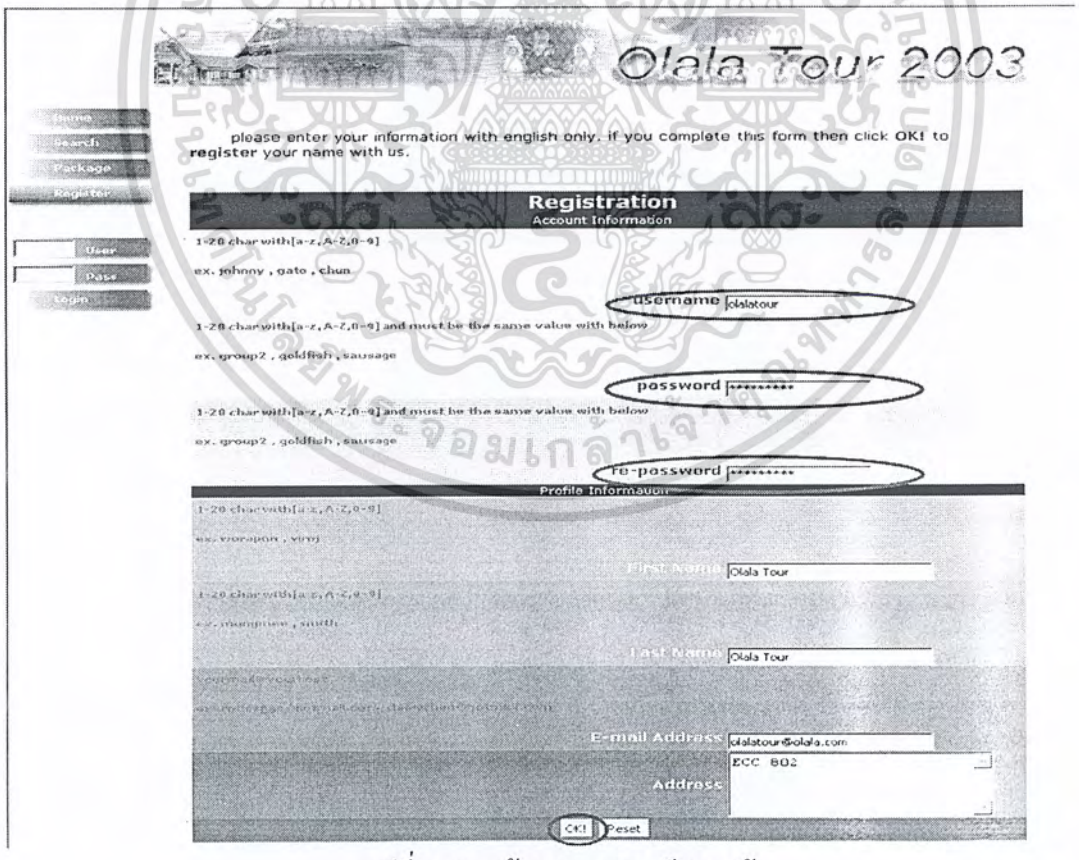
9.9.1 การลงทะเบียน

9.9.1.1 กด Register ที่ เมนูทางซ้าย



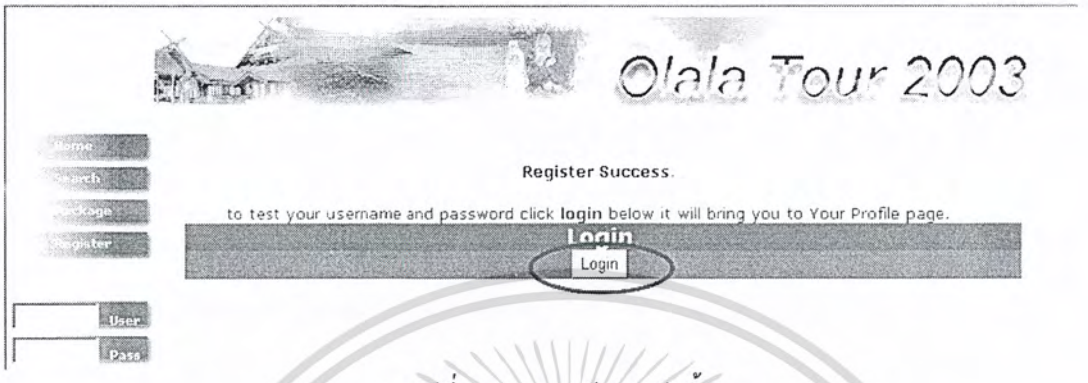
รูปที่ 9-11 Register Button

9.9.1.2 กรอกรายละเอียดต่างๆ แล้วกด OK



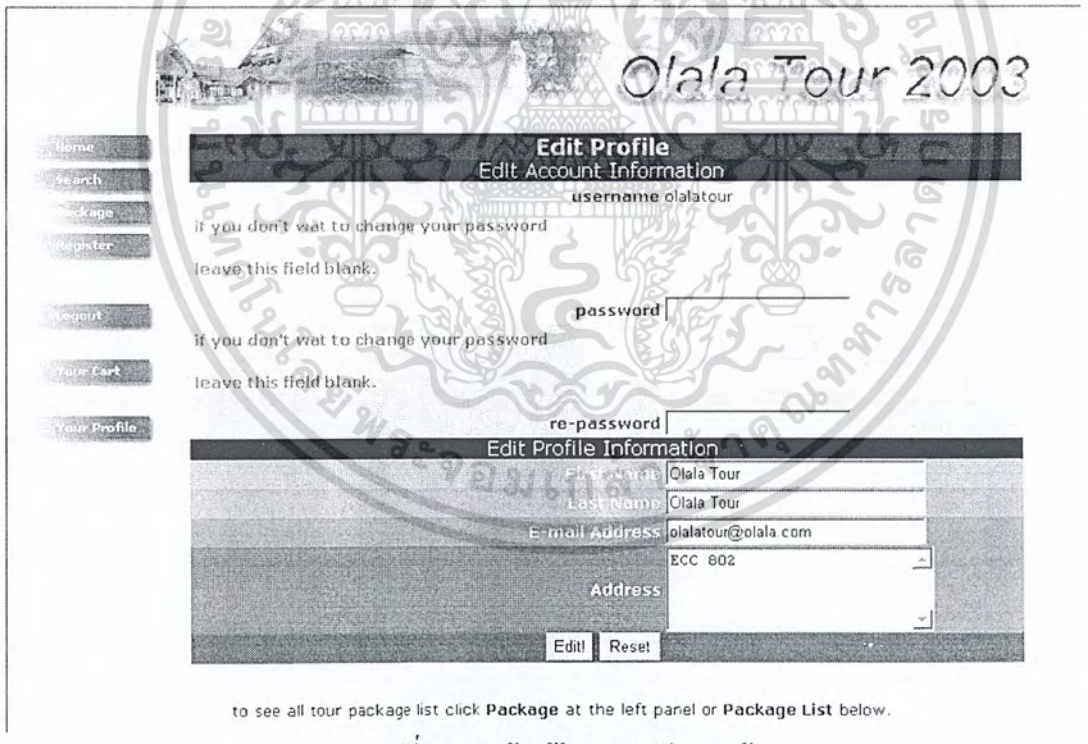
รูปที่ 9-12 หน้ากรอกรายละเอียดลูกค้า

9.9.1.3 ลงทะเบียนเสร็จสิ้น กด Login เพื่อทดลอง Login เป็นครั้งแรก



รูปที่ 9-13 ลงทะเบียนเสร็จสิ้น

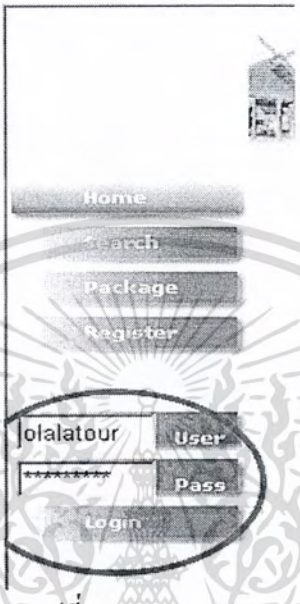
9.9.1.4 หน้าแก้ไขรายละเอียดลูกค้า ซึ่งสามารถเข้ามาได้ก็ต่อเมื่อ Login อย่างถูกต้องเท่านั้น



รูปที่ 9-14 หน้าแก้ไขรายละเอียดลูกค้า

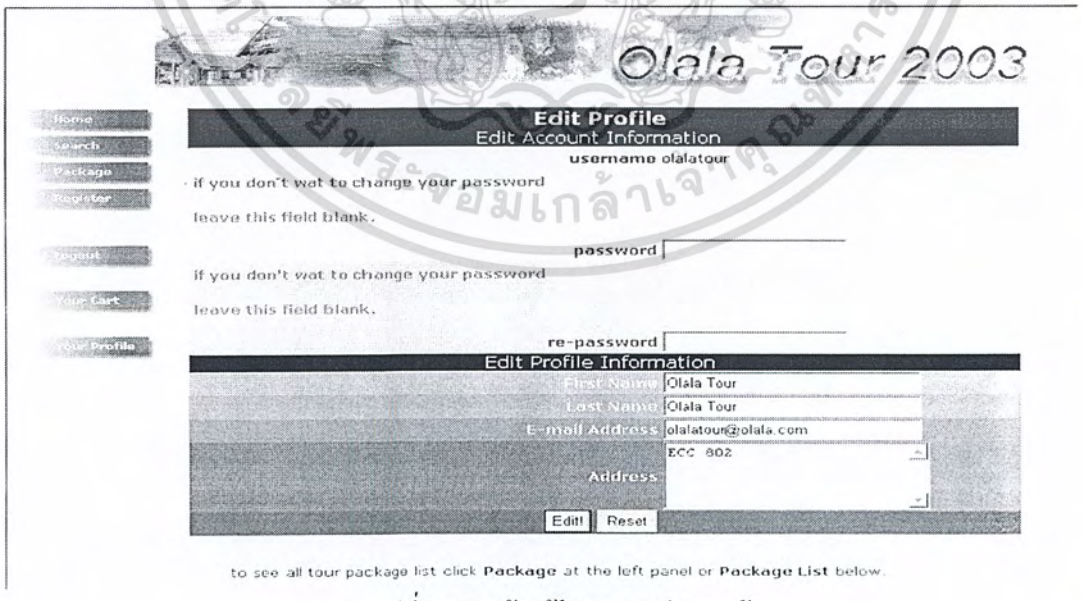
9.9.2 การเข้าสู่ระบบ

9.9.2.1 กรอก username , password แล้วกด Login



รูปที่ 9-15 Login Menu

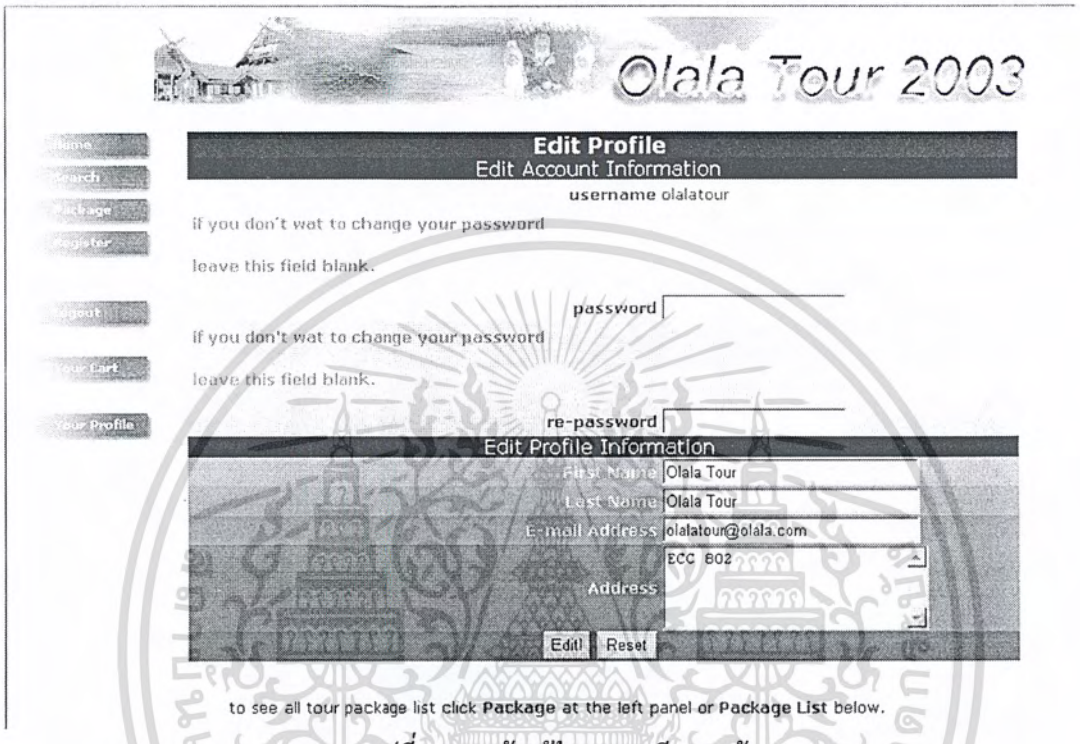
9.9.2.2 เข้าสู่หน้าแก้ไขรายละเอียดลูกค้า



รูปที่ 9-16 หน้าแก้ไขรายละเอียดลูกค้า

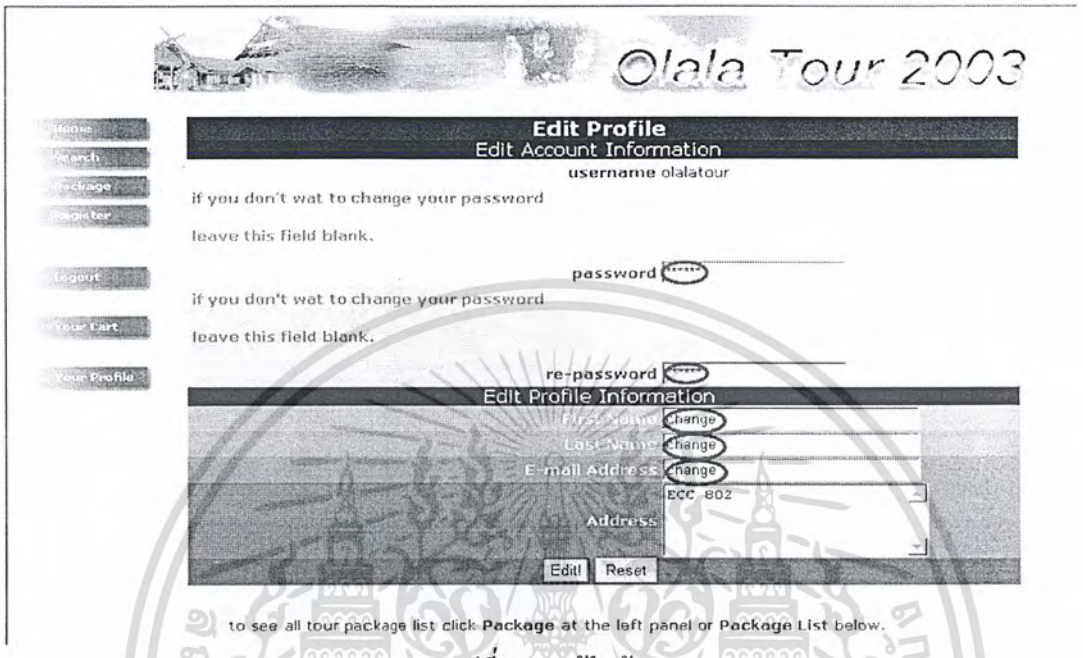
9.9.3 การแก้ไขรายละเอียดผู้ใช้

9.9.3.1 จากหน้าแก้ไขรายละเอียดลูกค้า



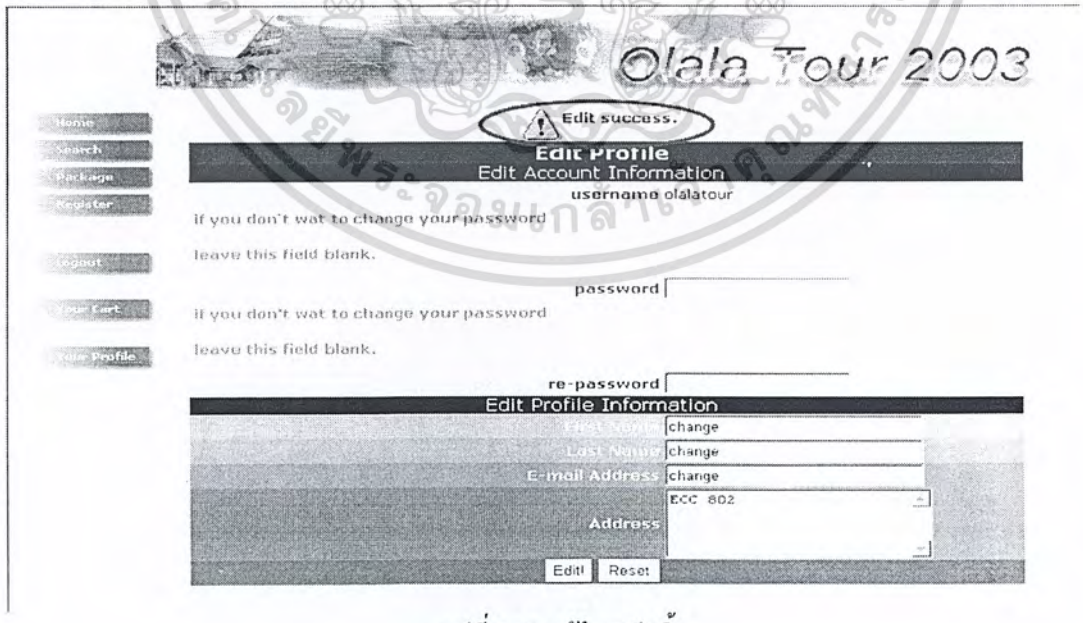
รูปที่ 9-17 หน้าแก้ไขรายละเอียดลูกค้า

9.9.3.2 แก้ไขค่าที่เปลี่ยนแปลงได้ แล้วกด Edit



รูปที่ 9-18 แก้ไขข้อมูล

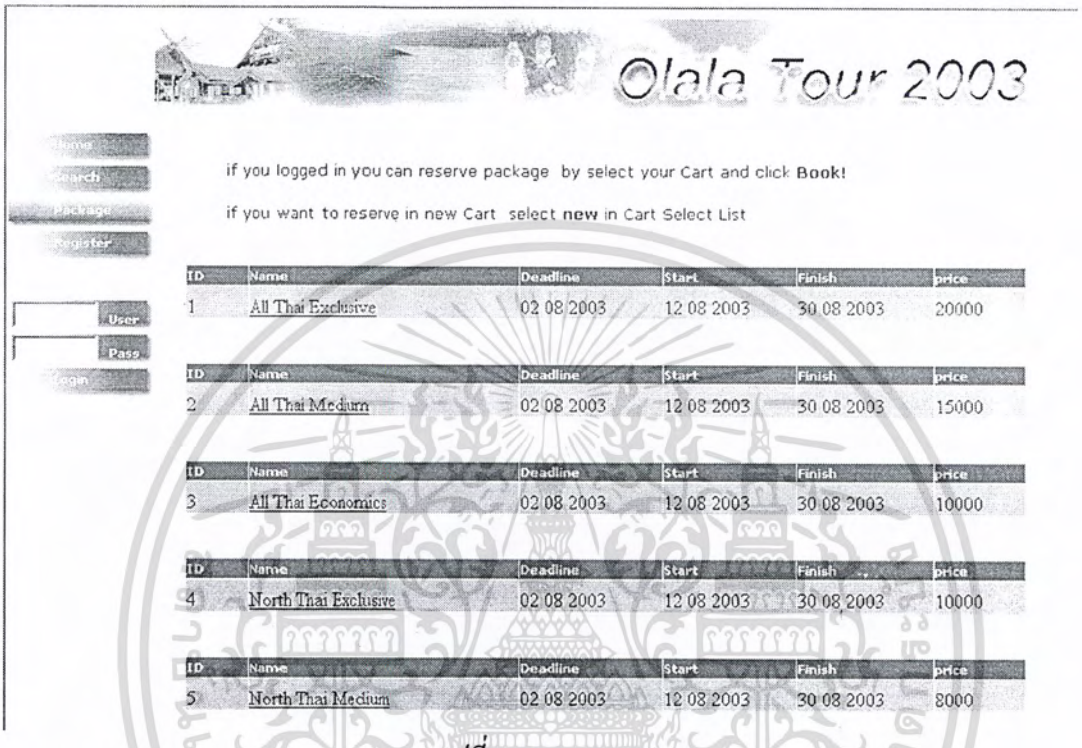
9.9.3.3 แก้ไขเสร็จสิ้น



รูปที่ 9-19 แก้ไขเสร็จสิ้น

9.9.4 การดูรายการแพ็กเกจ

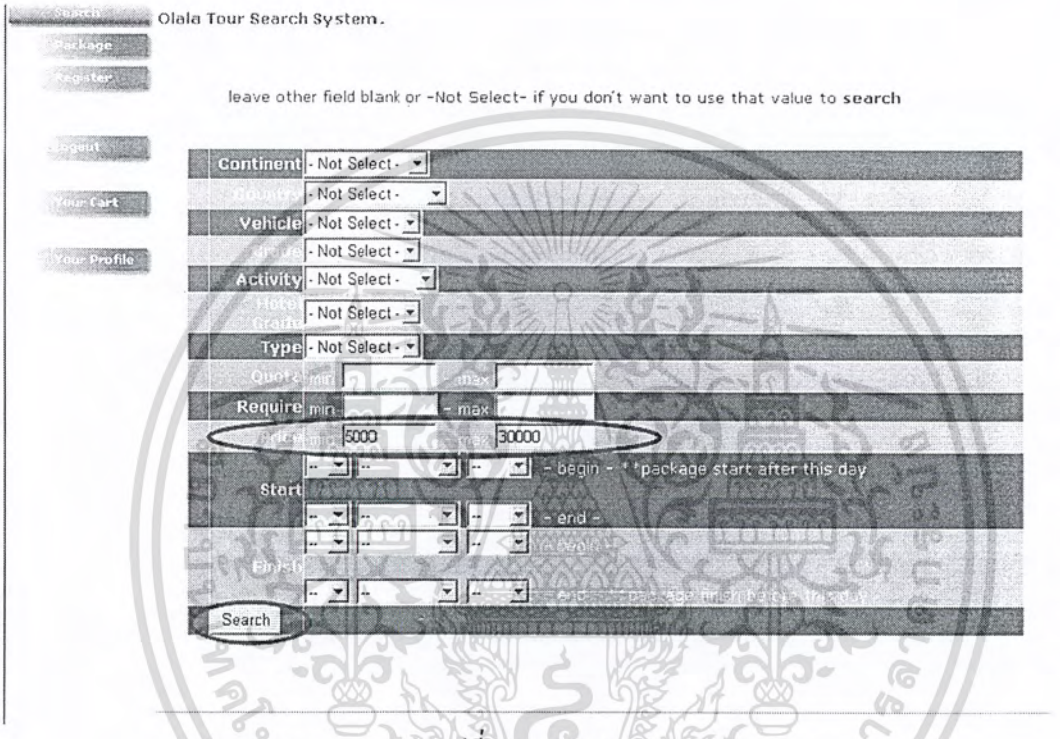
9.9.4.1 เข้าไปยังหน้าอสดงรายการแพ็กเกจโดยกดปุ่ม Package



รูปที่ 9-20 Package List

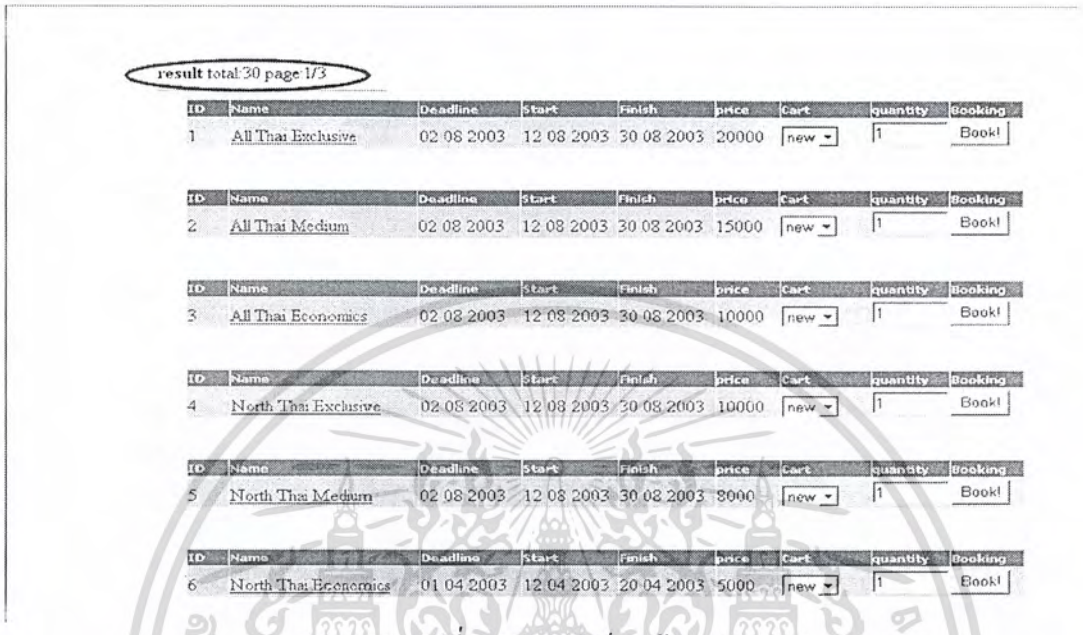
9.9.5 การค้นหาแพ็กเกจ

9.9.5.1 กด Search เพื่อไปยังหน้าค้นหา แล้วกรอกรายละเอียดแพ็กเกจที่ต้องการหาลงไป แล้วกด Search



รูปที่ 9-21 Search Menu

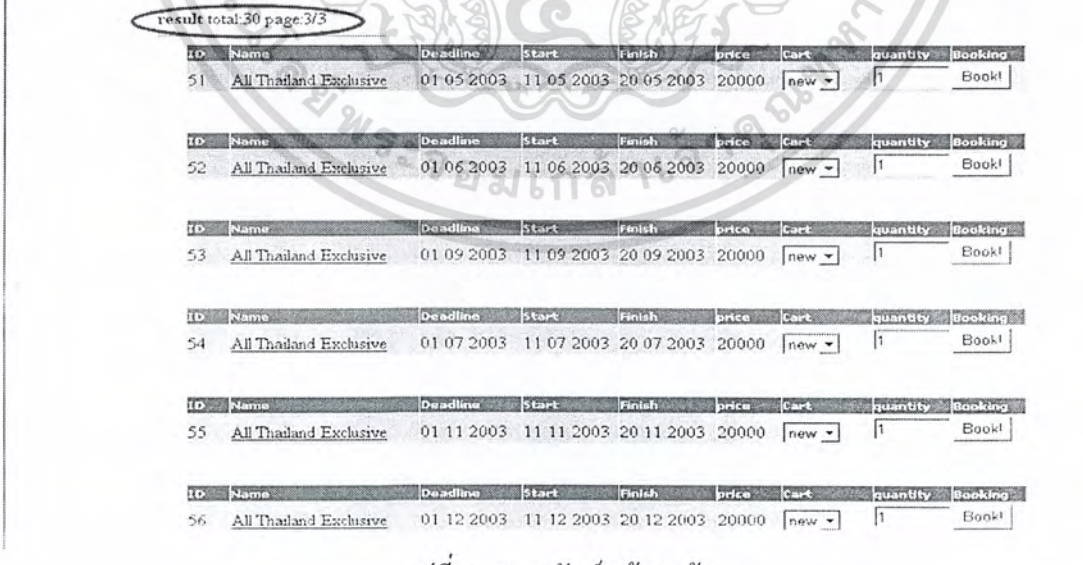
9.9.5.2 จะได้ผลลัพธ์ดังรูป


result total:30 page:1/3

ID	Name	Deadline	Start	Finish	price	Cart	quantity	Booking
1	All Thai Exclusive	02 08 2003	12 08 2003	30 08 2003	20000	new	1	Book!
2	All Thai Medium	02 08 2003	12 08 2003	30 08 2003	15000	new	1	Book!
3	All Thai Economics	02 08 2003	12 08 2003	30 08 2003	10000	new	1	Book!
4	North Thai Exclusive	02 08 2003	12 08 2003	30 08 2003	10000	new	1	Book!
5	North Thai Medium	02 08 2003	12 08 2003	30 08 2003	8000	new	1	Book!
6	North Thai Economics	01 04 2003	12 04 2003	20 04 2003	5000	new	1	Book!

รูปที่ 9-22 ผลลัพธ์การค้นหา

9.9.5.3 เลือกไปยังหน้าสุดท้าย

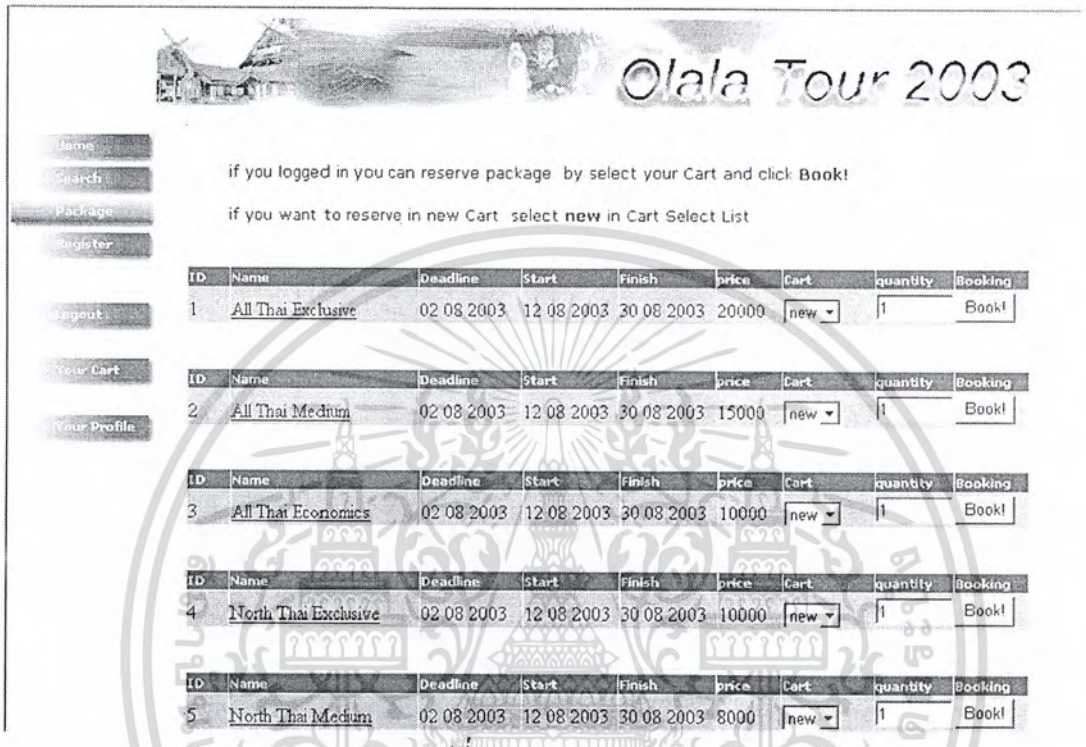

result total:30 page:3/3

ID	Name	Deadline	Start	Finish	price	Cart	quantity	Booking
51	All Thailand Exclusive	01 05 2003	11 05 2003	20 05 2003	20000	new	1	Book!
52	All Thailand Exclusive	01 06 2003	11 06 2003	20 06 2003	20000	new	1	Book!
53	All Thailand Exclusive	01 09 2003	11 09 2003	20 09 2003	20000	new	1	Book!
54	All Thailand Exclusive	01 07 2003	11 07 2003	20 07 2003	20000	new	1	Book!
55	All Thailand Exclusive	01 11 2003	11 11 2003	20 11 2003	20000	new	1	Book!
56	All Thailand Exclusive	01 12 2003	11 12 2003	20 12 2003	20000	new	1	Book!

รูปที่ 9-23 ผลลัพธ์หน้าสุดท้าย

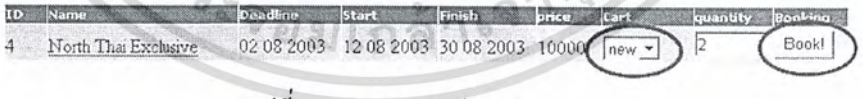
9.9.6 การจอง

9.9.6.1 เลือกแพ็กเกจที่ต้องการ



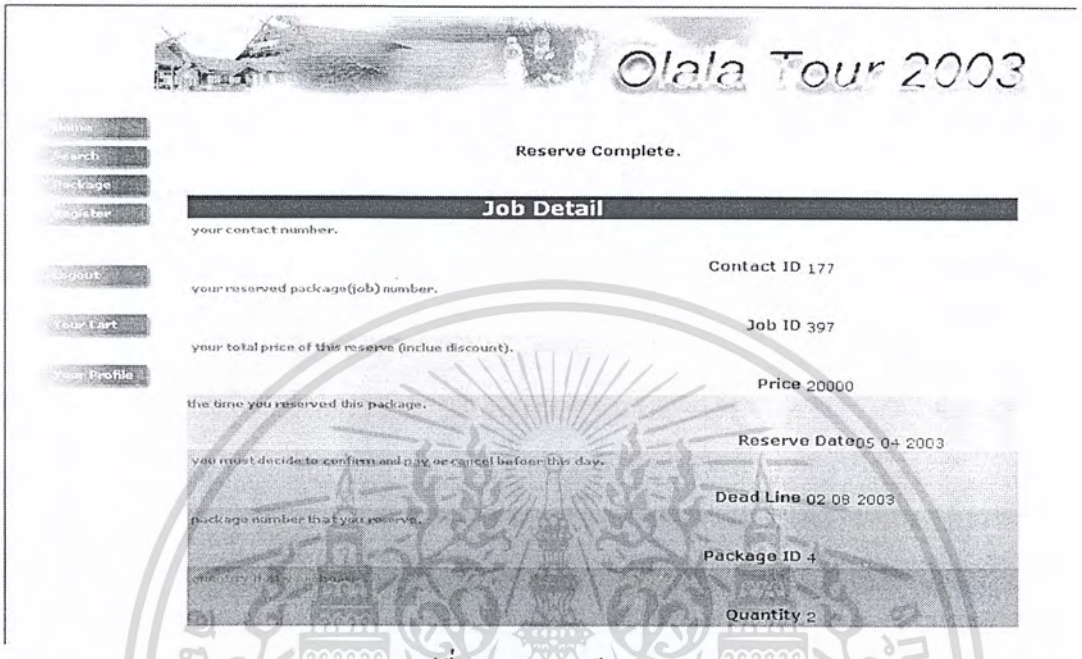
รูปที่ 9-24 รายการแพ็กเกจ

9.9.6.2 เลือกรถเข็น และจำนวน



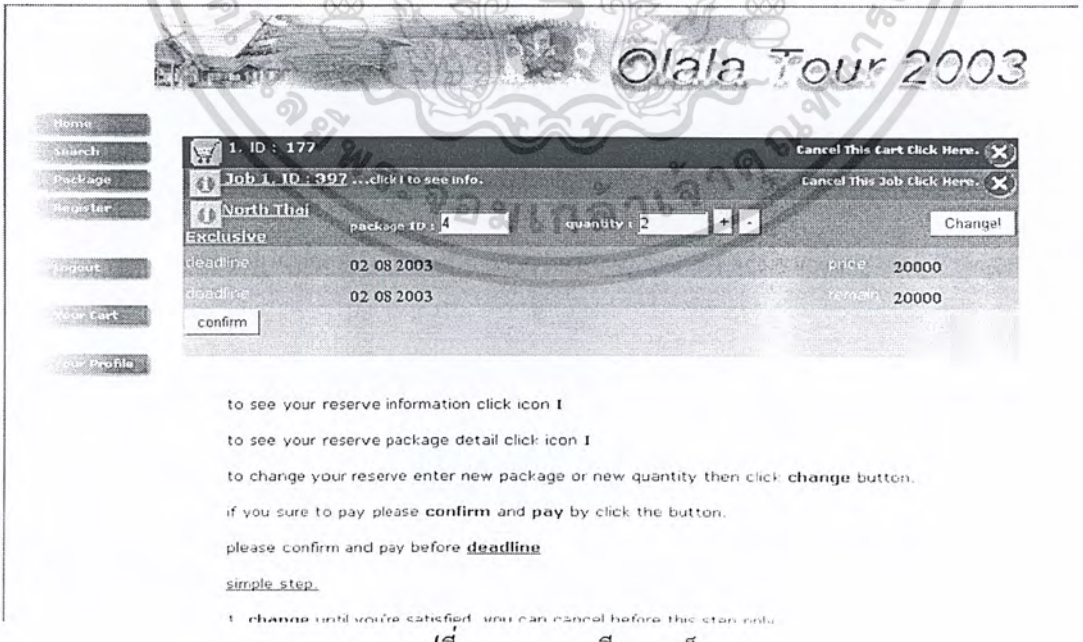
รูปที่ 9-25 รายการแพ็กเกจ

9.9.6.3 จองเสร็จสิ้น



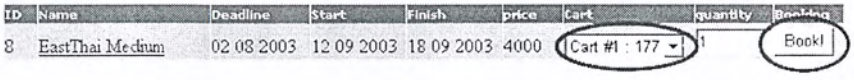
รูปที่ 9-26 รายละเอียดการจอง

9.9.6.4 แสดงรายละเอียดที่ Your Cart



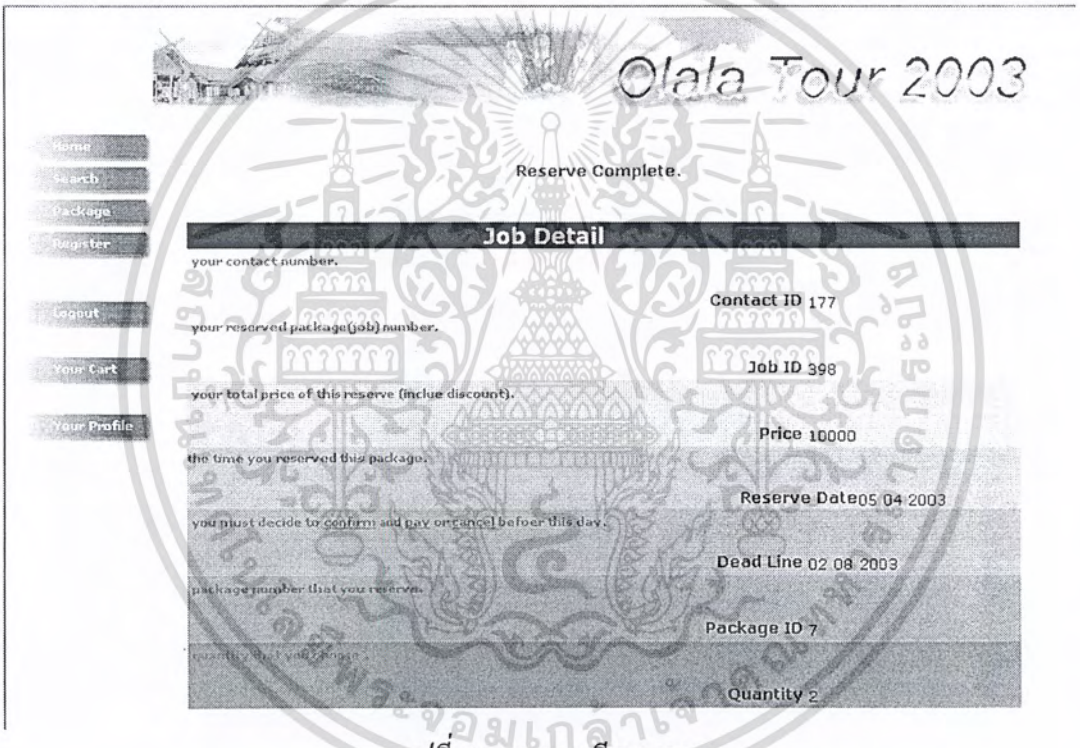
รูปที่ 9-27 รายละเอียดรถเข็น

9.9.6.5 เลือกแพ็กเกจใหม่ที่รถเข็นคันเดิม



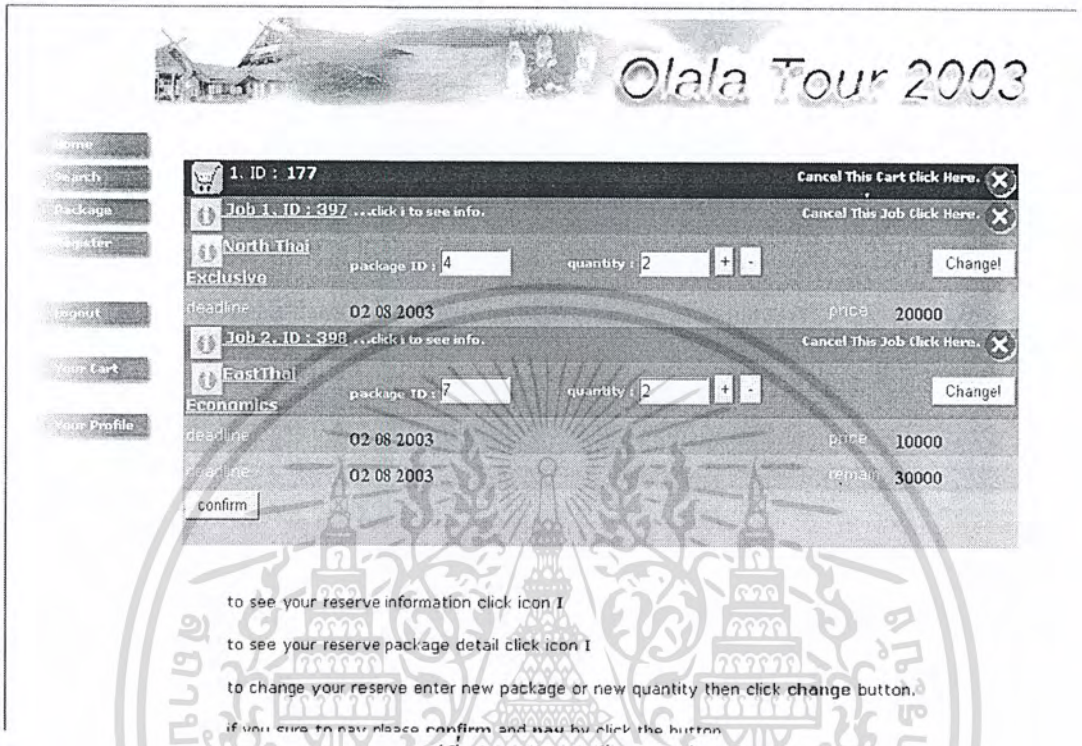
รูปที่ 9-28 รายการแพ็กเกจ

9.9.6.6 จองเสร็จสิ้น



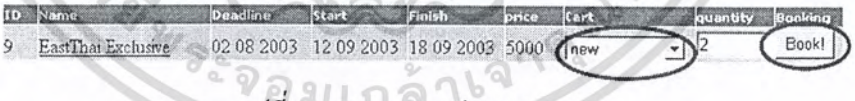
รูปที่ 9-29 รายละเอียดการจอง

9.9.6.8 มีการจองเพิ่มในรถเข็น



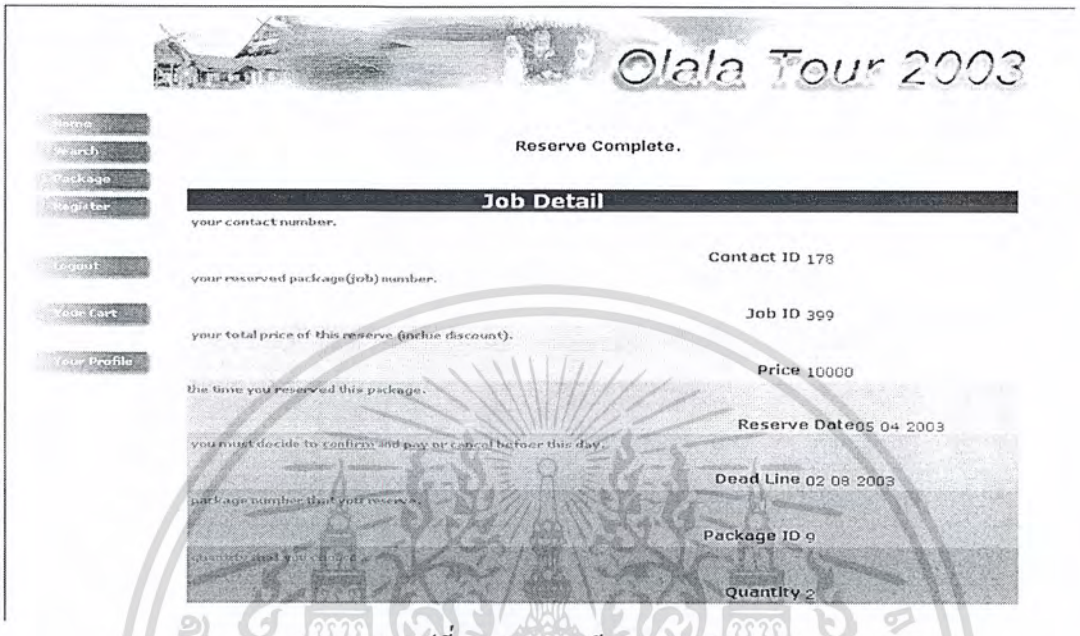
รูปที่ 9-30 รายละเอียดรถเข็น

9.9.6.9 เลือกแพ็กเกจในรถเข็นคันใหม่



รูปที่ 9-31 รายการแพ็กเกจ

9.9.6.10 จองเสร็จสิ้น



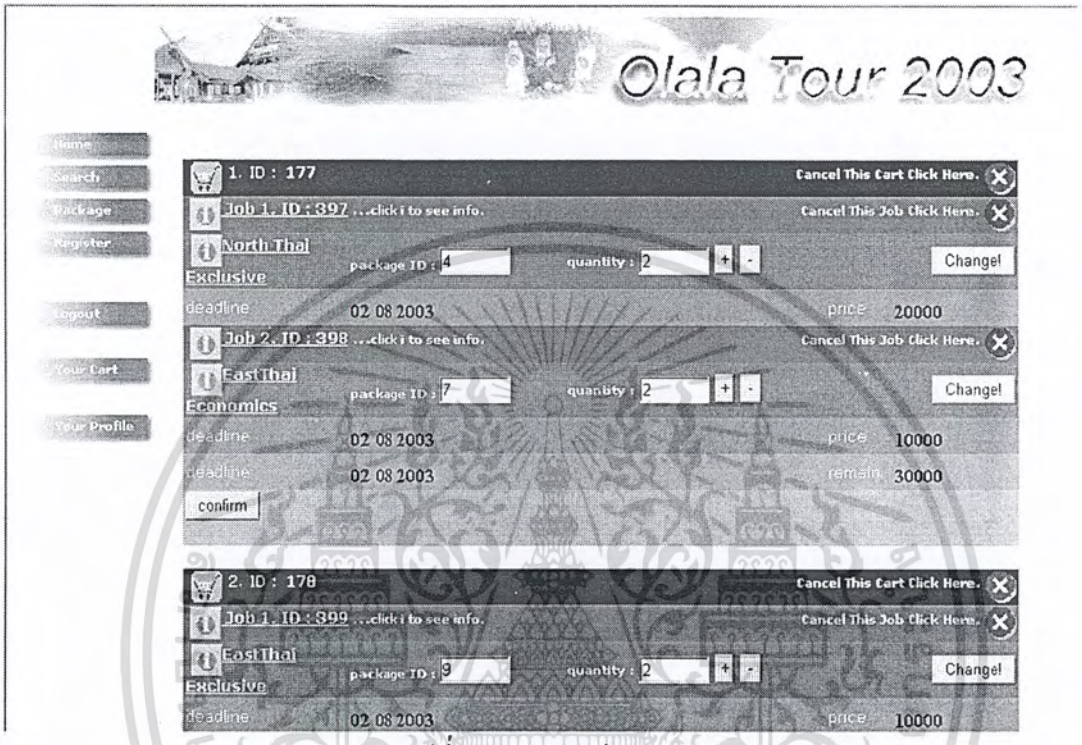
รูปที่ 9-32 รายละเอียดการจอง

9.9.6.11 เข้าไปดูรายละเอียดได้ที่ Your Cart

รูปที่ 9-33 รายละเอียดรถเข็น

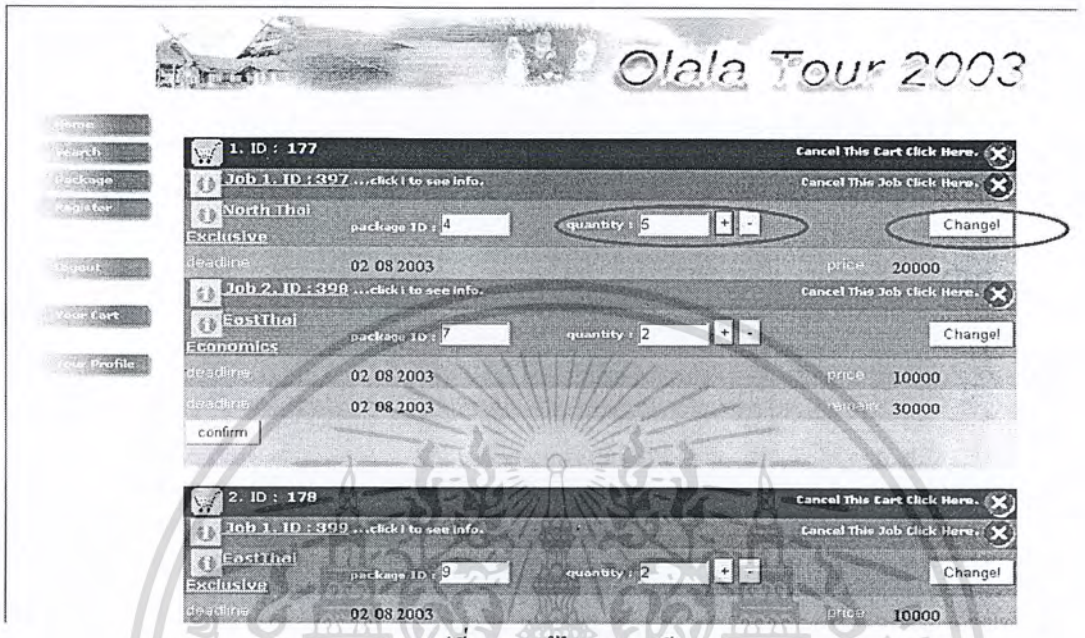
9.9.7 การแก้ไขการจอง

9.9.7.1 เลือกแพ็กเกจที่จะทำการแก้ไข



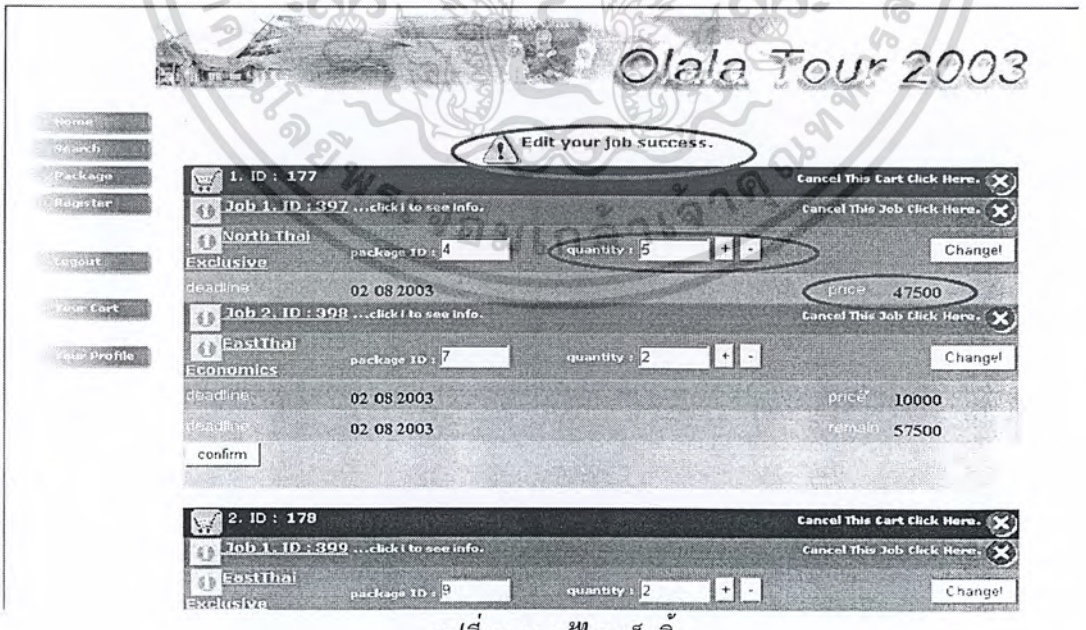
รูปที่ 9-34 รายละเอียดการจอง

9.9.7.2 แก้ไขรายละเอียด แล้วกด Change



รูปที่ 9-35 แก้ไขรายละเอียด

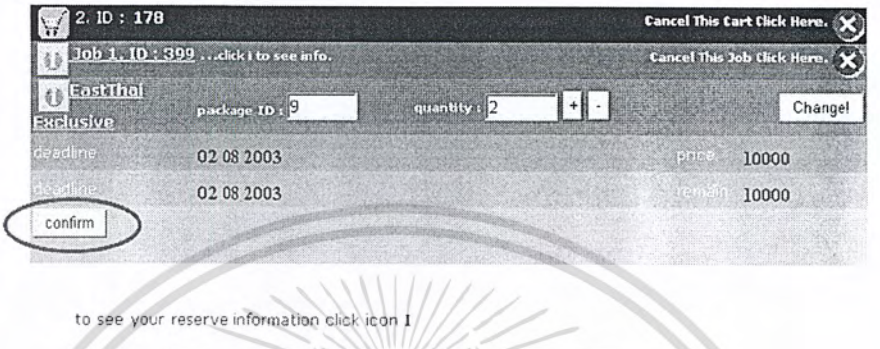
9.9.7.3 แก้ไขเสร็จสิ้น



รูปที่ 9-36 แก้ไขเสร็จสิ้น

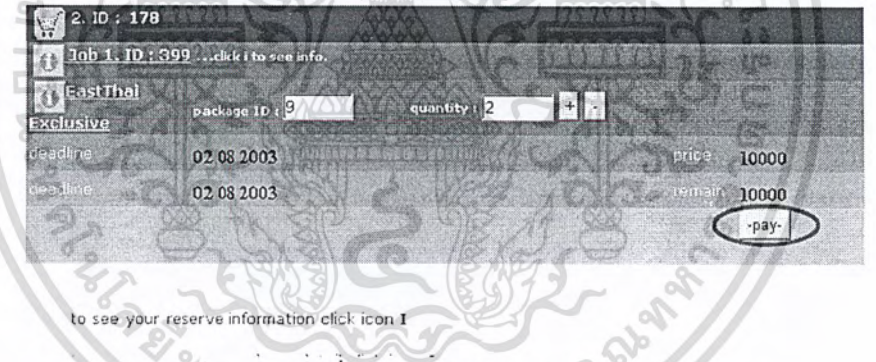
9.9.8 การจ่ายเงิน

9.9.8.1 กด Confirm ที่รายละเอียดการจอง



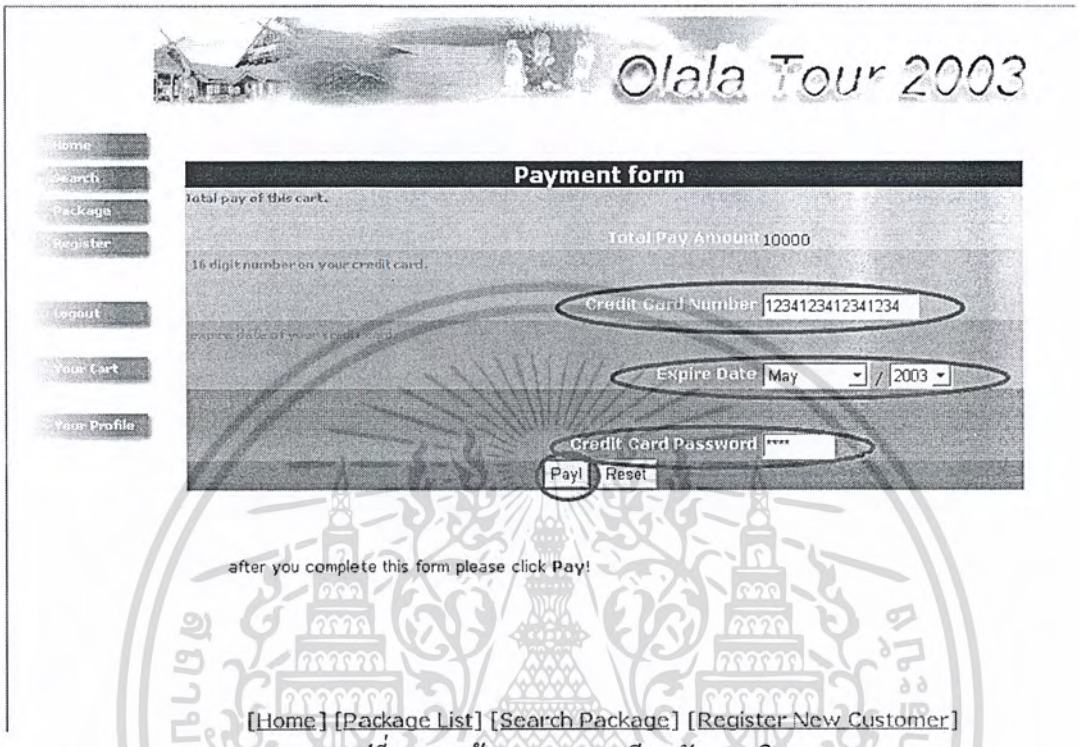
รูปที่ 9-37 Confirm Button

9.9.8.2 กด pay



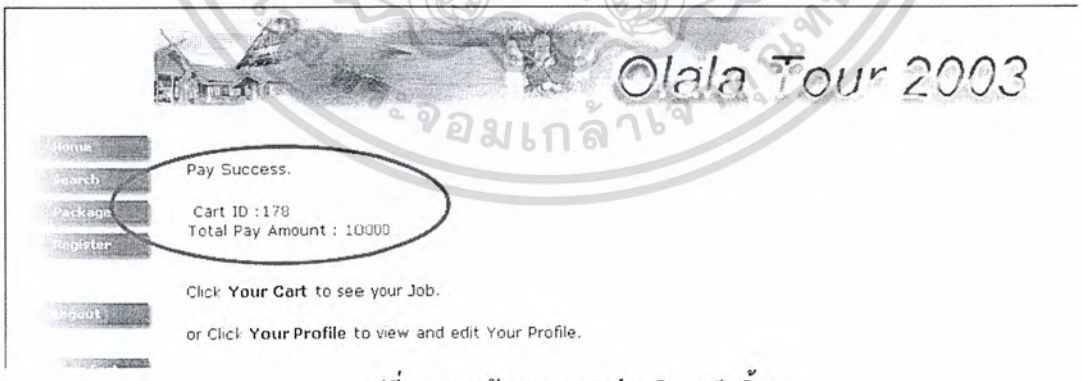
รูปที่ 9-38 pay button

9.9.8.3 กรอกรายละเอียดบัตรเครดิตของธนาคาร



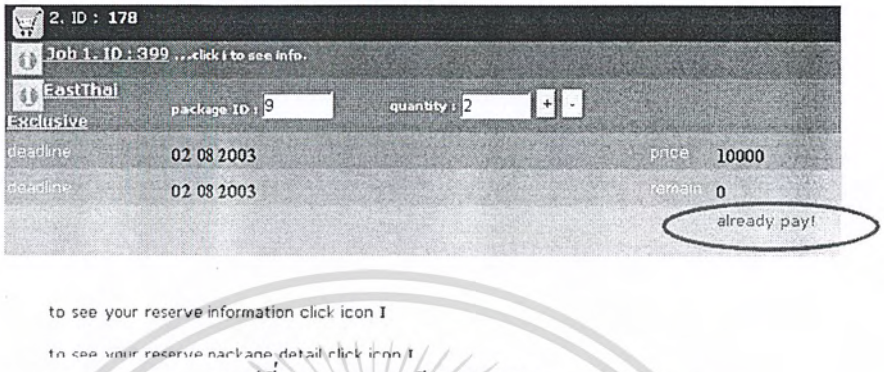
รูปที่ 9-39 หน้ากรอกรายละเอียดบัตรเครดิต

9.9.8.4 จ่ายเงินเสร็จสิ้น



รูปที่ 9-40 หน้าแสดงการจ่ายเงินเสร็จสิ้น

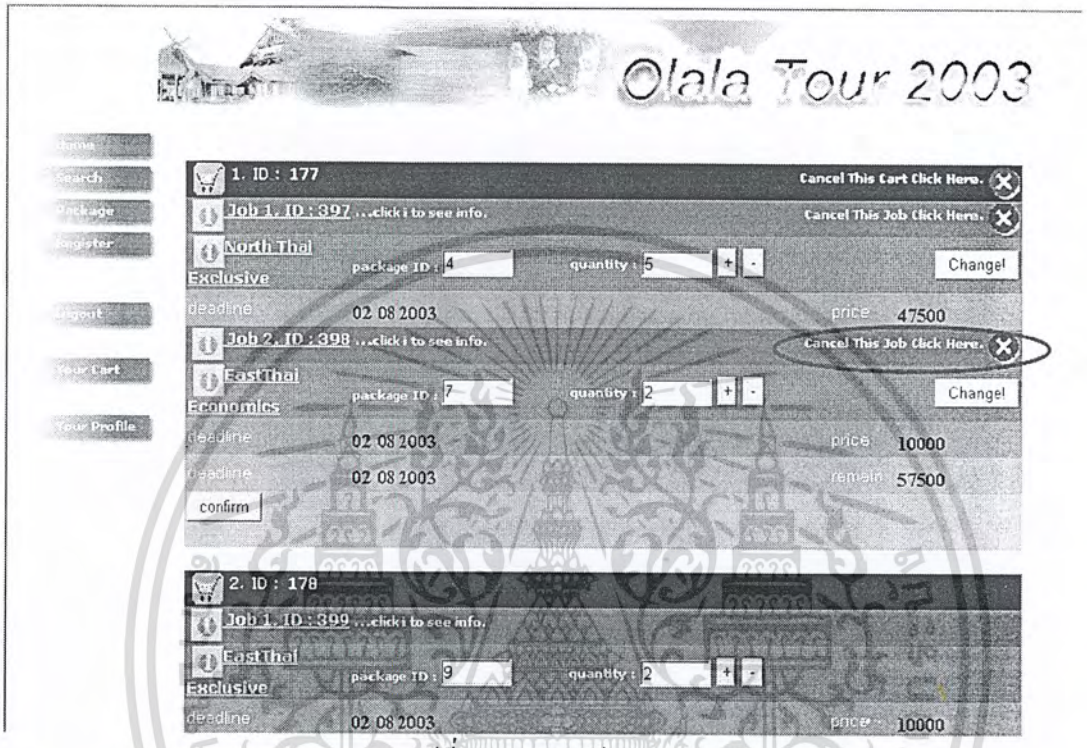
9.9.8.5 ปรากฏข้อความแสดงการจ่ายเงิน



รูปที่ 9-41 รายละเอียดการจอง

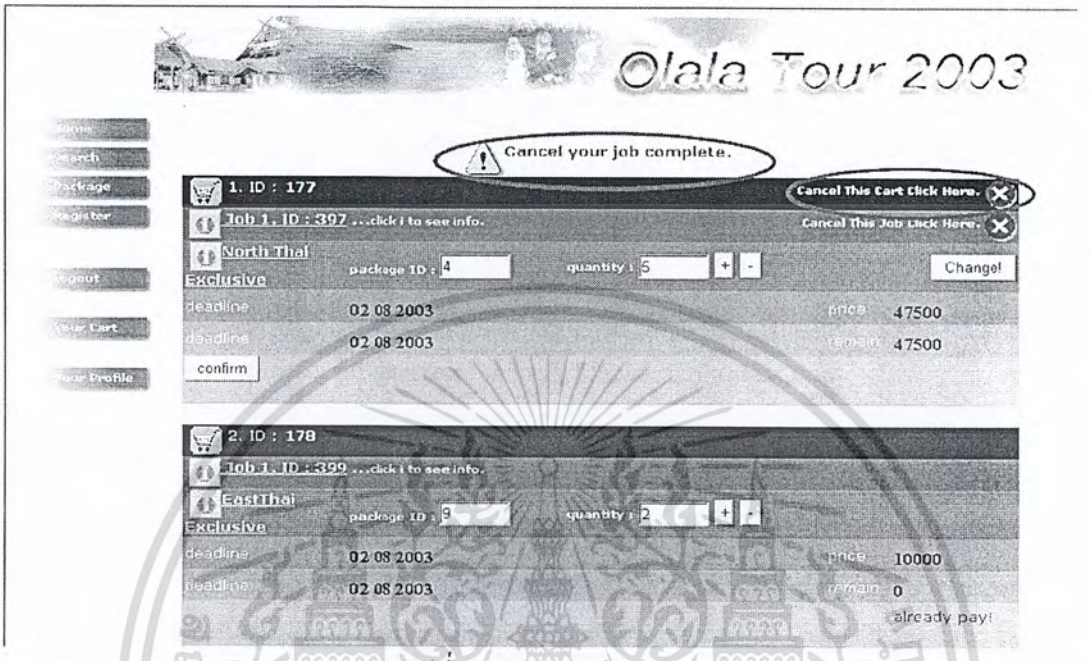
9.9.9 การยกเลิก

9.9.9.1 กดรูป X สีแดงที่แพ็กเกจที่ต้องการยกเลิก



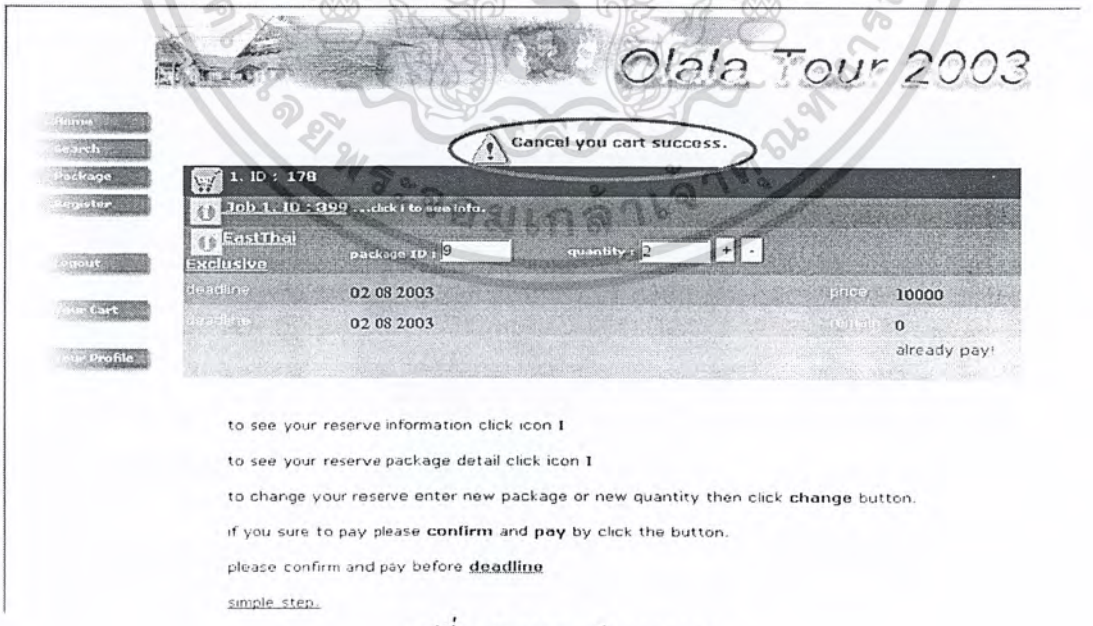
รูปที่ 9-42 รายละเอียดการจอง

9.9.9.2 ยกเลิกสำเร็จ , กด X สีแดงที่ขวาบนสุดเพื่อยกเลิกรถเข็น



รูปที่ 9-43 รายละเอียดการจอง

9.9.9.3 ลบรถเข็นเสร็จสิ้น



รูปที่ 9-44 รายละเอียดการจอง

บทที่ 10
บทวิจารณ์ และสรุป

10.1 สรุปผลการดำเนินงาน

ในโครงงานนี้ได้แบ่งการดำเนินงานออกเป็น 5 ส่วนหลัก คือ

1. ศึกษาทฤษฎีพื้นฐานของ คอมโพเน็นต์ , J2EE , EJB , เว็บเซอร์วิส และเทคโนโลยีอื่นๆ ที่เกี่ยวข้อง
2. ออกแบบระบบงาน เพื่อทำการทดสอบทฤษฎีต่างๆ ที่ได้ศึกษามา โดยตัวอย่างระบบงานที่สร้างขึ้นมา คือ ระบบการจองแพ็กเกจทัวร์ของบริษัททัวร์
3. ทำการอิมพลีเมนต์ระบบงานที่ได้ออกแบบไว้ โดยใช้ภาษาจาวาตามมาตรฐานของ J2EE
4. ทำการอิมพลีเมนต์ระบบงานในส่วนที่เรียกใช้ระบบธนาคารจำลองการโอนเงินที่พัฒนาจากเทคโนโลยีเว็บเซอร์วิสของ EJB ผ่านโพรโตคอลมาตรฐานที่เรียกว่า SOAP
5. ทดลองเพิ่มประสิทธิภาพของระบบโดยการนำ SSL มาใช้เพิ่มความปลอดภัยในการส่งข้อมูล

ระบบงานที่ทดลองสร้างขึ้น มีการใช้เทคโนโลยีต่างๆ ที่ได้ศึกษามา โดยได้ทำการสร้างระบบงานเป็น 3 เทียร์ คือ

1. พรีเซนเตชั่นเทียร์ มีไคลเอ็นต์เป็นเว็บบราวเซอร์ที่เรียกใช้บริการจากระบบและแอพพลิเคชันที่ทำหน้าที่เป็นเครื่องมือในการจัดการระบบ
2. บิสิเนสเทียร์ มี Weblogic Server ติดตั้ง
3. ดาต้าเทียร์ ใช้ดาตาเบสเซิร์ฟเวอร์เป็น Oracle 9.2i

10.2 แนวทางการพัฒนาต่อ

ถึงแม้ว่า การทดลองสร้างระบบงานเว็บเซอร์วิสให้สามารถเรียกใช้บริการจากระบบงานอื่นที่พัฒนามาจากเทคโนโลยีที่แตกต่างกันได้ (คือ .NET และ EJB) โดยผ่านทางโพรโตคอล SOAP นั้น สามารถทำงานได้เป็นอย่างดีก็ตาม แต่ก็ยังเป็นเพียงการแลกเปลี่ยนข้อมูลและเรียกใช้เซอร์วิสบริการแบบง่ายๆ เพียงแค่รับส่งข้อมูลผ่านกันได้รูปแบบของ XML เท่านั้น แต่การใช้งานความสามารถที่เพิ่มขึ้น เช่น การทำเสตทฟูลเว็บเซอร์วิสสามารถทำได้ภายในแพลตฟอร์มเดียวกัน หรือถ้าเป็นจาวาต้องภายในผลิตภัณฑ์ตัวเดียวกันด้วย การจัดการทรานแซกชันที่ไคลเอนต์ไม่สามารถทำ two phrase commit ระหว่างสองค่ายได้ รวมไปถึงหัวข้อสำคัญคือ ความปลอดภัยระหว่างการสื่อสารในขณะนี้ก็ทำโดยนำ โปรโตคอล SSL เข้ามาช่วยแต่ก็ส่งผลให้ความเร็วลดลงเป็นอย่างมาก

การนำเว็บเซอร์วิสเพื่อไปใช้งานนั้น ยังคงต้องมีการพัฒนามาตรฐานในเรื่องต่างๆออกมารองรับอีกมากมาย ดังเช่นเรื่องของทรานแซกชัน ที่คาดว่าจะมีมาตรฐานเปิดตัวออกมาในปี 2547 มาตรฐานทางด้านความปลอดภัยซึ่งทางฝั่งไมโครซอฟต์ได้ออกมาแล้ว แต่ทางฝั่งจาวาต้องรอถึงประมาณกลางปี 2546 จึงจะมีผลิตภัณฑ์ที่สนับสนุนมาตรฐานนี้ออกมา ซึ่งมาตรฐานเหล่านี้ก็เป็นสิ่งที่ควรศึกษาและนำมาพัฒนาต่อไป

บรรณานุกรม

[1] Ed Roman, Scott Ambler, Tyler Jewell (2002) : "Mastering Enterprise JavaBeans Second Edition" : John Wiley & Son, Inc., 2002

[2] http://e-docs.bea.com/wls/docs70/zip/beawlsdocs701-html.zip : Bea , 2002

[3] http://www.voelter.de/data/presentations/J2EE_vs_NET_MV.ppt : Voelter , 2002

[4] http://www.borland.com/jbuilder/download/addons.html : Borland, 2002