# การพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์ด้วยเอ็นเตอร์ไพรส์จาวาบีนส์

# Component-Based Software Development with Enterprise JavaBeans



นายทินพัฒน์ ชัยพานิช

นายประกิจ อิงคกิตติ



ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

# การพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์ด้วยเอ็นเตอร์ไพรส์จาวาบีนส์

# Component-Based Software Development with Enterprise JavaBeans

โดย

นายทินพัฒน์ ชัยพานิช

นายประกิจ อิงคกิตติ

อาจารย์ที่ปรึกษา

ดร. วรวัฒน์ ลิ้มโภคา

ดร. ชุติเมษฏ์ ศรีนิลทา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์ด้วยเอ็นเตอร์ไพรส์จาวาบีนส์

Component-Based Software Development with Enterprise JavaBeans

ผู้จัดทำ

1. นายทินพัฒน์ ชัยพานิช รหัสประจำตัว 41014160
2. นายประกิจ อิงคกิตติ รหัสประจำตัว 41014248

อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

ชุติเมษฎ์ ศรีนิลทา

อาจารย์ที่ปรึกษา

(ดร. ชุติเมษฎ์ ศรีนิลทา)

# การพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์ด้วยเอ็นเตอร์ไพรส์จาวาบีนส์

ทินพัฒน์ ชัยพานิช
ประกิจ อิงคกิตติ
ดร.วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ดร.ชุติเมษฏ์ ศรีนิลทา อาจารย์ที่ปรึกษา
ปีการศึกษา 2544

**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

นอกจากนั้นแอพพลิเคชันเซิร์ฟเวอร์ของ J2EE ยังสามารถทำคลัสเตอร์ได้ ซึ่งจะทำให้แน่ใจได้ว่าเอ็นเตอร์ไพรส์แอพพลิเคชันสามารถรองรับผู้ใช้บริการจำนวนมากได้ อีกทั้งยังป้องกันการล้มเหลวของระบบได้อีกด้วย

วิทยานิพนธ์ฉบับนี้ศึกษาถึงวิธีในการพัฒนาเอ็นเตอร์ไพรส์แอพพลิเคชัน โดยใช้ EJB และ API อื่น ๆ ของ J2EE การคลัสเตอร์แอพพลิเคชันเซิร์ฟเวอร์รวมไปถึงการเรียกคอมโพเน็นต์ข้ามสถาปัตยกรรม ซึ่งได้แก่ COM+ โดยใช้ SOAP

**Component-Based Software Development with Enterprise JavaBeans**

Tinnapat Chaipanich
Prakit Engkakitti
Dr. Worawat Limpoka Advisor
Dr. Chutimet Srinilta 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.

In addition, J2EE is designed to enable clustering service that ensures the scalability of application and also high-availability through fail over.

This thesis is to study the enterprise application development by component-based software development methodology using EJB and other J2EE API, application server clustering and services developed by COM+ Technology through SOAP.

## กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลที่มีส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์ วรวัฒน์ ลิ้มโภคา และอาจารย์ ชุติเมษฎ์ ศรีนิลทา อาจารย์ที่ปรึกษา ที่ให้ความเอาใจใส่ ช่วยแนะนำ สั่งสอน และช่วยเหลือในเรื่อง ๆ ต่าง ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก และขอขอบคุณพี่ ๆ เพื่อน ๆ ที่คอยช่วยเหลือ ให้คำปรึกษา และทำงานด้วยกันมา

สุดท้ายต้องขอขอบพระคุณ บิดา มารดา และญาติมิตร ที่อยู่เบื้องหลังความสำเร็จทั้งหมดนี้ จึงขอกราบขอบพระคุณมา ณ ที่นี้

ทินพัฒน์ ชัยพานิช
ประกิจ อิงคกิตติ

# สารบัญ

หน้าที่

# สารบัญตาราง

# สารบัญรูปภาพ

หน้าที่

# 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) เป็นสถาปัตยกรรมของบริษัทซันไมโครซิสเต็มส์ (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

นอกจากนั้นสถาปัตยกรรมทั้ง 2 นั้นยังมีความสามารถในการเพิ่มขนาด (Scalability) โดยการคลัสเตอร์แอพพลิเคชันเซิร์ฟเวอร์ ทำให้สามารถรองรับผู้ใช้บริการได้เป็นจำนวนมาก แต่ภายใน specification ของ J2EE ไม่ได้มีการระบุรายละเอียดเกี่ยวกับการทำคลัสเตอร์ไว้ แต่ละผลิตภัณฑ์จึงมีวิธีการคลัสเตอร์และเทคนิคที่ใช้ต่างกัน

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

**1.2 วัตถุประสงค์ของโครงงาน**

- ศึกษาการออกแบบและพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์
- ศึกษาสถาปัตยกรรมที่ทำงานบนฝั่งเซิร์ฟเวอร์ ข้อดีและข้อเสียของแต่ละสถาปัตยกรรม
- ศึกษาวิธีการในการทำคลัสเตอร์ และประโยชน์ของคลัสเตอร์
- ศึกษา XML และ SOAP
- สร้างแอพพลิเคชันโดยใช้สถาปัตยกรรมคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์ ที่สามารถรองรับผู้ใช้งานได้เป็นจำนวนมากและสามารถนำคอมโพเน็นต์มาใช้ใหม่ข้ามสถาปัตยกรรมได้

**1.3 ขอบเขตของโครงงาน**

โครงงานนี้พัฒนาระบบอี-คอมเมิร์ซที่เกี่ยวกับการท่องเที่ยว โดยใช้ EJB พัฒนาคอมโพเน็นต์ และแสดงผลด้วย Servlet และ JSP ที่เรียกใช้คอมโพเน็นต์ผ่านทางเว็บบราวเซอร์ ใช้ Simple Object Access Protocol (SOAP) ในการทำหน้าที่เป็นสื่อกลางในเรียกใช้คอมโพเน็นต์ของไมโครซอฟท์ (COM+) เนื่องจากสถาปัตยกรรมทั้ง 2 ตัวนี้ไม่สามารถทำงานด้วยกันได้โดยตรง นอกจากนี้ยังมีการคลัสเตอร์แอพพลิเคชันเซิร์ฟเวอร์ เพื่อเพิ่มความสามารถในการรองรับผู้ใช้บริการได้เป็นจำนวนมาก และป้องกันการล้มเหลวของระบบได้ ซึ่งเป็นคุณสมบัติที่สำคัญของเว็บแอพพลิเคชันที่ต้องให้บริการได้ตลอดเวลา โดยระบบที่จะพัฒนาคือ ระบบรถโดยสาร และ ระบบสายการบิน และระบบทัวร์ที่จะมีการเรียกใช้ระบบรถโดยสาร , ระบบสายการบิน และระบบโรงแรมที่สร้างด้วยคอมโพเน็นต์ของไมโครซอฟท์โดยใช้ SOAP

**1.4 วิธีการดำเนินงาน**

การศึกษาในโครงงานเริ่มจากศึกษาทฤษฎีพื้นฐานของการพัฒนาเชิงคอมโพเน็นต์ , J2EE ซึ่งเป็นเทคโนโลยีสถาปัตยกรรมบนฝั่งเซิร์ฟเวอร์ที่ใช้ภาษาจาวาในการพัฒนา ประกอบด้วย EJB , Servlet , JSP , JNDI และ JDBC เป็นต้น โดยใช้แนวคิดของการทำซอฟต์แวร์ให้มีลักษณะที่นำกลับมาใช้ใหม่ได้และขยายเพิ่มเติมได้ง่าย และยังมีส่วนที่ใช้ในการเรียกใช้คอมโพเน็นต์ข้ามสถาปัตยกรรม ประกอบด้วย XML (eXtensible Markup Language) และ SOAP (Simple Object Access Protocol) โดยจะแบ่งการดำเนินงานเป็น 2 ส่วนคือ ส่วนแอพพลิเคชัน จะศึกษาและออกแบบระบบอี-คอมเมิร์ซ ส่วนสถาปัตยกรรม (Architecture) เป็นส่วนของการออกแบบการทำคลัสเตอร์เพื่อเพิ่มความสามารถในการรองรับผู้ใช้บริการได้เป็นจำนวนมาก และป้องกันการล้มเหลวของระบบได้

# บทที่ 2
# ทฤษฎีคอมโพเน็นต์

## 2.1 นิยามของคอมโพเน็นต์

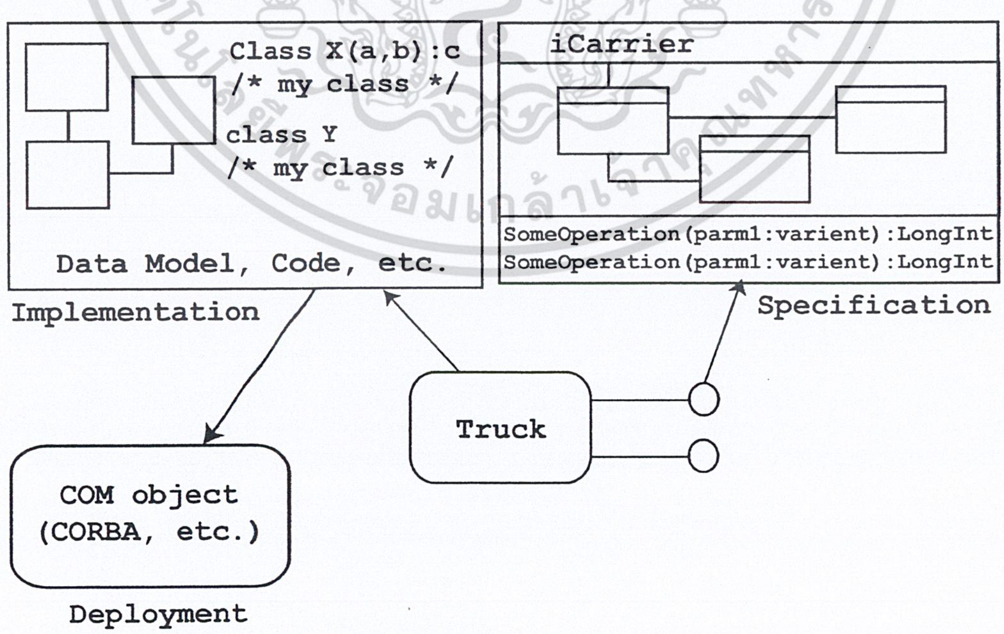
คอมโพเน็นต์คือชิ้นส่วนของซอฟต์แวร์ที่สามารถนำกลับมาใช้ใหม่ได้ (reuse) ในการที่จะนำคอมโพเน็นต์ที่มีอยู่ไปใช้จะต้องทำความเข้าใจว่าคอมโพเน็นต์นั้นใช้ทำอะไร คอมโพเน็นต์จึงเหมือนกล่องดำ (black box) ที่เราไม่จำเป็นต้องรู้ว่ามันมีขั้นตอนการทำงานอย่างไร เรารู้แค่ว่ามันใช้ทำอะไรและจะเรียกใช้งานมันอย่างไร

## 2.2 ส่วนประกอบของคอมโพเน็นต์

คอมโพเน็นต์จะประกอบด้วยส่วนสำคัญ 3 ส่วนดังนี้

- **อินเทอร์เฟซ** (interface) - คอมโพเน็นต์จะถูกเรียกใช้งานผ่านทางอินเทอร์เฟซ
- **อิมพลีเมนต์เตชัน** (implementation) - เป็นโค้ดที่กำหนดการทำงานของคอมโพเน็นต์
- **ดีพลอยเมนต์** (deployment) – เป็นเอ็กซิคิวต์ไฟล์จะใช้ในการทำให้คอมโพเน็นต์ทำงานได้ ทำหน้าที่จัดหารันไทม์เอ็นไวรอนเมนต์ในการควบคุมการทำงานของคอมโพเน็นต์และจัดหาเซอร์วิสที่จำเป็น

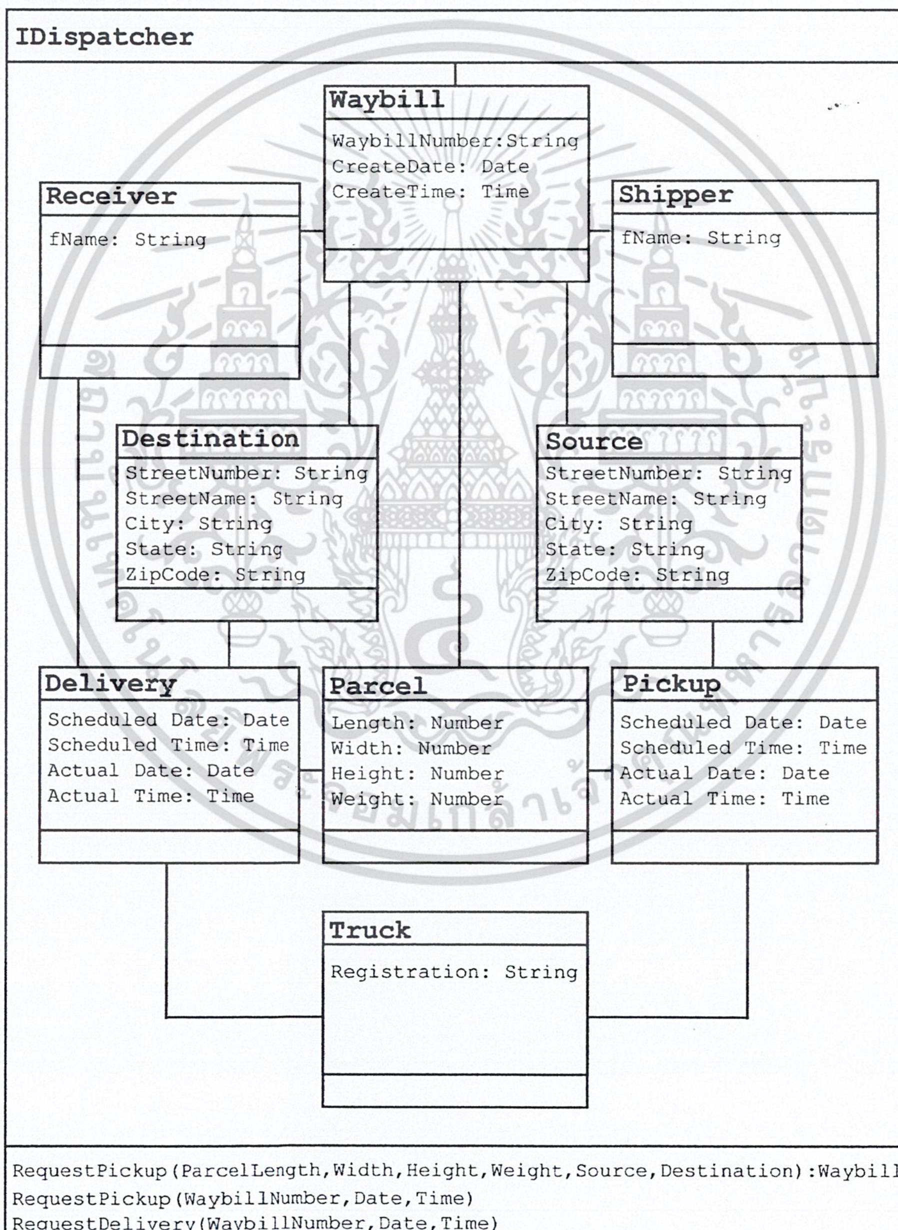
ส่วนประกอบต่าง ๆ ของคอมโพเน็นต์ดังรูปที่ 2-1




*รูปที่ 2-1 คอมโพเน็นต์และส่วนประกอบต่าง ๆ*

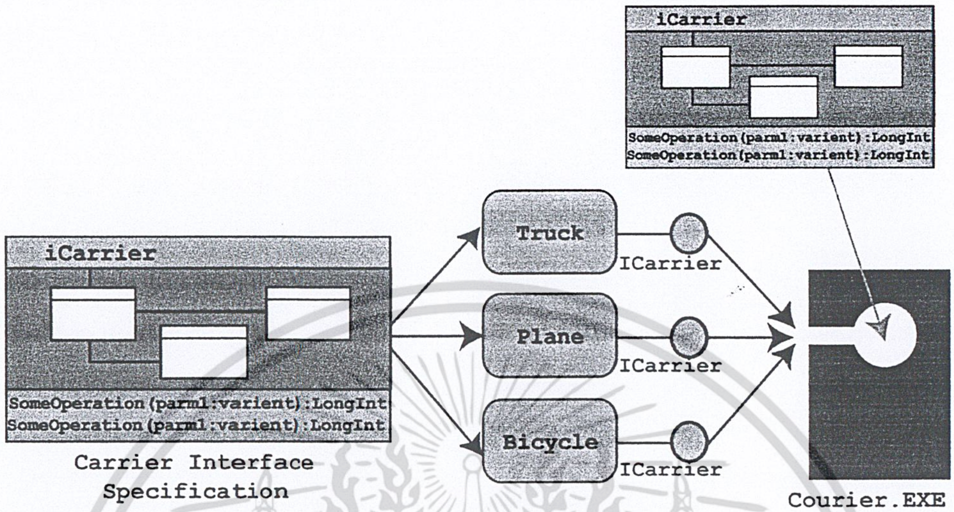
## 2.3 คุณสมบัติของคอมโพเน็นต์

- **Encapsulate** - เป็นกระบวนการในการซ่อนโค้ดหรืออิมพลีเมนต์เตชันของคอมโพเน็นต์ ผู้ใช้จะรู้เพียง อินเทอร์เฟซและจะใช้งานผ่านมัน โดยไม่จำเป็นต้องรู้ถึงอิมพลีเมนต์เตชัน ทำให้การเปลี่ยนแปลงอิมพลีเมนต์เตชันไม่มีผลกระทบต่อผู้ใช้ ตัวอย่างอินเทอร์เฟซดังรูปที่ 2-2
- **Descriptive** - เนื่องจากคอมโพเน็นต์จะต้องติดต่อผ่านอินเทอร์เฟซเท่านั้น มันจะต้องมีอินฟอร์เมชันของมันเองที่ผู้ใช้สามารถจะเข้าใจได้ โดยอินฟอร์เมชันนั้นจะต้องอธิบายถึงอินเทอร์เฟซ , อิมพลีเมนต์เตชัน และดีพลอยเมนต์



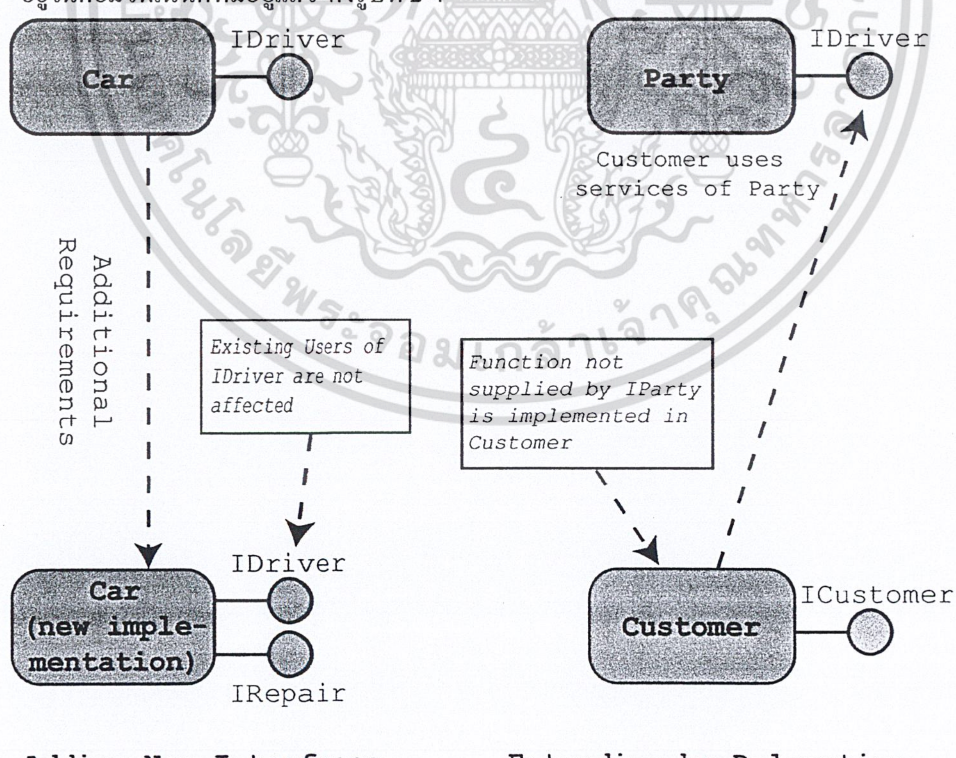

*รูปที่ 2-2 ตัวอย่างของอินเทอร์เฟซ*

- **Replaceable** - คอมโพเน็นต์มีความสามารถในการเปลี่ยนคอมโพเน็นต์หนึ่งกับคอมโพเน็นต์ใด ๆ ที่มีอินเทอร์เฟซเหมือนกัน ดังรูปที่ 2-3




*รูปที่ 2-3 คอมโพเน็นต์และการแทนที่*

- **Extensible** - สามารถที่จะเพิ่มความสามารถได้ มี 2 วิธีคือ การเพิ่มอินเทอร์เฟซและมอบหมายหน้าที่ (Delegating Responsibility) การสร้างคอมโพเน็นต์ใหม่สามารถมอบหมายหน้าที่ให้กับบริการที่มีอยู่ในคอมโพเน็นต์ที่มีอยู่แล้ว ดังรูปที่ 2-4



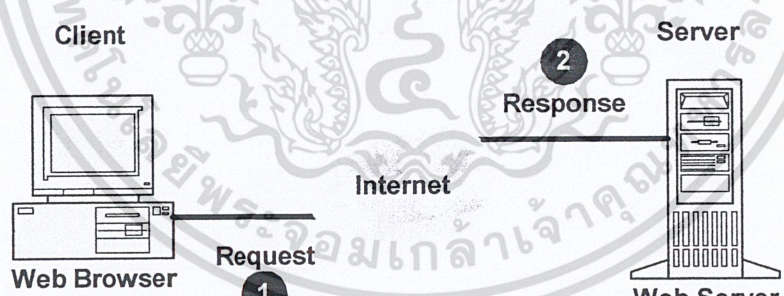

*รูปที่ 2-4 การเพิ่มความสามารถคอมโพเน็นต์*

# บทที่ 3
# สถาปัตยกรรมบนฝั่งเซิร์ฟเวอร์

Enterprise JavaBeans 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) ของแอพพลิเคชันในแต่ละมิดเดิลแวร์ของแต่ละผู้ผลิต

**3.1 วิวัฒนาการของแอพพลิเคชันบนฝั่งเซิร์ฟเวอร์ (Server-Side Application)**

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

**3.2 สถาปัตยกรรมแบบมัลติเทียร์ (Multi-tier Architecture)**

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

- **ธนาคาร -** เครื่องเอทีเอ็มจำนวนมากจะต้องติดต่อเข้าไปยังเครื่องเซิร์ฟเวอร์กลางของธนาคาร
- **สาขาร้านค้า -** เช่นร้านค้าในเครือข่ายของ Wal-Mart โดยร้านค้าแต่ละร้านจะส่งรายละเอียดการซื้อสินค้าของลูกค้าแต่ละรายไปยังเซิร์ฟเวอร์กลางของ 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

- **Presentation Layer** – ประกอบด้วยคอมโพเน็นต์ที่จัดการกับ user interface และ user interaction พรีเซนเทชันเลเยอร์ของแอพพลิเคชันแบบ stand-alone อาจเขียนด้วยภาษาวิชวลเบสิค ส่วนพรีเซนเทชันเลเยอร์ของดีพลอยเมนต์แบบ Web-base นั้น จะใช้ Java Servlet , JSP , หรือ Java Applet
- **Business Logic Layer** - ประกอบด้วยคอมโพเน็นต์ที่ใช้ในการแก้ปัญหาทางธุรกิจ คอมโพเน็นต์เหล่านี้อาจเป็น engine ที่มีความสามารถสูง เช่น engine ในการจัดการแคตาล็อก หรือ engine ในการคำนวณราคาสินค้า โดยปกติแล้วคอมโพเน็นต์เหล่านี้จะต้องถูกเขียนขึ้นโดยภาษาที่มีคุณสมบัติ type-safe เช่น ภาษาจาวา หรือภาษา C++

- **Data Layer** - ถูกเรียกใช้โดยบิซิเนสลอจิกเลเยอร์เพื่อเก็บข้อมูลแบบถาวร โดยทั่วไปดาต้าเลเยอร์จะเป็นระบบฐานข้อมูล

ข้อดีของการแบ่งคอมโพเน็นต์เป็นเลเยอร์คือ ทำให้สามารถเปลี่ยนแปลงแต่ละเลเยอร์ได้โดยจะไม่มีผลกระทบต่อเลเยอร์อื่น ๆ เช่น เราสามารถเปลี่ยนแปลงหน้าจออินเทอร์เฟซในพรีเซนเทชันเลเยอร์ได้ โดยไม่มีผลกระทบต่อเลเยอร์อื่น

แต่การแบ่งลักษณะการทำงานออกเป็นเลเยอร์นี้ เป็นการแบ่งในทางความคิดเท่านั้น ส่วนในทางกายภาพเราจะแบ่งออกเป็นหน่วยที่เรียกว่า "เทียร์" (tier) โดยสถาปัตยกรรมแบบ 2 เทียร์ คือการแบ่งเลเยอร์ทั้งสามออกเป็นสองส่วนในทางกายภาพ และสถาปัตยกรรมแบบ 3 เทียร์ คือการแบ่งเลเยอร์ทั้งสามออกเป็นสามส่วนในทางกายภาพ

**3.2.1 สถาปัตยกรรม 2 เทียร์ (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)

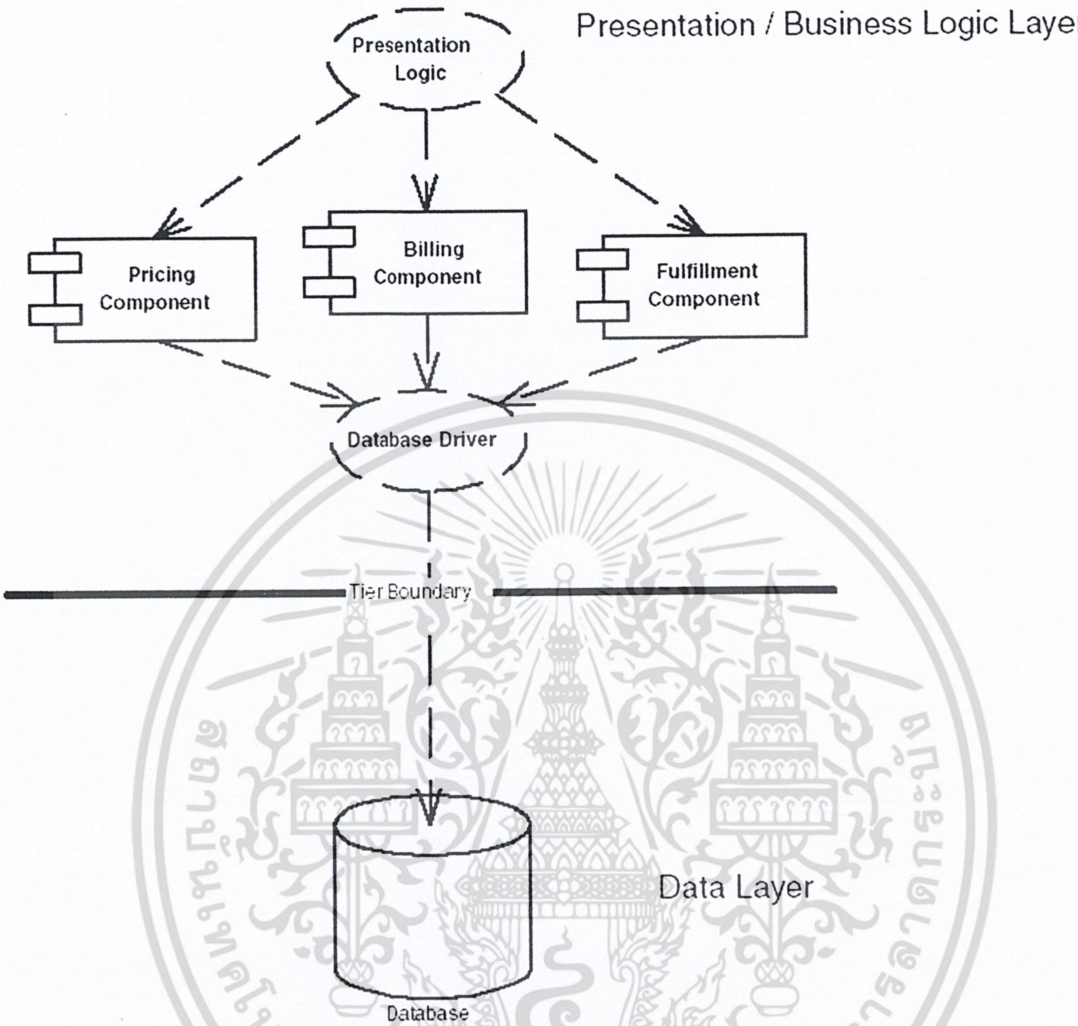
ลักษณะของสถาปัตยกรรมแบบ 2 เทียร์

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 ของระบบฐานข้อมูลเชิงสัมพันธ์ (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 จะยังอยู่และไคลเอ็นต์อื่นไม่สามารถนำมันไปใช้ได้

7) ประสิทธิภาพของระบบเครือข่ายลดลง

ในแต่ละครั้งที่ไคลเอ็นต์ทำการติดต่อกับฐานข้อมูล จะมีการส่งข้อมูลจำนวนมากไปมา ระหว่างบิซิเนสลอจิกเลเยอร์ และดาต้าเลเยอร์ ซึ่งหากว่าตัวกลางระหว่างเลเยอร์ทั้งสองนี้เป็นระบบเครือข่าย อาจทำให้ข้อมูลในระบบเครือข่ายนั้นมากจนเกิดการติดขัด ซึ่งจะมีผลให้การติดต่อกับฐานข้อมูลนั้นทำได้ช้าลง

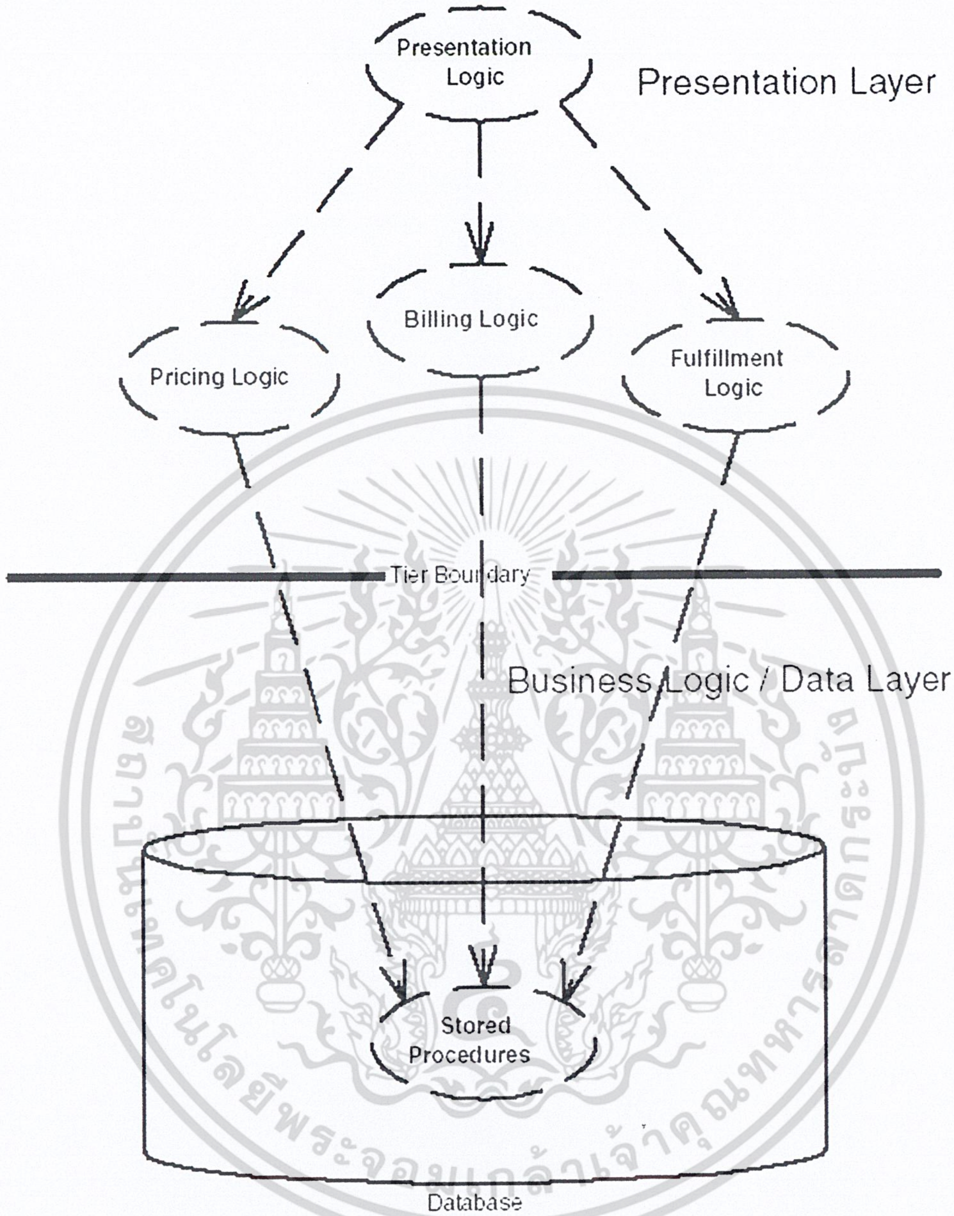
- **การรวมส่วนบิซิเนสลอจิกเลเยอร์บางส่วน เข้ากับดาต้าเลเยอร์**

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 และประสิทธิภาพในการทำงานเนื่องจากว่าลอจิกบางส่วนถูกรวมเข้ากับฐานข้อมูล 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)

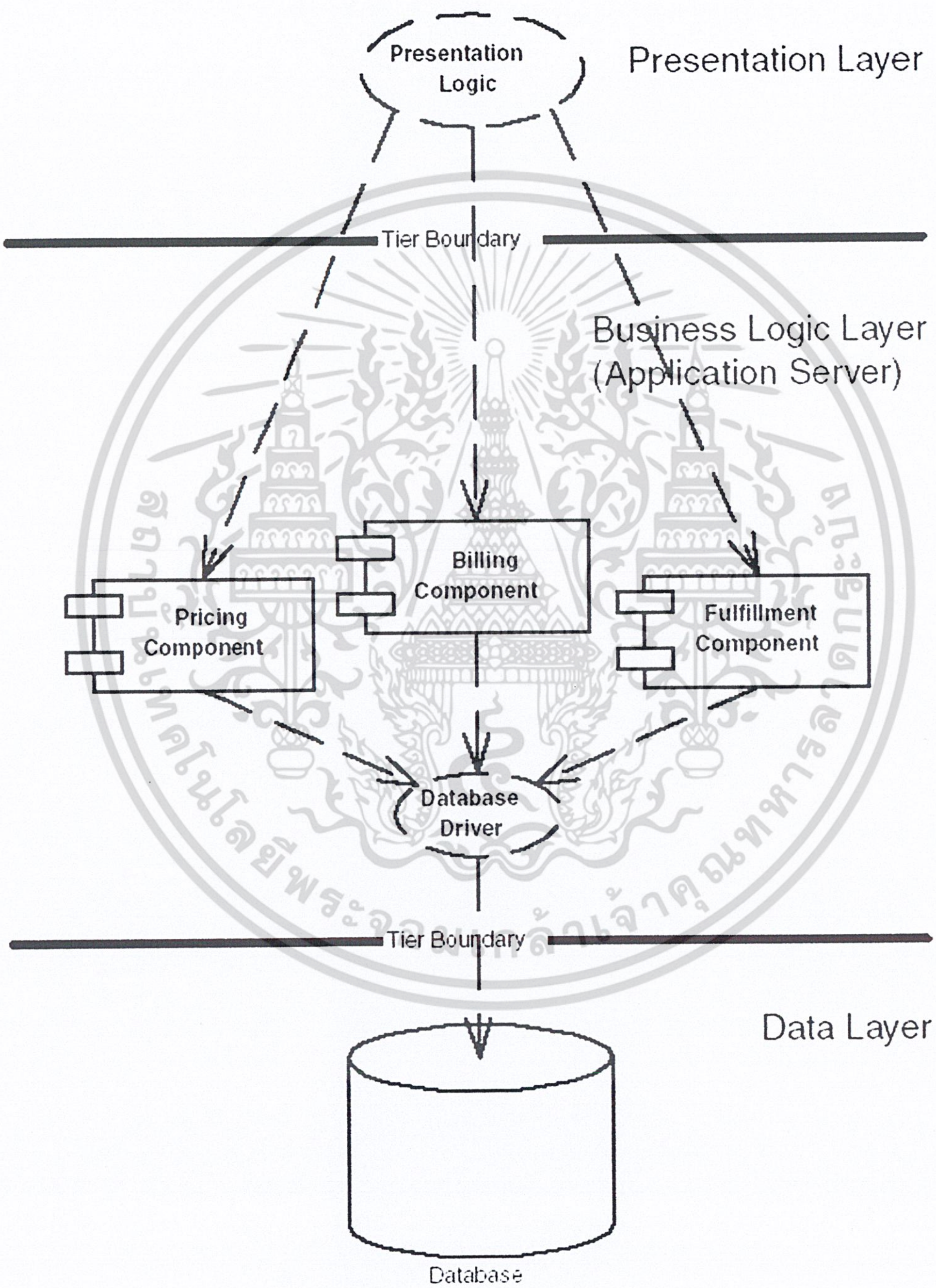



*รูปที่ 3-3 การรวมบิซิเนสลอจิกเลเยอร์บางส่วน เข้ากับดาต้าเลเยอร์*

**3.2.2 สถาปัตยกรรมแบบเอ็นเทียร์ (N-Tier Architecture)**

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

ตัวอย่างที่ดีของสถาปัตยกรรมแบบเอ็นเทียร์ได้แก่ web-base ดีพลอยเมนต์แบบ 3 เทียร์ ดังแสดงใน รูปที่ 3-4 ซึ่งเทียร์ทั้งสามนั้น ถูกแบ่งออกดังนี้




*รูปที่ 3-4 Web-base ดีพลอยเมนต์แบบ 3 เทียร์*

- พรีเซนเทชันเทียร์ (Presentation Tier)

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

- บิซิเนสลอจิกเทียร์ (Business Logic Tier)

อาจเป็นแอพพลิเคชันเซิร์ฟเวอร์เครื่องเดียวหรือมากกว่า แอพพลิเคชันเซิร์ฟเวอร์เป็นสิ่งจำเป็นในการให้สภาพแวดล้อมในการทำงานที่เหมาะสมกับคอมโพเน็นต์ของบิซิเนสลอจิก แอพพลิเคชันเซิร์ฟเวอร์จะจัดหาเอ็นไวรอนเมนต์ให้กับบิซิเนสลอจิกคอมโพเน็นต์ ทั้งยังทำหน้าที่จัดการคอมโพเน็นต์ และให้บริการที่จำเป็นแก่คอมโพเน็นต์เหล่านี้ ยกตัวอย่างเช่น แอพพลิเคชันเซิร์ฟเวอร์สามารถให้บริการติดต่อฐานข้อมูลกับบิซิเนสคอมโพเน็นต์ ซึ่งทำให้บิซิเนสคอมโพเน็นต์สามารถเก็บข้อมูลแบบถาวร (persistent) ลงไป และดึงข้อมูลขึ้นมาจากดาต้าเทียร์ได้ แอพพลิเคชันเซิร์ฟเวอร์ยังมีหน้าที่ในการจัดการคอมโพเน็นต์โดยการสร้างอินสแทนซ์ของคอมโพเน็นต์เมื่อจำเป็น

- ดาต้าเทียร์ (Data Tier)

ประกอบไปด้วยฐานข้อมูลจำนวนหนึ่งฐานข้อมูลหรือมากกว่า และอาจมีลอจิกเกี่ยวกับข้อมูลซึ่งถูกเก็บไว้ในรูปของ store procedure

ลักษณะของสถาปัตยกรรมแบบเอ็นเทียร์

1) ค่าใช้จ่ายในการดีพลอยต่ำ

เนื่องจากไดรเวอร์ของระบบฐานข้อมูลนั้นถูกติดตั้งอยู่ที่ฝั่งเซิร์ฟเวอร์ แทนที่จะเป็นเครื่องของไคลเอ็นต์ ซึ่งเป็นการสะดวกที่จะทำการติดตั้งซอฟต์แวร์บนฝั่งเซิร์ฟเวอร์ในสภาพแวดล้อมที่ควบคุมได้ มากกว่าการติดตั้งบนเครื่องไคลเอ็นต์จำนวนมากมาย

2) ค่าใช้จ่ายในการเปลี่ยนแปลงระบบฐานข้อมูลต่ำ

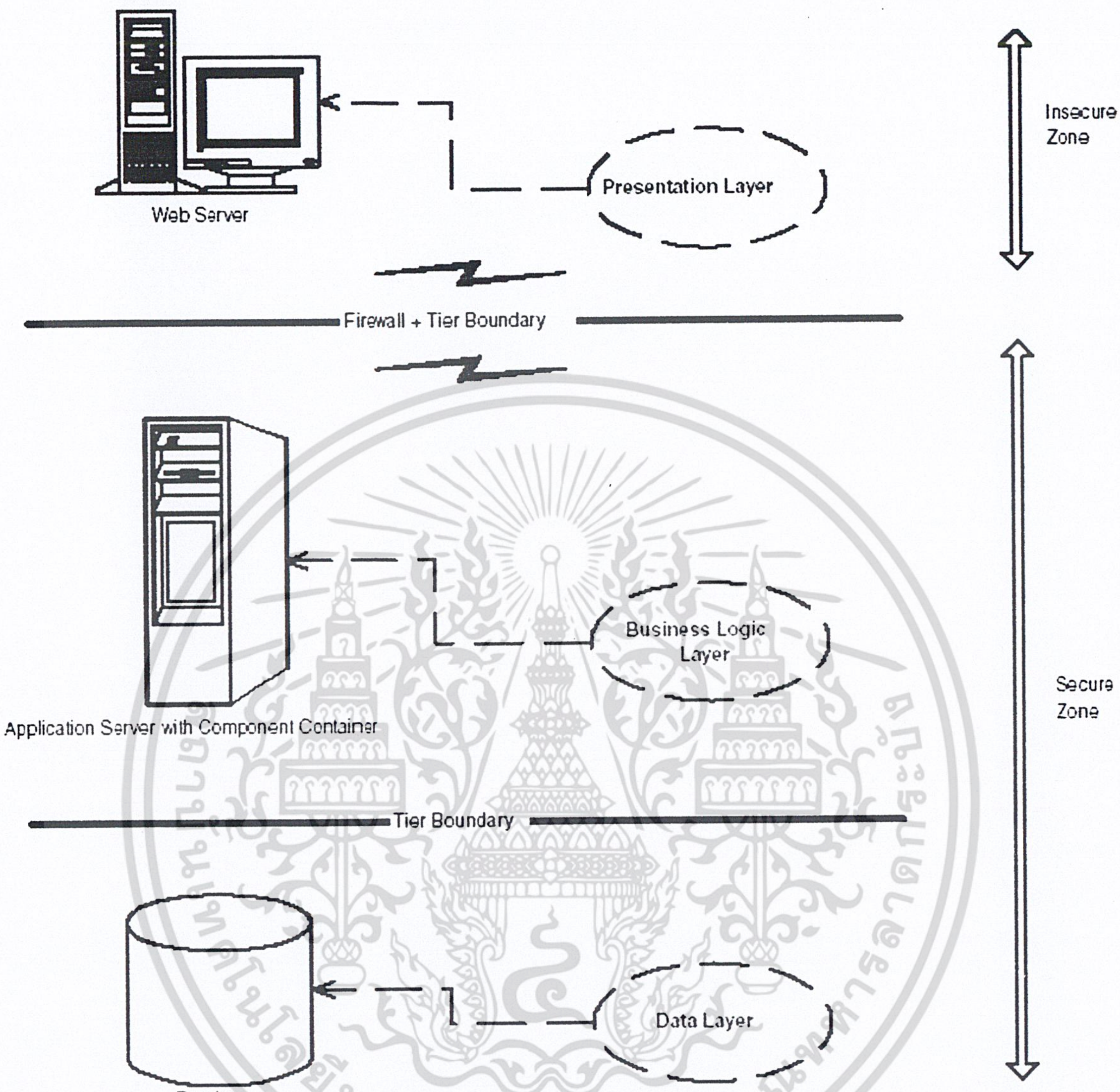
ไคลเอ็นต์จะทำการติดต่อกับฐานข้อมูลโดยผ่านทางเทียร์ซึ่งอยู่ตรงกลาง ทำให้เราสามารถเปลี่ยนแปลงโครงสร้างของฐานข้อมูล , ไดรเวอร์ของฐานข้อมูล แม้กระทั่งฐานข้อมูลโดยไม่จำเป็นต้องดีพลอยไคลเอ็นต์ทั้งหมดใหม่อีกครั้ง

3) ค่าใช้จ่ายในการเปลี่ยนแปลงบิซิเนสลอจิกต่ำ

เพราะการเปลี่ยนแปลงในบิซิเนสลอจิกเลเยอร์ไม่จำเป็นที่จะต้องมีการดีพลอยไคลเอ็นต์เทียร์ใหม่

4) สามารถสร้างเขตความปลอดภัยให้กับส่วนใดส่วนหนึ่งของดีพลอยเมนต์ได้ด้วยไฟล์วอลล์

ใน web-base ดีพลอยเมนต์ เราอาจไม่ต้องการให้บุคคลภายนอกเข้ามาเห็นข้อมูลในบิซิเนสลอจิกเลเยอร์ แต่อนุญาตให้บุคคลภายนอกเข้าถึงพรีเซนเทชันเลเยอร์ได้ ซึ่งสามารถแก้ปัญหานี้ได้โดยการวางไฟร์วอลล์ไว้ระหว่างส่วนพรีเซนเทชัน และบิซิเนสลอจิกเทียร์ ดังแสดงในรูปที่ 3-5

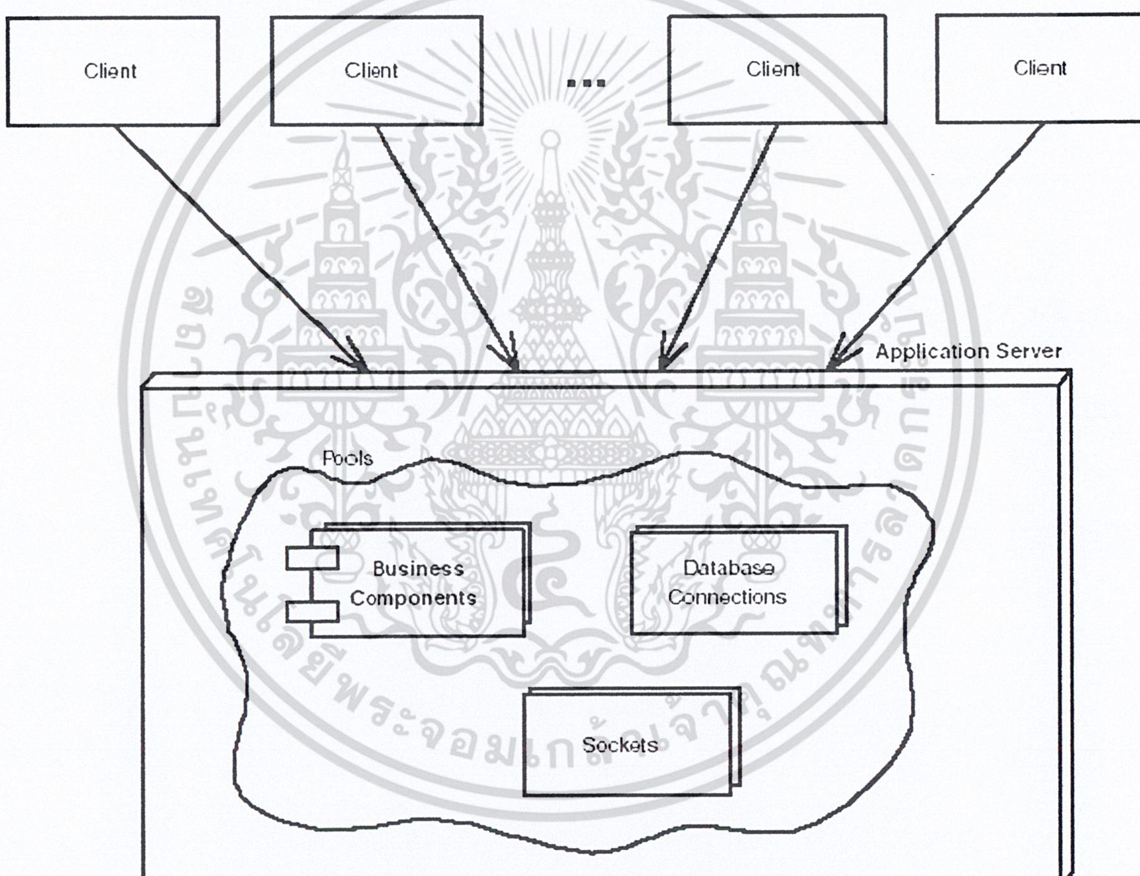



*รูปที่ 3-5 การรักษาความปลอดภัยในสถาปัตยกรรมแบบ 3 เทียร์*

5) ทรัพยากรสามารถถูกพูล (pool) และนำกลับมาใช้ใหม่ได้อย่างมีประสิทธิภาพ

ในสถาปัตยกรรมแบบเอ็นเทียร์ การเชื่อมต่อไปยังทรัพยากรภายนอกสามารถถูกจัดการได้อย่างมีประสิทธิภาพ การพูลทรัพยากร (resource pooling) นั้นใช้หลักความเป็นจริงที่ว่า ไคลเอ็นต์มักจะทำงานอื่นไปพร้อม ๆ กับการใช้ทรัพยากร เช่น การสร้างส่วนติดต่อกับผู้ใช้ แทนที่บิซิเนสคอมโพเน็นต์จะทำการเชื่อมต่อและปล่อยการติดต่อกับทรัพยากร (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 แล้วในสถาปัตยกรรมแบบเอ็นเทียร์นั้น บิซิเนสคอมโพเน็นต์เองสามารถที่จะพูลและให้บริการแก่ไคลเอ็นต์หลาย ๆ เครื่องได้ การพูลคอมโพเน็นต์นั้นหมายความว่าเราไม่จำเป็นที่จะต้องมีคอมโพเน็นต์ส่วนตัวสำหรับไคลเอ็นต์แต่ละเครื่อง การพูลทรัพยากรนั้นถูกแสดงไว้ในรูปที่ 3-6




***รูปที่ 3-6 การพูลทรัพยากร (Resource Pooling) ในสถาปัตยกรรมแบบเอ็นเทียร์***

6) ประสิทธิภาพถูกแบ่งเป็นส่วน ๆ

หากเทียร์ใดเทียร์หนึ่งเกิดการโอเวอร์โหลด (overload) ขึ้น เทียร์อื่น ๆ จะยังคงทำงานต่อไปได้อย่างปกติ เช่นในกรณีของเว็บดีพลอยเมนต์ ผู้ใช้สามารถที่จะเข้ามายังหน้าแรกของเว็บไซต์มาก ถึงแม้ว่าที่แอพพลิเคชันเซิร์ฟเวอร์จะเกิดการโอเวอร์โหลดก็ตาม

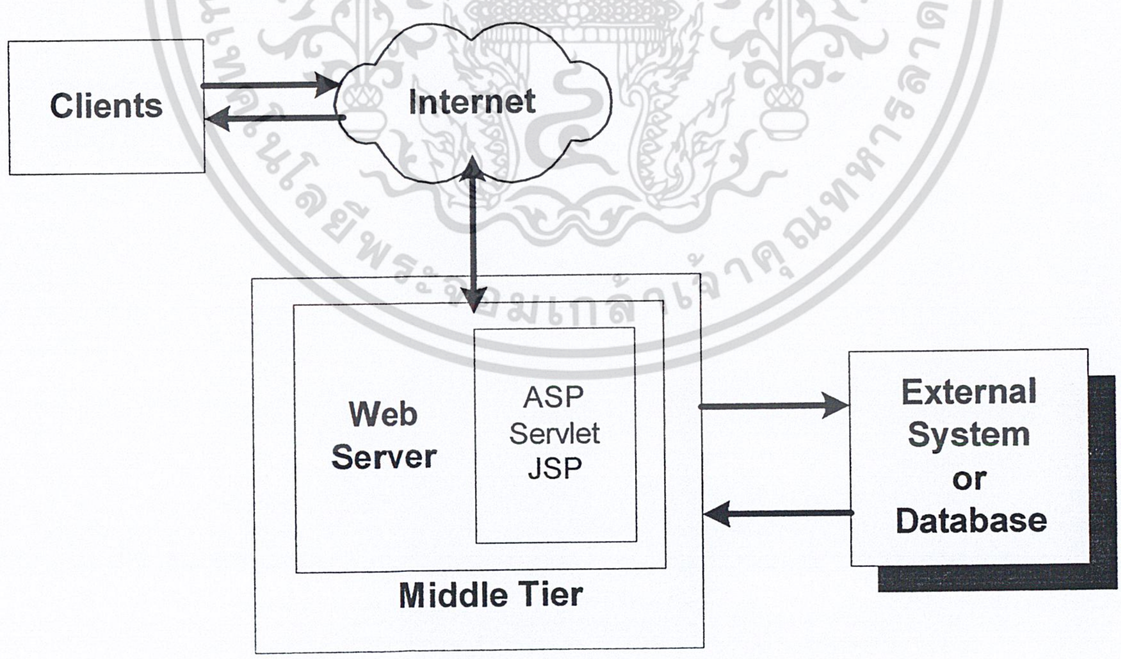
7) ความผิดพลาดในการทำงานถูกแบ่งเป็นส่วน ๆ

หากเกิดความผิดพลาดในการทำงานอย่างรุนแรงขึ้นที่เทียร์ใด ๆ ความผิดพลาดก็จะถูกจำกัดวงอยู่เพียงในเทียร์นั้นโดยไม่มีผลกระทบต่อเทียร์อื่น ๆ ยกตัวอย่างเช่น หากแอพพลิเคชันเซิร์ฟเวอร์เกิดความผิดพลาดในการทำงาน หรือแครช (crash) เว็บเซิร์ฟเวอร์สามารถทำการรายงานความผิดพลาดให้กับผู้ใช้ทราบได้

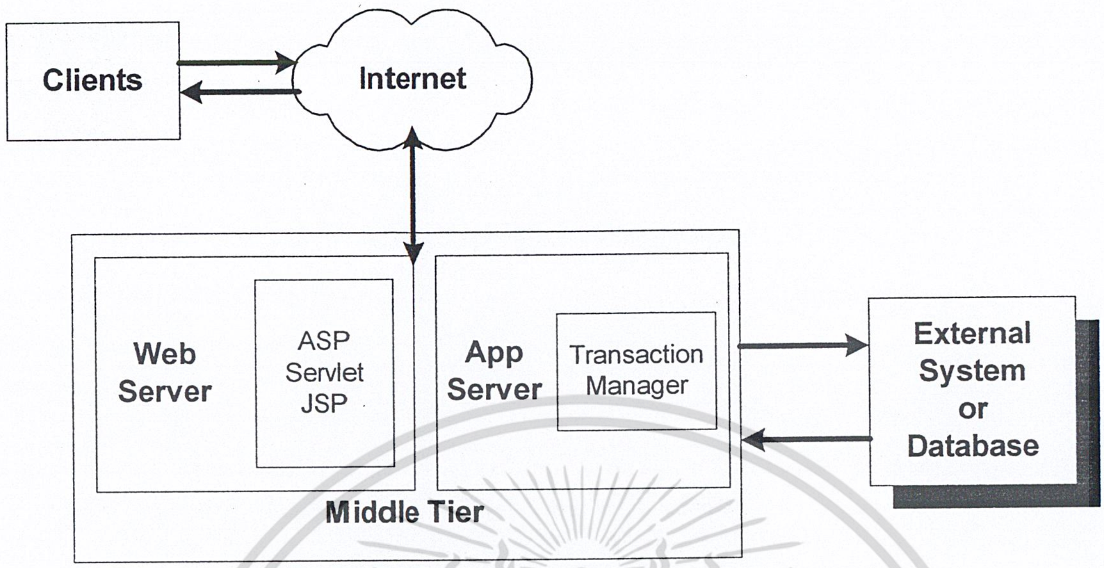
8) มีค่าใช้จ่ายในการบำรุงรักษาสูง

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*




*รูปที่ 3-8 ระบบไคลเอ็นต์/เซิร์ฟเวอร์ดีพลอยเมนต์แบบ 4 เทียร์*

**3.3 Java 2 Platform , Enterprise Edition (J2EE)**

ผลิตภัณฑ์สถาปัตยกรรมบนฝั่งเซิร์ฟเวอร์ (server-side component architecture solution) นั้น ได้มีออกมาจากผู้พัฒนาต่าง ๆ เช่น

- Microsoft's Distributed interNet Application Architecture (DNA)
- Sun Microsystem's Java 2 Platform, Enterprise Edition (J2EE)
- The Object Management Group's CORBA Standard

โดยในที่นี้จะกล่าวถึงสถาปัตยกรรมที่ใช้เทคโนโลยีของจาวาเท่านั้น ซึ่งก็คือ Java 2 Platform , Enterprise Edition (J2EE)

ภาษาจาวาเป็นภาษาที่สะดวกในการเขียนคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์ เนื่องจากเครื่องเซิร์ฟเวอร์ส่วนใหญ่เป็นเครื่องเมนเฟรม หรือเครื่องซึ่งใช้ระบบปฏิบัติการยูนิกซ์ ดังนั้นภาษาที่เราจะนำมาใช้สร้างคอมโพเน็นต์ซึ่งสามารถทำงานข้ามแพล็ตฟอร์มได้นั้นจะมีความสำคัญมาก เพราะผู้พัฒนาสามารถสร้างคอมโพเน็นต์ขึ้นมาเพียงครั้งเดียวแล้วนำไปดีพลอยในระบบเซิร์ฟเวอร์ของลูกค้า ซึ่งทำงานอยู่ในสภาพแวดล้อมที่แตกต่างกันออกไปได้

เพื่อสนับสนุนความสามารถในการทำงานบนฝั่งเซิร์ฟเวอร์ ซันจึงได้สร้างแพล็ตฟอร์มสำหรับการพัฒนาซึ่งมีชื่อว่า Java 2 Platform, Enterprise Edition (J2EE) ซึ่งมีความสามารถในการทำงานแบบไม่ขึ้นกับแพล็ตฟอร์ม , รองรับการทำงานของผู้ใช้หลายคน และมีความปลอดภัย แกนหลักของสถาปัตยกรรม J2EE คือ Enterprise JavaBeans (EJB) ซึ่งเป็นมาตรฐานในการสร้างคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์ด้วยภาษาจาวา

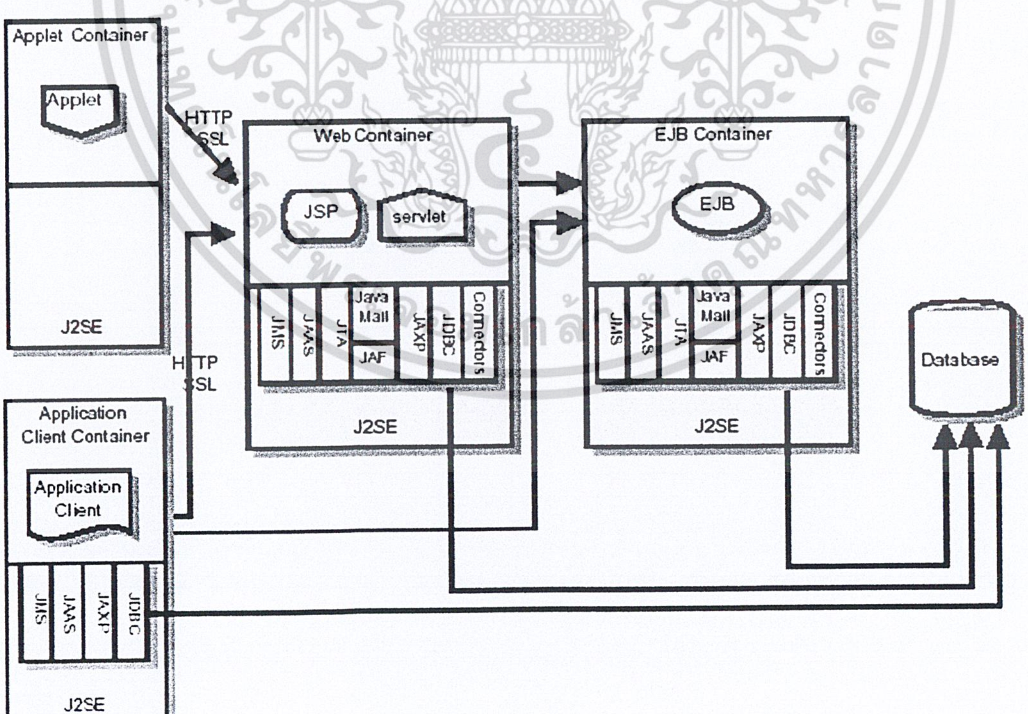
J2EE เป็น specification ไม่ใช่ผลิตภัณฑ์ J2EE จะกำหนดกฎในการเขียนเอ็นเตอร์ไพรส์ซอฟต์แวร์ ผู้ขายอื่น ๆ จะนำ J2EE specification ไปอิมพลีเมนต์เป็นผลิตภัณฑ์ที่รองรับ J2EE ในเอกสารนี้จะอ้างอิง J2EE specification 1.2 เป็นหลัก โดยจะกล่าวถึง J2EE specification 1.3 บางส่วน

แพล็ตฟอร์มของจาวามี 3 แพล็ตฟอร์ม โดยแต่ละแพล็ตฟอร์มจะเป็นซูเปอร์เซต (superset) ของแพล็ตฟอร์มที่เล็กกว่า

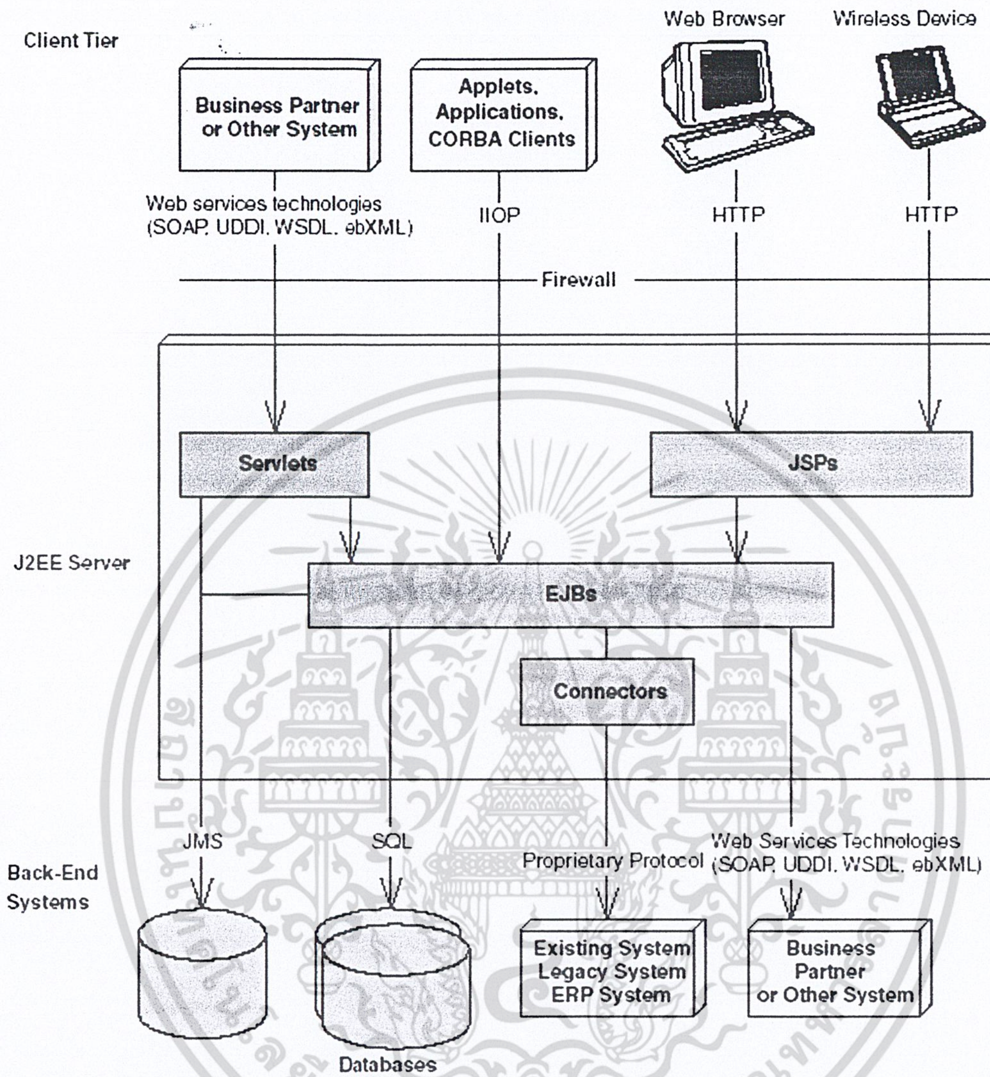
- **Java 2 Platform , Micro Edition (J2ME)** – เป็นแพล็ตฟอร์มพัฒนาสำหรับอุปกรณ์พกพา เช่น ปาล์ม , เพจเจอร์ เป็นต้น
- **Java 2 Platform , Standard Edition (J2SE)** – ประกอบด้วยบริการมาตรฐานของจาวาสำหรับ Applet และแอพพลิเคชัน จะรู้จักกันในชื่อ Java Development Kit (JDK)
- **Java 2 Platform , Enterprise Editon (J2EE)** – สร้างบน J2SE โดยเพิ่ม Java Enterprise API เป็นแพล็ตฟอร์มสำหรับพัฒนา enterprise-class ที่ต้องดีพลอยบนเซิร์ฟเวอร์

**3.3.1 เทคโนโลยีต่าง ๆ ในสถาปัตยกรรม J2EE**

Java 2 Platform Enterprise Edition เป็นชุดของบริการมิดเดิลแวร์ซึ่งช่วยอำนวยความสะดวกในการพัฒนาแอพพลิเคชันบนฝั่งเซิร์ฟเวอร์ เทคโนโลยีซึ่งเป็นส่วนประกอบของ J2EE ถูกแสดงไว้ดังรูปที่ 3-9 และ 3-10 ซึ่งประกอบด้วยเทคโนโลยีต่าง ๆ ดังนี้




*รูปที่ 3-9 ไดอะแกรมของ Java 2 Platform, Enterprise Edition*




*รูปที่ 3-10 ดีพลอยเมนต์ของ Java 2 Platform, Enterprise Edition*

- **Enterprise JavaBean (EJB)**

EJB เป็น specification ที่ระบุวิธีการเขียนคอมโพเน็นต์ และมาตรฐานในการติดต่อระหว่างคอมโพเน็นต์กับตัวคอนเทนเนอร์ซึ่งจัดการมันอยู่ (ในเอกสารนี้จะอ้างถึง EJB specification 1.1 เป็นหลักโดยมีรายละเอียดของ EJB specification 2.0 ในบางส่วน) EJB สนับสนุนให้เกิดตลาดคอมโพเน็นต์ ซึ่งเหล่าผู้พัฒนาสามารถขายคอมโพเน็นต์เพื่อให้บริษัทอื่น ๆ นำคอมโพเน็นต์นี้ไปประยุกต์ใช้กับธุรกิจของตนได้ enterprise bean จะต้องรันอยู่บน J2EE เซิร์ฟเวอร์ซึ่งเป็นมิดเดิลเซิร์ฟเวอร์ในระบบ 3 เทียร์ไคลเอ็นต์/เซิร์ฟเวอร์ (three-tier client/server system) EJB ถือว่าเป็นหัวใจของสถาปัตยกรรม J2EE เลยทีเดียว และเป็นจุดสนใจหลักของโครงงานชิ้นนี้

- **Java Remote Method Invocation (RMI) และ RMI-IIOP**

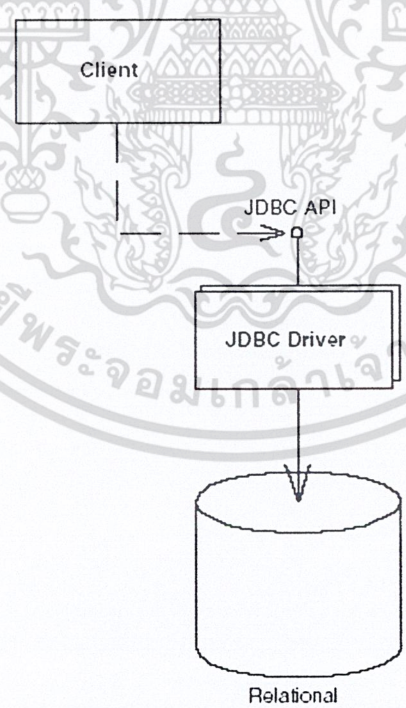
RMI เป็นข้อกำหนดในการติดต่อสื่อสารระหว่างออบเจ็กต์แบบกระจาย (distributed object) เช่น 2 ออบเจ็กต์ที่ต่างกันที่ทำงานอยู่คนละเครื่อง ส่วน RMI-IIOP เป็นส่วนเพิ่มขยายของ RMI ที่ใช้โพรโตคอล Internet InterORB Protocol (IIOP) และใช้ในการอินทิเกรตเข้ากับ CORBA เป็น API ที่ใช้ใน J2EE ไม่ใช่ใน RMI

- **Java Naming and Directory Interface (JNDI)**

ทำหน้าที่ระบุตำแหน่งของคอมโพเน็นต์ และทรัพยากรต่าง ๆ ในระบบเครือข่าย Enterprise JavaBeans จะใช้ JNDI ในการหา (look up) คอมโพเน็นต์แบบกระจายที่อยู่บนเครือข่าย JNDI เป็นคีย์เทคโนโลยีที่ทำให้ไคลเอ็นต์ติดต่อกับ EJB คอมโพเน็นต์ได้

- **Java Database Connectivity (JDBC)**

เป็นมาตรฐานของจาวาที่ใช้การเข้าถึงข้อมูล โปรแกรมเมอร์จาวาจะเขียนโค้ดติดต่อกับ API ของ JDBC ซึ่งเป็นมาตรฐานการติดต่อกับฐานข้อมูลเชิงสัมพันธ์ ไคลเอ็นต์จะติดต่อกับ JDBC API ซึ่งจะถูกอิมพลีเมนต์โดย JDBC ไดรเวอร์ JDBC ไดรเวอร์จะทำหน้าที่แปลงข้อมูลให้สามารถทำงานกับฐานข้อมูลได้ ดังรูปที่ 3-11




*รูปที่ 3-11 Java Database Connectivity*

- **Java Transaction API (JTA) และ Java Transaction Service (JTS)**

เป็น API ที่ใช้จัดการกับทรานแซ็กชัน JTA เป็นอินเทอร์เฟซของทรานแซ็กชันในระดับบนที่แอพพลิเคชันใช้ในการจัดการทรานแซ็กชัน JTS เป็นอินเทอร์เฟซของทรานแซ็กชันในระดับต่ำที่ EJB ใช้โดยจะถูกจัดการให้โดย EJB คอนเทนเนอร์ JTS เป็นมาตรฐานที่ช่วยให้สามารถทำทรานแซ็กชันแบบกระจายได้ภายใต้เซิร์ฟเวอร์ของผู้ผลิตต่าง ๆ กัน

- **Java Messaging Service (JMS)**

JMS ช่วยให้ออบเจ็กต์แบบกระจายสามารถสื่อสารกันแบบอะซิงโครนัส อยู่ใน EJB specification 2.0 ซึ่งจะไม่ใช้ในโครงงานนี้

- **Java Servlet และ Java Server Page (JSP)**

Servlet และ JSP เป็นคอมโพเน็นต์ในระบบเครือข่ายซึ่งเหมาะสมกับการทำงานในรูปแบบของการร้องขอและการตอบรับ (request/response) เช่น การติดต่อกับไคลเอ็นต์ผ่านทางโพรโตคอล HTTP Servlet จะถูกเขียนเป็นจาวาคลาสที่มีความสามารถในการจัดการ request และสามารถส่ง response กลับไปให้ไคลเอ็นต์ได้ แต่ JSP เป็นสคริปต์ที่ทำงานร่วมกับ HTML ทำให้การออกแบบหน้าเว็บทำได้ง่าย โดย JSP จะถูกคอมไพล์เมื่อเรียกใช้ครั้งแรกและจะกลายเป็น Servlet คลาส

- **Java IDL**

Java IDL เป็นการอิมพลีเมนต์เทคโนโลยี CORBA ด้วยภาษาจาวา Java IDL จะช่วยในการอินทิเกรตระบบเข้ากับภาษาอื่น ๆ ที่ไม่ใช่ภาษาจาวา นอกจากนี้ Java IDL ยังทำให้ออบเจ็กต์แบบกระจายใช้บริการของ CORBA ได้อย่างครบถ้วน

- **JavaMail**

บริการจาวาเมล์ ช่วยให้เราสามารถส่งข้อความอีเมล์โดยไม่ขึ้นกับแพลตฟอร์มและโพรโตคอลจากโปรแกรมที่สร้างขึ้นด้วยภาษาจาวา โดยจาวาเมล์จะขึ้นกับ JavaBeans Activation Framework (JAF)

- **J2EE Connector Architecture (JCA)**

Connector ทำให้ J2EE ดีพลอยเมนต์สามารถที่จะแอ็กเซสเอ็นเตอร์ไพรส์อินฟอร์เมชันซิสเต็ม (enterprise information system) รวมถึงระบบที่มีอยู่แล้วเช่นระบบที่เป็นเมนเฟรมซึ่งทำงานเกี่ยวกับทรานแซ็กชันแบบไฮเอนด์ โดยระบบอาจถูกดีพลอยด้วย CISC ของ IBM หรือ TUXEDO ของ BEA รวมถึงระบบ Enterprise Resource Planning (ERP)

- **Java API for XML Parsing (JAXP)**

JAXP เป็น API ที่ใช้ parse เอกสาร XML ซึ่งสามารถใช้ได้ใน Servlet , JSP หรือ EJB คอมโพเน็นต์

- **Java Authentication and Authorization Service (JAAS)**

JAAS เป็น API มาตรฐานที่ใช้จัดการกับความปลอดภัยใน J2EE

# บทที่ 4

# Enterprise JavaBeans

## 4.1 ส่วนประกอบของ EJB

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

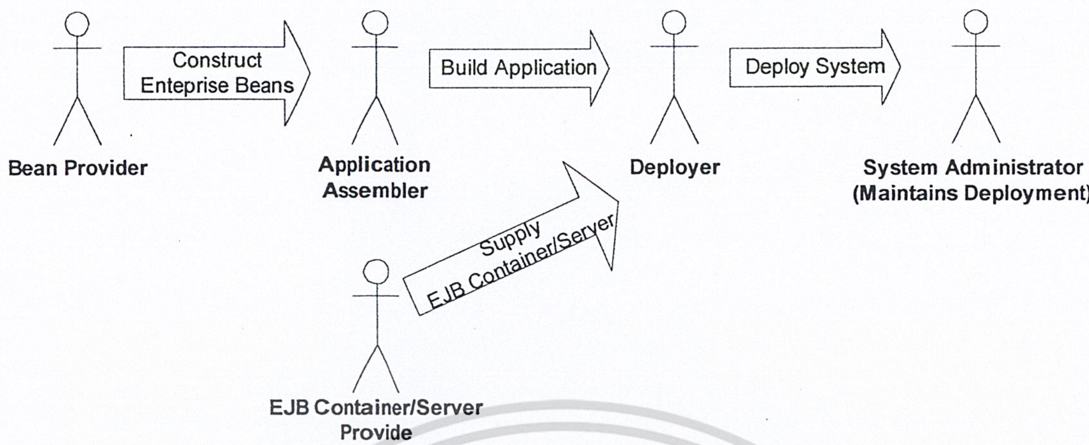
หน้าที่ที่เกี่ยวข้องกับการพัฒนาระบบด้วย EJB มีดังนี้

1) **Bean Provider** - เป็นผู้สร้างคอมโพเน็นต์ทางธุรกิจที่สามารถนำกลับมาใช้ใหม่ได้ (Reusable) ซึ่งธุรกิจอื่น ๆ สามารถหาซื้อคอมโพเน็นต์ไปใช้ในธุรกิจของตนเองได้ bean นั้นไม่ใช่แอพพลิเคชันที่สมบูรณ์ในตัวเอง แต่เป็นคอมโพเน็นต์ซึ่งสามารถนำไปดีพลอย และประกอบกันขึ้นมาเป็นแอพพลิเคชันที่สมบูรณ์ได้
2) **Container Provider** – เป็นผู้พัฒนาคอนเทนเนอร์ที่จะทำหน้าที่ในการจัดการกับเอ็นไวรอนเมนต์ในการทำงานให้กับ EJB
3) **Server Provider** – เป็นผู้พัฒนาแอพพลิเคชันเซิร์ฟเวอร์ แอพพลิเคชันเซิร์ฟเวอร์คือเซิร์ฟเวอร์ในมิดเดิลเทียร์ ทำหน้าที่ในการจัดการและดีพลอยคอมโพเน็นต์ต่าง ๆ โดยในขณะนี้ยังไม่มีข้อแตกต่างที่เห็นได้ชัดเจนระหว่าง Container Provider และ Server Provider
4) **Application Assembler** - เป็นผู้มีหน้าที่รับผิดชอบในการนำคอมโพเน็นต์ต่าง ๆ มาประกอบกันและเขียนแอพพลิเคชันที่ใช้คอมโพเน็นต์เหล่านั้น โดยที่อาจจะต้องเขียนคอมโพเน็นต์บางตัวขึ้นมาเอง Application Assembler อาจเป็นบริษัทภายนอกที่รับทำงานนี้โดยเฉพาะ หรืออาจจะเป็นโปรแกรมเมอร์ภายในบริษัทเองก็ได้
5) **Deployer** – เป็นผู้นำเอาคอมโพเน็นต์ต่าง ๆ ที่ได้สร้างไว้แล้วมาทำการติดตั้งลงในแอพพลิเคชันเซิร์ฟเวอร์เพื่อใช้งาน
6) **System Administrator** - มีหน้าที่ดูแลระบบที่ถูกดีพลอยเรียบร้อยแล้วให้ทำงานเป็นปกติ

ความสัมพันธ์ในการทำงานร่วมกันของแต่ละส่วน ถูกแสดงในรูปที่ 4-1

## 4.2 EJB Server และ EJB Container

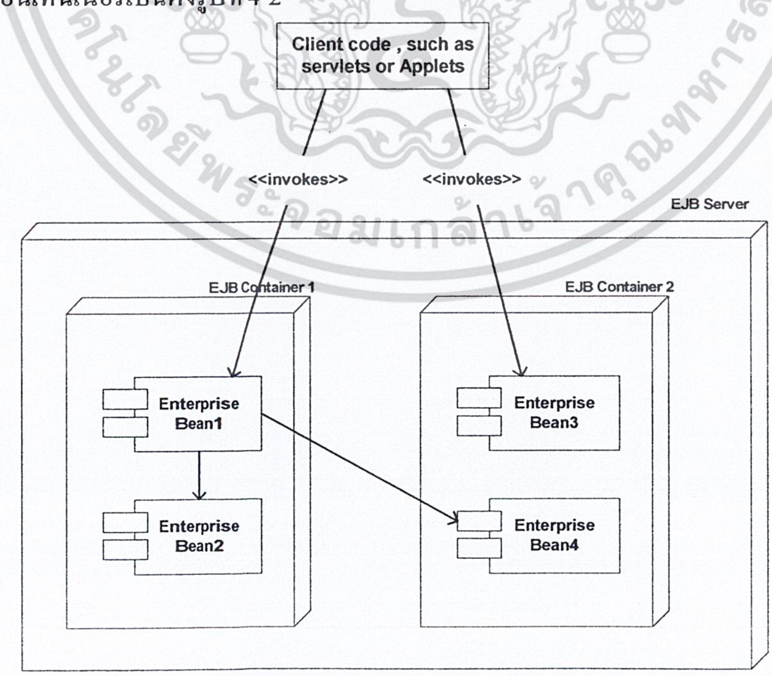
แอพพลิเคชันเซิร์ฟเวอร์จะจัดหาบริการมิดเดิลแวร์ (middleware service) ให้กับแอพพลิเคชัน เช่น ทรานแซ็กชัน , ความปลอดภัย และอื่น ๆ บริการเหล่านี้จำเป็นสำหรับแอพพลิเคชันที่ต้องการความสามารถ




*รูปที่ 4-1 ความสัมพันธ์ของแต่ละหน้าที่ในการพัฒนา EJB*

ในการเพิ่มขยาย , มีประสิทธิภาพ และมีความปลอดภัย ในการให้บริการกับผู้ใช้จำนวนมากพร้อม ๆ กัน แอพพลิเคชันเซิร์ฟเวอร์จะแบ่งเป็น 2 ส่วนคือ

- **EJB Container** – เป็นที่อยู่ที่ enterprise bean สามารถทำงานได้ ภายในคอนเทนเนอร์หนึ่งสามารถมี bean จำนวนมากทำงานอยู่ คอนเทนเนอร์มีหน้าที่ในการควบคุม bean ที่รันอยู่ภายในตัวมัน
- **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** | v5.0 | 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 v5.1 | 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<br>1.3 | 1.1 | 1.1 | 1.0.2 | ✔ | ✔ |
| | Total-e-Mobile v1.1 | 1.2.2<br>1.3 | 1.1 | 1.1 | 1.0.2 | | ✔ |
| **IBM Websphere** | Standard v3.5.3 | 1.2.2 | | 1.1 | 1.0.2 | ✔ | |
| | Advanced v4.0 | 1.3 | 1.1 | 1.1 | 1.0.2<br>+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<br>+XA | | ✔ |
| **In-Q-My Application Server** | v4.1 | 1.2.2<br>1.3 | 1.1 | 1.1 | 1.0.2 | ✔ | ✔ |
| **IONA iPortal** | Standard v3.0 | 1.3<br>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<br>1.3 | 1.1<br>2.0** | 1.1 | 1.0.2 | | |

*ตารางที่ 4-1 แอพพลิเคชันเซิร์ฟเวอร์ (ต่อ)*

| Vendor<br>Product | Edition<br>Version | JDK | EJB | JSP | JMS | J2EE<br>License | J2EE<br>Certif. |
|---|---|---|---|---|---|---|---|
| **Lutris** | Lutris EAS | 1.3 | 1.1 | 1.1 | 1.0.2 | | |
| | Enhydra v3.1 | 1.2.2<br>1.3 | | 1.1 | | | |
| | Lutris Enhydra v3.5.2 | 1.2.2<br>1.3 | | 1.1 | | | |
| **Oracle**<br>**9i AS** | Standard v1.0.2 | 1.1.8<br>1.2.2 | 1.1 | 1.1 | | | |
| | Enterprise v1.0.2 | 1.1.8<br>1.2.2 | 1.1 | 1.1 | | ✓ | |
| | Wireless v1.0.2 | 1.1.8<br>1.2.2 | 1.1 | 1.1 | | | |
| **Persistence**<br>**Power Tier for J2EE** | v7.0 | 1.2<br>1.3<br>1.3.1 | 1.1 | 1.1 | 1.0.2 | ✓ | ✓ |
| **Pramati Technologies**<br>**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**<br>**Application Server** | Developer v3.7.3 | 1.2<br>1.3 | 1.1 | 1.1 | 1.0.2 | ✓ | ✓ |
| | Enterprise v3.7.3 | 1.2<br>1.3 | 1.1 | 1.1 | 1.0.2 | | ✓ |
| **Sybase**<br>**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 Enterprise Bean**

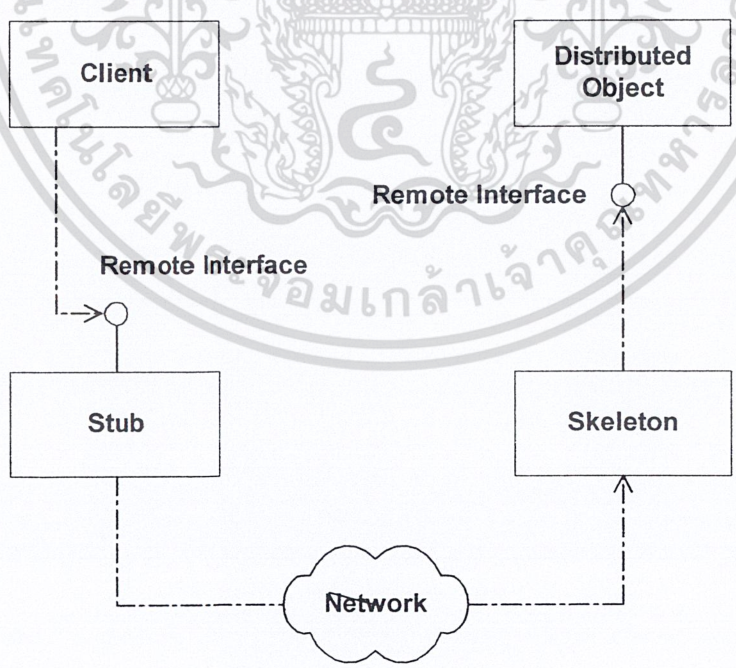
enterprise bean เป็นคอมโพเน็นต์ที่ทำงานบนฝั่งเซิร์ฟเวอร์ (Server-side software component) ซึ่งสามารถนำไปดีพลอยในเอ็นไวรอนเมนต์มัลติเทียร์แบบกระจาย (Distributed multi-tier environment) ได้ ภายในหนึ่ง enterprise bean สามารถประกอบไปด้วยจาวาออบเจ็กต์มากกว่าหนึ่งชิ้นได้ เนื่องจากคอมโพเน็นต์นั้นไม่จำเป็นจะต้องเป็นออบเจ็กต์ชิ้นเดียว

ไม่ว่า enterprise bean นั้นจะมีองค์ประกอบภายในอย่างไรก็ตาม ไคลเอ็นต์ที่มาเรียกใช้ enterprise bean นั้นต้องติดต่อผ่านทางอินเทอร์เฟซของ enterprise bean เท่านั้น อินเทอร์เฟซเหล่านี้รวมทั้ง enterprise bean จะต้องเป็นไปตามข้อกำหนดของ Enterprise JavaBeans ข้อกำหนดเหล่านี้จะระบุถึงเมธอดที่เป็นเมธอดบังคับจำนวนหนึ่ง ซึ่งเมธอดเหล่านี้จะทำให้ EJB คอนเทนเนอร์ควบคุม enterprise bean ได้ในแนวทางเดียวกัน โดยไม่ขึ้นอยู่กับคอนเทนเนอร์ที่ enterprise bean ทำงานอยู่

ไคลเอ็นต์ของ enterprise bean นั้นจะเป็นอะไรก็ได้ เช่น Servlet , Applet หรือ enterprise bean อื่น โดยในกรณีหลัง ไคลเอ็นต์ที่ทำการร้องขอไปยัง bean จะทำให้เกิดการร้องขอไปยัง bean ตัวอื่น ๆ ต่อไปเรื่อย ๆ ซึ่งเป็นแนวคิดที่ดี เนื่องจากเราสามารถแบ่ง bean ที่ทำงานที่ซับซ้อนเป็นส่วนย่อย ๆ ได้

**4.3.1 ออบเจ็กต์แบบกระจาย**

EJB คอมโพเน็นต์มีพื้นฐานอยู่บน RMI ซึ่งเป็นเทคโนโลยีออบเจ็กต์แบบกระจาย รูปที่ 4-3 จะแสดงการเรียกออบเจ็กต์แบบกระจายของไคลเอ็นต์




*รูปที่ 4-3 ออบเจ็กต์แบบกระจาย*

1) ไคลเอ็นต์เรียก stub ซึ่งเป็น proxy ออบเจ็กต์ทางฝั่งไคลเอ็นต์ (client-side proxy object) stub จะซ่อนการติดต่อผ่านเน็ตเวิร์ก (network communication) จากไคลเอ็นต์ stub จะรู้ว่าจะเรียกผ่านเน็ตเวิร์ก (call over network) อย่างไรผ่านทางซ็อกเก็ต
2) stub จะเรียกผ่านเน็ตเวิร์กไปยัง skeleton ซึ่งเป็น proxy ออบเจ็กต์ทางฝั่งเซิร์ฟเวอร์ (server-side proxy object) skeleton จะซ่อนการติดต่อผ่านเน็ตเวิร์กจากออบเจ็กต์แบบกระจาย skeleton จะรู้ว่าจะรับการเรียกอย่างไรผ่านทางซ็อกเก็ต
3) 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**

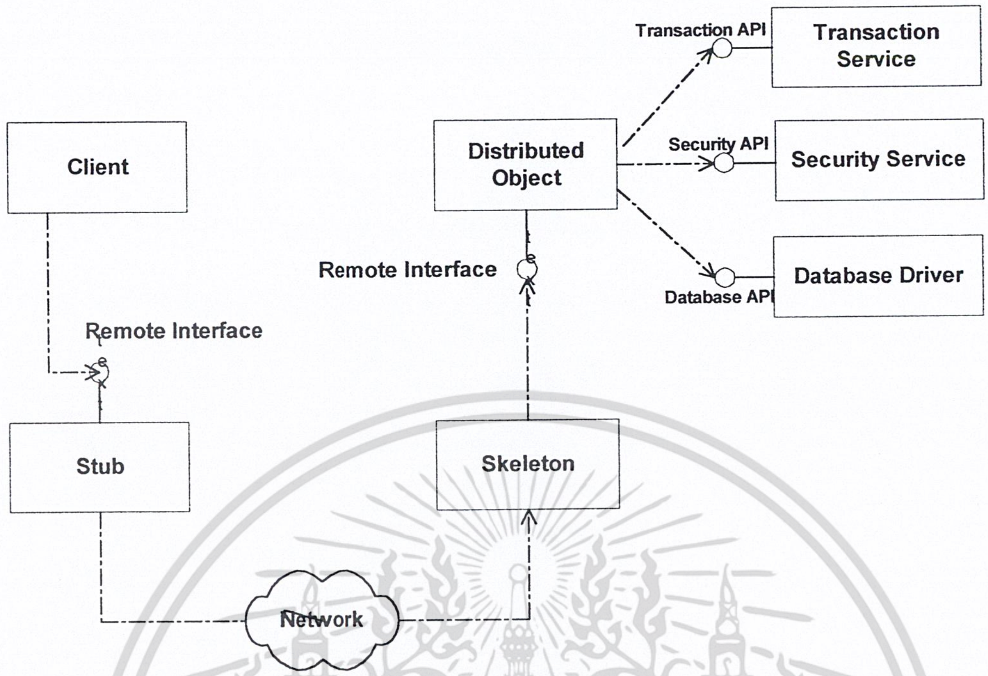
ออบเจ็กต์แบบกระจายนั้นมีประโยชน์มาก ทำให้สามารถเรียกใช้คอมโพเน็นต์บนเน็ตเวิร์กได้ แต่มันก็ยังต้องการบริการจากมิดเดิลแวร์ เช่น ทรานแซ็กชันและความปลอดภัย

- **Explicit Middleware**

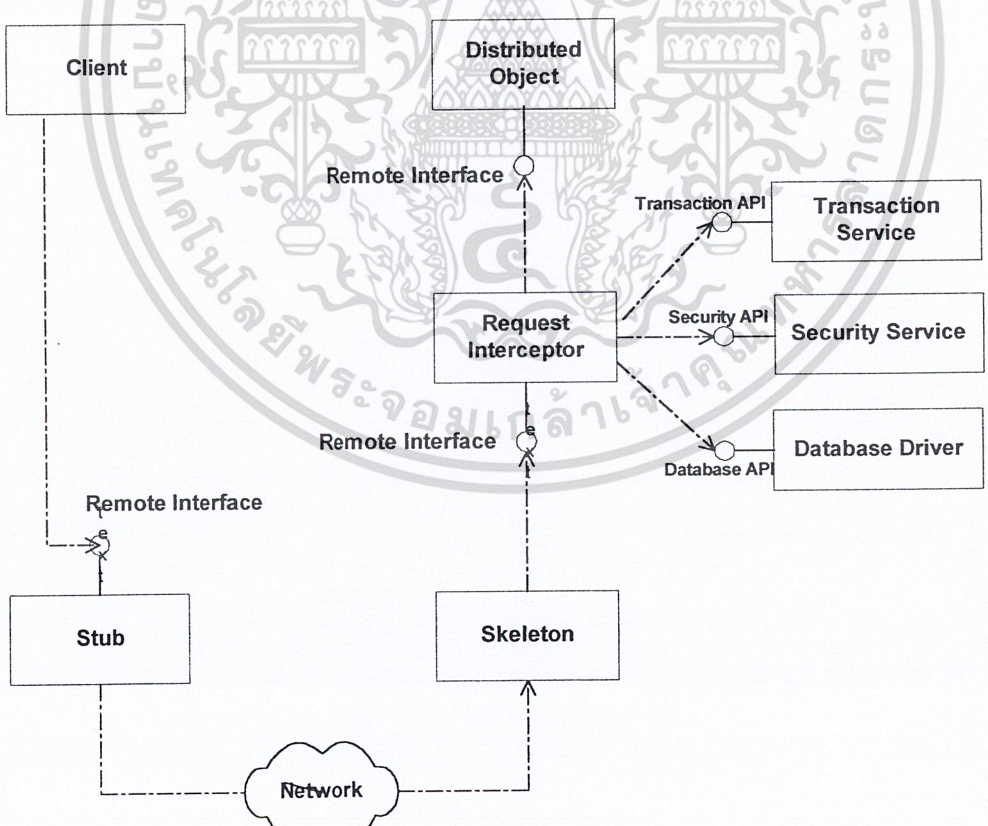
โดยปกติออบเจ็กต์แบบกระจายจะติดต่อกับมิดเดิลแวร์โดยเขียนโปรแกรมติดต่อกับ API ของมิดเดิลแวร์ เช่นหากต้องการการจัดการทางด้านทรานแซ็กชันจะต้องเขียนโค้ดติดต่อกับ transaction API ดังรูปที่ 4-4 มิดเดิลแวร์ประเภทนี้มีข้อเสียคือต้องเขียนโปรแกรมติดต่อกับ API ของมิดเดิลแวร์และหากต้องการเปลี่ยนมิดเดิลแวร์ที่ใช้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

- **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

- **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**

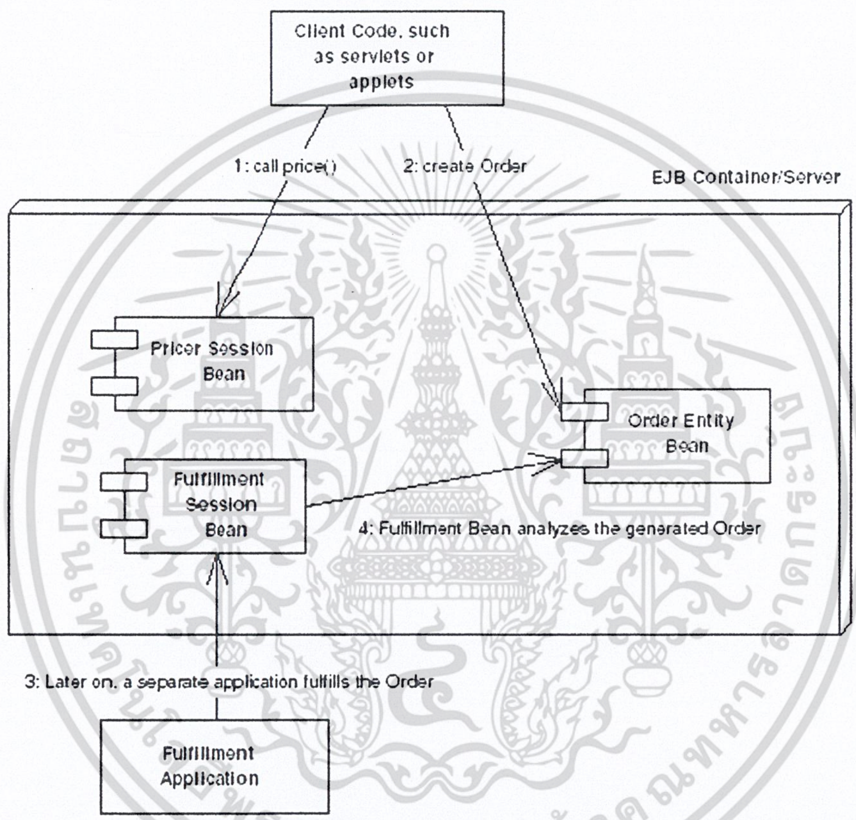
ส่วนประกอบพื้นฐานของธุรกิจอีกส่วนหนึ่งก็คือ ข้อมูลถาวร (Permanent Data) ซึ่งกระบวนการทางธุรกิจจะต้องนำไปใช้ ดังตัวอย่างเช่น

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

ในแต่ละสถานการณ์ที่กล่าวมานี้ คอมโพเน็นต์ของกระบวนการทางธุรกิจ ต้องเกี่ยวข้องกับข้อมูลซึ่งถูกเก็บอยู่ในแหล่งเก็บข้อมูลต่าง ๆ เช่นฐานข้อมูลเชิงสัมพันธ์ (Relational Database) เป็นต้น entity bean คือ

คอมโพเน็นต์ซึ่งเป็นตัวแทนของข้อมูลเหล่านี้ 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

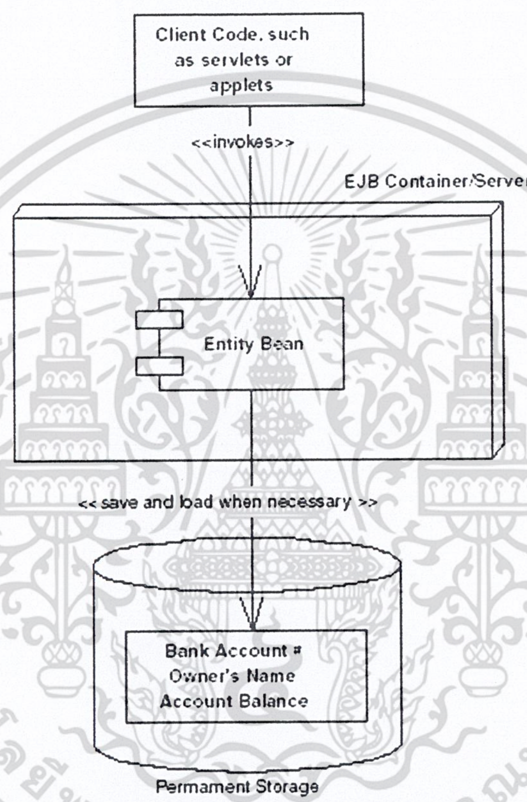



***รูปที่ 4-6 การนำ Session Bean และ Entity Bean มาใช้ร่วมกันในแอพลิเคชันการคิดราคาของสินค้าที่สั่งซื้อ***

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 เนื่องจากว่าโครงสร้างของฐานข้อมูลนั้นไม่ขึ้นกับ

ภาษาใด ๆ (Language-Independent) เรคอร์ดในฐานข้อมูลสามารถที่จะแปลงเป็นออบเจ็กต์ได้ในเกือบจะทุกภาษา ซึ่ง EJB ได้ใช้ประโยชน์ตรงจุดนี้ โดยอนุญาตให้มีการแปลงข้อมูลในฐานข้อมูลให้เป็นออบเจ็กต์ในภาษาจาวา กระบวนการแปลงข้อมูลนี้ เราอาจจะจัดการด้วย bean หรือให้คอนเทนเนอร์จัดการให้โดยอัตโนมัติก็ได้

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

- **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 ซึ่งจะบันทึกออบเจ็กต์จริง ๆ 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

- **Container-managed Persistent Entity Bean**

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

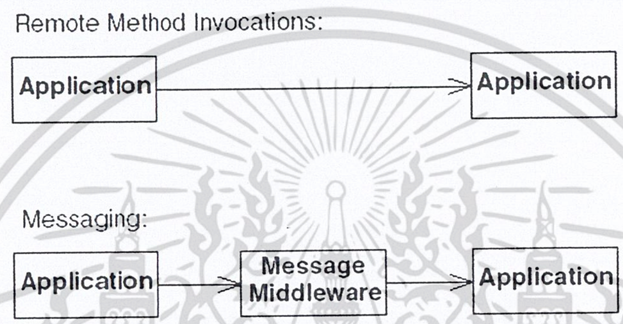
คอนเทนเนอร์จะทำฟังก์ชันในการเข้าถึงข้อมูลโดยอัตโนมัติ รวมถึงการเก็บข้อมูล การอ่านข้อมูล และการค้นหาข้อมูลของคอมโพเน็นต์ โดยที่ผู้พัฒนาไม่ต้องเขียนโค้ดในการติดต่อกับฐานข้อมูลด้วยตัวเองซึ่งจะช่วยประหยัดเวลาในขั้นตอนการพัฒนาลงไปได้มาก สิ่งที่ผู้พัฒนาต้องทำก็คือการระบุว่าต้องการให้ฟิลด์ใดบ้างที่ต้อง persistent หลังจากนั้น คอนเทนเนอร์ก็จะดูแลการ persistent ข้อมูลให้ตามที่กำหนด

ด้วยวิธีการดังกล่าว จะทำให้เราได้โค้ดที่ไม่ขึ้นอยู่กับระบบฐานข้อมูล ทำให้เราสามารถเปลี่ยนแปลงแหล่งเก็บข้อมูลได้อย่างอิสระ เนื่องจากใน Container-managed ไม่มีการเขียนโค้ดติดต่อกับ API ของระบบฐานข้อมูล

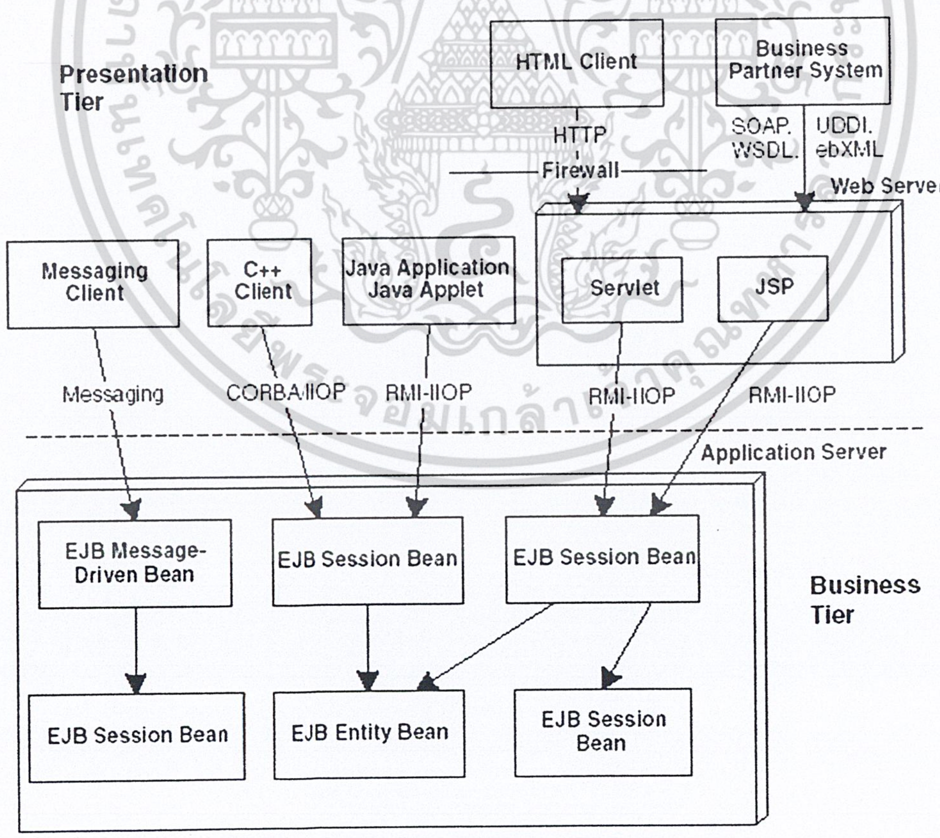
**4.4.3 Message-driven bean**

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*

## 4.5 การสร้าง Enterprise Bean

**1) Enterprise Bean Class**

ตามข้อกำหนดของ Enterprise JavaBeans การที่ bean จะทำงานได้ในทุก ๆ คอนเทนเนอร์ และทำงานได้กับทุก ๆ ไคลเอ็นต์ ซึ่งมาจากผู้ผลิตต่าง ๆ กันนั้น bean จะต้องเขียนโดยยึดตามอินเทอร์เฟซที่ถูกกำหนดไว้ก่อนแล้ว และเป็นไปตามกฎที่วางไว้

EJB กำหนดให้การอิมพลีเมนต์ของคอมโพเน็นต์อยู่ใน enterprise bean class ซึ่งก็คือคลาสในภาษาจาวาที่เขียนขึ้นตามข้อกำหนดข้างต้น ถึงแม้ว่าจะไม่มีกฎตายตัวใน Enterprise JavaBeans ก็ตาม การอิมพลีเมนต์ session bean จะแตกต่างออกไปจากการอิมพลีเมนต์ entity bean โดย session bean ประกอบด้วยลอจิกที่เกี่ยวข้องกับบิซิเนสโพรเซส ส่วน entity bean เป็นลอจิกเกี่ยวกับข้อมูล จะกล่าวถึงในรายละเอียดต่อไป

ข้อกำหนดของ EJB ระบุถึงอินเทอร์เฟซมาตรฐานจำนวนหนึ่ง ซึ่ง bean class จะต้องอิมพลีเมนต์อินเทอร์เฟซเหล่านี้จะบังคับให้ bean class ต้องมีบริการเมธอดมาตรฐาน ให้คอนเทนเนอร์เรียกใช้เพื่อจัดการกับ bean และแจ้งให้ bean ทราบเมื่อมีเหตุการณ์ (event) ที่เกี่ยวข้องเกิดขึ้น

อินเทอร์เฟซพื้นฐานที่ทุก ๆ bean class ต้องอิมพลีเมนต์ (ทั้ง session bean และ entity bean) คือ อินเทอร์เฟซ *javax.ejb.EnterpriseBean* ดังนี้

```
public interface javax.ejb.EnterpriseBean extends java.io.Serializable
{
}
```

อินเทอร์เฟซนี้เป็นเพียงอินเทอร์เฟซต้นแบบเท่านั้น ในการนำไปใช้งานจริง session bean และ entity bean จะอิมพลีเมนต์อินเทอร์เฟซเฉพาะที่สืบทอดจากอินเทอร์เฟซต้นแบบนี้อีกทีตามชนิดของ bean ซึ่งจะกล่าวถึงต่อไป จุดที่น่าสังเกตคือ อินเทอร์เฟซ *javax.ejb.EnterpriseBean* สืบทอดมาจาก *java.io.Serializable* ซึ่งหมายความว่า enterprise bean สามารถแปลงเป็นบิตข้อมูล และมีคุณสมบัติของออบเจ็กต์ที่สามารถทำ serialization ได้

**2) EJB Object**

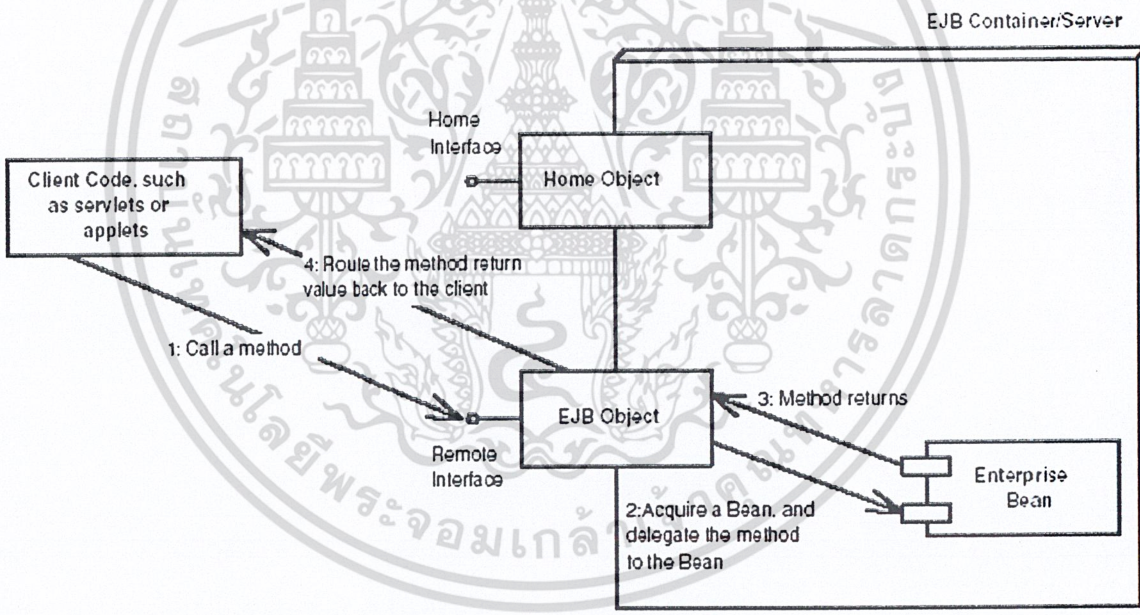
เมื่อไคลเอ็นต์ต้องการใช้งานอินสแทนซ์ของ enterprise bean class ไคลเอ็นต์ไม่สามารถเรียกเมธอดของ bean class ได้โดยตรง แต่การเรียกใช้งานนั้นจะถูกดักจับ (intercept) โดยคอนเทนเนอร์ และถูกส่งต่อ (delegate) ไปยังอินสแทนซ์ของ bean อีกที เหตุผลในการทำเช่นนี้มีหลายประการ เช่น

- enterprise bean class ไม่สามารถถูกเรียกใช้ผ่านทางเครือข่ายได้โดยตรง เนื่องจาก enterprise bean class ไม่มีความสามารถทางด้านเครือข่าย EJB คอนเทนเนอร์จึงทำหน้าที่จัดการเกี่ยวกับเครือข่ายให้โดยการหุ้ม bean เอาไว้ภายในออบเจ็กต์ที่มีความสามารถทางด้านเครือข่าย ออบเจ็กต์นี้จะรับการ

ร้องขอจากไคลเอ็นต์ แล้วส่งต่อไปยังอินสแทนซ์ของ bean class ทำให้เราไม่ต้องสนใจปัญหาเกี่ยวกับเครือข่าย

- ด้วยการดักจับการร้องขอ EJB คอนเทนเนอร์สามารถจัดการบริการที่จำเป็นบางอย่างได้อัตโนมัติ เช่น ลอจิกการจัดการทรานแซ็กชัน , การพูลอินสแทนซ์ของ bean , การรักษาความปลอดภัย และลอจิกอื่น ๆ ที่คอนเทนเนอร์ต้องการ
- EJB คอนเทนเนอร์ สามารถติดตามเก็บข้อมูลการเรียกใช้เมธอดในขณะต่าง ๆ ได้ แล้วแสดงเป็นรายงานให้ผู้ดูแลระบบรับทราบ หรือนำข้อมูลไปใช้เพื่อการกระจายงานระหว่างเซิร์ฟเวอร์ เป็นบริการเพิ่มเติมที่คอนเทนเนอร์สามารถทำได้

จะเห็นว่า EJB คอนเทนเนอร์เป็นเหมือนชั้นที่กั้นการติดต่อโดยตรงระหว่างไคลเอ็นต์และ bean เอาไว้ โดยชั้นที่กั้นการติดต่อดังกล่าวทำหน้าที่เป็นออบเจ็กต์ที่มีความสามารถทางเครือข่าย เรียกว่า EJB object




*รูปที่ 4-10 EJB Object*

EJB object เป็นออบเจ็กต์ที่มีความสามารถในด้านเครือข่าย , การจัดการทรานแซ็กชัน , การรักษาความปลอดภัย และอื่นๆ มันจะทำลอจิกที่คอนเทนเนอร์ต้องการก่อนที่การเรียกใช้เมธอดจะถูกส่งไปยัง bean อินสแทนซ์ EJB object จะเชื่อมการทำงานระหว่างไคลเอ็นต์และ bean โดยจะแสดงบิซิเนสเมธอดทุกเมธอดที่ bean ให้บริการ และทำการส่งต่อการร้องขอจากไคลเอ็นต์ไปยัง bean EJB object แสดงได้ดังรูปที่ 4-10 EJB object ถือเป็นส่วนหนึ่งของคอนเทนเนอร์ และถูกคอนเทนเนอร์สร้างขึ้นมาโดยอัตโนมัติ

**3) Remote Interface**

จากหลักการข้างต้น ไคลเอ็นต์ของ bean จะเรียกใช้งานเมธอดบน EJB object แทนการเรียกใช้งานเมธอดบนตัว bean ดังนั้น EJB object จะต้องมีทุกบิซิเนสเมธอดที่ bean class นั้นให้บริการ โดยผู้พัฒนา bean จะต้องสร้างอินเทอร์เฟซเพื่อบอกเมธอดที่สามารถเรียกใช้ผ่าน EJB object ได้ เรียกว่า remote interface

remote interface ต้องเขียนตามกฎที่ระบุโดยข้อกำหนดของ EJB เช่น ทุก ๆ remote interface จะต้องสืบทอดมาจากอินเทอร์เฟซมาตรฐานของซันที่ชื่อว่า *javax.ejb.EJBObject* แสดงดังนี้

```
public interface javax.ejb.EJBObject
extends java.rmi.Remote
{
  public abstract javax.ejb.EJBHome getEJBHome()
    throws java.rmi.RemoteException;
  public abstract java.lang.Object getPrimaryKey()
    throws java.rmi.RemoteException;
  public abstract void remove()
    throws java.rmi.RemoteException,javax.ejb.RemoveException;
  public abstract javax.ejb.Handle getHandle()
    throws java.rmi.RemoteException;
  public abstract boolean isIdentical(javax.ejb.EJBObject)
    throws java.rmi.RemoteException;
}
```

*javax.ejb.EJBObject* ประกอบด้วยเมธอดที่น่าสนใจซึ่งจะอธิบายในตารางที่ 4-2 โดยเมธอดเหล่านั้นเป็นเมธอดที่จำเป็นสำหรับ EJB object แต่ไม่จำเป็นที่จะต้องโค้ดเอง เนื่องจาก EJB คอนเทนเนอร์จะสร้างมันขึ้นมาโดยอัตโนมัติ

ไคลเอ็นต์ที่ต้องการทำงานกับ bean จะเรียกใช้งานเมธอดใน *javax.ejb.EJBObject* ในที่นี้ไคลเอ็นต์อาจเป็นแอพพลิเคชันทั่วไป Applet , Servlet หรืออะไรก็ได้ รวมถึง enterprise bean อื่น ๆ ด้วย

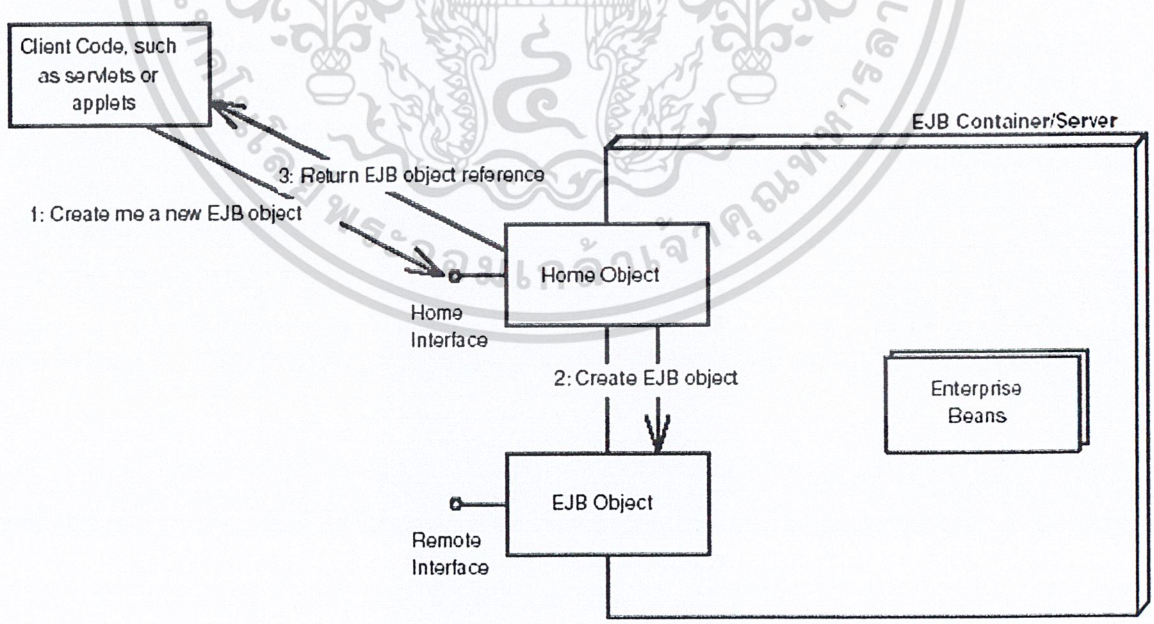
**4) Home Object**

ไคลเอ็นต์ไม่สามารถสร้างอินสแทนซ์ของ EJB object ได้โดยตรง เนื่องจาก EJB object นั้น อาจอยู่ต่างเครื่องกับไคลเอ็นต์ก็ได้ ในทำนองเดียวกันกับ EJB object EJB สนับสนุนการทำโลเคชัน transparancy โดยไคลเอ็นต์ไม่จำเป็นต้องทราบตำแหน่งที่แน่นอนของ EJB object ว่าอยู่ที่ใด

| เมธอด | คำอธิบาย |
|---|---|
| getEJBHome() | รับค่า reference ของ home object |
| getPrimaryKey() | คืนค่า primay key ของ EJB object นี้ โดย primary key นี้จะถูกใช้โดย entity bean เท่านั้น |
| remove() | ทำลาย EJB object เมื่อเรียกใช้เมธอดนี้ ทรัพยากรต่าง ๆ จะถูกคืนให้กับระบบ แต่ในกรณีของ enity bean จะเป็นการลบข้อมูลที่ bean อ้างถึงอยู่ที่อยู่ใน storage |
| getHandle() | รับ handle ของ EJB object EJB handle คือ reference ไปยัง EJB object ที่ไคลเอ็นต์ได้เก็บเอาไว้ หลังจากนั้นไคลเอ็นต์สามารถใช้ handle ในการรับค่า EJB object กลับมาและเริ่มใช้มันอีกครั้ง |
| isIdentical() | ใช้ทดสอบว่า 2 EJB object เหมือนกันหรือไม่ |

***ตารางที่ 4-2 เมธอดของ EJB Object***

เพื่อที่จะได้รับ reference ไปยัง EJB object ไคลเอ็นต์จะต้องขอ EJB object จาก EJB object factory ซึ่งมีหน้าที่สร้างและทำลาย EJB object ตามข้อกำหนดของ EJB เรียกว่า home object หน้าที่หลัก ๆ ของ home object คือ สร้าง EJB object , ค้นหา EJB object ที่มีอยู่แล้ว และทำลาย EJB object




***รูปที่ 4-11 Home Object***

ในลักษณะเดียวกับ EJB object home object ขึ้นอยู่กับชนิดของคอนเทนเนอร์ โดยมีลอจิกซึ่งขึ้นอยู่กับชนิดคอนเทนเนอร์ เช่นการทำ load-balancing , ลอจิกในการติดตามและรายงานข้อมูลการใช้งานคอมโพเน็นต์ เป็นต้น home object ถือเป็นส่วนหนึ่งของคอนเทนเนอร์ และถูกคอนเทนเนอร์สร้างขึ้นมาโดยอัตโนมัติ

**5) Home Interface**

home object เปรียบเสมือนโรงงานของ EJB object ซึ่งคอนเทนเนอร์ต้องการข้อมูลการกำหนดค่าเริ่มต้นให้ EJB object เพื่อสร้าง home object ผู้พัฒนา bean ต้องสร้าง home interface ระบุถึงเมธอดที่ใช้ในการสร้าง , ทำลาย และค้นหา EJB object home object ของคอนเทนเนอร์จะเป็นตัวอิมพลีเมนต์ home interface ลักษณะของ home object แสดงดังรูป 4-11

Enterprise JavaBeans กำหนดเมธอดบังคับสำหรับ home interface ทุกตัว ระบุไว้ในอินเทอร์เฟซ *javax.ejb.EJBHome* ซึ่งเป็นอินเทอร์เฟซที่ home interface จะต้องสืบทอดมา สำหรับอินเทอร์เฟซ *javax.ejb.EJBHome* แสดงดังนี้

```
public interface javax.ejb.EJBHome
extends java.rmi.Remote
{
  public abstract EJBMetaData getEJBMetaData()
    throws java.rmi.RemoteException;
  public abstract void remove(Handle handle)
    throws java.rmi.RemoteException,javax.ejb.RemoveException;
  public abstract void remove(Object primaryKey)
    throws java.rmi.RemoteException,javax.ejb.RemoveException;
}
```

อินเทอร์เฟซ *javax.ejb.EJBHome* สืบทอดมาจาก *java.rmi.Remote* ดังนั้น home object ถือเป็นจาวา RMI รีโมตออบเจ็กต์ที่มีความสามารถด้านเครือข่าย สามารถถูกเรียกใช้งานข้าม virtual machine ได้ โดยพารามิเตอร์ที่ส่งเข้ามาในเมธอดของ home interface ต้องอยู่ในรูปแบบที่ถูกต้องตามจาวา RMI ด้วย

เมธอดของ *javax.ejb.EJBHome* ดังตารางที่ 4-3

| เมธอด | คำอธิบาย |
| --- | --- |
| getEJBMetaData() | ถูกใช้ในการเข้าถึงอินฟอร์เมชันที่เกี่ยวกับ enterprise bean ที่ทำงานอยู่ เช่น bean นั้นเป็น session bean หรือ entity bean อินฟอร์เมชันเหล่านี้จะถูกซ่อนใน *EJBMetaData* object |
| remove() | เมธอดนี้จะทำลาย EJB object โดยจะเรียก remove() ได้ 2 วิธีดังนี้<br>1. โดย pass *javax.ejb.Handle* object ซึ่งจะ remove EJB object ที่ได้รับมาจาก EJB handle นั้น<br>2. โดย pass primary key วิธีนี้จะใช้กับ entity bean เท่านั้นซึ่งจะกล่าวถึงในภายหลัง |

***ตารางที่ 4-3 เมธอดของ Home Object***

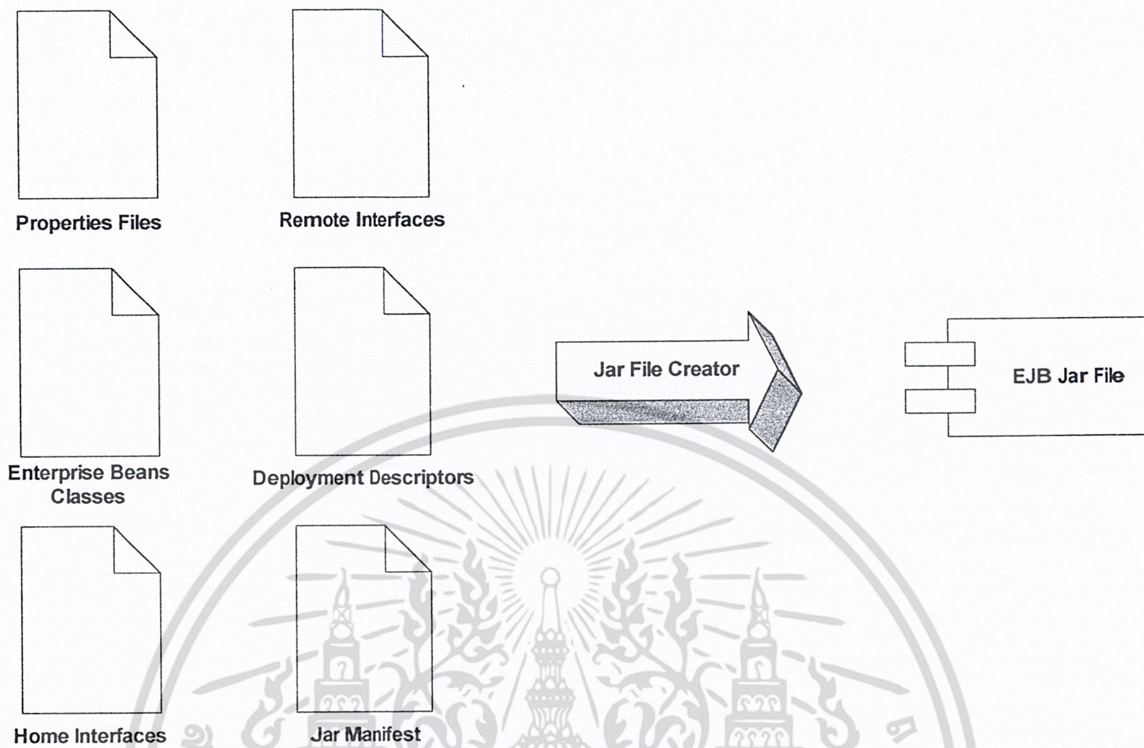
**6) Deployment Descriptor**

deployment descriptor ทำให้ EJB คอนเทนเนอร์สามารถให้บริการแก่คอมโพเน็นต์ enterprise bean ได้อย่างอัตโนมัติ (implicit middleware services) โดยที่ bean ไม่ต้องโค้ดในระดับ API ของมิดเดิลแวร์ แต่จะได้รับบริการเหล่านี้จากมิดเดิลแวร์โดยอัตโนมัติ

จุดประสงค์ของ descriptor เพื่อบอกความต้องการบริการจากคอนเทนเนอร์ โดยระบุความต้องการไว้ในไฟล์ deployment descriptor ตัวอย่างเช่น การจัดการกับวงจรชีวิตของ bean , persistent , การควบคุมทรานแซ็กชัน และบริการด้านความปลอดภัย เป็นต้น คอนเทนเนอร์จะอ่านข้อมูลจาก deployment descriptor และให้บริการต่าง ๆ แก่ bean ตามที่ระบุไว้ การระบุความต้องการใน descriptor จะขึ้นอยู่กับผลิตภัณฑ์ เนื่องจากแต่ละผลิตภัณฑ์มีความสามารถต่างกัน จะมีเฉพาะไฟล์ ejb-jar.xml เท่านั้นที่เป็นมาตรฐานเดียวกันทั้งหมด โดยไฟล์นี้จะใช้ระบุข้อมูลของ bean ในโครงงานนี้ใช้ WebLogic Server 6.0 descriptor ของมันจะประกอบด้วย ejb-jar.xml ดังที่กล่าวมาแล้ว , weblogic-ejb-jar.xml ที่ใช้ระบุค่าคอนฟิกต่าง ๆ ที่เกี่ยวกับ bean และไฟล์ที่ใช้แมปข้อมูลในฐานข้อมูลกับ entity bean ในกรณีที่ใช้ container-managed persistent entity bean

**7) ไฟล์ ejb-jar**

เมื่อสร้าง bean class , home interface , remote interface , deployment descriptor และไฟล์ระบุคุณสมบัติของ bean เสร็จเรียบร้อยแล้ว รวบรวมไฟล์ทั้งหมดเข้าด้วยกันเป็นไฟล์เดียวเรียกว่าไฟล์ ejb-jar ไฟล์ ejb-jar นี้เป็นไฟล์ .jar ซึ่งถูกบีบอัดเอาไว้ในรูปแบบ .zip การรวมไฟล์ทั้งหมดที่เกี่ยวข้องเป็นไฟล์เดียวนี้ทำให้เกิดความสะดวกในการเคลื่อนย้ายซอฟต์แวร์เพื่อนำไปดีพลอย ขั้นตอนการสร้างไฟล์ ejb-jar แสดงดังรูป 4-12




*รูปที่ 4-12 ขั้นตอนการสร้างไฟล์ Ejb-jar*

หลังจากสร้างไฟล์ ejb-jar แล้ว ถือว่า enterprise bean เสร็จสมบูรณ์ สามารถนำไปดีพลอยในแอพพลิเคชันเซิร์ฟเวอร์ได้ เมื่อนำไปดีพลอย เครื่องมือของผู้พัฒนาคอนเทนเนอร์จะทำหน้าที่ขยายไฟล์ ejb-jar และนำข้อมูลต่าง ๆ ในไฟล์มาใช้งาน

# บทที่ 5

# Session Bean

## 5.1 แนะนำ Session Bean

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

### 5.1.1 ลักษณะของ Session Bean

- **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

- **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-คอมเมิร์ซ รถเข็นแบบออนไลน์จะต้องเก็บสถานะของสินค้าที่สั่งไว้ เป็นต้น

- **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

**5.1.2 การเขียน Session Bean**

การเขียน session enterprise bean class จะต้องอิมพลีเมนต์อินเทอร์เฟซของ *javax.ejb.SessionBean* อินเทอร์เฟซจะกำหนดเมธอดที่จำเป็นต้องมี โดยจะเป็นเมธอดที่ใช้ในการจัดการที่ EJB คอนเทนเนอร์จะเรียกกับ bean เพื่อเตือนเหตุการณ์สำคัญบางอย่าง ไคลเอ็นต์ของ bean จะไม่เรียกเมธอดเหล่านี้เนื่องจากเมธอดเหล่านี้ไม่ได้มีไว้สำหรับให้ไคลเอ็นต์เรียกกับ EJB object อินเทอร์เฟซ *javax.ejb.SessionBean* เป็นดังนี้

```
public interface javax.ejb.SessionBean
extends javax.ejb.EnterpriseBean
{
  public abstract void setSessionContext(SessionContext ctx)
    throws java.rmi.RemoteException;
  public abstract void ejbPassivate()
    throws java.rmi.RemoteException;
  public abstract void ejbActivate()
    throws java.rmi.RemoteException;
  public abstract void ejbRemove()
    throws java.rmi.RemoteException;
}
```

สังเกตว่าอินเทอร์เฟซ *javax.ejb.SessionBean* ขยายมาจากอินเทอร์เฟซ *javax.ejb.EnterpriseBean* โดย entity bean มีอินเทอร์เฟซ *javax.ejb.EntityBean* ที่ขยายมาจาก *javax.ejb.EnterpriseBean* เช่นกัน

การเขียน session bean จะต้องประกอบด้วยเมธอดดังนี้

- **setSessionContext(SessionContext ctx)**

คอนเทนเนอร์จะเรียกเมธอดกับ bean โดยใช้ session context bean และคอนเทนเนอร์จะติดต่อกันผ่านทาง session context เช่น bean จะใช้ session context ในการ query คอนเทนเนอร์เพื่อขอข้อมูลเช่นสถานะทรานแซ็กชันของ bean นั้น

โดยทั่ว ๆ ไปการอิมพลีเมนต์ bean จะเก็บ context เป็นสมาชิกของมันเพื่อจะได้นำมา query ในภายหลัง ตัวอย่างดังนี้

```
import javax.ejb.*;
public class MyBean implements SessionBean
{
    private SessionContext ctx;
    public void setSessionContext(SessionContext ctx)
    {
        this.ctx=ctx;
    }
    ...
}
```

- **ejbCreate(...)**

*ejbCreate(...)* จะใช้เริ่มต้นการทำงานของ session bean โดยเมธอดนี้อาจมีพารามิเตอร์ได้หลากหลายขึ้นอยู่กับว่า session bean นั้นใช้ทำอะไร เมธอดนี้ต้องมีอย่างน้อย 1 เมธอด โดยอาจใช้ *ejbCreate(...)* ในการกำหนดค่าเริ่มต้นให้กับ bean เช่นกำหนดค่าให้กับตัวแปร ตัวอย่างดังนี้

```
import javax.ejb.*;
public class MyBean implements SessionBean
{
    private int memberVariable;
    public void ejbCreate(int initialValue)
    {
        this.memberVariable=initialValue;
    }
```

```
  ...
}
```

*ejbCreate(...)* เป็น callback เมธอดที่คอนเทนเนอร์จะเรียกใช้ ไคลเอ็นต์จะไม่เรียก *ejbCreate(...)* เพราะไคลเอ็นต์ไม่ได้ทำงานกับ bean โดยตรง แต่ต้องทำผ่านคอนเทนเนอร์ แต่ไคลเอ็นต์ต้องมีวิธีในการ pass ค่าพารามิเตอร์ให้กับ *ejbCreate(...)* โดยไคลเอ็นต์จะ initial bean ผ่านทาง home interface เช่น หากมีเมธอด *ejbCreate()* ดังนี้ใน bean class

```
public void ejbCreate(int i) throws ...
```

จะต้องมีเมธอด *create()* ใน home interface ดังนี้

```
public  void create(int i) throws ...
```

- **ejbPassivate()**

ถ้ามีอินสแทนซ์ของ session bean มากเกินไปในคอนเทนเนอร์ คอนเทนเนอร์สามารถ passivate บางส่วน ซึ่งหมายถึงการเก็บ bean ลงในที่เก็บชั่วคราวเช่นฐานข้อมูลหรือไฟล์ คอนเทนเนอร์สามารถปล่อยทรัพยากรที่ใช้โดย bean นั้นได้ โดยก่อนที่ bean จะถูก passivate คอนเทนเนอร์จะเรียกเมธอด *ejbPassivate()*

bean ที่อิมพลีเมนต์ *ejbPassivate()* ควรจะปล่อยทรัพยากรที่ bean ยึดครองอยู่ ตัวอย่างดังนี้

```
import javax.ejb.*;
public class MyBean implements SessionBean
{
  public void ejbPassivate()
  {
    <close socket connections , etc...>
  }
  ...
}
```

passivate จะไม่ถูกนำมาใช้กับ stateless session bean เนื่องจาก stateless session bean ไม่ได้เก็บสถานะใด ๆ ไว้ ดังนั้นในการเรียกเมธอดแต่ละครั้ง จึงมีช่วงเวลาที่ไม่ยาวนานเหมือนในกรณีของ stateful session bean ที่อาจจะนานเป็นวันหรือมากกว่านั้น หากไม่ได้กำหนดเวลา time out ไว้

- **ejbActivate()**

เมื่อไคลเอ็นต์จำเป็นที่จะต้องใช้ bean ที่ถูก passivate ไว้ กระบวนการในการนำ bean กลับสู่เมโมรีคือการ activate bean โดยก่อนที่ bean จะถูก activate คอนเทนเนอร์จะเรียกเมธอด *ejbActivate()*

ในขณะที่ bean กลับมาอยู่ในเมโมรี การอิมพลีเมนต์ *ejbActivate()* ควรจะนำทรัพยากรที่ bean ต้องใช้กลับคืนมาด้วย ตัวอย่างดังนี้

```
import javax.ejb.*;
public class MyBean implements SessionBean
{
  public void ejbActivate()
  {
    <open socket connections , etc...>
  }
  ...
}
```

เช่นเดียวกับการ passivate เมธอดนี้จะไม่ถูกนำไปใช้กับ stateless session bean ด้วยเหตุผลเดียวกัน

- **ejbRemove()**

เมื่อคอนเทนเนอร์ต้องการจะ remove อินสแทนซ์ของ session bean มันจะเรียก bean ด้วย callback เมธอด *ejbRemove()* แนวคิดของมันจะเหมือนกับ destructor ในภาษา C++

อิมพลีเมนต์ของ *ejbRemove()* ควรจะปล่อยทรัพยากรทั้งหมดที่ได้จองไว้ ตัวอย่างดังนี้

```
import javax.ejb.*;
public class MyBean implements SessionBean
{
  public void ejbRemove()
  {
    <prepare for destruction>
  }
  ...
}
```

- **Business Method**

  เป็นเมธอดที่ใช้ในการประมวลผลงานที่ต้องการ ตัวอย่างดังนี้

```
import javax.ejb.*;
public class MyBean implements SessionBean
{
  public int add(int i,int j)
  {
    return (i+j);
  }
  ...
}
```

**5.1.3 การเรียกใช้ Session Bean**

ไคลเอ็นต์โค้ดที่จะเรียกใช้ session bean จะต้องทำดังนี้

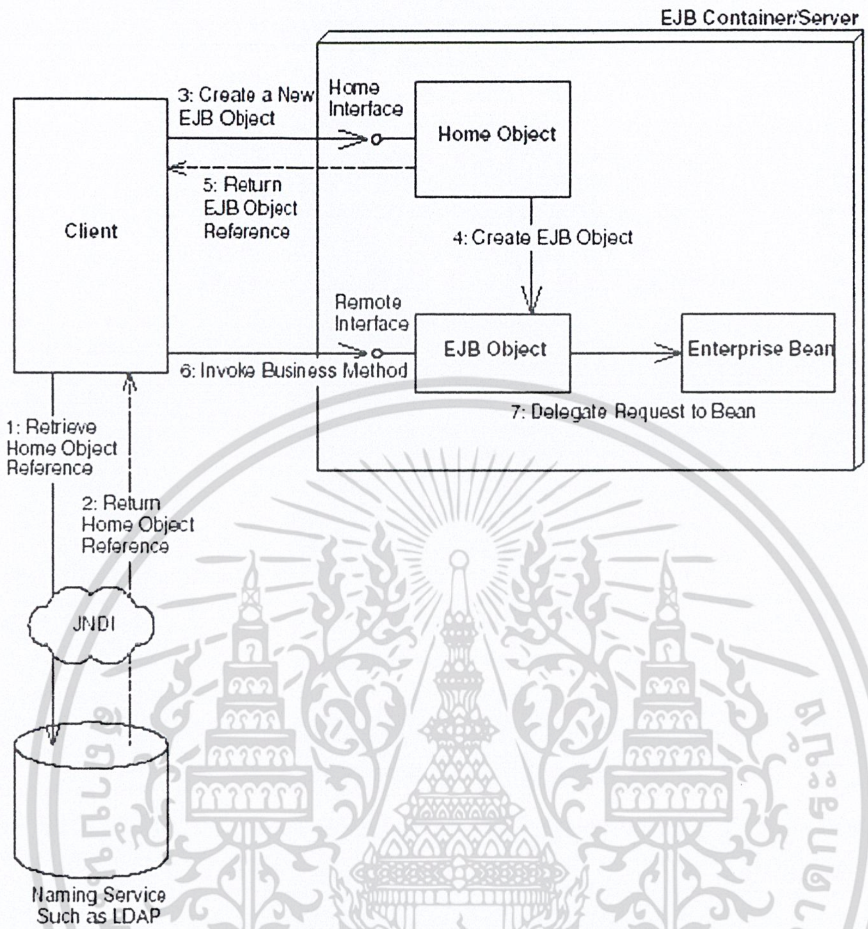
1) หา home object
2) ใช้ home object สร้าง EJB object
3) เรียกบิซิเนสเมธอดกับ EJB object
4) remove EJB object

- **การหา Home Object**

ในการหา home object ไคลเอ็นต์โค้ดจะต้องใช้ JNDI เนื่องจาก home object อาจจะอยู่บนเครื่องใด ๆ ในเน็ตเวิร์ก ตัวอย่างเช่น bean มีชื่อว่า MyBean อาจจะถูกกำหนดให้มีชื่อที่ใช้ในการ bind กับ JNDI ว่า MyHome ซึ่งจะต้องกำหนดใน deployment descriptor โดยคอนเทนเนอร์จะ bind MyHome เข้ากับ home object เมื่อไคลเอ็นต์ look up JNDI โดยใช้ชื่อ MyHome ก็จะได้ reference ของ home object ของ MyBean กลับไปดังรูปที่ 5-1

ไคลเอ็นต์โค้ดจะต้อง look up JNDI ดังนี้

1) ตั้งค่าเอ็นไวรอนเมนต์ – ตั้งค่าเน็ตเวิร์กโลเคชันที่เซอร์วิสอยู่ให้ถูกต้อง รวมถึง username และ password ในกรณีที่มีการตรวจสอบสิทธิ
2) สร้าง initial context
3) รับค่า home object




***รูปที่ 5-1 การรับ reference ไปยัง Home Object***

ตัวอย่างโค้ดเป็นดังนี้

```
/*
 *  Get System properties for JNDI initialization
 */
Properties props = System.getProperties();
/*
 * Form an initial context
 */
Context ctx = new InitialContext(props);
/*
 * Get a reference to the home object - the
```

```
* factory for EJB objects
*/
MyHome home = (MyHome) ctx.lookup("MyHome");
```

- **สร้าง EJB Object**

เมื่อได้ reference ไปยัง home object ให้ใช้ home object ในการ create EJB object ดังนี้

```
myRemoteInterface ejbObject = home.create();
```

ในกรณีนี้สมมติว่าเป็น stateless session bean จึงไม่ต้อง pass พารามิเตอร์ แต่หากเป็น bean ชนิดอื่นจะต้องเรียกเมธอด *create(...)* ให้ตรงตามที่ปรากฏในคลาสที่อิมพลีเมนต์ home interface

- **Call Method**

เมื่อได้ EJB object มาแล้วให้เรียกบิซิเนสเมธอดกับมัน ดังนี้

```
ejbObject.add();
```

- **Destroy EJB Object**

สุดท้าย เมื่อต้องการทำลาย EJB object ให้เรียกเมธอด *remove()* กับ EJB object หรือ home object เพื่อให้คอนเทนเนอร์ทำลาย EJB object ดังนี้

```
ejbObject.remove();
```

**5.2 Stateless Session Bean**

**5.2.1 ลักษณะของ Stateless Session Bean**

- **ไม่มีการเก็บสถานะ**

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

- **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

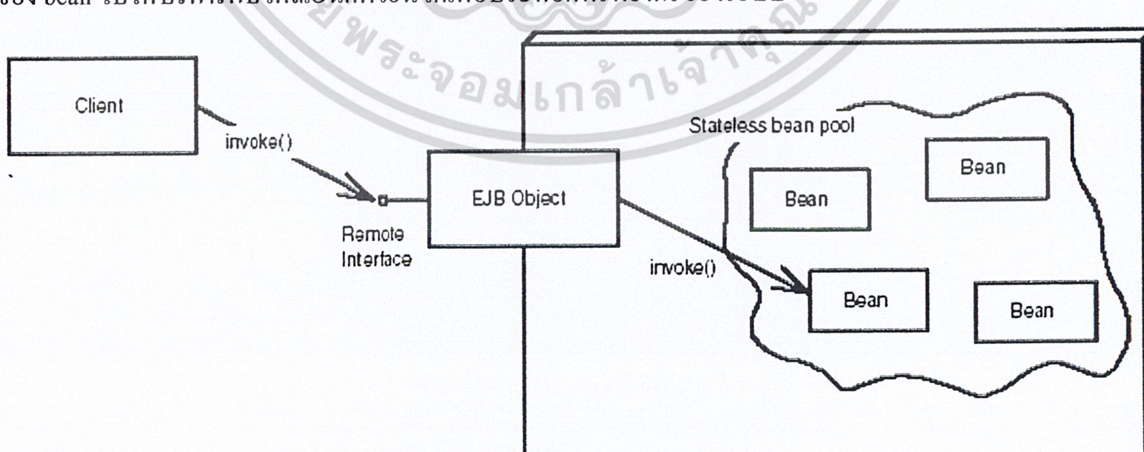
จากเหตุผลดังกล่าว stateless session bean จึงมีเมธอด *ejbCreate()* ได้เพียงตัวเดียว ที่ไม่รับค่าพารามิเตอร์ใด ๆ และใน home object จะมีเพียงเมธอด *create()* เข้าคู่กันซึ่งไม่รับพารามิเตอร์ใด ๆ เช่นกัน

- **คอนเทนเนอร์สามารถพูล และนำ Stateless Session Bean กลับมาใช้ใหม่ได้**

เนื่องจากเมธอด *ejbCreate()* ของ stateless session bean ไม่รับพารามิเตอร์ใด ๆ ไคลเอ็นต์จึงไม่มีการส่งข้อมูลเพื่อเริ่มต้นการทำงานของอินสแทนซ์ของ bean ดังนั้นคอนเทนเนอร์สามารถสร้างอินสแทนซ์ของ stateless session bean ไว้ก่อนที่ไคลเอ็นต์จะติดต่อเข้ามาได้ เมื่อไคลเอ็นต์เรียกใช้งานเมธอดคอนเทนเนอร์ก็ดึงอินสแทนซ์จากในพูลไปให้บริการกับไคลเอ็นต์ แล้วนำอินสแทนซ์นั้นกลับเข้าไปเก็บไว้ในพูลเช่นเดิมเมื่อไคลเอ็นต์เลิกใช้งาน 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



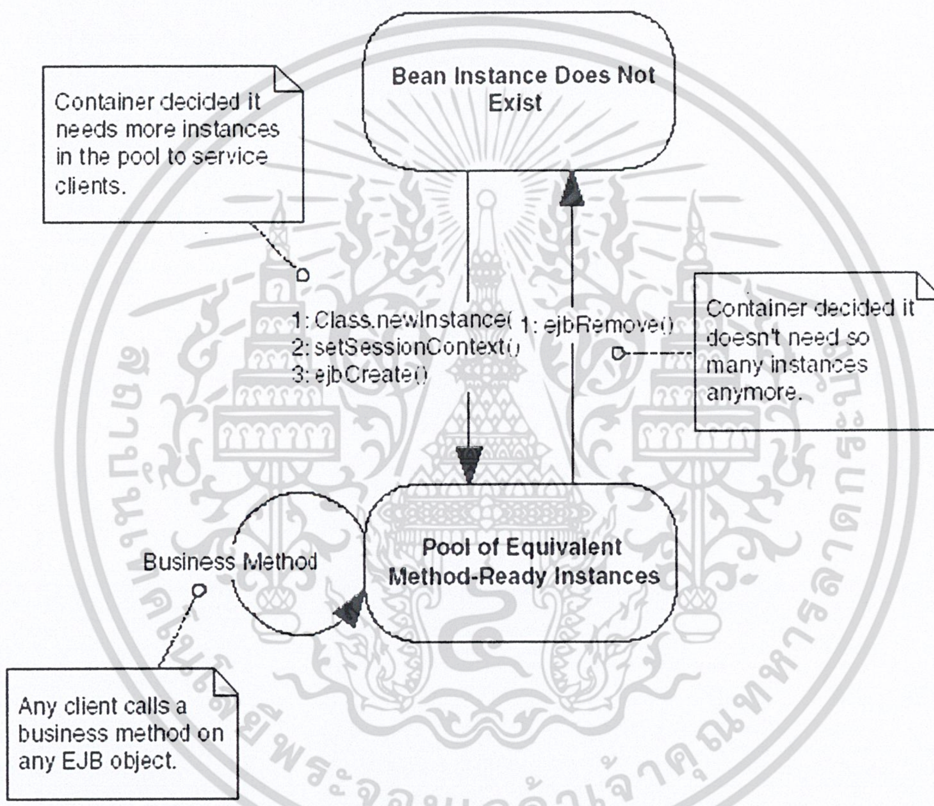

***รูปที่ 5-2 การพูล Stateless Session Bean***

ขนาดพูลของ bean ไม่จำเป็นต้องกำหนดไว้ตายตัว คอนเทนเนอร์ที่มีความสามารถสูงสามารถปรับขนาดของพูลได้อย่างไดนามิก โดยอัตโนมัติขณะโหลดเปลี่ยนแปลง เช่น ถ้าหากมีไคลเอ็นต์เข้ามาใช้งานตอนกลางวันมากกว่าตอนกลางคืน คอนเทนเนอร์อาจจะสร้างพูลขนาดใหญ่ในตอนกลางวัน และขนาดเล็กกว่าในตอนกลางคืน ช่วยให้ระบบมีทรัพยากรเหลือมากขึ้นสำหรับทำงานอย่างอื่นในช่วงเวลาที่ระบบมีโหลดไม่มาก

การพูลอินสแทนซ์ของ stateless session bean แสดงในรูปที่ 5-2

- **วงจรชีวิตของ Stateless Session Bean**

วงจรชีวิตของ stateless session bean ดังรูปที่ 5-3




***รูปที่ 5-3 วงจรชีวิตของ Stateless Session Bean***

### 5.2.2 ตัวอย่างโค้ด Stateless Session Bean

ตัวอย่างของ stateless session bean นี้เป็น bean ที่ใช้ในการแปลงค่าเงิน โดยในการโค้ดจะต้องประกอบด้วยส่วนต่าง ๆ ดังนี้

- **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 ซึ่งเป็นข้อแตกต่างของบิซิเนสเมธอดในอินเทอร์เฟซและ enterpise bean class

- **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;
  }
}
```

- **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*

- **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>
```

- **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 ไปดีพลอย โดยดูวิธีการติดตั้ง WebLogic Server และวิธีดีพลอยได้ในภาคผนวก ก

- **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**

- **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 จะต้องคงอยู่ในการเรียกใช้เมธอดครั้งต่อไปของไคลเอ็นต์ตัวเดิมนั้น ดังนั้นคอนเทนเนอร์จะไม่สามารถพูล bean ขึ้นมาและนำไปให้บริการการเรียกใช้

งานเมธอดต่าง 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

- **ข้อบังคับในการเก็บสถานะของไคลเอ็นต์**

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

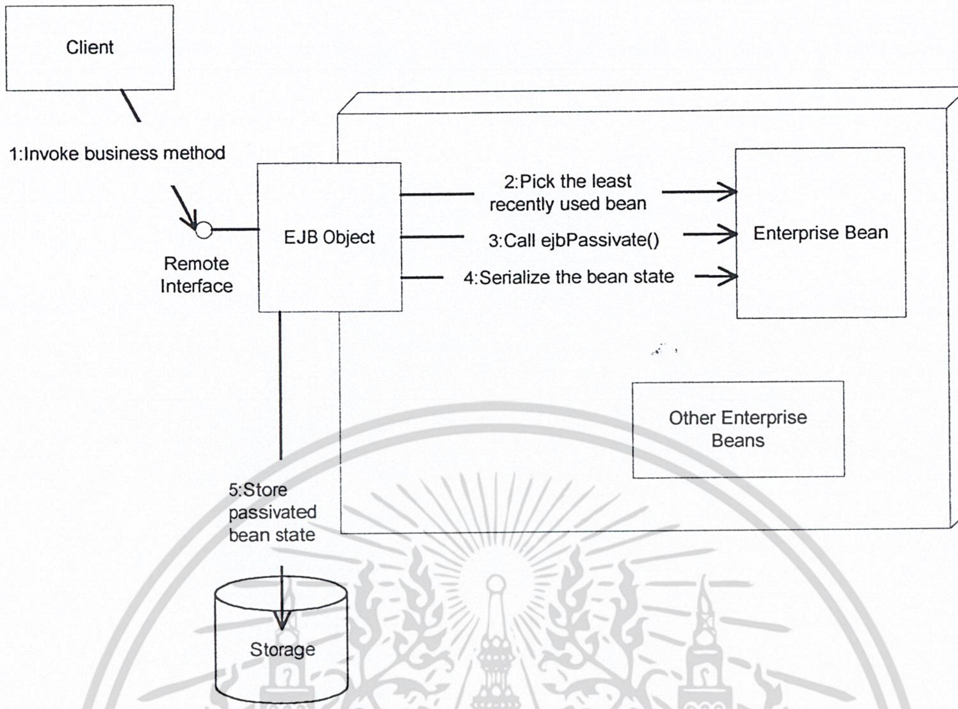
สำหรับจาวาออบเจ็กต์ที่เป็นส่วนหนึ่งของสถานะของ bean ก็ใช้หลักการเดียวกันคือออบเจ็กต์นั้นจะต้องอิมพลีเมนต์ *java.io.Serializable* ด้วย แม้ว่า bean จะต้องเป็นไปตามกฎของ object serialization แต่ EJB คอนเทนเนอร์ไม่จำเป็นต้องใช้โพรโตคอลมาตรฐานในการทำ serialization เสมอไป อาจใช้โพรโตคอลของตัวเองซึ่งสามารถช่วยเพิ่มความยืดหยุ่นและความแตกต่างระหว่างคอนเทนเนอร์ของแต่ละผู้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

- **ขั้นตอนการทำ Passivation และ Activation**

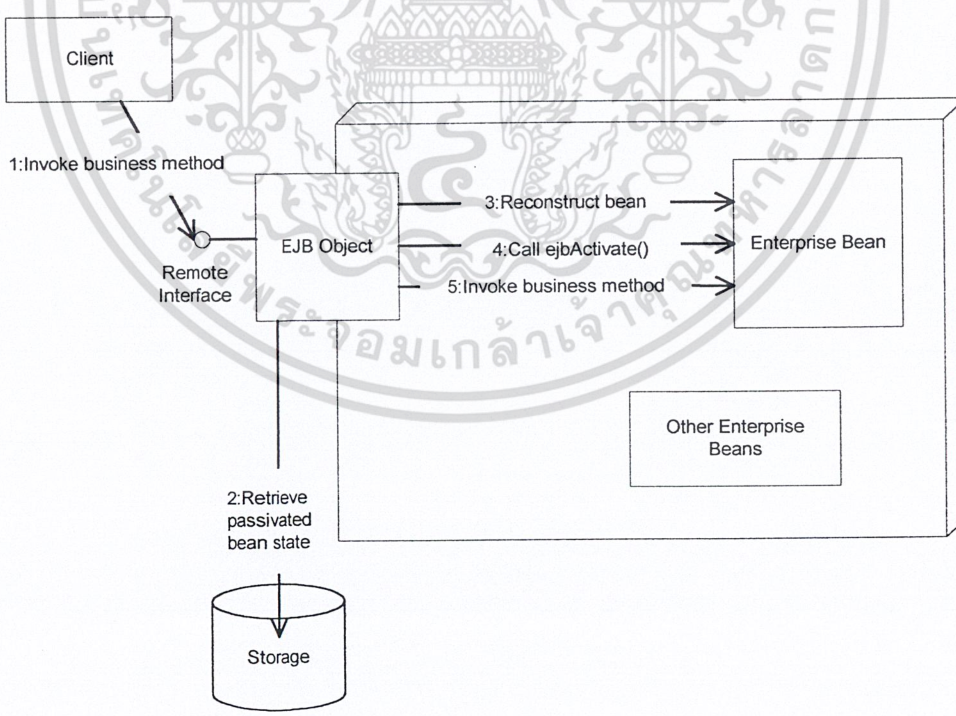
เมื่อ EJB คอนเทนเนอร์ passivate bean คอนเทนเนอร์จะเก็บสถานะการทำงานของ bean นั้นลงไปในแหล่งเก็บข้อมูล เช่นไฟล์ หรือฐานข้อมูล โดยคอนเทนเนอร์จะบอกให้ bean ทราบว่า มันกำลังจะทำการ passivate ด้วยการเรียกใช้เมธอด *ejbPassivate()* เมธอดนี้เป็นการเตือน bean ว่า สถานะการทำงานของมันกำลังจะถูก swap ออกไป

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 5-4

ขั้นตอนตรงกันข้ามจะเกิดขึ้นในกรณีของการ activate bean ซึ่งสถานะของการทำงานที่ถูก serialize จะถูกอ่านกลับเข้ามาในหน่วยความจำ และคอนเทนเนอร์จะสร้างสถานะขึ้นมาใหม่ในหน่วยความจำ โดยใช้ Object Serialization หรือสิ่งที่เหมือนกัน (equivalent) ในการแปลง หลังจากนั้นคอนเทนเนอร์จะเรียกเมธอด *ejbActivate()* ทำให้ bean สามารถเรียกใช้ทรัพยากรต่าง ๆ ที่จำเป็น ซึ่งถูกปล่อยไปขณะถูก passivate ด้วยเมธอด *ejbPassivate()* กลับมา ขั้นตอนการทำ activation แสดงในรูปที่ 5-5



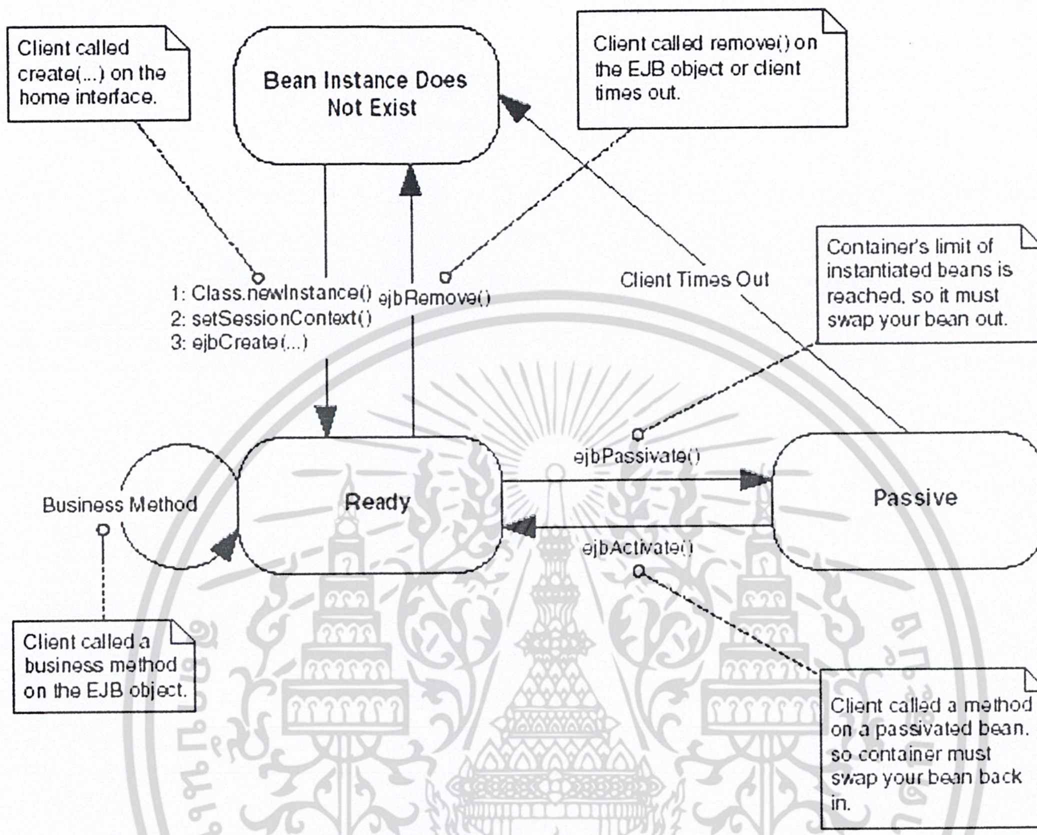

*รูปที่ 5-4 การ Passivate Stateful Session Bean*




*รูปที่ 5-5 การ Activate Stateful Session Bean*

- **วงจรชีวิตของ Stateful Session Bean**

วงจรชีวิตของ stateful session bean ดังรูปที่ 5-6




*รูปที่ 5-6 วงจรชีวิตของ Stateful Session Bean*

### 5.3.2 ตัวอย่างโค้ด Stateful Session Bean

ตัวอย่างของ bean คือ bean ที่ใช้นับ

- **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;
}
```

- **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

- **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;
}
```

- **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>
```

- **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

- 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 แนะนำ Entity Bean

### 6.1.1 แนวคิดของการทำ persistent

- **Java Object Serialization**

ในการทำงานกับออบเจ็กต์ที่เขียนด้วยภาษาจาวา อาจต้องการเก็บสถานะ (state) ของออบเจ็กต์ลงใน permanent storage วิธีหนึ่งที่ทำได้คือใช้วิธีการ object serialization ซึ่งเป็นการแปลงออบเจ็กต์ให้อยู่ในรูปสตรีมของไบต์ข้อมูล (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

- **Object-Relational Mapping**

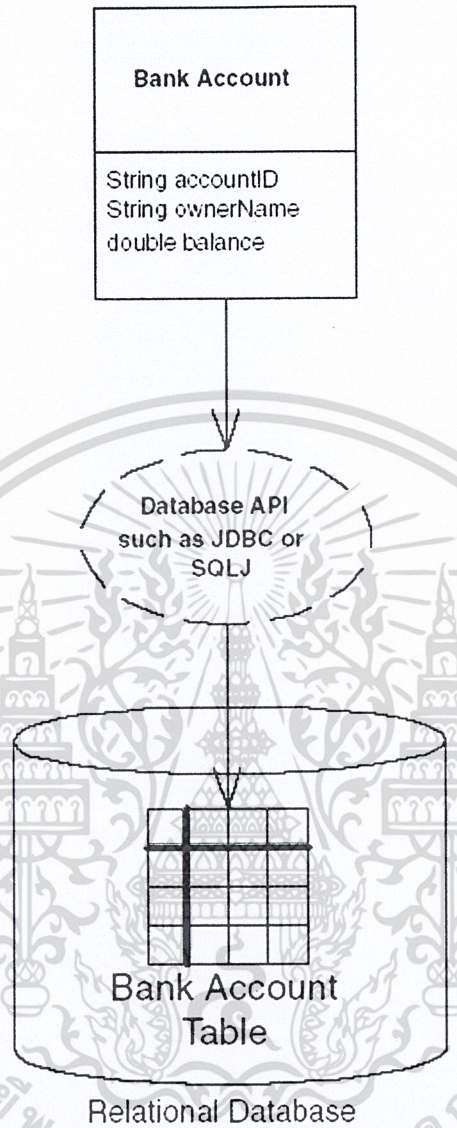
อีกวิธีหนึ่งที่ได้รับความนิยมในการเก็บจาวาออบเจ็กต์ก็คือ การใช้ฐานข้อมูลเชิงสัมพันธ์ (relational database) เช่น Oracle หรือ Microsoft SQL Server แทนที่จะทำการ serialize ออบเจ็กต์ การบันทึกจาวาออบเจ็กต์ใช้ JDBC หรือ SQL/J เพื่อแมปข้อมูลของออบเจ็กต์ไปยังฐานข้อมูลเชิงสัมพันธ์ นอกจากนี้ยังสามารถบันทึกชื่อของจาวาคลาสที่สัมพันธ์กับข้อมูลนี้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

- **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 มีความซับซ้อนมาก ๆ เช่น การเก็บฐานข้อมูลของ




***รูปที่ 6-1 Object-Relational Mapping***

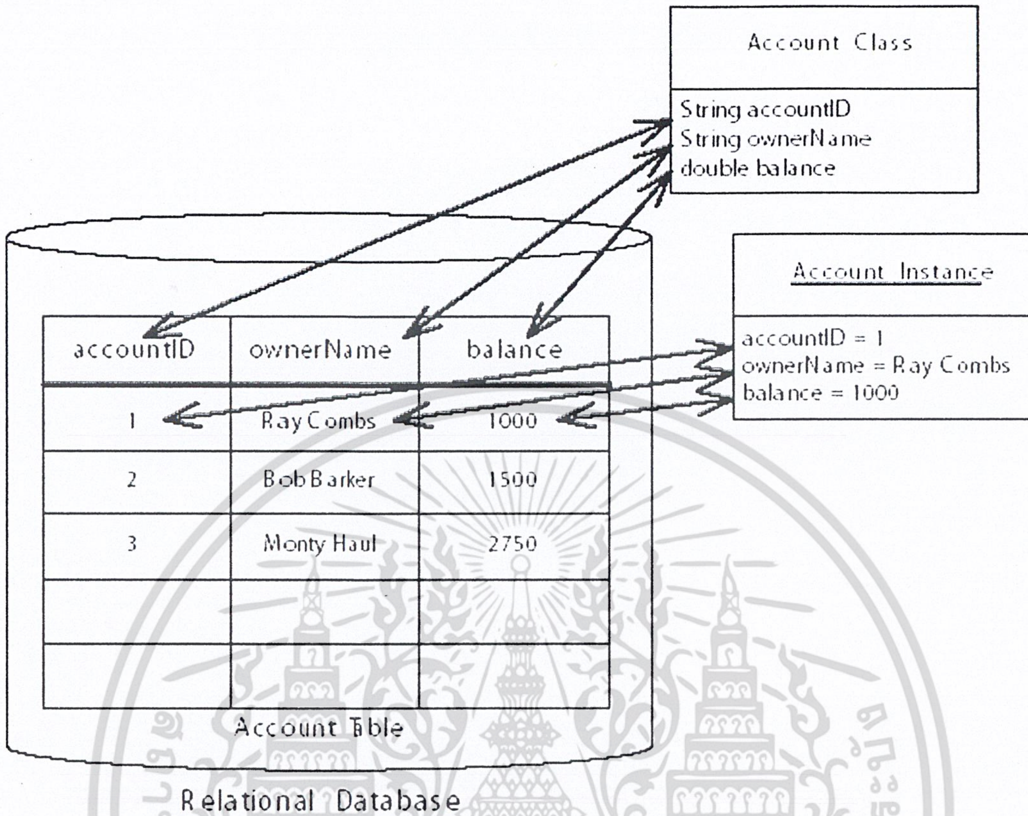
CAD/CAM แต่ถ้าออบเจ็กต์ไม่ซับซ้อนและสามารถแมปไปเป็นฐานข้อมูลเชิงสัมพันธ์ได้ง่าย เช่น แอพพลิเคชันทางธุรกิจต่าง ๆ ซึ่งเน้นที่ปริมาณงานจำนวนมากแล้ว ฐานข้อมูลเชิงสัมพันธ์จะมีประสิทธิภาพที่สูงกว่า

**6.1.2 แนวคิดของ Entity Bean**

คอมโพเน็นต์ที่ถูกดีพลอยแบ่งแยกความแตกต่างได้ดังนี้

1) **Application Logic Components** - เป็นคอมโพเน็นต์ที่ให้บริการสำหรับงานทั่ว ๆ ไปเช่น

- การคำนวณราคาคำสั่งซื้อ




***รูปที่ 6-2 ตัวอย่างการแมประหว่างออบเจ็กต์และฐานข้อมูลเชิงสัมพันธ์***

- การออกบิลบัตรเครดิตของลูกค้า
- การหาอินเวิร์สของเมตริก

โดยส่วนใหญ่คอมโพเน็นต์เหล่านี้จะแทนการกระทำ เหมาะกับการใช้กับบิซิเนสโพรเซส

2) **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 สามารถใช้เมธอดเพื่อจัดการกับกลุ่มข้อมูล

เหล่านี้ 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

**6.1.3 ส่วนประกอบของ Entity Bean**

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

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

3) **Home Interface**

เป็นอินเทอร์เฟซที่ไคลเอ็นต์จะต้องเรียกใช้เพื่อสร้าง , ค้นหา , และทำลาย EJB ออบเจ็กต์ของ entity bean ภายใน home interface ต้องระบุเมธอดบอกถึงวิธีที่มีทั้งหมดในการสร้าง EJB ออบเจ็กต์ใหม่ขึ้นมา , การค้นหาและการทำลาย EJB ออบเจ็กต์เดิมที่มีอยู่แล้ว ผู้พัฒนา EJB คอนเทนเนอร์ จะให้เครื่องมือสำหรับอิมพลีเมนต์ 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

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

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

**6.1.4 ลักษณะของ Entity Bean**

- **Entity Bean มีช่วงชีวิตยาวนาน**

session bean มีอายุเท่ากับ session ของไคลเอ็นต์เท่านั้น เมื่อไคลเอ็นต์จบการติดต่อก็สามารถทำลาย session bean ได้ทันที

ส่วน entity bean จะมีช่วงชีวิตยาวนานเท่ากับคอนเทนเนอร์ โดยเริ่มและจบการทำงานพร้อมกับคอนเทนเนอร์ 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

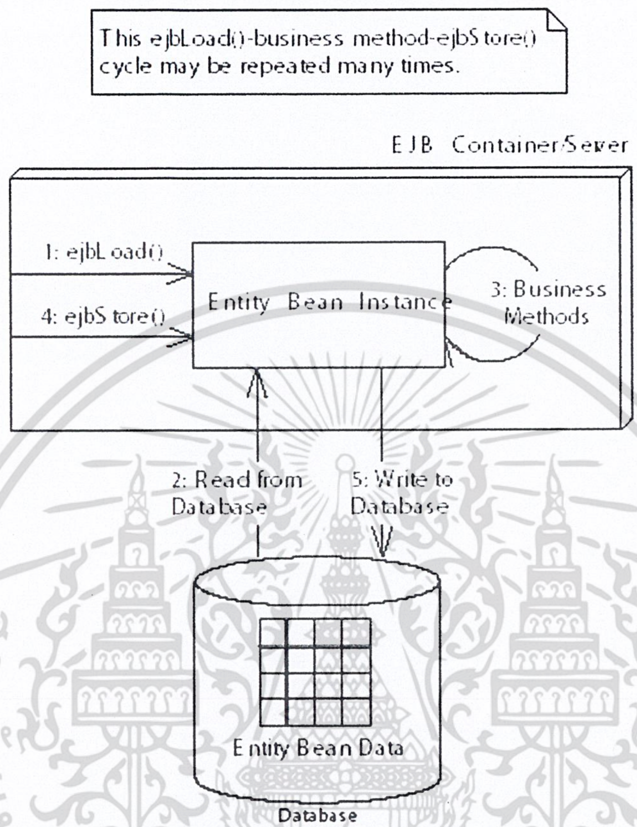
ในความเป็นจริงแล้ว ข้อมูลหนึ่ง ๆ จะมีอยู่สองชุดด้วยกัน คืออินสแทนซ์ของ entity bean ซึ่งอยู่ในหน่วยความจำ และตัว entity bean จริง ๆ ที่ถูกเก็บอยู่ในฐานข้อมูล ดังนั้นจึงต้องมีวิธีการส่งผ่านข้อมูลไปมาระหว่างจาวาออบเจ็กต์และฐานข้อมูล การส่งผ่านข้อมูลนี้สามารถทำได้โดยเมธอดที่ทุก entity bean class ต้องอิมพลีเมนต์ 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 ขึ้นอยู่กับความ

สามารถของคอนเทนเนอร์ 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

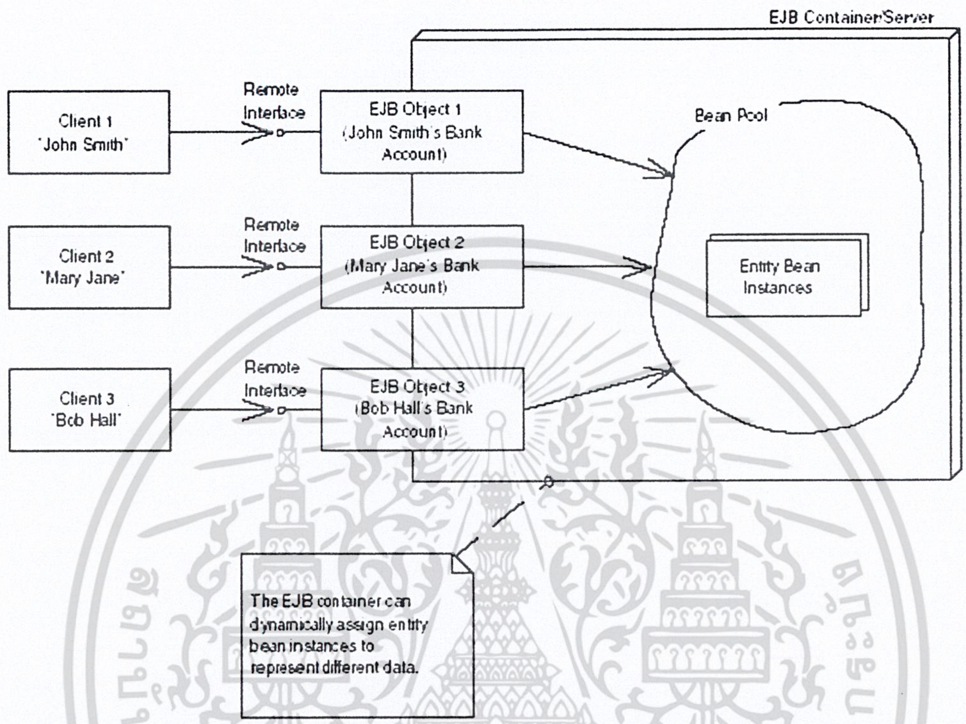
นอกจากจะช่วยประหยัดเวลาในการสร้างและทำลายอินสแทนซ์ของ bean โดยไม่จำเป็นแล้ว วิธีการนี้ยังช่วยลดทรัพยากรที่ระบบจำเป็นต้องใช้อีกด้วย ดังแสดงในรูปที่ 6-4

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

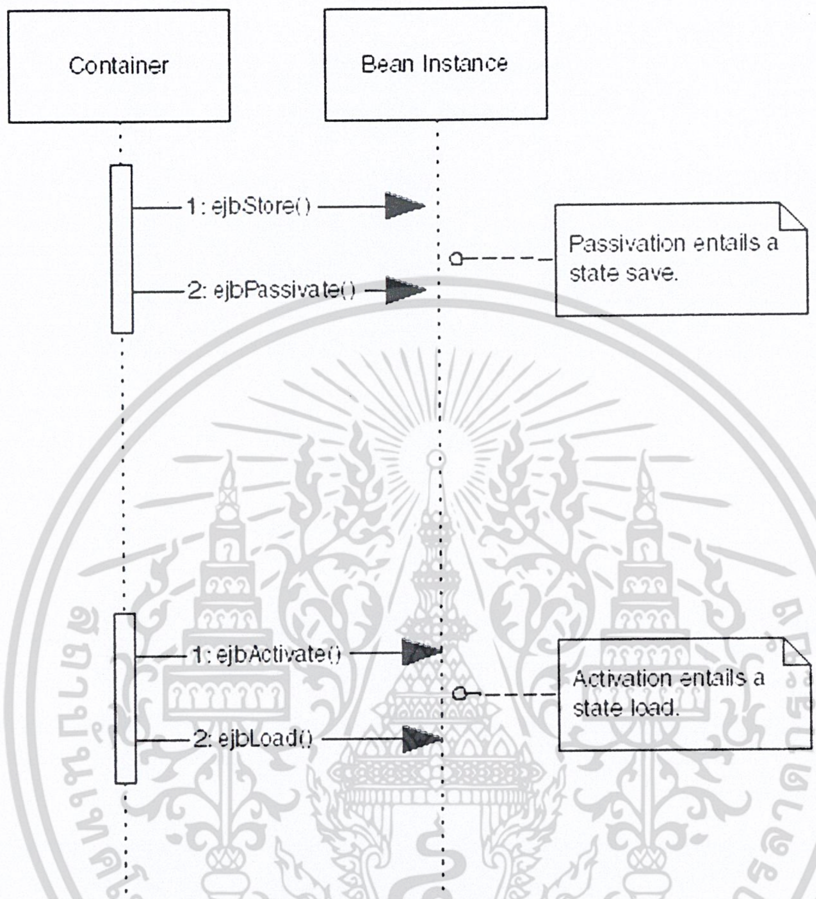
เมธอด *ejbPassivate()* ควรทำการปล่อยทรัพยากรที่ถือครองอยู่ทั้งหมด เช่นซ็อกเก็ตซึ่ง bean ได้ถือครองไว้ขณะ activation หลักการนี้เป็นหลักการเดียวกับการทำ passivation ของ stateful session bean




***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 และเมธอด *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()* เพื่อจัดการเกี่ยวกับการเก็บค่าสถานะ 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 คอนเทนเนอร์ว่า ฟิลด์ใดเป็น persistent ฟิลด์ โดยระบุไว้ใน deployment descriptor เมื่อคอนเทนเนอร์ทราบแล้วว่าฟิลด์ใดบ้างที่ต้อง 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

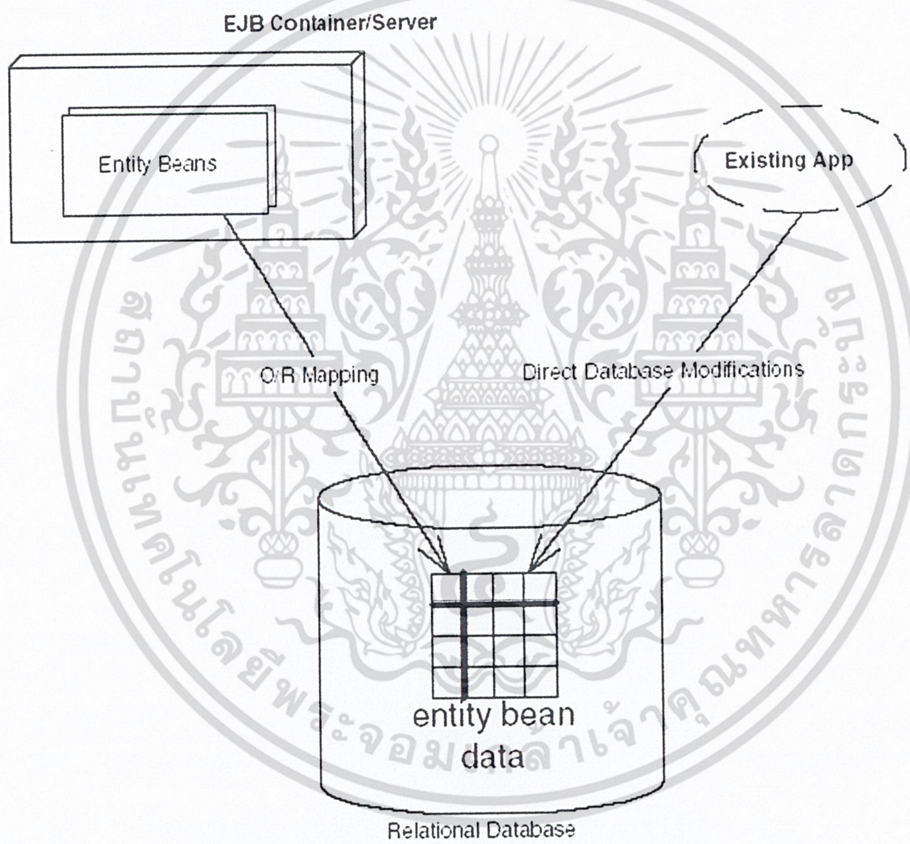
วิธีค้นหา entity bean มีได้หลายวิธีด้วยกัน โดยจะระบุวิธีการเหล่านี้เป็นเมธอดไว้ใน home interface ของ entity bean เรียกว่า finder method นอกเหนือจากเมธอดที่ใช้ในการสร้างและทำลาย entity bean จุดนี้เป็นจุดที่ home interface ของ session bean และ entity bean แตกต่างกัน เพราะใน home interface ของ session bean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

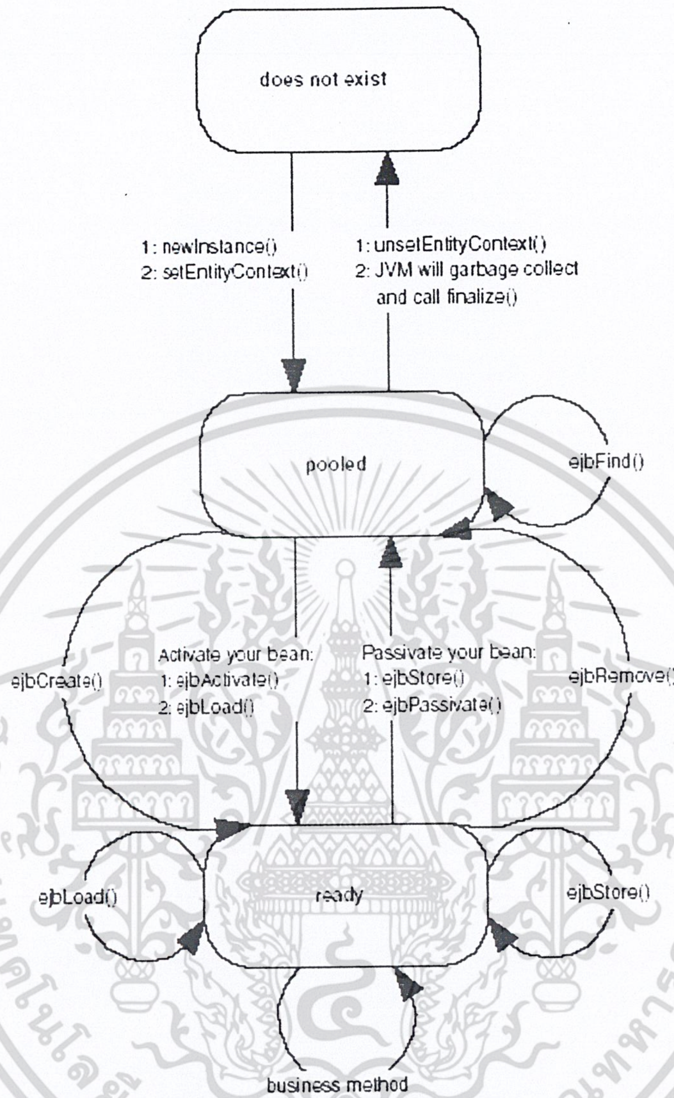
ปกติ การสร้าง , ทำลาย และค้นหาข้อมูลของ entity bean ทำโดยใช้ home interface แต่สามารถติดต่อกับ entity bean ได้อีกวิธีหนึ่ง คือการแก้ไขฐานข้อมูลซึ่งบันทึกข้อมูลนั้นอยู่โดยตรง เช่น ถ้าอินสแทนซ์ของ bean ถูกแมปเข้ากับฐานข้อมูลเชิงสัมพันธ์ จึงสามารถลบแถวออกจากฐานข้อมูลซึ่งสัมพันธ์กับอินสแทนซ์ของ entity bean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




*รูปที่ 6-6 การแก้ไขฐานข้อมูลที่สัมพันธ์กับ Entity Bean โดยตรง*

■ **วงจรชีวิตของ Entity Bean**

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




***รูปที่ 6-7 วงจรชีวิตของ Entity Bean***

**6.1.5 การเขียน 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) ...
```

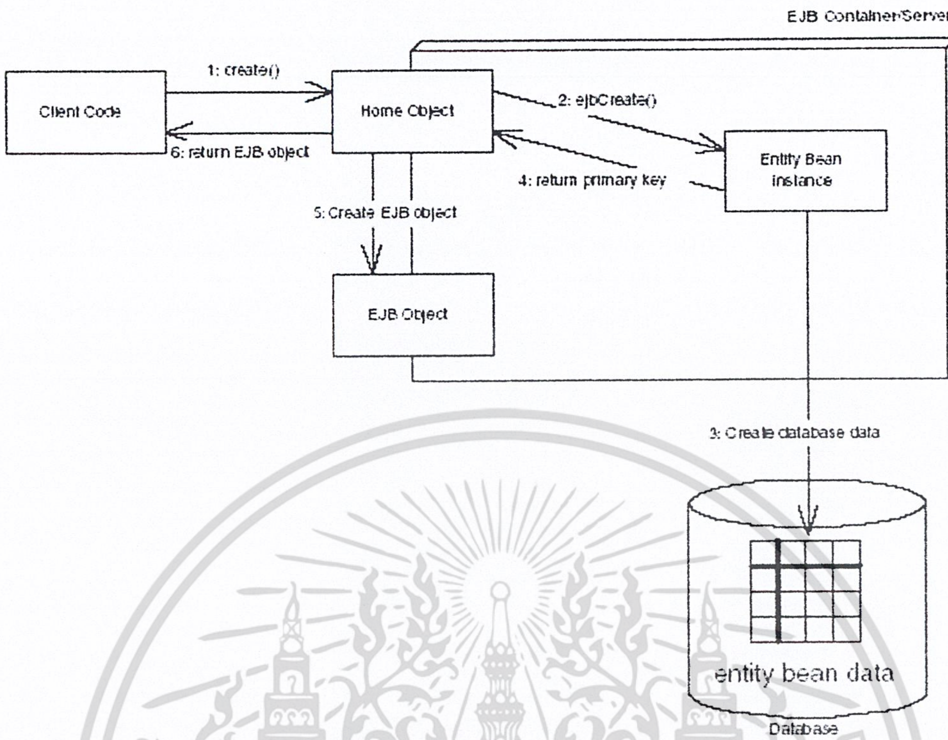
จะต้องมีเมธอด *create()* ใน home interface ดังนี้

```
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) …
```

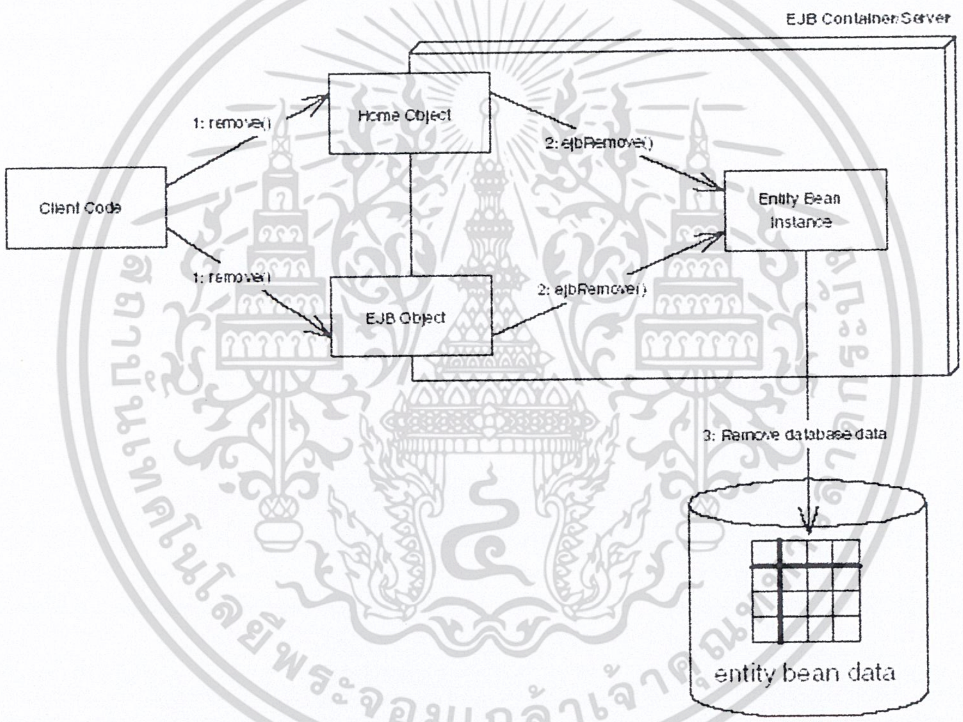
แล้วจะต้องมีเมธอดใน home interface ดังนี้

```
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*

### 6.2 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()* คือต้องมีพารามิเตอร์เหมือนกัน 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***

**6.2.1 ตัวอย่างโค้ด Bean-Managed Persistent Entity Bean**

ตัวอย่าง bean นี้คือ bean ที่เก็บข้อมูลของบัญชีธนาคาร สคริปต์ที่ใช้สร้างเทเบิลดังนี้

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

- **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;
}
```

- **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.CreateExeption* ออกมาได้ และเมธอด *find()* จะกำหนดให้มีการโยน *javax.ejb.FinderException* ออกมาได้

- **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;
  }
}
```

- **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;
  }
```

- **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);
  }
}
```

- **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>
```

- **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

- **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

### 6.3 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 โดยเมธอดนี้จะไม่ถูกเรียกระหว่างทรานแซ็กชัน |
| ejbLoad() | คอนเทนเนอร์จะเรียกเมธอดนี้เพื่อโหลดข้อมูลจากฐานข้อมูลไปยังอินสแทนซ์ของ bean ไม่ต้องอิมพลีเมนต์เมธอดนี้ |
| ejbStore() | คอนเทนเนอร์จะเรียกเมธอดนี้เพื่ออัพเดตข้อมูลจากอินสแทนซ์ของ bean ลงฐานข้อมูล ไม่ต้องอิมพลีเมนต์เมธอดนี้ |
| ejbPassivate() (เหมือนกับ bean-managed persistent) | คอนเทนเนอร์จะเรียกเมธอดนี้เมื่อนำ entity bean เก็บลงในพูล |
| ejbRemove() | จะลบข้อมูลในฐานข้อมูล ไม่ต้องอิมพลีเมนต์เมธอดนี้ |
| unsetEntityContext() (เหมือนกับ bean-managed persistent) | เมธอดนี้ใช้ในการลบ bean ออกจากเอ็นไวรอนเมนต์ คอนเทนเนอร์จะเรียกเมธอดนี้ก่อนที่อินสแทนซ์ของ entity bean จะถูกทำลาย |

***ตารางที่ 6-2 คำอธิบายเมธอดใน Container-Managed Persistent Entity Bean (ต่อ)***

### 6.3.1 ตัวอย่างโค้ด Container-Managed Persistent Entity Bean

ตัวอย่างคือ bean ที่เก็บข้อมูลผลิตภัณฑ์ สคริปต์ที่ใช้สร้างเทเบิลดังนี้

create table products (productId varchar(64), name varchar(64), basePrice numeric(18), description varchar(64));

- **Remote Interface**

  โค้ดดังนี้

**Product.java**

```
package olala;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Product extends EJBObject
{
  public String getProductId()
    throws RemoteException;
  public String getName()
    throws RemoteException;
  public void setName(String name)
    throws RemoteException;
  public double getBasePrice()
    throws RemoteException;
  public void setBasePrice(double basePrice)
    throws RemoteException;
  public String getDescription()
    throws RemoteException;
  public void setDescription(String description)
    throws RemoteException;
}
```

- **Home Interface**

  โค้ดดังนี้

**ProductHome.java**

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

```java
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;
}
```

- **Enterprise Bean Class**

  container-managed persistent ช่วยลดการโค้ดติดต่อด้วย JDBC เป็นจำนวนมาก โค้ดดังนี้

**ProductEJB.java**

```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)
{ }
}
```

- **Deployment Descriptor**

container-managed persistent จะต้องมีไฟล์ descriptor เพิ่มจากเดิม โดยไฟล์นั้นจะเป็นไฟล์ที่ใช้แมป ข้อมูลจากออบเจ็กต์ไปเป็นข้อมูลในฐานข้อมูล 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

- **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

- 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","P5-350","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..
```

**6.4 เปรียบเทียบ Bean-Managed Persistent กับ Container-Managed Persistent Entity Bean**

การเลือกระหว่าง entity bean 2 ชนิดนี้ขึ้นอยู่กับงานที่จะทำ เนื่องจากทั้ง 2 ชนิดนี้มีข้อดีและข้อเสียต่างกัน โดยจะเปรียบเทียบในหัวข้อต่าง ๆ ดังนี้

- **จำนวนโค้ดและเวลาในการพัฒนา**

container-managed persistent มีจำนวนโค้ดน้อยกว่าและพัฒนาได้เร็วกว่า เนื่องจากไม่ต้องเขียนโค้ดแมปข้อมูลด้วย JDBC

- **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

- **การควบคุม**

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

- **ความเป็นอิสระจากแอพพลิเคชันเซิร์ฟเวอร์และฐานข้อมูล**

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

- **การเรียนรู้และค่าใช้จ่าย**

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

# บทที่ 7
# ทรานแซ็กชัน

## 7.1 เหตุผลที่ต้องใช้ทรานแซ็กชัน

ตัวอย่างปัญหาที่ต้องใช้ทรานแซ็กชันในการแก้

- **Atomic Operation**

กรณีที่ต้องการทำคำสั่งหลาย ๆ คำสั่ง โดยมองทั้งหมดเป็นเหมือนคำสั่งเดียว (atomic operation) เช่น กรณีบัญชีธนาคาร เมื่อต้องการโอนเงินจากบัญชีหนึ่งไปยังอีกบัญชีหนึ่ง ต้องถอนเงินออกจากบัญชีหนึ่ง จากนั้นจึงฝากเงินเข้าอีกบัญชีหนึ่ง ถ้ามีคำสั่งใดคำสั่งหนึ่งล้มเหลว ต้องให้คำสั่งทั้งคู่ล้มเหลว ไม่เช่นนั้นยอดเงินคงเหลือในบัญชีจะไม่ถูกต้อง จึงต้องไม่เกิดสถานการณ์ที่คำสั่งหนึ่งทำงานสำเร็จ ในขณะที่อีกคำสั่งทำงานล้มเหลว เนื่องจากทั้งสองคำสั่งนั้นเป็นทรานแซ็กชันเดียวกัน

การแก้ปัญหานี้วิธีหนึ่งคือการใช้ exception ดังตัวอย่างต่อไปนี้

```
try {
// Withdraw funds from account 1
}
catch (Exception e) {
// If an error occurred, do not proceed.
return;
}
try {
// Otherwise, deposit funds into account 2
}
catch (Exception e) {
// If an error occurred, do not proceed,
// and redeposit the funds back into account 1.
return;
}
```

ตัวอย่างโค้ดข้างต้น ถอนเงินจากบัญชีที่ 1 ก่อน หากเกิดข้อผิดพลาดขึ้น แอพพลิเคชันจะจบการทำงาน และไม่มีผลของการทำงานใด ๆ เกิดขึ้น หากไม่เกิดข้อผิดพลาดก็จะฝากเงินเข้าไปยังบัญชีที่ 2 ในขณะฝากเงินนั้น หากเกิดข้อผิดพลาด จะต้องทำการฝากเงินกลับเข้าไปในบัญชีที่ 1 ใหม่ จากนั้นจึงจบการทำงาน

ปัญหาที่เกิดขึ้นจากการแก้ปัญหาด้วยวิธีนี้คือ

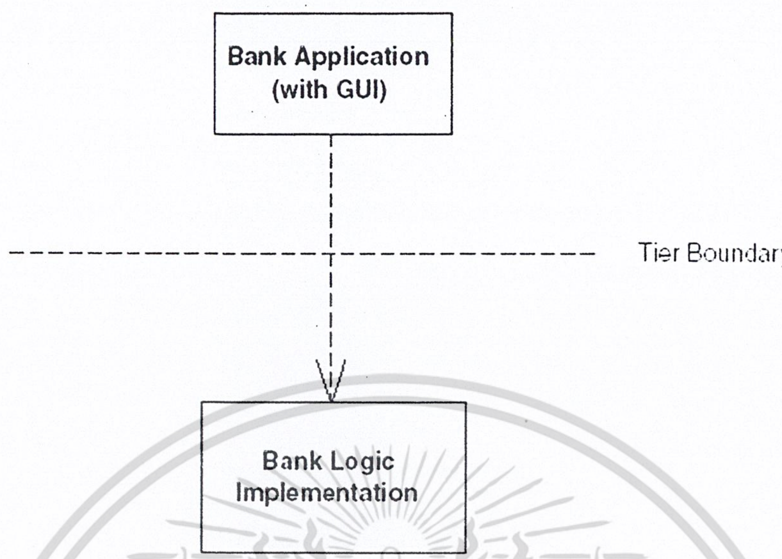
- โปรแกรมมีความซับซ้อน
- ต้องตรวจสอบทุกความเป็นไปได้ที่จะทำให้เกิดการทำงานผิดพลาดในทุก ๆ ขั้นตอน และต้องเขียนโค้ดเพื่อยกเลิกการเปลี่ยนแปลงที่ได้ทำไปก่อนที่จะเกิดความผิดพลาดนั้นเอง
- ในการตรวจสอบความผิดพลาด แทบจะไม่สามารถควบคุมได้หากกระบวนการทั้งหมดมีความซับซ้อน เช่นการปรับปรุงข้อมูลทางการเงิน ที่ต้องอาศัยการทำงานทั้งหมด 10 ขั้นตอนด้วยกัน จะต้องหาวิธีการในการรองรับการทำงานที่ผิดพลาดในทุกขั้นตอนที่เป็นไปได้
- การตรวจสอบการทำงานได้ยาก เพราะต้องทำการจำลองการทำงานที่ผิดพลาดที่เกิดขึ้นได้ในทุกขั้นตอน

ในทางอุดมคตินั้นต้องการวิธีการที่สามารถทำคำสั่งทั้งสองได้ โดยมองเป็นคำสั่งใหญ่เพียงคำสั่งเดียว และรับประกันว่าคำสั่งทั้งสองจะต้องทำงานสำเร็จทั้งคู่ หรือทำงานผิดพลาดทั้งคู่เท่านั้น

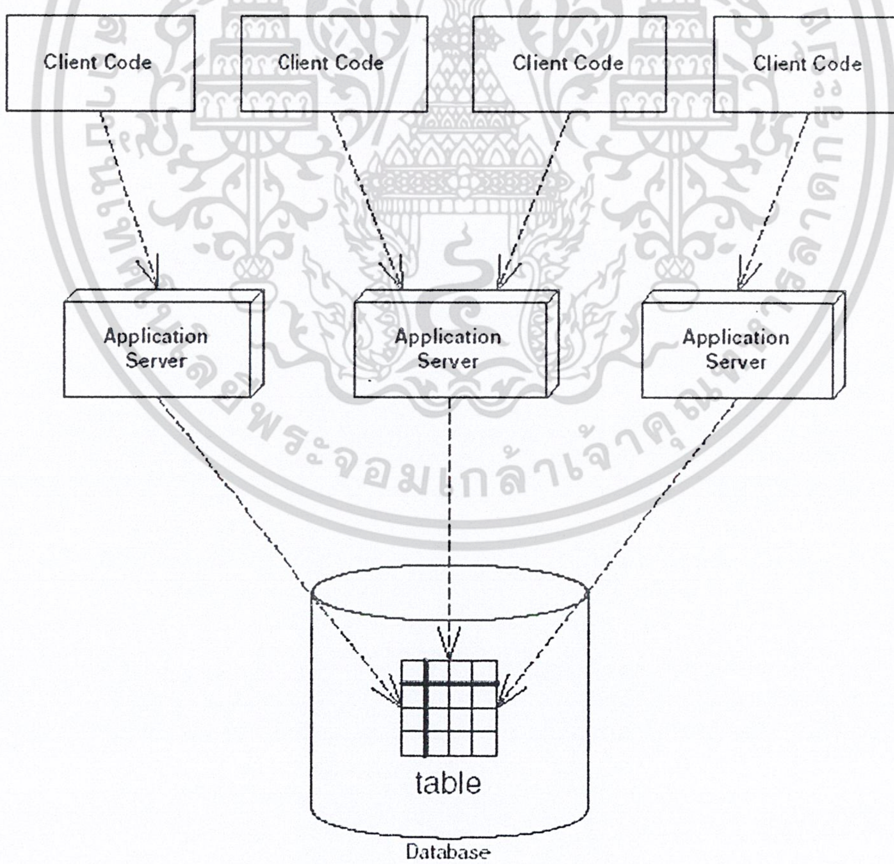
- **การทำงานผิดพลาดของเครื่องคอมพิวเตอร์ หรือระบบเครือข่าย**

จากตัวอย่างข้างต้น ถ้าลอจิกต่าง ๆ ที่ทำงานกับบัญชีธนาคารถูกกระจายบนเครือข่ายโดยดีพลอยแบบมัลติเทียร์ เพื่อเหตุผลทางด้านความปลอดภัย , การเพิ่มขยาย หรือประโยชน์จากการแบ่งโปรแกรมออกเป็นโมดูล ในการดีพลอยแบบมัลติเทียร์ ไคลเอ็นต์ที่ต้องการใช้งานแอพพลิเคชันบัญชีธนาคารนี้ จะต้องกระทำผ่านทาง remote method invocation แสดงในรูปที่ 7-1 ทำให้ต้องคำนึงถึงเรื่องของการทำงานผิดพลาดและความน่าเชื่อถือ เช่น ระบบเครือข่ายแครชระหว่างที่โปรแกรมบัญชีธนาคารกำลังทำคำสั่งใด ๆ อยู่ แม้ว่าเมื่อระบบเครือข่ายแครชจะมีการสร้าง exception ขึ้นมา แต่การแครชของระบบเครือข่ายนั้นอาจจะเกิดขึ้นก่อนหน้าการถอนเงิน หรืออาจเกิดขึ้นภายหลังการถอนเงินก็ได้ ซึ่งไม่อาจแยกความแตกต่างระหว่าง exception ที่เกิดขึ้นจากกรณีทั้งสองนี้ได้เลย

ในความเป็นจริง ระบบเครือข่ายไม่ได้เป็นจุดเดียวที่ทำให้เกิดปัญหานี้ขึ้น เนื่องจากข้อมูลบัญชีธนาคารก็คือข้อมูลแบบ persistent ที่อยู่ในฐานข้อมูล มีความเป็นไปได้เสมอที่ระบบฐานข้อมูลจะแครช หรือเครื่องคอมพิวเตอร์ที่ดีพลอยระบบฐานข้อมูลไว้จะเกิดแครช ถ้าหากการแครชนั้นเกิดขึ้นในระหว่างที่กำลังเขียนข้อมูลลงไปในฐานข้อมูล อาจทำให้ฐานข้อมูลอยู่ในสถานะที่ไม่ consistent ได้




*รูปที่ 7-1 แอพพลิเคชันที่ทำงานในระบบแบบกระจาย*




*รูปที่ 7-2 การใช้งานฐานข้อมูลพร้อม ๆ กัน*

- **การที่ผู้ใช้หลายคนสามารถใช้งานข้อมูลเดียวกันพร้อมกันได้**

ในระบบแบบ enterprise-level distributed system เป็นรูปแบบที่ไคลเอ็นต์หลาย ๆ ตัวติดต่อกับแอพพลิเคชันเซิร์ฟเวอร์หลาย ๆ ตัว ถ้าแอพพลิเคชันเซิร์ฟเวอร์ทั้งหมดใช้ฐานข้อมูลเดียวกัน (ดังรูปที่ 7-2) เซิร์ฟเวอร์สามารถที่จะแก้ไขข้อมูลตัวเดียวกันที่อยู่ในฐานข้อมูลได้

ถ้ามีไคลเอ็นต์สองตัวพยายามแก้ไขข้อมูลเดียวกันที่อยู่ในฐานข้อมูลในเวลาเดียวกันแล้ว คำสั่งของไคลเอ็นต์ทั้งสองอาจทำงานซ้อนกัน และได้ผลลัพธ์สุดท้ายผิดพลาด ซึ่งการมีข้อมูลที่ไม่ถูกต้องในฐานข้อมูลในระบบทั่วไปถือว่าเป็นสิ่งที่ยอมรับไม่ได้ (unacceptable) และอาจสร้างความเสียหายอย่างมากให้กับองค์กร ดังนั้นจึงจำเป็นต้องหาวิธีการที่จะจัดการกับการแก้ไขข้อมูลพร้อมกันของไคลเอ็นต์หลาย ๆ ตัว โดยจะต้องไม่ทำให้ผลลัพธ์สุดท้ายที่ได้นั้นเป็นผลลัพธ์ที่ไม่ถูกต้อง

## 7.2 ประโยชน์ของทรานแซ็กชัน

ทรานแซ็กชัน คือเซ็ตของคำสั่งที่ถูกเอ็กซิคิวต์โดยมองว่าเป็นคำสั่งใหญ่เพียงอันเดียว ทรานแซ็กชันรับประกันว่าคำสั่งทั้งหมดทำงานสำเร็จ มิฉะนั้นจะต้องไม่มีผลการทำงานจากคำสั่งใดเกิดขึ้นเลย ทรานแซ็กชันสามารถแก้ปัญหาที่เกิดจากการทำงานผิดพลาดของระบบเครือข่ายได้ อนุญาตให้ไคลเอ็นต์หลาย ๆ ตัวใช้งานข้อมูลเดียวกันพร้อมกันได้ และรับประกันว่าข้อมูลทั้งหมดที่คำสั่งเหล่านี้ทำการแก้ไขจะถูกต้องสมบูรณ์โดยไม่ถูกรบกวนจากการแก้ไขข้อมูลโดยไคลเอ็นต์อื่น ๆ

การใช้ทรานแซ็กชันอย่างถูกต้องทำให้สามารถมีหลายไคลเอ็นต์ทำงานกับฐานข้อมูลได้โดยไม่ขึ้นต่อกัน เช่น ถ้ามีไคลเอ็นต์สองตัวอ่านและเขียนข้อมูลลงในฐานข้อมูล ไคลเอ็นต์ทั้งสองจะ mutual exclusive กัน ระบบจัดการฐานข้อมูลจะจัดการเกี่ยวกับการควบคุมการทำงานพร้อมกันที่จำเป็นบนฐานข้อมูล เช่น การล็อคข้อมูล เพื่อไม่ให้การทำงานของแต่ละเธรดมีผลกระทบซึ่งกันและกัน

### 7.2.1 ACID Properties

เมื่อใช้งานทรานแซ็กชันอย่างถูกต้อง คำสั่งจะถูกเอ็กซิคิวต์ โดยมีการรับประกันผลการทำงานทั้งสิ้น 4 อย่างด้วยกัน เรียกว่า ACID properties โดยคำว่า ACID ย่อมาจาก Atomicity , Consistency , Isolation และ Durability

- **Atomicity** - รับประกันว่าคำสั่งทั้งหมดจะถูกรวมเข้าไว้ด้วยกัน และมองเป็นหน่วยของการทำงานเพียงอันเดียว การทำงานของทรานแซ็กชันจะเป็นไปในลักษณะของ All-or-Nothing คือการอัพเดตข้อมูลในฐานข้อมูลทั้งหมดจะต้องเสร็จสิ้น หรือถ้าเกิดข้อผิดพลาดใด ๆ ขึ้นระหว่างการทำงาน การอัพเดตฐานข้อมูลทั้งหมดนั้นจะต้องไม่เกิดขึ้นเลย โดยการอัพเดตฐานข้อมูลที่เกิดขึ้นก่อนแล้ว จะต้องถูกยกเลิก
- **Consistency** - รับประกันว่าทรานแซ็กชันจะต้องทำให้ระบบอยู่ในสถานะที่ consistent เมื่อทรานแซ็กชันทำงานเสร็จเรียบร้อย เช่น การโอนเงินจะต้องไม่มีเงินหายไปจากระบบ แต่ในระหว่างการ

ทำงานของทรานแซ็กชันนี้ ขณะที่เงินถูกถอนออกมาจากบัญชีแรก ข้อมูลอยู่ในสถานะที่ไม่ consistent แต่เนื่องจากทรานแซ็กชันจะมองเห็นคำสั่งทั้งหมดเปรียบเสมือนเป็นคำสั่งเดียว ดังนั้นถ้าทรานแซ็กชันสำเร็จก็คือคำสั่งทั้งหมดจะต้องสำเร็จ แต่ถ้าทรานแซ็กชันทำงานไม่สำเร็จคำสั่งทั้งหมดก็จะต้องไม่สำเร็จ และการเปลี่ยนแปลงใด ๆ ที่ได้กระทำไประหว่างนั้นจะต้องถูกยกเลิก ตามคุณสมบัติข้อแรกของทรานแซ็กชัน ดังนั้นข้อมูลจะอยู่ในสถานะที่ consistent เสมอ

- **Isolation** - เป็นการป้องกันทรานแซ็กชันที่ทำงานพร้อมกัน ไม่ให้เห็นผลลัพธ์ที่ยังไม่สมบูรณ์ที่เกิดจากทรานแซ็กชันอื่น isolation อนุญาตให้หลายทรานแซ็กชันสามารถอ่านหรือเขียนฐานข้อมูลโดยไม่รู้ถึงการมีอยู่ของทรานแซ็กชันอื่น ถ้าไม่มีคุณสมบัติ isolation สถานะของข้อมูลอาจจะอยู่ในสถานะที่ไม่ consistent ได้ ซึ่งมีประโยชน์มากในกรณีไคลเอ็นต์หลายตัวแก้ไขฐานข้อมูลในเวลาเดียวกัน โดยแต่ละไคลเอ็นต์จะทำงานโดยเปรียบเสมือนว่ามีเพียงตัวมันเองเท่านั้นที่กำลังใช้งานฐานข้อมูลอยู่ ทรานแซ็กชันสร้างคุณสมบัติ isolation โดยการใช้ synchronization โพรโตคอลต่าง ๆ กับข้อมูลในฐานข้อมูล เช่น ระหว่างการทำงานของทรานแซ็กชัน ข้อมูลจะถูกล็อคและปลดล็อคโดยอัตโนมัติตามความจำเป็น และการอัพเดตข้อมูลของทรานแซ็กชันอื่น จะไม่มีผลกับข้อมูลที่ทรานแซ็กชันนี้กำลังทำงานอยู่
- **Durability** - รับประกันว่าการเปลี่ยนแปลงข้อมูลใด ๆ ในฐานข้อมูลที่เสร็จสิ้น (commit) แล้ว จะต้องมีความถาวร คือต้องไม่สูญหายเมื่อเกิดความผิดพลาดใด ๆ ขึ้น เช่น เครื่องคอมพิวเตอร์แครช , ระบบเครือข่ายแครช , ฮาร์ดดิสก์แครช หรือไฟฟ้าดับ โดยต้องมีการเก็บ log file ของการอัพเดตข้อมูลใด ๆ ในทรานแซ็กชัน เมื่อระบบเกิดแครชขึ้น ข้อมูลสามารถสร้างขึ้นมาใหม่โดยการทำงานตามลำดับใน log file อีกครั้งหนึ่ง

### 7.3 ทรานแซ็กชันโมเดล

ทรานแซ็กชันโมเดลเป็นวิธีการที่ต่างกันในการทำทรานแซ็กชัน มีหลายโมเดลที่ใช้จัดการทรานแซ็กชัน แต่ละโมเดลจะมีความซับซ้อนและฟีเจอร์ที่ต่างกัน 2 โมเดลที่ได้รับความนิยมมากที่สุดคือ flat transaction และ nested transaction

- **Flat Transactions**

flat transaction เป็นทรานแซ็กชันโมเดลที่เข้าใจได้ง่าย flat transaction เป็นกลุ่มของ operation ที่ทำงานเป็นหน่วยเดียว ถ้า operation ไม่สำเร็จทั้งหมดจะเกิด rollback ขึ้น แต่ถ้าสำเร็จจะเกิด commit กระบวนการของ flat transaction ดังรูปที่ 7-3

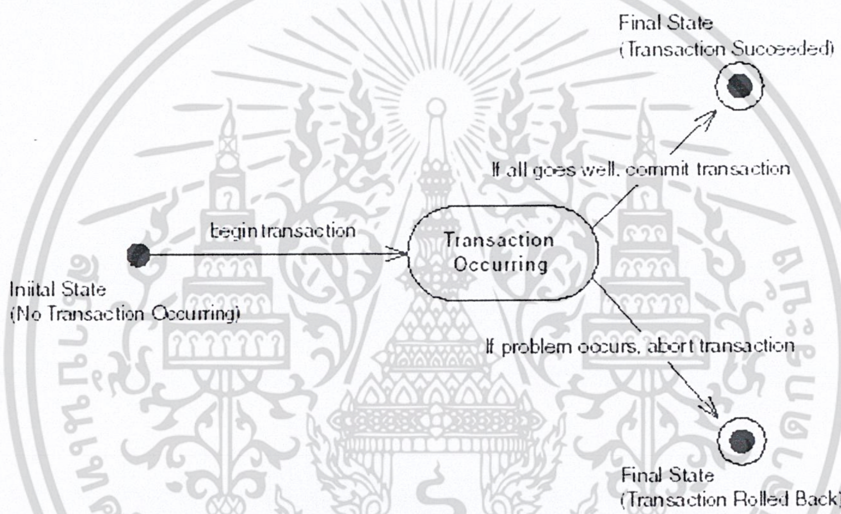
- **Nested Transactions**

ตัวอย่างของงานที่ต้องใช้ nested transaction คือปัญหา trip-planning โดยแอพพลิเคชันจะมี operation ดังนี้

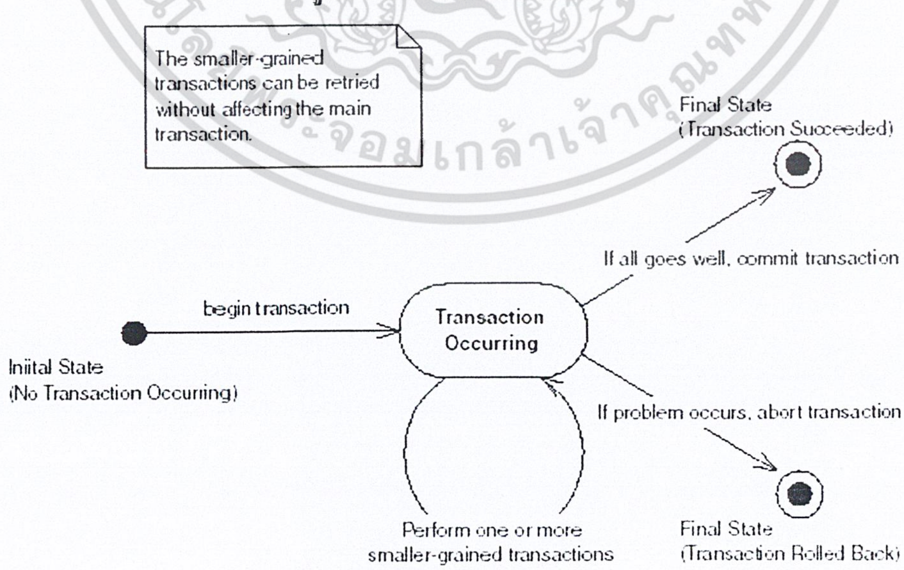
1. ซื้อตั๋วรถไฟจากบอสตันไปยังนิวยอร์ค
2. ซื้อตั๋วเครื่องบินจากนิวยอร์คไปยังลอนดอน
3. ซื้อตั๋วบอลลูนจากลอนดอนไปยังปารีส
4. แอพพลิเคชันพบว่าไม่มีไฟลท์ไปยังปารีส

ถ้าลำดับของ operation นี้ทำกับ flat transaction จะเกิด rollback เท่านั้น เนื่องจากไม่มีไฟลท์ไปยังปารีส แอพพลิเคชันจะต้องไม่เก็บการจองใด ๆ ไว้เลย แต่ตรงจุดนี้อาจมีทางในการเปลี่ยนการเดินทางด้วยบอลลูนไปเป็นทางอื่น ทำให้เก็บข้อมูลการจองไว้ได้

nested transaction ช่วยในการแก้ปัญหานี้ มันจะยอมให้เกิดทรานแซ็กชันซ้อนกันได้ รูปที่ 7-4 แสดงแนวคิดของ nested transaction




***รูปที่ 7-3 Flat Transaction***




***รูปที่ 7-4 Nested Transaction***

- **โมเดลอื่น ๆ**

ทรานแซ็กชันโมเดลยังมี chained transaction และ sagas แต่จะไม่อธิบายถึง เพราะ EJB specification ไม่รองรับโมเดลเหล่านั้น รวมถึง nested transaction ด้วย มันจะรองรับเฉพาะ flat transaction เท่านั้น

## 7.4 การใช้งานทรานแซ็กชันกับ EJB

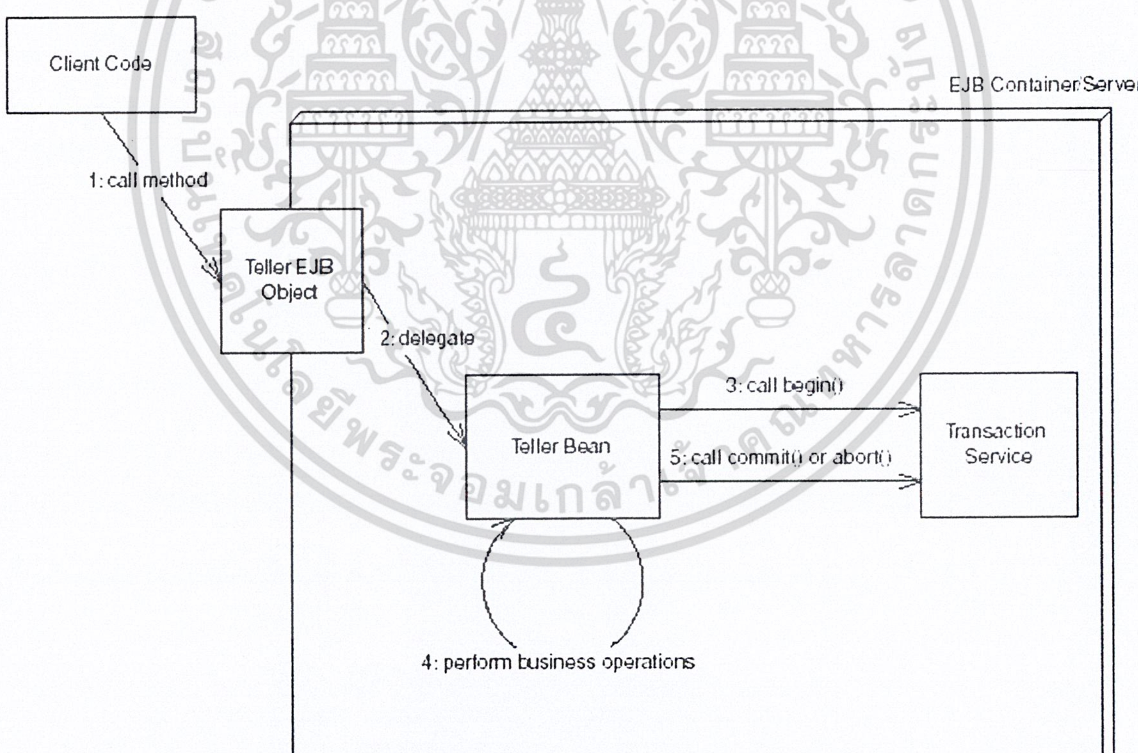
### 7.4.1 วิธีการกำหนดการจัดการทรานแซ็กชัน

ทรานแซ็กชันใด ๆ ผลการทำงานของทรานแซ็กชันจะจบลงด้วยการ commit หรือ abort อย่างใดอย่างหนึ่งเท่านั้น โดยวิธีการกำหนดขอบเขตของทรานแซ็กชันสำหรับ EJB มี 2 วิธี

**1) Programmatic Transaction**

ผู้พัฒนาแอพพลิเคชันเป็นผู้เขียนโปรแกรมลอจิกเกี่ยวกับการควบคุมทรานแซ็กชันลงไปในโค้ดของแอพพลิเคชันเอง ต้องเป็นผู้สั่งคำสั่ง begin , commit หรือ abort เอง

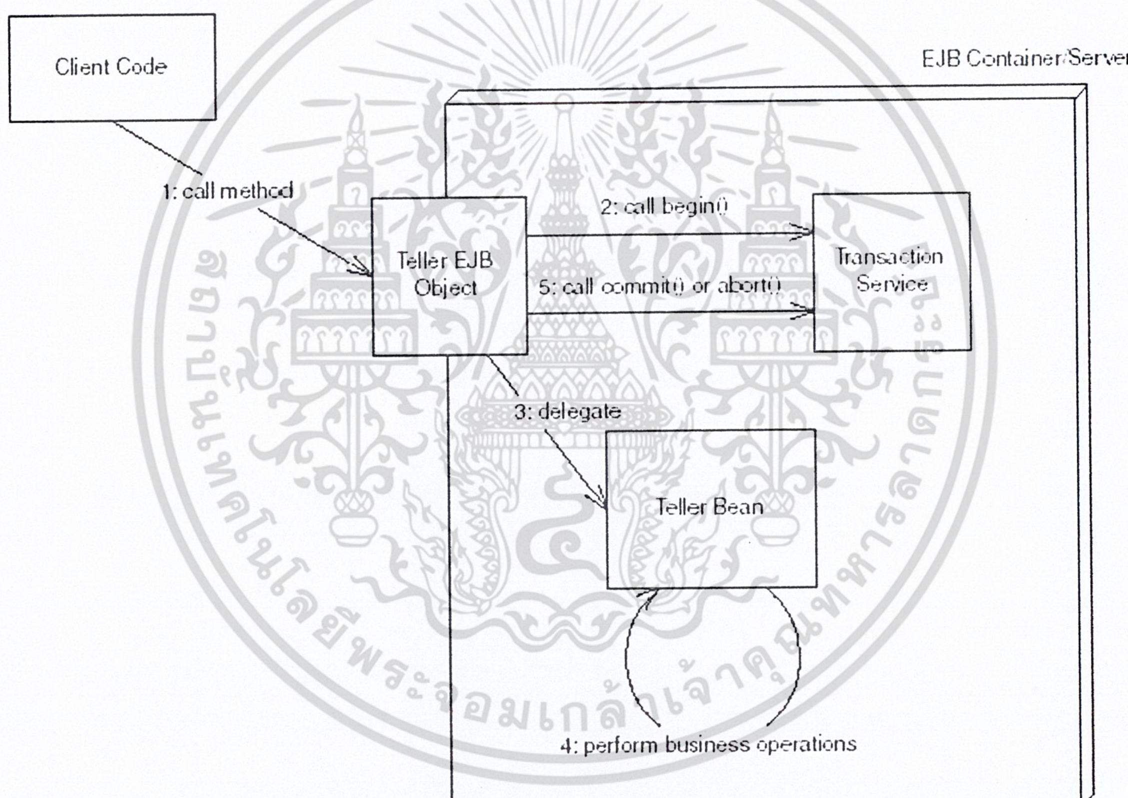
การทำงานของ programmatic transaction แสดงดังรูปที่ 7-5




*รูปที่ 7-5 bean ที่ทำงานกับ Programmatic Transaction*

**2) Declarative Transaction**

declarative transaction อนุญาตให้คอมโพเน็นต์ทำงานในรูปแบบทรานแซ็กชันโดยอัตโนมัติ enterprise bean ไม่จำเป็นต้องสั่ง begin , commit หรือ abort เอง คอนเทนเนอร์จัดการให้อัตโนมัติ EJB คอนเทนเนอร์จะดักจับการร้องขอที่มีเข้ามาและเริ่มต้นทรานแซ็กชันอัตโนมัติโดยการสั่ง begin หลังจากนั้นคอนเทนเนอร์ก็จะส่งต่อการร้องขอไปยัง enterprise bean ซึ่งจะทำงานอยู่ในทรานแซ็กชันที่เกิดขึ้น ถ้าหากมีข้อผิดพลาดใด ๆ bean จะส่งไปบอกคอนเทนเนอร์ให้ยกเลิกทรานแซ็กชัน เมื่อการทำงานเสร็จสิ้น bean จะส่งการควบคุมกลับมาให้คอนเทนเนอร์ คอนเทนเนอร์จะสั่ง commit หรือ abort โดยขึ้นอยู่กับว่ามีข้อผิดพลาดเกิดขึ้นระหว่างการทำงานหรือไม่ การทำงานของ declarative transaction แสดงในรูปที่ 7-6




*รูปที่ 7-6 bean ที่ทำงานใน Declarative Transaction*

declarative transaction ช่วยให้ bean ไม่จำเป็นต้องติดต่อกับ API ของทรานแซ็กชันโดยตรง โดยที่โค้ดของ bean และไคลเอ็นต์จะไม่รู้เลยว่ามันทำงานอยู่ภายใต้ทรานแซ็กชัน

การระบุว่าต้องการทำงานในแบบ declarative หรือ programmatic transaction นั้น EJB กำหนดให้ระบุไว้ในแท็ก persistence-type ที่อยู่ใน ejb-jar.xml ถ้ามีค่าเท่ากับ Bean จะเป็นการระบุให้ bean ควบคุมเอง แต่ถ้าเป็น Container จะให้คอนเทนเนอร์เป็นผู้ควบคุมจะมีค่าดังนี้

```
<persistence-type>Bean</persistence-type>
<persistence-type>Container</persistence-type>
```

### 7.4.2 การจัดการทรานแซ็กชันของคอนเทนเนอร์

transaction attribute เป็นค่าที่ระบุว่าคอนเทนเนอร์จะควบคุมทรานแซ็กชันของ bean อย่างไร โดยสามารถระบุ transaction attribute ที่แตกต่างกันสำหรับแต่ละ bean ในระบบได้โดยไม่ต้องคำนึงถึงว่าจะมี bean ทั้งหมดกี่ตัวที่ทำงานอยู่ร่วมกัน

transaction attribute เป็นส่วนหนึ่งของ deployment descriptor ของแต่ละ bean deployment descriptor ที่มากับ bean จะระบุคุณสมบัติต่าง ๆ ในการทำงานของ bean คอนเทนเนอร์จะจัดการกับทรานแซ็กชันของ bean ตามที่ได้ดูจาก transaction attribute ใน deployment descriptor และสามารถกำหนด transaction attribute สำหรับทุก ๆ เมธอดของ bean หรือจะระบุแตกต่างกันไปในแต่ละเมธอดก็ได้ โดยถ้ามีการระบุค่าทั้งสองแบบ คอนเทนเนอร์จะยึดถือค่าที่ระบุให้แต่ละเมธอดเป็นหลัก ตัวอย่างของการกำหนด transaction attribute ใน ejb-jar.xml ดังนี้

```
<assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>Product</ejb-name>
            <method-name>*</method-name>
        </method>
        <!—
        Transaction attribute. Can be “NotSupported” , “Supports” , “Required” ,
        “RequiresNew” , “Mandatory” , or “Never”
        -->
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
</assembly-descriptor>
```

ค่า transaction attribute ที่เป็นไปได้ มีดังนี้

■ **Required**

จะใช้งานโหมด Required เมื่อต้องการให้ bean ทำงานในทรานแซ็กชันเสมอ ถ้ามีทรานแซ็กชันกำลังทำงานอยู่แล้ว bean จะทำงานร่วมกับทรานแซ็กชันเดิม แต่ถ้าไม่มีทรานแซ็กชันกำลังทำงานอยู่ bean จะเริ่มต้นการทำงานในทรานแซ็กชันใหม่

■ **RequiresNew**

จะใช้ค่านี้เมื่อต้องการให้มีการเริ่มต้นทรานแซ็กชันใหม่ทุกครั้งเมื่อ bean ถูกเรียกใช้งาน ถ้ามีทรานแซ็กชันเดิมกำลังทำงานอยู่ก่อนแล้ว ทรานแซ็กชันนั้นจะถูกหยุดไว้ชั่วคราว คอนเทนเนอร์จะเริ่มต้นการทำงานของทรานแซ็กชันใหม่และส่งต่อการร้องขอมายัง bean ใหม่ที่ถูกเรียกใช้งานนี้ เมื่อ bean ทำงานเสร็จสิ้น คอนเทนเนอร์ก็จะสั่ง commit หรือ abort ทรานแซ็กชันนี้ และกลับไปทำงานต่อในทรานแซ็กชันเก่า

■ **Supports**

เมื่อ bean ถูกเรียกใช้งานด้วย Supports มันจะทำงานในทรานแซ็กชันเมื่อไคลเอ็นต์นั้นทำงานอยู่ในทรานแซ็กชันก่อนแล้ว โดยทำงานในทรานแซ็กชันเดิม แต่ถ้าไคลเอ็นต์ไม่ได้ทำงานในทรานแซ็กชัน bean ก็จะไม่เริ่มต้นการทำงานในทรานแซ็กชันใหม่

■ **Mandatory**

ระบุว่าต้องมีทรานแซ็กชันกำลังทำงานอยู่แล้วเมื่อเมธอดของ bean ถูกเรียกใช้งาน ถ้าหากไม่มีทรานแซ็กชันทำงานอยู่ก่อน exception *javax.ejb.TransactionRequired* จะถูกส่งกลับไปยังผู้เรียก

■ **NotSupported**

bean จะไม่สามารถทำงานในทรานแซ็กชันได้ ยกตัวอย่างเช่น สมมติว่าเรามี enterprise bean A และ B เมื่อ A เริ่มต้นทรานแซ็กชัน และเรียกใช้งาน B หาก B มีการตั้งค่า transaction attribute เป็น NotSupported แล้ว ทรานแซ็กชันที่ A เริ่มต้นไว้จะหยุดชั่วคราว คำสั่งของ B ทั้งหมดจะทำงานโดยไม่อยู่ในทรานแซ็กชัน เมื่อ B ทำงานเสร็จเรียบร้อย ทรานแซ็กชันของ A จะกลับมาทำงานต่อ

■ **Never**

ใช้ระบุว่า bean จะต้องไม่มีทรานแซ็กชัน ถ้าไคลเอ็นต์ที่เรียก bean อยู่ในทรานแซ็กชันจะเกิด exception คืนมาให้ไคลเอ็นต์

ตารางที่ 7-1 แสดงผลลัพธ์ที่เกิดจาก transaction attribute

**7.5 Transactional Isolation**

คุณสมบัติ isolation รับประกันว่า ผู้ใช้ที่ทำงานอยู่พร้อม ๆ กันจะไม่รู้สึกและได้รับผลกระทบจากผู้ใช้คนอื่น ๆ แม้ว่าผู้ใช้ทั้งหมดนั้นกำลังทำงานกับข้อมูลเดียวกันในฐานข้อมูลก็ตาม โดยสามารถระบุระดับของ isolation ที่ต้องการได้

| TRANSACTION ATTRIBUTE | CLIENT'S TRANSACTION | BEAN'S TRANSACTION |
|---|---|---|
| **Required** | none | T2 |
| | T1 | T1 |
| **RequiresNew** | none | T2 |
| | T1 | T2 |
| **Supports** | none | none |
| | T1 | T1 |
| **Mandatory** | none | error |
| | T1 | T1 |
| **NotSupported** | none | none |
| | T1 | none |
| **Never** | none | none |
| | T1 | error |

*ตารางที่ 7-1 ผลลัพธ์ที่เกิดจาก Transaction Attribute*

การระบุระดับของ isolation สามารถควบคุมได้ว่าจะให้แต่ละทรานแซ็กชัน isolate จากกันมากน้อยเพียงใด ถ้าตั้งระดับของ isolation ที่สูง ก็มั่นใจได้ว่าแต่ละทรานแซ็กชันนั้น จะถูก isolate ออกจากทรานแซ็กชันอื่น ๆ ทั้งหมด แต่การตั้งระดับของ isolation สูงก็ทำให้ประสิทธิภาพในการทำงานที่ได้ลดลง เนื่องจากในการทำ isolation นั้นใช้วิธีการล็อคค่าในฐานข้อมูลเป็นหลัก แต่ถ้าตั้งระดับของ isolation ไว้ต่ำ ทรานแซ็กชันก็จะไม่ isolate แต่ประสิทธิภาพในการทำงานร่วมกันของหลาย ๆ ทรานแซ็กชันจะดีขึ้น

ดังนั้น ต้องพิจารณาให้รอบคอบว่า ต้องการความเป็น isolation สูงต่ำเพียงใดในการทำงาน การกำหนด transaction isolation มีอยู่ทั้งหมด 4 ระดับด้วยกัน ดังนี้

- **READ UNCOMMITTED**

เป็นระดับการทำงานที่ต่ำที่สุด ไม่มีการรับประกันคุณสมบัติ isolation เลย แต่จะให้ประสิทธิภาพในการทำงานสูงสุด การทำงานในระดับนี้อาจเกิดปัญหา dirty read ขึ้นได้ โดยหากทรานแซ็กชันหนึ่งทำงานพร้อมกันไปกับทรานแซ็กชันอื่น แล้วอีกทรานแซ็กชันนั้นเขียนข้อมูลโดยยังไม่ได้ commit ทรานแซ็กชันเดิมจะอ่านค่านั้นเข้ามาได้ทำให้ได้ค่าที่ผิดพลาดไป โดยไม่ขึ้นกับระดับของ isolation ที่อีกทรานแซ็กชันหนึ่งใช้งานอยู่ นอกจากนี้ยังเกิดปัญหา unrepeatable read และ phantom phenomenon ได้อีกด้วย

- **READ COMMITTED**

การทำงานในระดับนี้คล้ายกับระดับ READ UNCOMMITTED แตกต่างกันที่ในระดับนี้โค้ดจะอ่านค่าข้อมูลที่ commit แล้วเท่านั้น จะไม่อ่านค่าที่ถูกเขียนลงไปยังฐานข้อมูลแล้วแต่ยังไม่ commit ดังนั้นระดับ

ของ isolation ระดับนี้จึงสามารถแก้ปัญหา dirty read ได้ แต่ยังไม่สามารถแก้ปัญหาที่ซับซ้อน เช่นปัญหา unrepeatable read และ phantom phenomenon

- **REPEATABLE READ**

เป็นการรับประกันที่สูงขึ้นไปจาก READ COMMITTED เมื่อใดก็ตามที่อ่านค่าข้อมูลที่ commit แล้วจากฐานข้อมูล สามารถจะอ่านค่าข้อมูลนั้นซ้ำได้และค่าที่ได้จะเป็นค่าที่เหมือนกับค่าที่เราอ่านขึ้นมาได้ครั้งแรก ซึ่งระดับของ isolation ที่ READ COMMITTED หรือต่ำกว่า ทรานแซ็กชันอื่นที่ทำงานอยู่พร้อมกันอาจจะ commit ข้อมูลในระหว่างการอ่านข้อมูลแต่ละครั้งได้ ทำให้ค่าของข้อมูลเดียวกันที่อ่านได้ในแต่ละครั้งไม่ตรงกัน การทำงานในโหมดนี้แก้ปัญหาที่เรียกว่า unrepeatable read

- **SERIALIZABLE**

สามารถแก้ปัญหา phantom phenomenon คือการที่เซตของข้อมูลใหม่ถูกเพิ่มเข้ามาในฐานข้อมูลระหว่างการอ่านข้อมูลของทรานแซ็กชัน รวมถึงแก้ปัญหาอื่น ๆ ที่กล่าวมาก่อนหน้านี้ทั้งหมด รับประกันว่าทุก ๆ ทรานแซ็กชันจะทำงานเป็นลำดับกับทรานแซ็กชันอื่น ๆ และมีคุณสมบัติ ACID properties ของฐานข้อมูลครบถ้วน หมายความว่าแต่ละทรานแซ็กชันจะทำงานแยกกับทรานแซ็กชันอื่น ๆ อย่างแท้จริง

ตารางที่ 7-2 จะแสดงถึงปัญหาที่อาจเกิดในแต่ละระดับของ isolation

| **ISOLATION LEVEL** | **DIRTY READ?** | **UNREPEATABLE READS?** | **PHANTOM READ?** |
|---|---|---|---|
| READ UNCOMMITTED | Yes | Yes | Yes |
| READ COMMITTED | No | Yes | Yes |
| REPEATABLE READ | No | No | Yes |
| SERIALIZABLE | No | No | No |

***ตารางที่ 7-2 ระดับของ Isolation***

การกำหนด isolation ใน EJB ในกรณีของ bean-managed persistent จะกำหนด isolation โดยใช้ resource manager API เช่น JDBC โดยการเรียก *java.sql.Connection.SetTransactionIsolation(...)* ส่วนในกรณีของ container-managed persistent จะไม่สามารถกำหนดระดับของ isolation ใน deployment descriptor ได้ การกำหนดระดับของ isolation จะขึ้นอยู่กับผลิตภัณฑ์ที่ใช้เป็นคอนเทนเนอร์หรือฐานข้อมูล

## 7.6 ทรานแซ็กชันแบบกระจาย

นอกจากทรานแซ็กชันพื้นฐาน คือแอพพลิเคชันเซิร์ฟเวอร์หนึ่งตัวเชื่อมต่อกับฐานข้อมูลหนึ่งตัวแล้ว ยังสามารถที่จะทำ distributed transaction ได้เช่นเดียวกัน โดยขึ้นอยู่กับการรองรับของแอพพลิเคชันเซิร์ฟเวอร์

ซึ่ง distributed transaction ยึดหลักการเดียวกับทรานแซ็กชันธรรมดา โดยที่หากมีคอมโพเน็นต์หนึ่งบนเครื่องคอมพิวเตอร์เครื่องหนึ่ง abort ทรานแซ็กชัน ทรานแซ็กชันทั้งหมดจะต้องถูก abort ด้วยเช่นกัน distributed transaction สามารถทำงานได้กับหลายแอพพลิเคชันเซิร์ฟเวอร์และหลายฐานข้อมูล หมายความว่าคอมโพเน็นต์ที่ดีพลอยในแอพพลิเคชันเซิร์ฟเวอร์หนึ่ง ทำงานในทรานแซ็กชันเดียวกันร่วมกับอีกคอมโพเน็นต์ซึ่งถูกดีพลอยในอีกแอพพลิเคชันเซิร์ฟเวอร์ ในการทำงานบิซิเนสเมธอดหนึ่ง ๆ เครื่องหลายเครื่องอาจจำเป็นต้องทำงานร่วมกัน distributed transaction จึงอนุญาตให้หลายแอพพลิเคชันเซิร์ฟเวอร์ซึ่งอาจเป็นซอฟต์แวร์จากผู้พัฒนาคนละรายกันทำงานร่วมกันในทรานแซ็กชันได้

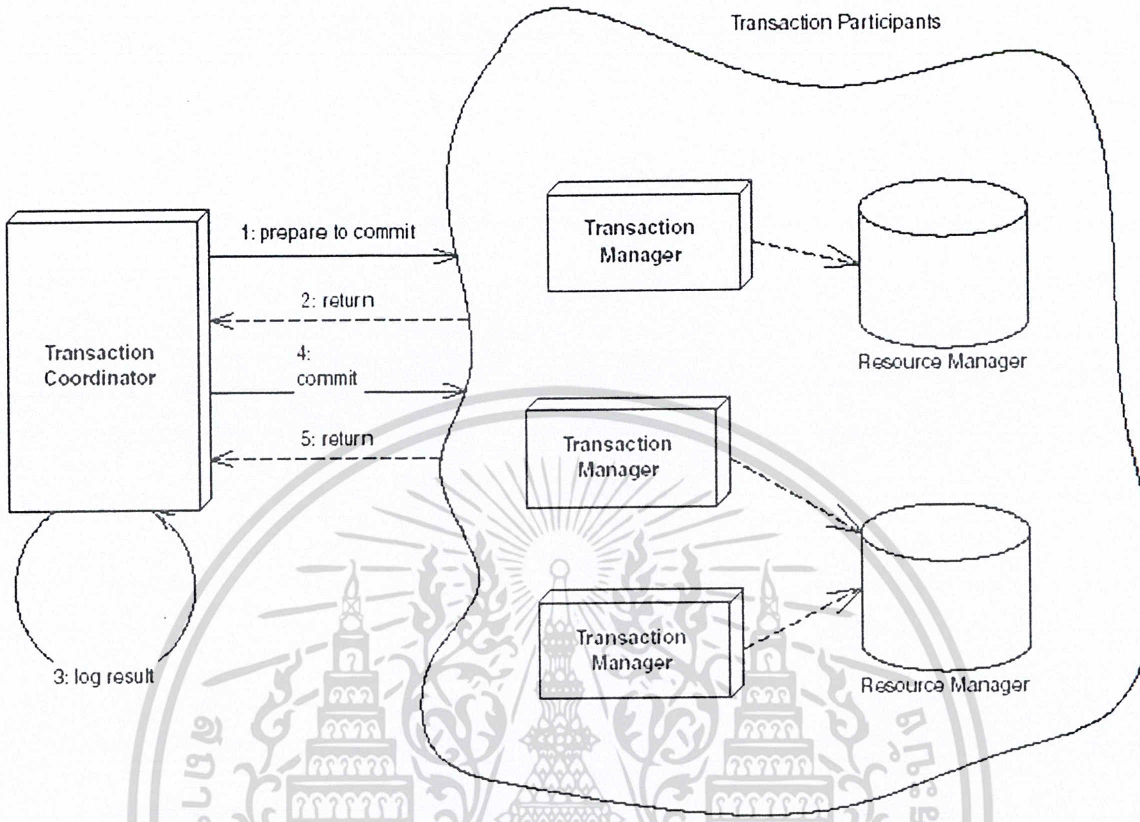
### 7.6.1 Durability และโพรโตคอล Two-Phase Commit

durability รับประกันว่า การอัพเดต resource ที่ commit แล้วทุกครั้งจะต้องมีความถาวร สามารถอิมพลีเมนต์ได้ไม่ยากถ้าทำงานกับฐานข้อมูลเพียงฐานข้อมูลเดียว แต่กรณีที่มีหลาย resource manager เข้ามาเกี่ยวข้องด้วย หากมี resource หนึ่งเกิดความผิดพลาดในการทำงานใด ๆ ขึ้น เช่น ฐานข้อมูลแครชจะต้องมีวิธีที่จะกู้ข้อมูลกลับคืนมาให้ได้

วิธีหนึ่งในการแก้ปัญหานี้คือ การบันทึก log ของคำสั่งที่ทำงานกับฐานข้อมูลทั้งหมดก่อนที่คำสั่งนั้นจะทำงานจริง จะช่วยให้สามารถกู้ข้อมูลคืนมาได้ โดยดูข้อมูลได้จาก log file และทำการอัพเดตข้อมูลต่าง ๆ ใหม่อีกครั้ง ด้วยวิธีการนี้ทรานแซ็กชันจะเสร็จสิ้นในสองเฟส

- **เฟสที่หนึ่ง** - เริ่มต้นโดยส่งข้อความ before commit ไปยังทุก resource ที่เกี่ยวข้องในทรานแซ็กชัน ณ จุดนี้ resource ที่เกี่ยวข้องในทรานแซ็กชันจะมีโอกาสที่จะยกเลิกทรานแซ็กชันเป็นครั้งสุดท้าย ถ้าหากมี resource ใดตัดสินใจยกเลิกทรานแซ็กชัน ทรานแซ็กชันทั้งหมดก็จะถูกยกเลิก และการอัพเดต resource ต่าง ๆ ก็จะไม่เกิดขึ้น มิฉะนั้นทรานแซ็กชันก็จะทำงานต่อไปและไม่มีการหยุดนอกจากจะมีความผิดพลาดเกิดขึ้น เพื่อป้องกันความผิดพลาด การอัพเดต resource ใด ๆ ในแต่ละครั้งจะถูกบันทึกลงใน log หรือ journal ที่มีความคงทน (persistent) โดยข้อมูลเหล่านี้จะไม่สูญหายไปเมื่อมีการแครชเกิดขึ้น และสามารถใช้ข้อมูลเหล่านี้ในการอัพเดต resource ต่าง ๆ ได้อย่างถูกต้อง
- **เฟสที่สอง** - จะเกิดขึ้นก็ต่อเมื่อเฟสที่หนึ่งจบลงโดยไม่มีการยกเลิกการทำงาน ณ จุดนี้ทุก resource manager ซึ่งอาจถูกควบคุมจากคนละที่ จะทำการอัพเดตข้อมูลจริง ๆ

การแบ่งจุดที่ทรานแซ็กชันทำงานสำเร็จออกเป็นสองเฟสนี้ เรียกว่าโพรโตคอล two-phase commit หรือ *2PC* มันมีประโยชน์มากเนื่องจากทำให้หลาย transaction manager และ resource manager ทำงานร่วมกันในทรานแซ็กชันแบบกระจายได้ หากมีใครก็ตามระบุว่าทรานแซ็กชันควรถูกยกเลิก ทุก resource ก็จะต้องถูก roll back ดังรูปที่ 7-7




*รูปที่ 7-7 ทรานแซ็กชันแบบกระจายที่มีการใช้โพรโตคอล **Two-Phase Commit***

ในระบบ two-phase commit แบบกระจาย มี transaction manager หลักอยู่หนึ่งตัว เรียกว่า distributed transaction coordinator ทำหน้าที่ควบคุมทรานแซ็กชันทั้งหมด และจัดการกับแต่ละคำสั่งที่มาจากหลาย ๆ transaction manager ในเครือข่าย ขั้นตอนต่อไปนี้เป็นขั้นตอนที่เกิดขึ้นในทรานแซ็กชัน two-phase commit แบบกระจาย

1) transaction coordinator ส่งข้อความ prepare to commit ไปยังทุก transaction manager ที่เกี่ยวข้อง
2) แต่ละ transaction manager อาจส่งผ่านข้อความนี้ไปยังแต่ละ resource manager ที่สัมพันธ์กับ transaction manager นั้น
3) แต่ละ transaction manager รายงานกลับไปยัง transaction coordinator หากทุก transaction manager ยินยอมที่จะ commit คำสั่ง commit ที่กำลังจะเกิดขึ้นจะถูกบันทึกลง log เพื่อใช้ในกรณีที่เกิดความผิดพลาดขึ้น
4) สุดท้าย transaction coordinator จะบอกแต่ละ transaction manager ให้ทำการ commit แต่ละ transaction manager จะเรียก resource manager ให้อัพเดตข้อมูล ถ้าหากมีข้อผิดพลาดใด ๆ เกิดขึ้น log จะถูกใช้ในการทำขั้นตอนนี้ซ้ำอีกครั้ง

# บทที่ 8

# XML

## 8.1 แนะนำ XML

XML ย่อมาจาก eXtensible Markup Language XML ถูกออกแบบมาเพื่อใช้อธิบายข้อมูล ใน XML จะไม่มีแท็ก (tag) ที่กำหนดไว้ล่วงหน้า เราจะต้องสร้างแท็กของเราขึ้นมาเอง และจะใช้ Document Type Definition (DTD) ในการอธิบายรูปแบบของข้อมูล

ข้อแตกต่างที่สำคัญระหว่าง XML และ HTML คือ XML ถูกออกแบบมาเพื่อใช้แลกเปลี่ยนข้อมูล ไม่ใช่มาแทนที่ HTML XML กับ HTML ถูกออกแบบมาเพื่อจุดประสงค์ที่ต่างกัน HTML จะเกี่ยวข้องกับการแสดงข้อมูล XML จะเกี่ยวข้องกับการอธิบายข้อมูล แท็กของ HTML เช่น <H1></H1> เป็นแท็กสำหรับบอกรูปแบบการแสดงผล ในขณะที่แท็กของ XML ต้องสร้างขึ้นมาเองเช่น

```
<BOOK>
   <NAME>Beginning XML</NAME>
   <ISDN>1-861003-41-2</ISDN>
   <PAGE>800</PAGE>
<BOOK>
```

จะใช้แสดงข้อมูลของหนังสือที่ประกอบด้วย element BOOK , NAME , ISDN และ PAGE ข้อมูลที่อยู่ในรูปเท็กซ์นี้เป็นอิสระต่อซอฟต์แวร์และฮาร์ดแวร์ จึงเหมาะที่จะนำมาใช้ในการแลกเปลี่ยนข้อมูลระหว่างระบบบนอินเทอร์เน็ต การเปลี่ยนข้อมูลให้อยู่ในรูป XML เป็นการลดความซับซ้อนและสามารถสร้างข้อมูลที่ถูกอ่านโดยแอพพลิเคชันใด ๆ ก็ได้ นอกจากนั้นการเพิ่มเติม element จะไม่ทำให้แอพพลิเคชันเสียหาย เนื่องจากการอ่านข้อมูลโดย XML parser จะใช้วิธีแยกข้อมูลจาก element ที่ต้องการ จาก XML ข้างต้นนี้จะแยกข้อมูล element NAME , ISDN และ PAGE หากเพิ่ม element AUTHOR เข้าไป แอพพลิเคชันก็ยังคงดึงข้อมูลใน element เดิมโดยจะมองข้าม element ที่ไม่ได้กำหนดให้แยกข้อมูลออกมาในตอนพัฒนาแอพพลิเคชัน

## 8.2 ไวยากรณ์ของ XML

กฎไวยากรณ์ของ XML นั้นง่ายและเคร่งครัดมาก กฎของมันง่ายต่อการเรียนรู้และง่ายต่อการใช้ ด้วยเหตุนี้การสร้างซอฟต์แวร์ในการอ่านและจัดการกับข้อมูลจึงเป็นเรื่องง่าย ตัวอย่างของเอกสาร XML ดังนี้

```
<?XML version="1.0"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

ในบรรทัดแรกคือการประกาศเอกสาร XML ที่บอกถึง XML เวอร์ชัน

บรรทัดต่อมาคือ root element (note) อีก 4 บรรทัดถัดมาเป็นการอธิบายถึง 4 child element ของ root (to , from , heading , body) และบรรทัดสุดท้ายเป็นการกำหนดจุดสิ้นสุดของ root

กฎของเอกสาร XML มีดังนี้

- **XML Element ทุกตัวต้องมีการปิดแท็ก**

ใน HTML บาง element ไม่จำเป็นต้องมีการปิดแท็ก แต่ใน XML ทุก element จำเป็นที่จะต้องมีการปิดแท็ก

```
<p>This is a paragraph              X
<p>This is a paragraph</p>          √
```

- **XML แท็กมีความแตกต่างระหว่างตัวพิมพ์เล็กและตัวพิมพ์ใหญ่ (case sensitive)**

ใน XML แท็ก <Letter> นั้นต่างจาก <letter> เนื่องจาก XML นั้นมีคุณสมบัติที่แยกความแตกต่างระหว่างตัวพิมพ์เล็กและตัวพิมพ์ใหญ่

```
<Message>This is incorrect</message>       X
<message>This is correct</message>         √
```

- **XML Element ทุกตัวต้องมีการซ้อนกันอย่างเหมาะสม**

ใน HTML บาง element สามารถเหลื่อมกันได้ แต่ใน XML นั้นต้องซ้อนกันเป็นขั้นโดยสมบูรณ์

```
<b><i>This text is bold and italic</b></i>       X
<b><i>This text is bold and italic</i><b>       √
```

- **ทุกเอกสาร XML ต้องมี root แท็ก**

แท็กแรกในเอกสาร XML นั้นจะเป็น root แท็กและจะมีได้เพียงหนึ่งเดียวเท่านั้น element อื่น ๆ ทั้งหมดจะต้องถูกซ้อนอยู่ภายใน root element เท่านั้น และ element เหล่านั้นก็สามารถมี element ย่อย (child element) ได้อีก ดังนี้

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

- **ค่าของ Attribute ต้องมีอยู่ภายในเครื่องหมายคำพูด (“”)**

XML element สามารถมี attribute ได้ โดยที่ค่าของ attribute จะต้องอยู่ภายในเครื่องหมายคำพูด (“) เอกสาร XML ข้างล่างนี้ อันแรกผิด อันที่สองถูก

```
<note date=12/11/99>        X
<note date="12/11/99">      √
```

**8.3 XML Element**

XML element มีคุณสมบัติดังนี้

- **XML Element สามารถขยายได้**

ตามตัวอย่างนี้

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

ถ้าเราจะสร้างแอพพลิเคชันที่สามารถแยก element to , from และ body จาก เอกสาร XML เพื่อแสดงผลลัพธ์ดังนี้

MESSAGE

To:Tove

From: Jani

Don't forget me this weekend!

และ ถ้าเพิ่มข้อมูลเพิ่มเติมลงไปดังนี้

```
<note>
    <date>1999-08-01</date>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

เมื่อเอกสาร XML เป็นดังนี้แล้ว แอพพลิเคชันที่สร้างไว้จะไม่เสียหาย เนื่องจาก แอพพลิเคชันจะยังคงค้นหา element to , from และ heading ในเอกสาร XML และยังคงให้ผลลัพธ์เหมือนเดิม

- **XML Element มีความสัมพันธ์กัน**

ในการที่จะเข้าใจถึงภาพรวมของ XML จะต้องเข้าใจถึงความสัมพันธ์ระหว่างชื่อของ XML element ที่ตั้งและ content ของ element

คำอธิบายของหนังสือดังนี้

```
Book Title: My First XML
Chapter 1: Introduction to XML
    What is HTML
    What is XML
Chapter 2: XML Syntax
    Elements must have a closing tag
    Elements must be correctly nested
```

สามารถมีเอกสาร XML ที่อธิบายหนังสือนั้น ได้ดังนี้

```
<book>
    <title>My First XML</title>
    <prod id="33-657" media="paper"></prod>
    <chapter>Introduction to XML
        <para>What is HTML</para>
       <para>What is XML</para>
    </chapter>
    <chapter>XML Syntax
        <para>Elements must have a closing tag</para>
        <para>Elements must be properly nested</para>
    </chapter>
</book>
```

book เป็น root element title และ chapter เป็น child element ของ book book เป็น parent element ของทั้ง title และ chapter title และ chapter จึงเป็น sibling กันเนื่องจากมี parent เดียวกัน

- **Element มี Content**

XML element คือทั้งหมดตั้งแต่เปิดแท็กจนถึงปิดแท็ก element หนึ่งสามารถมี element content , mixed content , simple content หรือ empty content และยังมี attribute ได้ด้วย

จากตัวอย่างข้างบน book มี element content เพราะมันประกอบด้วย element อื่น chapter เป็น mixed content เพราะมันประกอบด้วยเท็กซ์และ element อื่น ๆ para เป็น simple content (text content) เพราะมันประกอบด้วยเท็กซ์เท่านั้น prod เป็น empty element เนื่องจากมันไม่มีข้อมูลใดๆ และเฉพาะ prod element เท่านั้นที่มี attribute attribute ที่ชื่อ id มีค่าเท่ากับ “33-657” attribute ที่ชื่อ medis มีค่าเท่ากับ “paper”

- **การตั้งชื่อ Element**

XML element มีกฎการตั้งชื่อดังนี้

  - ชื่อประกอบด้วย ตัวอักษร , ตัวเลข และอักขระอื่นๆ
  - ชื่อต้องไม่ขึ้นต้นด้วยตัวเลขหรือ “_ “(underscore)
  - ชื่อต้องไม่ขึ้นด้วยคำว่า XML (หรือ Xml หรือ xml)
  - ชื่อไม่สามารถมีช่องว่างได้ (space)

- ชื่อต้องไม่ควรประกอบด้วย ":" เพราะมันถูกสงวนไว้กับการใช้ namespace

ทุกๆชื่อสามารถใช้ได้ ไม่มีคำใดเป็นคำสงวน แต่จะเป็นการดีในการที่จะหลีกเลี่ยงการใช้ "–" หรือ "." แล้วใช้เฉพาะ "_" เท่านั้น เนื่องจากอาจเกิดการสับสนของซอฟต์แวร์ในกรณีพยายามที่จะตัดคำ (first-name) หรือคิดว่า "name" เป็นคุณสมบัติของ object "first" (first.name)

**8.4 XML Attribute**

XML attribute มีคุณสมบัติดังนี้

- **รูปแบบของ Quote , "female" or 'female'**

ค่าของ attribute ต้องอยู่ในเครื่องหมายคำพูดเสมอ แต่อาจเป็น ' (single quote) หรือ " (double quote) ก็ได้

```
<person sex="female">                √
หรือ
<person sex='female'>                √
```

Double quote จะเป็นที่ใช้โดยทั่วไปมากกว่า แต่ในบางครั้ง (ถ้าค่าของ attribute จำเป็นที่จะต้องมี quote) จึงจำเป็นที่จะต้องใช้ single quote ดังนี้

```
<gangster name='George "Shotgun" Ziegler'>
```

- **การใช้ Element และ Attribute**

ข้อมูลสามารถที่จะถูกเก็บใน child element หรือ attribute ดังนี้

```
<person sex="female">
   <firstname>Anna</firstname>
   <lastname>Smith</lastname>
</person>
```

หรือ

```
<person>
   <sex>female</sex>
```

```
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
```

ตัวอย่างแรก sex เป็น attribute ตัวอย่างที่ 2 sex เป็น child element ซึ่งทั้งสองตัวอย่างให้ความหมายเดียวกัน มันไม่มีกฎว่าจะใช้ attribute เมื่อไร หรือ child element เมื่อไร แต่แนะนำว่าควรจะหลีกเลี่ยงการใช้ attribute ควรจะใช้ child element ถ้าอินฟอร์เมชันนั้นดูเหมือนจะเป็นข้อมูล (data)

- **หลีกเลี่ยงการใช้ Attribute**

  บางปัญหาของการใช้ attribute

  - attribute ไม่สามารถมีหลายค่า (child element สามารถมีได้)
  - attribute ไม่ง่ายที่จะขยาย (ในอนาคต)
  - attribute ไม่สามารถอธิบายโครงสร้าง (child element สามารถทำได้)
  - attribute ยากที่จะจัดการด้วยโค้ดของโปรแกรม
  - ค่าของ attribute ไม่ง่ายในการจะตรวจสอบกับ DTD ที่มีอยู่

แต่อย่างไรก็ตาม จะสามารถใช้ attribute ก็ต่อเมื่อ อินฟอร์เมชันนั้นไม่สัมพันธ์กับข้อมูล ตัวอย่างเช่น ในบางครั้ง ID reference สามารถถูกใช้ในการเข้าถึง XML element

```
<messages>
    <note ID="501">
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
    </note>
    <note ID="502">
        <to>Jani</to>
        <from>Tove</from>
          <heading>Re: Reminder</heading>
          <body>I will not!</body>
    </note>
</messages>
```

ID ในตัวอย่างนี้เป็น counter หรือ unique identifier ในการอ้างถึง note ที่ต่างกันของไฟล์ XML ไม่ใช่ส่วนหนึ่งของข้อมูล note

### 8.5 XML Validation

เอกสาร XML แบ่งการตรวจสอบความถูกต้องของเอกสารเป็น 2 ชนิดคือ

**1) "Well Formed" XML Document**

"Well Formed" XML document เป็นเอกสารที่เป็นไปตามกฎไวยากรณ์ XML ที่ได้กล่าวมาข้างต้น

**2) "Valid" XML Document**

"Valid" XML document เป็นเอกสาร XML ที่มีรูปแบบที่เป็นไปตามกฎของ DTD โดยจะต้องตรงตามโครงสร้างข้อมูลของเอกสารที่ได้ประกาศโดย DTD

- **XML DTD**

  วัตถุประสงค์ของ DTD คือกำหนดโครงสร้างของเอกสาร XML เพื่อใช้ในการตรวจสอบว่าเอกสาร XML นั้นถูกต้องตามรูปแบบหรือไม่ เช่น element BOOK จะต้องมี element NAME เสมอ อาจจะมี element PAGE หรือไม่ก็ได้ เป็นต้น

- **XML Schema**

  XML schema มีวัตถุประสงค์เหมือนกับ DTD จะถูกนำมาใช้แทน DTD เนื่องจาก

  - ง่ายในการเรียนรู้มากกว่า DTD
  - ใช้ได้ดีกว่า DTD
  - schema เขียนด้วย XML
  - รองรับ datatype
  - รองรับ namespace

### 8.6 แนะนำ DTD

**1) การประกาศ DOCTYPE ภายใน**

ถ้า DTD ถูกรวมอยู่ในไฟล์ XML นั้น มันจะต้องอยู่ภายในการให้คำจำกัดความ DOCTYPE ตามไวยากรณ์ นี้

```
<!DOCTYPE root-element [element-declarations]>
```

ตัวอย่างดังนี้

```
<?XML version="1.0"?>
<!DOCTYPE note [
   <!ELEMENT note (to,from,heading,body)>
   <!ELEMENT to      (#PCDATA)>
   <!ELEMENT from    (#PCDATA)>
   <!ELEMENT heading (#PCDATA)>
   <!ELEMENT body    (#PCDATA)>
]>
<note>
   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend</body>
</note>
```

DTD ข้างบนนี้สามารถแปลความหมายได้ดังนี้

!DOCTYPE note (บรรทัดที่ 2) กำหนดว่าเป็นเอกสารของ type note

!ELEMENT note (บรรทัดที่ 3) กำหนดว่า note element มี 4 element คือ to , from , heading และ body

!ELEMENT to (บรรทัดที่ 4) กำหนดว่า to element เป็นข้อมูลชนิด #PCDATA

!ELEMENT from (บรรทัดที่ 5) กำหนดว่า from element เป็นข้อมูลชนิด #PCDATA

**2) การประกาศ DOCTYPE ภายนอก**

ถ้า DTD อยู่ภายนอกไฟล์ XML มันจะต้องอยู่ภายในการให้คำจำกัดความ DOCTYPE ตาม ไวยากรณ์นี้

```
<!DOCTYPE root-element SYSTEM "filename">
```

ตัวอย่างไฟล์ XML เป็นดังนี้

```
<?XML version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
```

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

ไฟล์ของ DTD ชื่อ note.dtd เป็นดังนี้

**note.dtd**

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

**8.6.1 โครงสร้างของ XML**

เอกสาร XML จะถูกสร้างจากส่วนต่างๆ ดังนี้

- Element – เป็นโครงสร้างหลักของ XML
- Tag - ใช้ในการกำหนด markup element
- Attribute - ให้ข้อมูลเพิ่มเติมกับ element
- Entity – เป็นตัวแปรที่ใช้ในการกำหนดเท็กซ์ทั่ว ๆ ไป entity reference เป็นการอ้างถึง entity เช่น entity ชื่อ domain มีค่าเท่ากับ www.w3.org entity reference คือ &domain หากภายในเอกสาร XML มี &domain ค่าของมันจะมีค่าเท่ากับ www.w3.org
- PCDATA -หมายถึง parse character data PCDATA เป็นเท็กซ์ที่จะถูก parse โดย parser
- CDATA -หมายถึง character data เป็นเท็กซ์ที่จะไม่ถูก parse โดย parser

**8.6.2 การกำหนด Element ใน DTD**

วิธีการในการประกาศ element จะเป็นดังนี้

- **การประกาศ Element**

  ประกาศตามไวยากรณ์นี้

```
<!ELEMENT element-name category>
```

หรือ

```
<!ELEMENT element-name (element-content)>
```

- **Element ว่างเปล่า (Empty)**

  ถูกประกาศโดยคีย์เวิร์ด EMPTY

```
<!ELEMENT element-name EMPTY>
```

```
DTD example:
<!ELEMENT br EMPTY>
XML example:
<br />
```

- **Element ที่เป็นข้อมูลชนิดตัวอักษร**

  element ที่เป็นตัวอักษรเท่านั้นจะถูกประกาศด้วยคีย์เวิร์ด PCDATA ที่เปิดและปิดด้วย “(“ “)”

```
<!ELEMENT element-name (#PCDATA)>
```

```
DTD example:
<!ELEMENT note (#PCDATA)>
```

- **Element ที่เป็นข้อมูลใด ๆ**

  ประกาศกับคีย์เวิร์ด ANY content จะสามารถเป็นข้อมูลชนิดใดก็ได้

```
<!ELEMENT element-name ANY>
```

```
DTD example:
<!ELEMENT note ANY>
```

- **Element และ Children (Sequences)**

ประกาศโดยใช้ “,” ในการแยก child element แล้วเปิดและปิดด้วย “(“ “)”

```
<!ELEMENT element-name (child-element-name)>
```

หรือ

```
<!ELEMENT element-name (child-element-name,child-element-name,.....)>
```

```
DTD example:
<!ELEMENT note (to,from,heading,body)>
```

ถ้ามีการประกาศ child element จะต้องประกาศ child element ให้สมบูรณ์ด้วยดังตัวอย่างข้างล่างนี้

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to      (#PCDATA)>
<!ELEMENT from    (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body    (#PCDATA)>
```

- **การประกาศเพียงครั้งเดียวของ Element เดียวกัน**

```
<!ELEMENT element-name (child-name)>
```

```
DTD example:
<!ELEMENT note (message)>
```

จากตัวอย่างหมายความว่า note element จะประกอบด้วย child element message เท่านั้น

- **การประกาศอย่างน้อย 1 ครั้งของ Element เดียวกัน**

```
<!ELEMENT element-name (child-name+)>
```

```
DTD example:
<!ELEMENT note (message+)>
```

เครื่องหมาย + ถูกใช้ในความหมายหนึ่งหรือมากกว่า จึงหมายความว่า note element จะประกอบด้วย child element message ตั้งแต่ 1 ค่าขึ้นไป

- **การไม่ประกาศหรือประกาศเท่าไรก็ได้ของ Element เดียวกัน**

```
<!ELEMENT element-name (child-name*)>
```

```
DTD example:
<!ELEMENT note (message*)>
```

เครื่องหมาย * ถูกใช้ในความหมายว่าไม่มีหรือมีเท่าไรก็ได้ จึงหมายความว่า note element จะประกอบด้วย child element message กี่ค่าก็ได้ หรือไม่มีเลยก็ได้

- **การไม่ประกาศหรือประกาศเพียงครั้งเดียวของ Element เดียวกัน**

```
<!ELEMENT element-name (child-name?)>
```

```
DTD example:
<!ELEMENT note (message?)>
```

เครื่องหมาย ? ถูกใช้ในความหมาย ไม่มีหรือมีเพียงหนึ่ง จึงหมายความว่า note element จะประกอบด้วย child element message หนึ่งค่า หรือไม่มีก็ได้

- **การประกาศ Content แบบให้เลือก**

  จะใช้เครื่องหมาย | แทนความหมายหรือ

```
DTD example:
<!ELEMENT note (to,from,header,message|body)>
```

ดังนั้นตัวอย่างนี้จึงหมายถึง note element จะต้องประกอบด้วย to element , from element , header element แล้วก็ message element หรือ body element อย่างใดอย่างหนึ่งเท่านั้น

- **การประกาศ Content แบบผสม**

  เป็นการผสมการประกาศหลาย ๆ อย่างเข้าด้วยกัน

```
DTD example:
<!ELEMENT note (#PCDATA|to|from|header|message)*>
```

ตัวอย่างนี้หมายถึง note element จะประกอบด้วย character data หรือ element to , from , header , และ message จำนวนเท่าไรก็ได้

**8.6.3 การกำหนด Attribute ใน DTD**

วิธีการประกาศ attribute จะเป็นดังนี้

- **การประกาศ Attribute**

  attribute ต้องถูกประกาศตามไวยากรณ์ นี้

```
<!ATTLIST element-name attribute-name attribute-type default-value>
```

```
DTD example:
<!ATTLIST payment type CDATA "check">
XML example:
<payment type="check">
```

ชนิดของ attribute สามารถมีค่าได้ดังตารางที่ 8-1

| ค่า | คำอธิบาย |
|---|---|
| CDATA | ค่าต้องเป็น character ที่จะไม่ถูก parse โดย parser |
| (en1\|en2\|..) | ค่าต้องเป็นอันใดอันหนึ่งในรายการนั้น |
| ID | ค่าต้องเป็น unique id |
| IDREF | ค่าต้องเป็น id ของ element อื่น |
| IDREFS | ค่าต้องเป็นรายการของ id อื่น โดยจะเว้นแต่ละรายการด้วยช่องว่าง |

***ตารางที่ 8-1 ชนิดของ Attribute ใน XML***

| ค่า | คำอธิบาย |
|---|---|
| NMTOKEN | ค่าต้องเป็นชื่อที่ประกอบด้วยตัวอักษร , ตัวเลข , เครื่องหมายจุด (.) , เครื่องหมายขีดกลาง (-) , เครื่องหมายขีดล่าง (_) และสามารถมีเครื่องหมายโคลอน (:) ยกเว้นในตำแหน่งแรกได้ |
| NMTOKENS | ค่าต้องเป็นรายการของ NMTOKEN โดยจะเว้นแต่ละรายการด้วยช่องว่าง |
| ENTITY | ค่าต้องเป็น entity |
| ENTITIES | ค่าต้องเป็นรายการของ entity |
| NOTATION | ค่าต้องเป็นชื่อของ notation (notation คือชนิดของข้อมูลภายนอกที่กำหนดขึ้นเอง เช่น GIF เพื่อบอกว่าเป็นไฟล์รูปภาพ) |
| XML: | ค่าต้องขึ้นต้นด้วย XML |

***ตารางที่ 8-1 ชนิดของ Attribute ใน XML (ต่อ)***

default-value สามารถมีค่าได้ดังตารางที่ 8-2

| ค่า | คำอธิบาย |
|---|---|
| #DEFAULT value | ค่าดีฟอลต์ของ attribute |
| #REQUIRED | element ต้องมี attribute |
| #IMPLIED | element จะมี attribute หรือไม่ก็ได้ |
| #FIXED value | attribute ต้องเหมือนกับที่กำหนดไว้ |

***ตารางที่ 8-2 ค่าดีฟอลต์ของ Attribute ใน XML***

- **ตัวอย่างของการประกาศ Attribute**

```
DTD example:
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
XML example:
<square width="100"></square>
```

ตัวอย่างนี้หมายความว่า square element เป็น empty element กับ attribute width ซึ่งเป็นชนิด CDATA ถ้าไม่ระบุจะยึดค่า default เป็น 0

- **ค่าของ Attribute แบบดีฟอลต์**

```
<!ATTLIST element-name attribute-name attribute-type "default-value">
```

```
DTD example:
<!ATTLIST payment type CDATA "check">
XML example:
<payment type="check">
```

ถ้าในเอกสาร XML ไม่มีการระบุจะใช้ค่าดีฟอลต์ แต่ถ้าระบุก็จะใช้ตามที่ระบุไว้

- **Attribute โดยนัย**

```
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
```

```
DTD example:
<!ATTLIST contact fax CDATA #IMPLIED>
XML example:
<contact fax="555-667788">
```

จะใช้ #IMPLIED ในกรณีจะระบุ attribute หรือไม่ก็ได้ ถ้าไม่ระบุก็จะไม่มีค่า default ให้

- **Attribute ที่ต้องระบุเสมอ**

```
<!ATTLIST element-name attribute_name attribute-type #REQUIRED>
```

```
DTD example:
<!ATTLIST person number CDATA #REQUIRED>
XML example:
<person number="5677">
```

ใช้ในการบังคับให้ต้องระบุ attribute ถ้าในเอกสาร XML ไม่ระบุ attribute เอกสารนั้นจะไม่ถูกต้อง

- **ค่าของ Attribute คงที่**

```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
```

```
DTD example:
<!ATTLIST sender company CDATA #FIXED "Microsoft">
XML example:
<sender company="Microsoft">
```

ในแบบนี้ถ้าไม่ได้ระบุค่า attribute จะใช้ค่าดีฟอลต์ แต่ถ้าจะระบุต้องระบุให้ตรงกับค่าดีฟอลต์ มิฉะนั้นจะถือว่าผิด

- **ค่าของ Attribute กำหนดเอง**

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
```

```
DTD example:
<!ATTLIST payment type (check|cash) "cash">
XML example:
<payment type="check">
หรือ
<payment type="cash">
```

จะใช้เมื่อเราต้องการกำหนดค่าของ attribute เอง

**8.6.4 การกำหนด Entity ใน DTD**

- **การประกาศ Entity ภายใน**

```
<!ENTITY entity-name "entity-value">
```

```
DTD Example:
<!ENTITY writer "Donald Duck.">
```

```
<!ENTITY copyright "Copyright W3Schools.">
XML example:
<author>&writer;&copyright;</author>
```

- **การประกาศ Entity ภายนอก**

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

```
DTD Example:
<!ENTITY writer
 SYSTEM "http://www.w3schools.com/entities/entities.XML">
<!ENTITY copyright
 SYSTEM "http://www.w3schools.com/entities/entities.dtd">
XML example:
<author>&writer;&copyright;</author>
```

**8.7 XML Namespace**

XML namespace จะหาวิธีหลีกเลี่ยงความสับสนที่เกิดจากชื่อ element (Name Conflicts)

- **ความสับสนที่เกิดจากชื่อ**

ความสับสนที่เกิดจากชื่อ element จะเกิดขึ้นบ่อยเมื่อเอกสารที่แตกต่างกัน ใช้ชื่อเดียวกันในการอธิบายชนิดของ element ที่แตกต่างกัน เช่น table ใน 2 ตัวอย่างต่อไปนี้

```
<table>
  <tr>
  <td>Apples</td>
  <td>Bananas</td>
  </tr>
</table>
```

และ

```
<table>
  <name>African Coffee Table</name>
```

```
  <width>80</width>
  <length>120</length>
</table>
```

ถ้านำเอกสาร XML ทั้ง 2 มารวมกัน จะเกิดความสับสนจากชื่อขึ้นเพราะทั้ง 2 มี element (table)ซึ่งมีใช้ในความหมายที่ต่างกัน

- **การแก้ปัญหาความสับสนที่เกิดจากชื่อโดยใช้ Prefix**

  เอกสาร XML ทั้ง 2 คือ

```
<h:table>
  <h:tr>
  <h:td>Apples</h:td>
  <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

และ

```
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

ความสับสนที่เกิดจากชื่อจะถูกกำจัดไปโดยการเพิ่ม prefix ทำให้ได้ element ที่ต่างกัน 2 ชนิด (**<h:table>** และ **<f:table>**)

- **การใช้ Namespaces**

  จากเอกสาร XML ทั้ง2 คือ

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
  <h:td>Apples</h:td>
  <h:td>Bananas</h:td>
```

```
  </h:tr>
</h:table>
```

และ

```
<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

เราสามารถเพิ่ม attribute xmlns ลงในแท็ก เพื่อบอกชื่อที่ระบุไว้ ที่เกี่ยวข้องกับ namespace

- **Attribute Namespace**

attribute namespace จะวางในแท็กเริ่มต้น (start tag) ตามไวยากรณ์ ดังนี้

```
Xmlns:namespace-prefix="namespace"
```

จากตัวอย่างข้างบน namespace จะสามารถกำหนดด้วย Internet address

```
Xmlns:f="http://www.w3schools.com/furniture">
```

จาก namespace specification ของ W3C กล่าวไว้ว่า namespace สามารถเป็น Uniform Resource Identifier (URI)

ถ้า namespace ถูกกำหนดในแท็กเริ่มต้นแล้ว child element ทุกตัวที่มี prefix เหมือนกันจะต้อง มีความสัมพันธ์กับ namespace เดียวกันด้วย

และ address ที่ใช้บอก namespace ใช้เพื่อให้ชื่อมีความเป็นเอกลักษณ์ไม่ได้ใช้เป็น address เพื่ออ้างอิงหาข้อมูล แต่หลายบริษัทใช้ namespace เพื่อเป็นตัวชี้ไปยัง web page ที่มีอยู่จริงซึ่งเก็บข้อมูลเกี่ยวกับ namespace ไว้

- **Default Namespaces**

การกำหนด default namespace จะป้องกันการใช้ prefix ใน child element ทุกตัว มีไวยากรณ์ดังนี้

```
<element xmlns="namespace">
```

ซึ่งจากตัวอย่างที่ผ่านมาจะได้ดังนี้

```
<table xmlns="http://www.w3.org/TR/html4/">
  <tr>
  <td>Apples<td>
  <td>Bananas<td>
  <tr>
</table>
```

และ

```
<table xmlns="http://www.w3schools.com/furniture">
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

**8.8 XML PCDATA and CDATA**

Parsable Character Data (PCDATA) คือข้อมูลแบบเท็กซ์ที่จะถูก parse โดย parser

Character Data (CDATA) คือข้อมูลแบบเท็กซ์ที่ไม่ถูก parse โดย parser

**8.8.1 PCDATA**

- **XML Parser จะมองทุกเท็กซ์เป็น Parsable Characters (PCDATA)**

  เมื่อ XML ถูก parse เท็กซ์ที่อยู่ระหว่าง XML แท็กจะถูก parse ไปด้วย

```
<message>This text is also parsed</message>
```

parser ทำเช่นนี้เนื่องจาก XML element สามารถมีหลาย element อยู่ภายใน ดังในตัวอย่าง

```
<name><first>Bill</first><last>Gates</last></name>
```

และ parser จะแบ่งเป็น sub-element ดังนี้

```
<name>
```

```
  <first>Bill</first>
  <last>Gates</last>
</name>
```

- **Escape Character**

XML character ที่ไม่สามารถใช้ได้จะถูกแทนที่ด้วย entity reference

ถ้าใส่ตัวอักษร “<” ใน XML element จะทำให้เกิดความผิดพลาด เพราะ parser จะแปลเป็นจุดเริ่มต้นของ element ใหม่ ดังนั้นจึงไม่สามารถเขียน อย่างนี้ได้

```
<message>if salary < 1000 then</message>
```

การหลีกเลี่ยง สามารถทำได้โดย แทนที่ “<” ด้วย entity reference ดังนี้

```
<message>if salary &lt; 1000 then</message>
```

entity reference มาตรฐานที่ได้กำหนดมาให้ก่อนแล้วใน XML มีทั้งหมด 5 ตัว ดังตารางที่ 8-3

| Entity Reference | ค่า | ความหมาย |
|---|---|---|
| < | < | less than |
| > | > | greater than |
| & | & | ampersand |
| ' | ' | apostrophe |
| " | " | quotation mark |

*ตารางที่ 8-3 Entity Reference มาตรฐาน*

entity reference จะขึ้นต้นด้วย & และลงท้ายด้วย ;

เฉพาะ “<” และ “&” เท่านั้นที่ไม่สามารถใช้ได้ใน XML ส่วนตัวที่เหลือควรทำให้เป็นนิสัย

**8.8.2 CDATA**

parser จะไม่ parse ทุกอย่างที่อยู่ในส่วนของ CDATA

ถ้าเท็กซ์มี “<” หรือ “&” อยู่ภายใน XML element นั้นจะถูกกำหนดเป็นส่วนของ CDATA ส่วนของ CDATA จะเริ่มต้นด้วย “<![CDATA[“ และลงท้ายด้วย “]]>”

```
<script>
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
  {
  return 1
  }
else
  {
  return 0
  }
}
]]>
</script>
```

จากตัวอย่างข้างต้น ทุกอย่างที่อยู่ในส่วนของ CDATA จะถูกเพิกเฉยโดย parser

**8.9 XML Parser**

XML parser คือโปรแกรมที่ใช้ในการอ่าน สร้าง และจัดการกับเอกสาร XML รวมถึงตรวจความถูกต้องของเอกสาร XML parser สามารถเขียนได้ด้วยภาษาใด ๆ ก็ได้ ในบราวเซอร์ IE 5.0 ขึ้นไปก็มี Microsoft XML parser ทำให้ IE สามารถแสดงผลเอกสาร XML ได้ XML parser ของภาษาจาวาอย่างเช่น XML parser ของซัน (http://java.sun.com/xml) หากเราต้องการสร้างแอพพลิเคชันที่สามารถจัดการกับ XML เราก็ต้องมี parser ของภาษานั้นและเขียนโค้ดติดต่อกับ API ของมัน

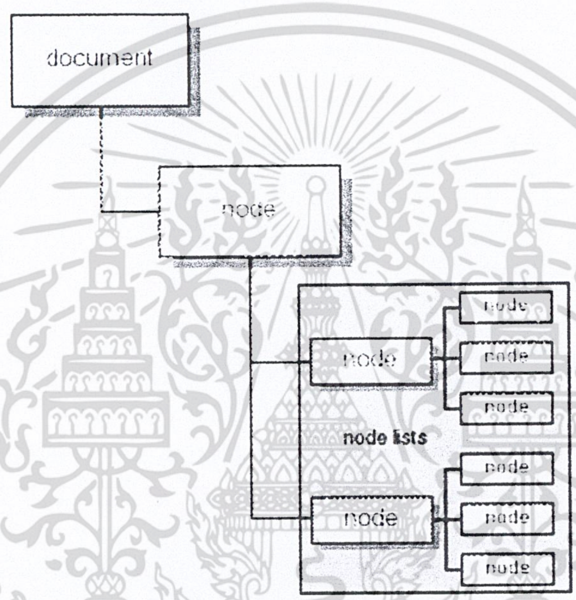
XML parser มี 2 ชนิดคือ

- DOM (Document Object Model)
- SAX (Simple API for XML)

### 8.9.1 DOM

DOM จะใช้วิธี parse เอกสาร XML เป็นลำดับชั้น (hierachy) ของออบเจ็กต์ดังนี้ (ดังรูปที่ 8-1)

1) Document Object – แหล่งข้อมูล XML
2) Node Object – Parent node ของ child node
3) Node List Object – list ของ sibling node
4) Parse Error – ใช้ในการรับค่าผิดพลาดที่เกิดจากการ parse

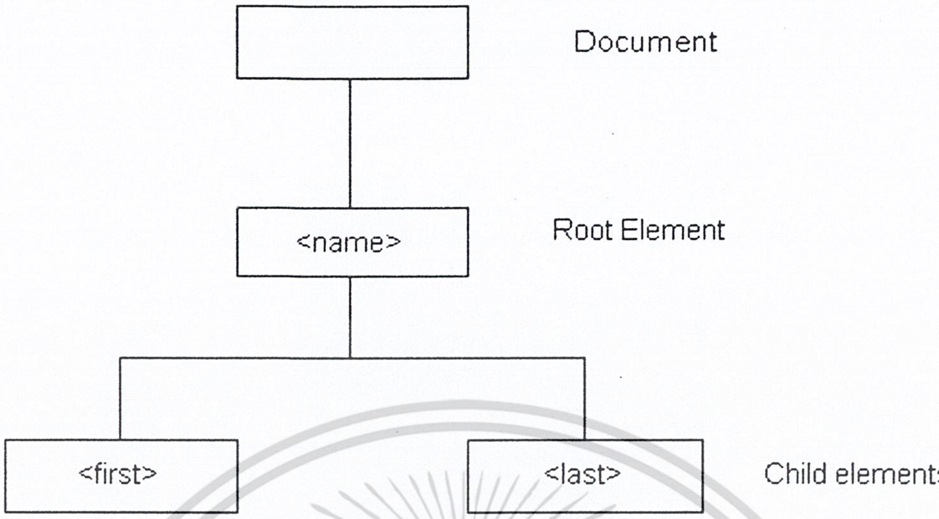



*รูปที่ 8-1 โครงสร้างของ DOM*

หากเอกสาร XML เป็นดังนี้

```
<name>
    <first>Jim(/first>
    <last>Jones</last>
</name>
```

จะได้ความสัมพันธ์ดังรูปที่ 8-2




***รูปที่ 8-2 ความสัมพันธ์ของการ parse ด้วย DOM***

**8.9.2 SAX**

SAX (Simple API for XML) วิธีนี้จะเป็นแบบ event-driven โดย event จะประกอบด้วย

1) Start Element – จะตอบสนองกับ event นี้เมื่อพบจุดเริ่มต้นของแท็ก XML
2) End element – จะตอบสนองกับ event นี้เมื่อพบจุดสิ้นสุดของแท็ก
3) Character – จะตอบสนองกับ event นี้เมื่อพบกับตัวอักษร
4) Start Document – จะตอบสนองกับ event นี้ในตอนเริ่ม parse
5) End Document - จะตอบสนองกับ event นี้ในเมื่อจบการ parse

ดังนั้นจากโค้ดข้างบนจะเกิด event ดังนี้

| | | |
|---|---|---|
| | → | Start Document |
| <name> | → | Start Element name |
| <first> | → | Start Element first |
| Jim | → | Character Jim |
| </first> | → | End Element first |
| <last> | → | Start Element last |
| Jones | → | Character Jones |
| </last> | → | End Element last |
| </name> | → | Start Element name |
| | → | End Document |

ทั้ง 2 วิธีมีข้อดีข้อเสียต่างกันคือ DOM จะใช้เมโมรีและมีการประมวลผลมาก แต่สามารถจัดการได้ง่าย สามารถที่จะแก้ไขค่าได้ ไม่เหมาะกับเอกสาร XML ที่มีขนาดใหญ่ ถูกนำไปใช้ในบราวเซอร์ ในขณะที่ SAX ไม่ต้อง parse เอกสาร XML ทั้งหมดจึงใช้โมเมรีและมีการประมวลผลน้อยกว่า เหมาะกับการสร้างและรับค่าจากเอกสาร จะถูกใช้ในการประมวลผลทางฝั่งเซิร์ฟเวอร์ ตัวอย่าง XML parser ดังตารางที่ 8-4

| ชื่อ | แพล็ตฟอร์ม | คำบรรยาย |
|---|---|---|
| ActiveDOM | Microsoft Window | http://www.vivid-creations.com |
| ActiveSAX | Microsoft Window | http://www.vivid-creations.com |
| JavaTM API for XML | Java | http://java.sun.com/xml |
| MS XML | Microsoft Window | http://www.microsoft.com |
| Oracle XML Developer's Kit | Java , C++ , PL/SQL | http://technet.oracle.com/tech/xml |
| Xerces | Java , C++ | http://xml.apache.org |
| XP | Java | http://www.jclark.com/xml |

***ตารางที่ 8-4 XML Parser***

# บทที่ 9

# SOAP

## 9.1 แนะนำ SOAP

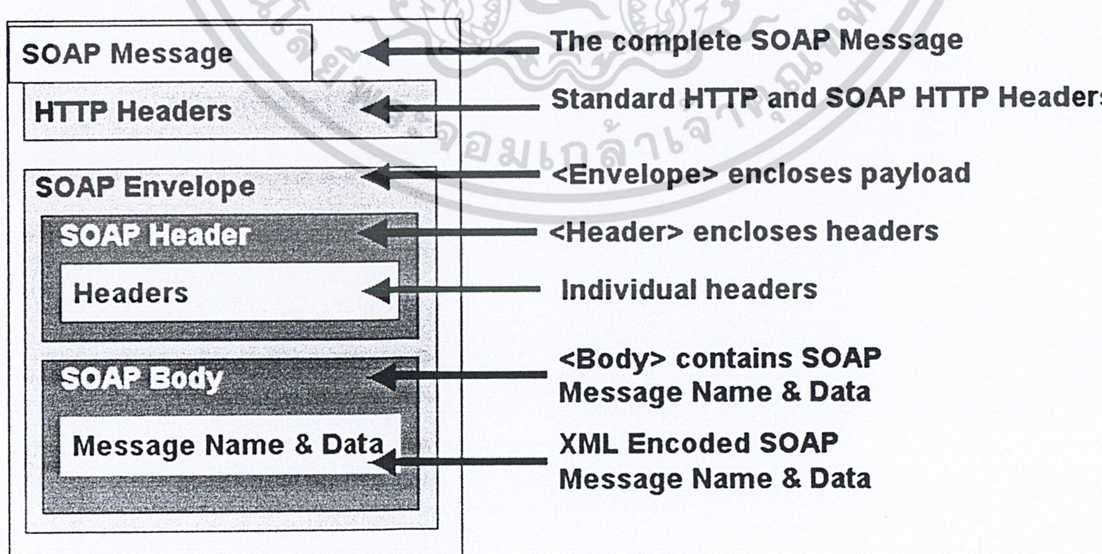
SOAP (Simple Object Access Protocal) เป็นโพรโตคอลที่ใช้ XML เป็นพื้นฐาน เพื่อให้ซอฟต์แวร์คอมโพเน็นต์ และแอพพลิเคชันสามารถติดต่อกันผ่าน HTTP ซึ่งเป็นมาตรฐานอินเตอร์เน็ตโพรโตคอลได้

เนื่องจากการสื่อสารระหว่างแอพพลิเคชัน เป็นสิ่งจำเป็นต่อการพัฒนาแอพพลิเคชันบนอินเตอร์เน็ตเป็นอย่างมาก แต่แอพพลิเคชันแบบกระจายกันทำงาน (distributed application) ในปัจจุบันใช้ Remote Procedure Call (RPC) สื่อสารกันระหว่างออบเจ็กต์ เช่น DCOM และ CORBA ซึ่งใช้ไม่ได้กับ HTTP ดังนั้น RPC จึงนำมาปรับใช้กับอินเตอร์เน็ตได้ยาก และมีปัญหาทางด้านความปลอดภัย ไฟร์วอลล์และ proxy server จะไม่ยอมให้ส่งข้อมูลชนิดนี้ได้ตามปกติ วิธีที่ดีกว่าคือใช้ HTTP เพราะเป็นที่ยอมรับโดยอินเตอร์เน็ตบราวเซอร์ และเซิร์ฟเวอร์ทุกชนิด ซึ่ง SOAP ถูกสร้างมาเพื่อใช้ในกรณีนี้

ข้อดีของ SOAP ก็คือ SOAP ไม่ขึ้นกับคอมโพเน็นต์เทคโนโลยี และภาษาการเขียนโปรแกรมใดๆ สามารถเขียนได้ง่ายและขยายเพิ่มเติมได้

## 9.2 ส่วนประกอบของ SOAP

SOAP เมสเซจใช้ไวยากรณ์ของ XML ในการสร้าง ประกอบด้วย 3 element มาตรฐาน คือ SOAP Envelope , SOAP Header และ SOAP Body โครงสร้างของ SOAP ดังรูปที่ 9-1




*รูปที่ 9-1 โครงสร้างของ SOAP*

SOAP เมสเซจ จะต้องเป็นไปตามกฎนี้

- **Envelope** - เป็น element ที่อยู่บนสุด จะต้องมี element นี้เสมอ
- **Header** - อาจจะมีหรือไม่มีก็ได้ แต่ถ้ามีต้องเป็น child element แรกของ envelope
- **Body** - ต้องเป็น childe element แรกของ envelope หรือ header

ตัวอย่างของ SOAP เมสเซจ ตัวอย่างนี้คือ GetLastTradePrice SOAP Request ที่ส่งไปยัง StockQuote service ประกอบด้วยสตริงพารามิเตอร์ symbol และคืนค่า float กลับมากับ SOAP response SOAP envelope element อยู่ที่จุดบนสุดของเอกสาร XML และใช้ XML namespace ใช้เพื่อป้องกันการสับสนของ SOAP identifier

SOAP request จะเป็นดังนี้

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
     <m:GetLastTradePrice xmlns:m="Some-URI">
        <symbol>DIS</symbol>
     </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP response จะเป็นดังนี้

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
```

```
Content-Length: nnnn

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

นอกจากนั้นใน SOAP เมสเซจ จะมีการใช้ XML namespace ทุก ๆ element ในเอกสารจะขึ้นต้นด้วย namespace namespace จะถูกกำหนดโดยใช้ xmlns attribute ดังนี้

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
```

แต่หากไม่ต้องการให้ขึ้นต้นด้วย namespace ก็สามารถทำได้ ดังนี้

```
<Envelope
      xmlns="http://schemas.xmlsoap.org/soap/envelope/">
```

**9.2.1 Envelope**

envelope element เป็น element บนสุดของเอกสาร XML ที่ใช้แสดงเมสเซจ อาจประกอบด้วย attribute คือ envelope namespace และ encodingStyle

1) **Envelope Namespace** – SOAP เมสเซจจะแสดงเวอร์ชันโดยใช้ namespace ของ envelope element namespace จะถูกนำไปใช้ขึ้นต้น envelope , header และ body element
2) **EncodingStyle Attribute** – ใช้แสดงวิธี serialization ของ SOAP เมสเซจ attribute นี้สามารถมีได้ในทุก element และจะมีผลกับ content ของ element นั้นและ child element ทั้งหมดของมันที่ไม่ได้ประกาศ attribute

### 9.2.2 Body

เป็นส่วนของข้อมูลที่ใช้ในการแลกเปลี่ยน โดยทั่ว ๆ ไปข้อมูลจะเป็นการ marshall RPC call และรายงานความผิดพลาด child element ของ body element จะถูกเรียกว่า body entry

body entry จะต้องเป็นไปตามกฎนี้

- body entry จะต้องแสดงด้วยชื่อเต็มที่ประกอบด้วย namespace URI และ ชื่อของมัน (local name)
- SOAP encodingStyle attribute อาจจะมีได้ เพื่อแสดงถึงวิธี encode ของ body entry นั้น

### 9.2.3 Header

เป็นส่วนเพิ่มเติมของเมสเซจสามารถประกอบด้วยอินฟอร์เมชันที่ระบุไปยังแอพพลิเคชัน ส่วนเพิ่มเติมนี้อาจนำไปใช้อิมพลีเมนต์เป็น authentication , การจัดการทรานแซ็กชัน เป็นต้น ทุก ๆ child element ของ header element จะถูกเรียกว่า header entry

header entry จะต้องเป็นไปตามกฎดังนี้

- header entry จะต้องแสดงด้วยชื่อเต็มที่ประกอบด้วย namespace URI และ ชื่อของมัน (local name)
- อาจมี SOAP encodingStyle attribute ที่ใช้สำหรับ header entry
- SOAP mustUnderStand attribute และ SOAP actor attribute จะมีหรือไม่ก็ได้ เพื่อใช้บอกว่าจะจัดการกับ entry นั้นอย่างไรและโดยใคร

```
<soap:Header>
   <m:local xmlns:m="http://www.w3schools.com/local/"
                soap:actor="http://www.w3schools.com/appml" />
      <m:language>en</m:language>
      <m:currency>USD</m:currency>
    </m:local>
</soap:Header>
```

1) **SOAP Actor Attribute** – SOAP เมสเซจสามารถถูกส่งไปยังปลายทางผ่านตัวกลางของ SOAP (SOAP intermediary) ตัวกลางของ SOAP คือแอพพลิเคชันที่มีความสามารถทั้งรับและส่งต่อ SOAP เมสเซจ ในการส่งต่อตัวกลางจะต้องไม่ส่งต่อ header element ไปให้กับแอพพลิเคชัน โดยตัวกลางนี้จะถูกกำหนดเป็น URI ถ้าไม่มี attribute นี้หมายถึงผู้รับ SOAP เมสเซจเป็นปลายทางสุดท้าย
2) **SOAP MustUnderStand Attribute** – ใช้แสดงว่า header entry นี้จำเป็นหรือเป็นเพียงออปชันสำหรับผู้รับที่จะนำไปประมวลผล ค่าของมันคือ “1” หรือ “0” ถ้าไม่มีการระบุจะมีค่าเท่ากับ “0”

"0" หมายถึงเป็นออปชัน "1" หมายถึงจำเป็น โดยจะต้องประมวลผลให้ถูกต้องตาม semantic หรือต้องแสดงความผิดพลาด เช่น เมสเซจเป็นส่วนหนึ่งของทรานแซ็กชัน ถ้าปลายทางไม่รองรับทรานแซ็กชันจะต้องไม่ประมวลผลและส่งผลผิดพลาดกลับไปในกรณีที่ mustUnderStand attribute เป็น "1" แต่ถ้าเป็น "0" จะยังคงประมวลผลต่อไป

**9.3 SOAP Fault Element**

ข้อความแสดงความผิดพลาดจากแอพพลิเคชันของ SOAP จะเก็บอยู่ใน fault element ซึ่งถ้ามีจะต้องปรากฏใน body element เพียงครั้งเดียวใน SOAP เมสเซจ ตัวอย่างอาจเป็นดังนี้

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
          soap:encodingStyle="http://schemas.xmlsoap.org/soap/
                    encoding/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:MustUnderstand</faultcode>
      <faultstring>Mandatory Header error.</faultstring>
      <faultactor>http://www.wrox.com/heroes/endpoint.asp</faultactor>
      <detail>
        <w:source xmlns:w="http://www.wrox.com/">
          <module>endpoint.asp</module>
          <line>203</line>
        </w:source>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

SOAP fault element มี element ย่อย ๆ ดังตารางที่ 9-1 และค่าของ faultcode สามารถมีค่าได้ดังตารางที่ 9-2

| Element ย่อย | คำอธิบาย |
|---|---|
| <faultcode> | โค้ดที่ระบุถึงการ error |
| <faultstring> | ข้อความการ error |
| <faultactor> | ใครเป็นสาเหตุของการ error |
| <detail> | ระบุอินฟอร์เมชันของการ error |

***ตารางที่ 9-1 SOAP Fault Element***

| ความผิดพลาด | คำอธิบาย |
|---|---|
| VersionMismatch | namespace ภายใน SOAP Envelope element ไม่ถูกต้อง |
| MustUnderstand | child element ของ Header element กับ mustUnderstand attribute ที่มีค่า "1" ผู้รับไม่รองรับ |
| Client | เมสเซจมีรูปแบบไม่ถูกต้อง หรือมีอินฟอร์เมชันที่ไม่ถูกต้อง |
| Server | เกิดปัญหากับเซิร์ฟเวอร์ ไม่สามารถประมวลผลได้ |

***ตารางที่ 9-2 ค่า Faultcode***

**9.4 SOAP Encoding**

SOAP encoding มีวิธีการแมปจาก type ของโปรแกรมมิ่งไปเป็น XML 2 วิธี คือจากภายนอกโดยใช้ WSDL (Web Services Description Language) ที่บอกถึง type ของข้อมูลที่รับหรือส่ง หรือใช้ xsi:type attribute ในกรณีที่ภาษาที่ใช้ไม่รองรับ WSDL โดยทั้ง 2 วิธีจะใช้ XML schema ในการระบุ type ตัวอย่างของการใช้ xsi จะเป็นดังนี้

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
          soap:encodingStyle="http://schemas.xmlsoap.org/soap
                         /encoding/"
          xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <soap:Body>
    <m:MixedMessage xmlns:m="http://www.wrox.com/mix/">
```

```
      <param1 xsi:type="xsd:string">OU812</param1>
      <param2 xsi:type="xsd:integer">2001</param2>
      <param3 xsi:type="xsd:double">3.14159</param3>
    </m:MixedMessage>
  </soap:Body>
</soap:Envelope>
```

SOAP รองรับ type ของข้อมูลได้ทั้ง simple type และ compound type เช่น struct และ array

**9.4.1 ตัวอย่างการแมป Struct**

struct book ที่เขียนด้วยภาษา c++ จะเป็นดังนี้

```
struct Book
{
    string author;
    string name;
    int page;
};
```

Soap เมสเซจที่ไม่ใช้ WSDL แต่ใช้ xsi จะเป็นดังนี้

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:searchBook xmlns:ns1="http://soapinterop.org/" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<bookResult xmlns:ns2="http://soapinterop.org/xsd" xsi:type="ns2:Book">
  <author xsi:type="xsd:string">Ed Roman</author>
  <name xsi:type="xsd:string">Mastering EJB</intro>
  <page xsi:type="xsd:int">500</page>
  </bookResult>
```

```
 </ns1:searchBook>
</SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

แต่ถ้าใช้ WSDL ในส่วนของการประกาศ schema ของ WSDL จะต้องประกอบด้วย schema ดังนี้

```
<types>
  <schema targetNamespace='http://soapinterop.org/xsd'
   xmlns='http://www.w3.org/2001/XMLSchema'
   xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
   xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
   elementFormDefault='qualified'>
     <complexType name="Book">
        <sequence>
           <element name='author' type='string'/>
           <element name='name' type='string'/>
           <element name='page' type='int'/>
        </sequence>
     </complexType>
  </schema>
</types>
```

รายละเอียดของ WSDL จะใช้บอกรายละเอียดการส่ง SOAP request และ SOAP response รวมถึงการแมป type ด้วย

**9.4.2 ตัวอย่างการแมป Array**

ตัวอย่างนี้เป็น array ของ struct book ในตัวอย่างก่อน Soap เมสเซจที่ไม่ใช้ WSDL แต่ใช้ xsi จะเป็นดังนี้

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/1999/XMLSchema">
```

```
<SOAP-ENV:Body>
    <ns1:BookArrayResult xmlns:ns1="http://soapinterop.org/" SOAP-
    ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <BookArray xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
        xsi:type="ns2:Array" xmlns:ns3="http://soapinterop.org/xsd"
        ns2:arrayType="ns3:Book[3]">
          <item xsi:type="ns3:Book">
              <author xsi:type="xsd:string">Ed Roman</author>
              <name xsi:type="xsd:string">Mastering EJB</name>
              <page xsi:type="xsd:int">500</page>
          </item>
          <item xsi:type="ns3:Book">
              <author xsi:type="xsd:string">Roger Session</author>
              <name xsi:type="xsd:string">The Battle of the  middleware</name>
              <page xsi:type="xsd:int">350</page>
          </item>
          <item xsi:type="ns3:Book">
              <author xsi:type="xsd:string">Chris Dix</author>
              <name xsi:type="xsd:string">Professional XML Web Service</name>
              <page xsi:type="xsd:int">550</page>
          </item>
      </BookArray>
    </ns1:BookArrayResult>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

แต่ถ้าใช้ WSDL ในส่วนของการประกาศ schema ของ WSDL จะต้องประกอบด้วย schema ดังนี้

```
<types>
  <schema targetNamespace='http://soapinterop.org/xsd'
   xmlns='http://www.w3.org/2001/XMLSchema'
```

```
    xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
    xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
    elementFormDefault='qualified'>
      <complexType name="Book">
         <sequence>
            <element name='author' type='string'/>
            <element name='name' type='string'/>
            <element name='page' type='int'/>
         </sequence>
      </complexType>
      <complexType  name ='ArrayOfBook'>
      <complexContent>
         <restriction base='SOAP-ENC:Array'>
            <attribute ref='SOAP-ENC:arrayType' wsdl:arrayType='typens:Book[]'/>
         </restriction>
     </complexContent>
   </complexType>
  </schema>
</types>
```

### 9.5 SOAP ใน HTTP

ในการส่ง SOAP ผ่านทาง HTTP จะต้องใช้ content-type เป็น text/xml แต่ใน SOAP request จะต้องมี header SOAPAction ภายใน HTTP header SOAPAction จะเป็นตัวบอกให้เซิร์ฟเวอร์รู้ว่า HTTP Post นั้นเป็น SOAP เมสเซจ และค่าของ header คือ URI ที่แสดงถึงจุดหมายของ SOAP เมสเซจ ส่วน SOAP response จะต้องมี status code ตามมาตรฐานของ HTTP โดย 200-299 แสดงว่าสำเร็จ แต่ถ้า response เมสเซจเป็นการ fault แล้ว status code จะต้องเป็น 500 ซึ่งแสดงถึง internal server error ตัวอย่างของ response ที่มี status code เป็น 500 อาจเป็นดังนี้

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml
Content-Length: ###
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
            soap:encodingStyle="http://schemas.xmlsoap.org/soap
                        /encoding/"
            xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:VersionMismatch</faultcode>
        <faultstring>The SOAP namespace is incorrect.</faultstring>
      <faultactor>http://www.wrox.com/endpoint.asp</faultactor>
      <detail>
        <w:errorinfo xmlns:w="http://www.wrox.com/">
          <desc>The SOAP namespace was blank.</desc>
        </w:errorinfo>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

### 9.6 SOAP สำหรับ RPC

จุดประสงค์ของการออกแบบ SOAP คือทำ RPC โดยใช้ XML การเรียก RPC นั้นจะแมปเข้ากับ HTTP request และ RPC response จะแมปกับ HTTP response

ในการทำ mechod call จำเป็นที่จะต้องมีอินฟอร์เมชันดังนี้

1) URI ของออบเจ็กต์ปลายทาง (target object)
2) ชื่อของเมธอด
3) คำอธิบายของเมธอด (method signature) ส่วนนี้เป็นออปชัน
4) พารามิเตอร์สำหรับเมธอด
5) ข้อมูลส่วนหัว (header data) เป็นออปชัน

พิจารณาเมธอดดังนี้

```
double GetStockQuote ( [in] string sSymbol);
```

ถ้า namespace ของเมธอดคือ “http://www.worxstox.com/” แล้วการเรียกเมธอดโดย request stock quote ใช้ symbol OU812 จะเป็นดังนี้

```
<q:GetStockQuote xmlns:q="http://www.wroxstox.com/">
  <q:sSymbol xsi:type="xsd:string">OU812</q:sSymbol>
</q:GetStockQuote>
```

ชื่อเมธอดและชื่อของ element จะต้อง match กันเช่นเดียวกับพารามิเตอร์

สำหรับ response จะต้องชื่อ element จะต้องเป็นชื่อเมธอดตามด้วย response ดังนี้

```
<q:GetStockQuoteResponse xmlns:q="http://www.wroxstox.com/"
    <q:ret xsi:type="xsd:double">100.0</q:ret>
</q:GetStockQuoteResponse>
```

ชื่อ element ของค่าที่คืนมาสามารถเปลี่ยนได้ โดยจะกำหนดได้จากโปรแกรมที่เขียนหรือจาก WSDL

**9.7 SOAP Toolkit**

SOAP toolkit คือเครื่องมือที่จะทำหน้าที่ในการประมวลผล SOAP request และส่ง response กลับไปให้ไคลเอ็นต์ โดย SOAP toolkit แต่ละตัวใช้ได้บนแพล็ตฟอร์มที่ต่างกัน รองรับเทคโนโลยี web service ต่างกัน บางตัวอาจจะรองรับ WSDL หรือ UDDI ในขณะที่บางตัวรองรับอย่างใดอย่างหนึ่งหรือไม่รองรับทั้งสองอย่าง และถึงแม้ว่า SOAP จะเป็นอิสระต่อแพล็ตฟอร์ม แต่ใน SOAP toolkit บางตัวยังไม่สามารถทำงานข้ามผลิตภัณฑ์ (interoperability) ได้ เนื่องจากรองรับเทคโนโลยีของ web service ไม่เหมือนกัน เช่น xsi ในบางผลิตภัณฑ์จะบังคับให้ SOAP เมสเซจจะต้องระบุ type ด้วย xsi หากไม่มีก็จะ falut ซึ่งในปัจจุบันแต่ละผลิตภัณฑ์ก็พยายามพัฒนาให้สามารถทำงานด้วยกันได้ นอกจากนั้นการแมป type ยังไม่ตรงกันทั้งหมดเช่นในกรณีของ date SOAP toolkit แต่ละตัวจะแมป type ระหว่างโปรแกรมมิ่งกับ xsi ไม่ตรงกัน ทำให้ไม่สามารถทำงานด้วยกันได้ ปัจจุบันได้มี SOAP toolkit ออกมาหลายตัว แต่ละตัวมีฟีเจอร์ต่างกัน บางผลิตภัณฑ์สามารถสร้าง WSDL ให้โดยอัตโนมัติจากซอร์สโค้ดหรือจากคลาสที่คอมไพล์แล้ว แต่ก็ยังมีปัญหาในส่วนที่ว่า WSDL ที่สร้างจากผลิตภัณฑ์หนึ่งไม่สามารถทำงานได้ในอีกผลิตภัณฑ์หนึ่ง ตัวอย่างของ SOAP toolkit ดังตารางที่ 9-3 โดยในโครงงานนี้จะใช้ Apache SOAP 2.2 ที่รองรับภาษาจาวาทำงานข้ามผลิตภัณฑ์กับ Visual Studio .NET ของไมโครซอฟท์ รายละเอียดการติดตั้ง Apache SOAP 2.2 ดูได้ในภาคผนวก ก

| ชื่อ | แพล็ตฟอร์ม | แหล่งที่มา |
|---|---|---|
| 4s4c | COM | http://www.4s4c.com |
| A SOAP for RPC NT Service | COM | http://www.whitemesa.com |
| Apache Axis | Java | http://xml.apache.org/axis |
| Apache SOAP | Java | http://xml.apache.org/soap |
| CapeConnect | Java | http://www.capeclear.com |
| Glue | Java | http://www.themindelectric.com |
| IBM Web Services Toolkit | Java | http://www.alphaworks.ibm.com |
| IdooXoap for Java and C++ | Java , C++ | http://www.zvon.org |
| Microsoft SOAP Toolkit | COM | http://www.microsoft.com |
| pocketSOAP | COM | http://www.pocketsoap.com |
| SOAP::Lite | Perl | http://www.soaplite.com |
| Visual Studio .NET | COM | http://www.microsoft.com |

***ตารางที่ 9-3 SOAP Toolkit***

# บทที่ 10
# คลัสเตอร์

## 10.1 แนะนำคลัสเตอร์

คลัสเตอร์ (cluster) คือการทำ multiprocessing แบบหนึ่งที่ใช้เมโมรีแยกส่วนกัน การคลัสเตอร์แอพพลิเคชันเซิร์ฟเวอร์คือการทำงานร่วมกันของแอพพลิเคชันเซิร์ฟเวอร์ แต่เนื่องจาก J2EE specification ไม่ได้ระบุรายละเอียดการทำคลัสเตอร์ไว้ แต่ละผลิตภัณฑ์จึงมีสถาปัตยกรรมและการจัดการคลัสเตอร์ต่างกัน เช่น ในส่วนของออบเจ็กต์เทียร์ (Object tier) บางผลิตภัณฑ์เป็นแบบ centralize บางผลิตภัณฑ์เป็น distribute ดังนั้นในบทนี้จะกล่าวถึงเฉพาะวิธีการคลัสเตอร์ของ WebLogic เซิร์ฟเวอร์เท่านั้น

เนื่องจากในบทนี้จะมีการใช้คำในความหมายที่ต่างจากเดิม โดยเฉพาะในเรื่องเทียร์ ในการคลัสเตอร์จะแบ่งเทียร์ทางความคิดต่างจากเดิม โดยจะแบ่งเทียร์ในเว็บแอพพลิเคชัน จึงจำเป็นต้องนิยามความหมายดังนี้

เว็บแอพพลิเคชัน (Web Application) จะถูกแบ่งเป็นหลายเทียร์ที่จะอธิบายถึง ลอจิกคอลเซอร์วิสที่แอพพลิเคชันมี ดังนี้

- **Web Tier** - เว็บเทียร์เป็นส่วนที่จัดการกับสแตติกคอนเท็นต์เช่น HTML
- **Presentation Tier** - พรีเซนเทชันเทียร์เป็นส่วนที่จัดการกับไดนามิกคอนเท็นต์เช่น Servlet หรือ JSP
- **Object Tier** - ออบเจ็กต์เทียร์เป็นส่วนที่จัดการกับจาวาออบเจ็กต์ได้แก่ EJB และ RMI คลาส

## 10.2 การคลัสเตอร์ WebLogic เซิร์ฟเวอร์

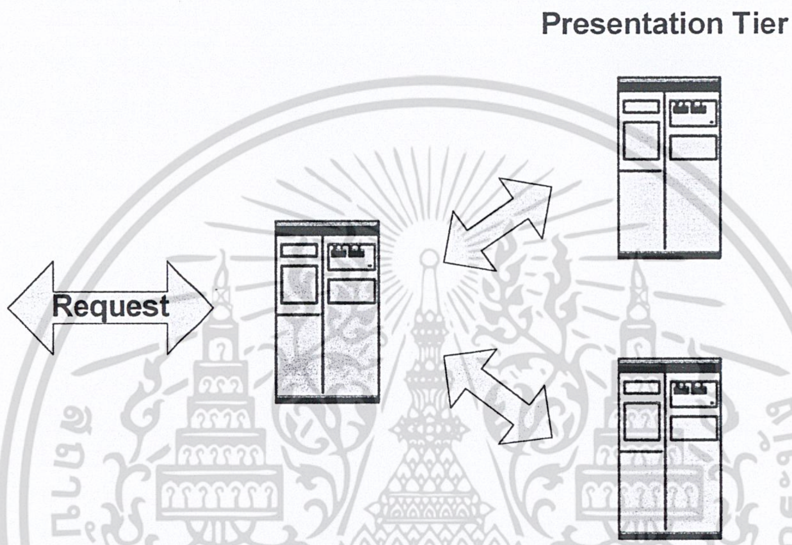
WebLogic เซิร์ฟเวอร์คลัสเตอร์เป็นกลุ่มของ WebLogic เซิร์ฟเวอร์ที่ทำงานร่วมกัน แต่ไคลเอ็นต์จะเห็นคลัสเตอร์เป็นเหมือนเป็นเซิร์ฟเวอร์ตัวเดียว คลัสเตอร์มีความสามารถที่เหนือกว่าเซิร์ฟเวอร์ที่ทำงานเครื่องเดียว ดังนี้

- **การเพิ่มขนาด (Scalability)** - ความสามารถของคลัสเตอร์ไม่ถูกจำกัดอยู่ที่จำนวนเซิร์ฟเวอร์ที่ทำงานเครื่องเดียว การเพิ่มเซิร์ฟเวอร์เครื่องใหม่จะช่วยเพิ่มความสามารถในการทำงาน ทำให้สามารถรองรับการทำงานภายใต้ผู้ใช้บริการเป็นจำนวนมาก ๆ ได้ และเป็นการเพิ่มแบบไดนามิกคือ หากเซิร์ฟเวอร์ไม่เพียงพอต่อการให้บริการ ก็สามารถเพิ่มเซิร์ฟเวอร์เข้าไปได้ทันที (ต้องไม่เกินขอบเขตความสามารถของคลัสเตอร์)
- **การคงเหลือ (High-Availability)** - คลัสเตอร์ใช้เซิร์ฟเวอร์หลาย ๆ ตัวที่ทำงานเหมือนกัน (redundancy) ในการป้องกันการขัดข้อง (failure) ของเซิร์ฟเวอร์ เซอร์วิสเดียวกันสามารถถูกเพิ่มบนเซิร์ฟเวอร์หลาย ๆ ตัวในคลัสเตอร์ได้ ถ้าเซิร์ฟเวอร์ตัวหนึ่งขัดข้อง ก็ยังเหลือเซิร์ฟเวอร์ตัวอื่นที่ให้บริการได้ รวมถึงการ failover ซึ่งเป็นกระบวนการในการย้ายการทำงานจากเครื่องหนึ่งไปอีกเครื่อง

หนึ่ง เมื่อเกิดการขัดข้องในเครื่องแรก โดยจะเกิดขึ้นโดยอัตโนมัติ ไคลเอ็นต์จะไม่รับรู้การขัดข้องของเซิร์ฟเวอร์ที่เกิดขึ้น

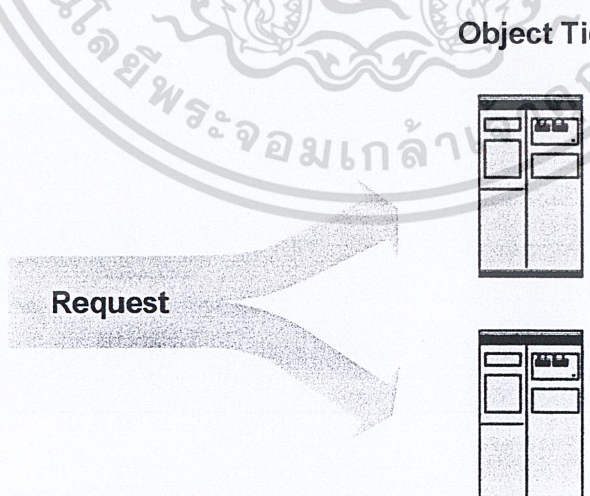
เซอร์วิสที่สามารถนำมาคลัสเตอร์ได้ประกอบด้วย

- **HTTP Session State** – เป็นการคลัสเตอร์พรีเซนเทชันเทียร์ (Presentation tier) ซึ่งได้แก่ Servlet และ JSP การกระจายโหลดไปยังพรีเซนเทชันเทียร์เป็นแบบ centralize ดังรูปที่ 10-1




*รูปที่ 10-1 การกระจายโหลดไปยังพรีเซนเทชันเทียร์*

- **EJB และ RMI ออบเจ็กต์** - การกระจายโหลดไปยังออบเจ็กต์เทียร์เป็นแบบ distribute ดังรูปที่ 10-2




*รูปที่ 10-2 การกระจายโหลดไปยังออบเจ็กต์เทียร์*

## 10.3 โครงสร้างพื้นฐานของคลัสเตอร์

### 10.3.1 การสื่อสารของเซิร์ฟเวอร์

การติดต่อสื่อสารของเซิร์ฟเวอร์ อินสแทนซ์ของ WebLogic เซิร์ฟเวอร์ในคลัสเตอร์จะสื่อสารกับเครื่องอื่น ๆ โดยใช้เน็ตเวิร์กเทคโนโลยีพื้นฐานสองชนิดดังนี้

- **IP มัลติคาสต์ (IP multicast)** – เป็นการสื่อสารแบบบรอดคาสต์ (broadcast)
- **IP ซ็อกเก็ต (IP socket)** - เป็นการสื่อสารแบบเพียร์ทูเพียร์ (peer-to-peer) ระหว่างอินสแทนซ์ของเซิร์ฟเวอร์ที่นำมาคลัสเตอร์

❑ **IP มัลติคาสต์**

IP มัลติคาสต์ใช้วิธีง่าย ๆ ในการบรอดคาสต์เมสเซจไปยังแอพพลิเคชัน แต่มันไม่รับรองว่าเมสเซจนั้นจะถึงผู้รับหรือไม่ เช่น ถ้ามัลติคาสต์บัฟเฟอร์เต็ม มัลติคาสต์เมสเซจใหม่จะไม่สามารถถูกเขียนลงบนบัฟเฟอร์และแอพพลิเคชันก็จะไม่รู้ด้วยว่าเมสเซจนั้นถูกทิ้งไป (drop)

WebLogic เซิร์ฟเวอร์ใช้ IP มัลติคาสต์ในการกระทำดังนี้

- **อัพเดต JNDI ของคลัสเตอร์ (Cluster-wide JNDI)** – เซิร์ฟเวอร์ทั้งหมดจะใช้มัลติคาสต์ในการประกาศออบเจ็กต์ที่ถูกคลัสเตอร์ ซึ่งมันอาจจะถูกดีพลอย (deploy) ลงไปหรือถอนออกไป (remove) ในเวลาใดก็ได้ เซิร์ฟเวอร์จะตรวจสอบการประกาศแล้วนำมาอัพเดต JNDI ทรี
- **"heartbeat"** – จะใช้มัลติคาสต์ในการบรอดคาสต์ "heartbeat" เมสเซจ เพื่อประกาศว่า เซิร์ฟเวอร์เครื่องนั้นยังทำงานอยู่ ซึ่งเซิร์ฟเวอร์ทั้งหมดจะคอยฟัง heartbeat เมสเซจ ในการที่จะนำมาตัดสินใจว่าเมื่อใดที่เซิร์ฟเวอร์เครื่องใดขัดข้อง

เนื่องจากการคลัสเตอร์จะใช้ heartbeat ในการตัดสินว่าเซิร์ฟเวอร์เครื่องใดขัดข้อง ดังนั้นในการคลัสเตอร์จึงต้องพิจารณาถึงสิ่งต่าง ๆ ต่อไปนี้

- **คลัสเตอร์ผ่าน WAN (Wide Area Network)**

ถ้าเลือกที่จะคลัสเตอร์ผ่าน WAN (ผ่านหลาย ๆ ซับเน็ต) จะต้องวางแผนและคอนฟิก เน็ตเวิร์ก topology เพื่อให้แน่ใจได้ว่ามัลติคาสต์เมสเซจจะส่งถึงเซิร์ฟเวอร์ทั้งหมดในคลัสเตอร์ โดยจะต้องเป็นไปตามความต้องการดังนี้

1) เน็ตเวิร์กต้องสนับสนุน IP มัลติคาสต์ หรือพูดอีกอย่างได้ว่าเราเตอร์ (router) ทั้งหมดและทันแนล (tunnel) อื่น ต้องถูกคอนฟิกเพื่อที่จะส่งมัลติคาสต์ไปยังอินสแทนซ์ของเซิร์ฟเวอร์ที่คลัสเตอร์ไว้ได้
2) เน็ตเวิร์ก latency ต้องไม่มากเกินไป ต้องแน่ใจว่ามัลติคาสต์เมสเซจจะต้องถึงปลายทางสุดท้ายภายในเวลา 200-300 มิลลิวินาที
3) ค่ามัลติคาสต์ Time To Live (TTL) ต้องมากพอที่จะมั่นใจว่าเราเตอร์จะไม่เลิกส่งมัลติคาสต์แพ็กเก็ตก่อนที่จะถึงปลายทางสุดท้าย

- **คลัสเตอร์ผ่านไฟร์วอลล์**

ถึงแม้ว่ามันอาจจะเป็นไปได้ที่จะมัลติคาสต์ผ่านไฟร์วอลล์ (firewall) แต่ไม่ควรทำ แต่ละ WebLogic เซิร์ฟเวอร์คลัสเตอร์ควรจะถูกทำเป็นกลุ่มเดียว (logical unit) ที่ไม่ถูกแบ่งให้อยู่คนละส่วนของความปลอดภัย (security zone) ยิ่งกว่านั้นเทคโนโลยีที่สามารถหน่วงเวลา (delay) หรืออินเทอร์รัปต์การสื่อสารผ่าน IP สามารถทำลายการคลัสเตอร์ โดยทำให้การวิเคราะห์การขัดข้องของเซิร์ฟเวอร์ผิดพลาด เนื่องจาก heartbeat หายไป

- **ใช้มัลติคาสต์แอดเดรสเฉพาะสำหรับคลัสเตอร์**

แม้การคลัสเตอร์สามารถที่จะใช้ IP มัลติคาสต์แอดเดรสและหมายเลขพอร์ตร่วมกันกับแอพพลิเคชันอื่น แต่ก็ไม่ควรที่จะทำเช่นนั้น เพราะจะทำให้ต้องประมวลผลเมสเซจโดยไม่จำเป็น ทั้งยังสิ้นเปลืองมัลติคาสต์บัฟเฟอร์และเป็นการหน่วงเวลาการส่ง heartbeat เมสเซจ

- **IP ซ็อกเก็ต**

อินสแทนซ์ของ WebLogic เซิร์ฟเวอร์ในคลัสเตอร์อาจจะใช้ IP ซ็อกเก็ตสำหรับ

- เข้าถึงออบเจ็กต์ที่ไม่ได้คลัสเตอร์ที่อยู่บนอินสแทนซ์ของเซิร์ฟเวอร์อื่นในคลัสเตอร์
- Replicate HTTP session state และ stateful session EJB state ระหว่าง primary และ secondary เซิร์ฟเวอร์ ในกรณีที่ primary เซิร์ฟเวอร์ขัดข้อง secondary เซิร์ฟเวอร์จะสามารถเข้ามาทำงานแทนได้
- เข้าถึงออบเจ็กต์ที่คลัสเตอร์ที่อยู่บนอินสแทนซ์ของเซิร์ฟเวอร์อื่นในคลัสเตอร์

ในการสื่อสารด้วย IP ซ็อกเก็ต การคอนฟิกซ็อกเก็ตเป็นสิ่งสำคัญต่อสมรรถนะ (performance) ของ WebLogic เซิร์ฟเวอร์คลัสเตอร์ ซึ่งมีตัวแปร (factor) ที่ต้องพิจารณา ดังนี้

- ระบบของเซิร์ฟเวอร์ใช้ซ็อกเก็ต reader ของจาวา (pure-Java) หรือของระบบปฏิบัติการ (native)
- สำหรับระบบที่ใช้จาวาซ็อกเก็ต reader เซิร์ฟเวอร์ได้ถูกคอนฟิกให้ใช้ซ็อกเก็ต reader เธรดที่เพียงพอหรือไม่

เนื่องจากกลไกในการอ่านค่าจากซ็อกเก็ตของซ็อกเก็ต reader ทั้ง 2 ชนิดนั้นต่างกัน ซ็อกเก็ต reader ของจาวา จะใช้วิธีวนตรวจสอบว่ามีข้อมูลเข้ามาหรือไม่ ดังนั้นถ้าซ็อกเก็ตที่เปิดมีจำนวนมากกว่าซ็อกเก็ต reader ข้อมูลจะไม่ถูกอ่านทันทีเพราะต้องวนตรวจสอบทุก ๆ ซ็อกเก็ตที่เปิด ดังนั้นจึงควรคอนฟิกให้มีจำนวนซ็อกเก็ต reader เท่ากับซ็อกเก็ตที่เปิดหรือเพียงพอ ในขณะที่ซ็อกเก็ต reader ของระบบปฏิบัติการจะมีประสิทธิภาพที่ดีกว่า เพราะมันจะรู้ว่ามีข้อมูลเข้ามาด้วยอินเทอร์รัปต์ จึงสามารถอ่านข้อมูลได้ทันที

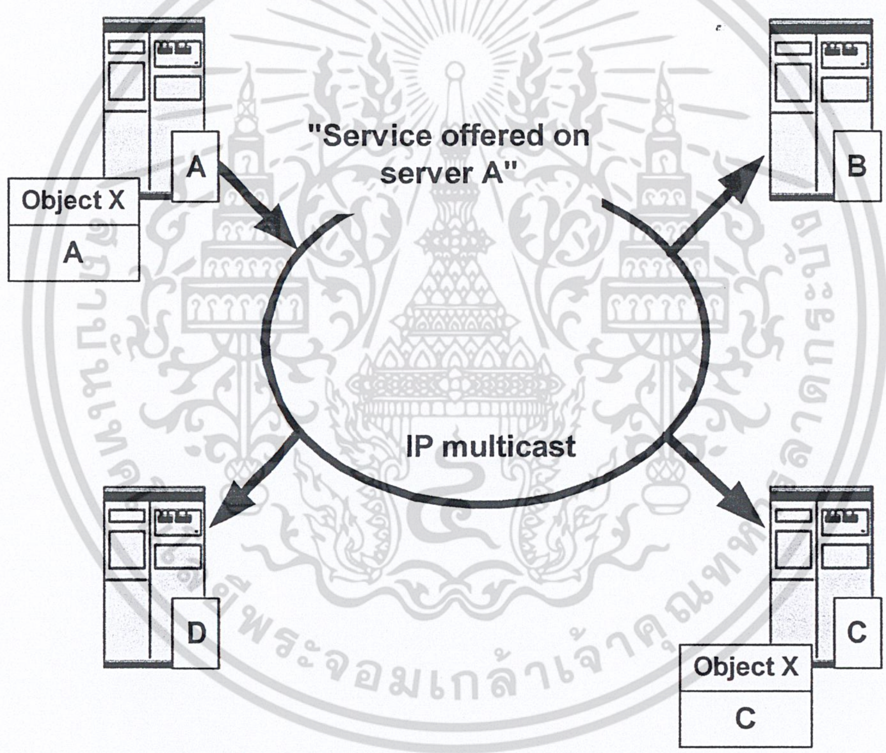
**10.3.2 JNDI ของคลัสเตอร์**

J2EE จะใช้ JNDI ในการระบุถึงเซอร์วิสที่ให้บริการ โดยจะมีโครงสร้างเป็นทรี WebLogic เซิร์ฟเวอร์จะให้บริการเซอร์วิสโดย bind มันลงใน JNDI ทรี ไคลเอ็นต์จะรับบริการโดยการติดต่อไปยังเซิร์ฟเวอร์และ look up ชื่อของเซอร์วิสที่ต้องการ

แต่ในคลัสเตอร์ อินสแทนซ์ของเซิร์ฟเวอร์ในคลัสเตอร์จะใช้ JNDI ทรีของคลัสเตอร์ (cluster-wide JNDI tree) ที่แสดงรายการของเซอร์วิสที่ให้บริการโดยสมาชิกทั้งหมดในคลัสเตอร์

การสร้าง JNDI ทรีของคลัสเตอร์ แต่ละ WebLogic เซิร์ฟเวอร์ในคลัสเตอร์จะสร้างและดูแล (maintain) JNDI ทรีของคลัสเตอร์บนเครื่องมันเอง เมื่อเซิร์ฟเวอร์บูต (boot) หรือดีพลอย (deploy) เซอร์วิสใหม่ลงไป เซิร์ฟเวอร์นั้นจะ bind เซอร์วิสลงบนโลคอล JNDI ทรี แล้วจึงส่ง stub ของออบเจ็กต์ไปยังสมาชิกอื่นในคลัสเตอร์ สมาชิกอื่นจะคอยตรวจสอบมัลติคาสต์แอดเดรสเพื่อตรวจดูว่าเมื่อใดที่เซิร์ฟเวอร์เครื่องอื่นจะมีเซอร์วิสใหม่

ตัวอย่างเซิร์ฟเวอร์ A จะ bind คลัสเตอร์ออบเจ็กต์ X ในโลคอล JNDI ทรีเสร็จแล้ว เซิร์ฟเวอร์ C ยังคงประมวลผลการ bind ออบเจ็กต์เดียวกัน (รูปที่ 10-3)

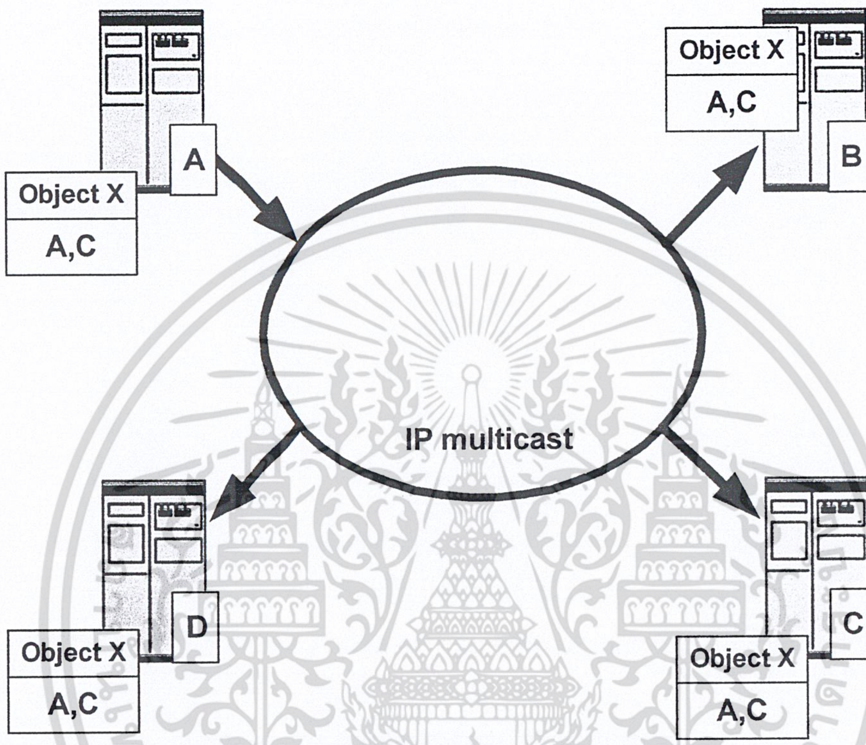



*รูปที่ 10-3 JNDI ทรีของคลัสเตอร์ก่อนการอัพเดต*

เซิร์ฟเวอร์อื่นจะคอยฟังมัลติคาสต์แอดเดรสจากเซิร์ฟเวอร์ A ที่มีเซอร์วิสใหม่คือออกเจ็กต์ X โดยที่การอัพเดตโลคอล JNDI ที่เกิดจากการ bind เกิดขึ้นได้โดย 1 ใน 2 วิธีนี้

- ถ้าเซอร์วิสที่คลัสเตอร์ยังไม่ bind ในโลคอล JNDI ทรี เซอร์วิสจะ bind replica-aware stub ใหม่ลงในโลคอลทรี เพื่อแสดงว่ามีคลัสเตอร์ออบเจ็กต์ X บนเซิร์ฟเวอร์ A เซิร์ฟเวอร์ B และ D อัพเดตโลคอล JNDI ทรีของพวกมันด้วยวิธีนี้ เพราะคลัสเตอร์ออบเจ็กต์ไม่ได้ถูกดีพลอยบนเซิร์ฟเวอร์เหล่านั้น

- ถ้ามีการ bind เซอร์วิสนั้นแล้ว มันจะอัพเดตโลคอล JNDI ทรี เพื่อแสดง replica ของเซอร์วิสว่ามีอยู่บนเซิร์ฟเวอร์ A ด้วย เซิร์ฟเวอร์ C จะอัพเดต JNDI ทรี ด้วยวิธีนี้ เพราะมันมีการ bind คลัสเตอร์ออบเจ็กต์ X อยู่แล้ว รูปที่ 10-4 แสดงภายหลังการอัพเดต JNDI เสร็จสิ้น




*รูปที่ 10-4 JNDI ทรีของคลัสเตอร์ ภายหลังอัพเดต*

**10.3.3 Load Balancing เซอร์วิสที่คลัสเตอร์**

พื้นฐานของแนวคิดของ load balancing คือการกระจายโหลด (distribute load) ระหว่างเซิร์ฟเวอร์ในคลัสเตอร์

- **Load Balancing สำหรับ HTTP Session State**

Load balancing สำหรับ Servlet และ JSP HTTP session state สามารถถูกสร้างจาก load balancing ฮาร์ดแวร์หรือใช้ความสามารถของ built-in load balancing ของ WebLogic proxy plug-in

สำหรับคลัสเตอร์ที่ใช้เว็บเซิร์ฟเวอร์และ WebLogic proxy plug-in proxy plug-in จะใช้ round-robin อัลกอริทึมสำหรับกระจาย request ไปยัง Servlet และ JSP ในคลัสเตอร์

ส่วนคลัสเตอร์ที่ใช้ load balancing ฮาร์ดแวร์ (load balancer) สามารถใช้อัลกอริทึมใด ๆ ที่รองรับด้วยฮาร์ดแวร์

■ **Load Balancing สำหรับคลัสเตอร์ออบเจ็กต์**

WebLogic เซิร์ฟเวอร์คลัสเตอร์สนับสนุนอัลกอริทึมดังนี้ สำหรับการ load balancing คลัสเตอร์ออบเจ็กต์

1) **Round-Robin (ดีฟอลต์)**

ข้อดีของอัลกอริทึมนี้คือง่าย ถูกและคาดเดาได้ง่าย ข้อเสียจะเกิดขึ้นเมื่อมีเซิร์ฟเวอร์ที่ทำงานได้ช้ากว่าเครื่องอื่น

2) **Weight-Based**

ถ้าใช้อัลกอริทึมนี้จะต้องเสียเวลาในการคำนวณความสัมพันธ์ของค่าน้ำหนัก (weight) ปัจจัยที่มีผลต่อการกำหนดค่าน้ำหนักให้เซิร์ฟเวอร์ประกอบด้วย

- ความสามารถทางด้านฮาร์ดแวร์ของเซิร์ฟเวอร์เมื่อเทียบกับเครื่องอื่น
- จำนวนของออบเจ็กต์ที่ไม่ได้คลัสเตอร์ของแต่ละเซิร์ฟเวอร์ที่เป็นโฮสต์

3) **Random**

วิธีนี้แนะนำให้ใช้กับการคลัสเตอร์ที่แต่ละเซิร์ฟเวอร์มีความสามารถเท่ากัน และให้เซอร์วิสเดียวกันเท่านั้น ข้อดีของมันคือง่าย ข้อเสียของมันคือเสียเวลาในการสร้างเลขสุ่มสำหรับทุก ๆ request

**10.3.4 Failover สำหรับคลัสเตอร์เซอร์วิส**

Failover คือกระบวนการในการย้ายการทำงานจากเครื่องหนึ่งไปอีกเครื่องหนึ่ง เมื่อเครื่องแรกเกิดการขัดข้อง โดยคลัสเตอร์สามารถ failover ได้เนื่องจากได้ replicate state ที่เซิร์ฟเวอร์อีกเครื่องหนึ่ง ถ้าเครื่องแรกขัดข้อง เครื่องที่ replicate state ไว้ก็จะเข้ามาทำแทน

ในส่วนนี้จะกล่าวถึงว่า WebLogic เซิร์ฟเวอร์ ตรวจสอบความขัดข้องในคลัสเตอร์อย่างไรและ failover ทำงานอย่างไร

■ **วิธีตรวจสอบการขัดข้องของ WebLogic เซิร์ฟเวอร์**

อินสแทนซ์ของ WebLogic เซิร์ฟเวอร์ในคลัสเตอร์จะตรวจสอบการขัดข้องด้วยการมอนิเตอร์

1) การสื่อสารผ่านซ็อกเก็ตไปยังเพียร์เซิร์ฟเวอร์ (peer server)
2) Heartbeat เมสเซจ

■ **Failover สำหรับคลัสเตอร์ Servlet and JSP**

Proxy plug-in จะจัดการ failover โดยที่ไคลเอ็นต์ไม่รู้ ถ้าเซิร์ฟเวอร์ที่ให้บริการขัดข้อง plug-in จะกำหนดตำแหน่งของ replication HTTP session state บน secondary เซิร์ฟเวอร์และเปลี่ยนทิศทางการส่ง request ของไคลเอ็นต์ไปยัง secondary เซิร์ฟเวอร์

สำหรับคลัสเตอร์ที่ใช้ load balancing ฮาร์ดแวร์จะเปลี่ยนทิศทางการส่ง request ของไคลเอ็นต์ไปยังเซิร์ฟเวอร์ที่เหลือในคลัสเตอร์และจะเอา replica ของ HTTP session state ของไคลเอ็นต์นั้น จาก secondary เซิร์ฟเวอร์ในคลัสเตอร์

- **Failover สำหรับคลัสเตอร์ออบเจ็กต์**

สำหรับคลัสเตอร์ออบเจ็กต์ failover ถูกจัดการโดยใช้ replica-aware stub ของออบเจ็กต์ เมื่อไคลเอ็นต์เรียกผ่าน replica-aware stub ไปยังเซอร์วิสที่ขัดข้อง stub จะพบความผิดพลาดและจะพยายามเรียกไปยัง replica อื่น

**- Idempotent Object**

กับคลัสเตอร์ออบเจ็กต์ failover อัตโนมัติจะเกิดขึ้นก็ต่อเมื่อออบเจ็กต์นั้นเป็น idempotent โดยออบเจ็กต์จะเป็น idempotent ถ้าเมธอดใด ๆ สามารถถูกเรียก (call) ได้หลายครั้งโดยไม่มีผลต่างจากการเรียกครั้งเดียว เช่น บริการของรถเข็นแบบออนไลน์เรียก addItem() เพื่อเพิ่มไอเท็ม (item) ไปยังรถเข็นแบบออนไลน์ สมมติให้ ไคลเอ็นต์ C เรียกกับ replica บนเซิร์ฟเวอร์ S1 ภายหลัง S1 ได้รับการเรียก แต่ก่อนที่มันจะคืนค่า (return) ผลลัพธ์กลับไปยัง C S1 ได้แครช (crash) ที่จุดนี้ไอเท็มได้มีการเพิ่มไปแล้วแต่ replica-aware stub จะได้รับ exception ถ้า stub พยายามที่จะทำเมธอดนี้อีกครั้งบนเซิร์ฟเวอร์ S2 จะทำให้ไอเท็มถูกเพิ่มเป็น 2 ครั้ง ด้วยเหตุนี้ replica-aware stub จึงไม่สามารถ failover ได้ในกรณีที่ไม่ใช่ idempotent โดยจะต้องกำหนดเมธอดให้เป็น idempotent ใน deployment descriptor

**10.4 การ Replica HTTP Session State**

ในการ failover แบบอัตโนมัติสำหรับ Servlet และ JSP HTTP session state WebLogic เซิร์ฟเวอร์จะ replicate session state ออบเจ็กต์ในเมโมรี กระบวนการนี้จะสร้าง primary session state ที่อยู่บน WebLogic เซิร์ฟเวอร์ที่ไคลเอ็นต์ติดต่อเข้ามาครั้งแรกและ secondary replica ของ session state บนอินสแทนซ์ของ WebLogic เซิร์ฟเวอร์อื่นในคลัสเตอร์ replica นั้นจะตรงกันเสมอและพร้อมจะถูกใช้เมื่อเซิร์ฟเวอร์หนึ่งขัดข้อง กระบวนการของการก๊อบปี้ state จากอินสแทนซ์หนึ่งไปยังอินสแทนซ์อื่นเรียกว่า in-memory replication

**10.4.1 ความต้องการสำหรับการ Replica HTTP Session State**

ในการที่จะใช้ in-memory replication สำหรับ HTTP session state จะต้องแอ็กเซส (access) WebLogic เซิร์ฟเวอร์คลัสเตอร์โดยใช้วิธีดังต่อไปนี้

- Load balancing ฮาร์ดแวร์
- เว็บเซิร์ฟเวอร์กับ WebLogic proxy plug-in

**10.4.1.1 ความต้องการของ Proxy**

WebLogic proxy plug-in จะดูแลรายการ (maintain list) อินสแทนซ์ของ WebLogic เซิร์ฟเวอร์ที่เป็นโฮสต์ของคลัสเตอร์ของ Servlet หรือ JSP และส่งต่อ HTTP request ไปยังอินสแทนซ์อื่นโดยใช้วิธี round-robin proxy จะมีวิธีการกำหนดตำแหน่ง replica HTTP session state ของไคลเอ็นต์ ถ้าอินสแทนซ์ของ WebLogic เซิร์ฟเวอร์ขัดข้อง

เว็บเซิร์ฟเวอร์ และ proxy ที่รองรับมีดังนี้

- WebLogic Server กับ HttpClusterServlet
- Netscape Enterprise Server กับ Netscape (proxy) plug-in
- Apache with the Apache Server (proxy) plug-in
- Microsoft Internet Information Server with the Microsoft-IIS (proxy) plug-in

**10.4.1.2 ความต้องการของ Load Balancer**

ถ้าเลือกที่จะใช้ load balancing ฮาร์ดแวร์แทนที่การใช้ proxy plug-in แล้วจะต้องใช้ฮาร์ดแวร์ที่สนับสนุน SSL persistence และ passive cookie persistence passive cookie persistence จะทำให้ WebLogic เซิร์ฟเวอร์สามารถเขียนคุกกี้ผ่าน load balancer ได้

**10.4.1.3 ความต้องการของ Session**

เมื่อจะพัฒนา Servlet หรือ JSP ที่จะดีพลอยในเอ็นไวรอนเมนต์ที่มีการคลัสเตอร์ จะต้องเป็นไปตามความต้องการนี้

- **ข้อมูลของ Session ต้อง Serialize**

เพื่อให้สนับสนุน in-memory replication สำหรับ HTTP session state Servlet และ JSP session ทั้งหมดจะต้อง serialize ถ้า Servlet หรือ JSP ใช้ออบเจ็กต์ที่ serialize และไม่ serialize ปนกัน WebLogic เซิร์ฟเวอร์จะไม่สามารถ replicate session state ได้

- **ใช้ setAttribute() ในการเปลี่ยนแปลง Session State**

Servlet ต้องใช้ *setAttribute()* หรือ *removeAttribute()* ในการเปลี่ยนออบเจ็กต์ของ session ถ้าใช้เมธอดเซต (set) อื่นในการเปลี่ยนออบเจ็กต์ภายใน session WebLogic เซิร์ฟเวอร์จะไม่สามารถ replicate การเปลี่ยนแปลงเหล่านั้นได้

- **ต้องคำนึงถึง Overhead ที่จะเกิดขึ้นกับการ Serialize ออบเจ็กต์ใน Session**

Serializing session data จะนำมาซึ่ง overhead สำหรับการ replicate session state overhead จะเพิ่มขนาดตามขนาดของออบเจ็กต์ที่ serialize ถ้าวางแผนที่จะสร้างออบเจ็กต์ขนาดใหญ่ใน session ควรจะทดสอบสมรรถนะของมันเพื่อให้แน่ใจว่ายอมรับได้

**10.4.2 การใช้กลุ่มของการ Replicate**

โดยดีฟอลต์ (default) WebLogic เซิร์ฟเวอร์พยายามที่จะสร้าง session state replica บนคนละเครื่อง (different machine) ใน WebLogic Server 6.0 จะอนุญาตให้ระบุที่ที่ secondary state อยู่โดยใช้ replication group replication group นั้นก็คือกลุ่มของอินสแทนซ์ในคลัสเตอร์ที่ใช้ในการเก็บ session state replica

เมื่อคอนฟิกอินสแทนซ์ของเซิร์ฟเวอร์ที่คลัสเตอร์ จะต้องกำหนดสมาชิกใน replication group เช่นเดียวกับกำหนด secondary replication group เพื่อเป็นที่เก็บ replica ของ primary HTTP session state

เมื่อไคลเอ็นต์ติดต่อไปยังเซิร์ฟเวอร์ในคลัสเตอร์และสร้าง primary session state เซิร์ฟเวอร์ที่เป็นโฮสต์ของ primary state จะจัดอันดับเซิร์ฟเวอร์อื่นในคลัสเตอร์ เพื่อที่จะตัดสินใจว่าจะใช้เซิร์ฟเวอร์ใดเป็นโฮสต์ของ secondary เซิร์ฟเวอร์ อันดับของเซิร์ฟเวอร์จะถูกกำหนดจากโลเคชันของเซิร์ฟเวอร์และ replication group ดังตารางที่ 10-1

| อันดับของเซิร์ฟเวอร์ | เซิร์ฟเวอร์อยู่บนคนละเครื่องกัน (machine) หรือไม่ | เซิร์ฟเวอร์อยู่ใน replication group เดียวกันหรือไม่ |
|---|---|---|
| 1 | Yes | Yes |
| 2 | No | Yes |
| 3 | Yes | No |
| 4 | No | No |

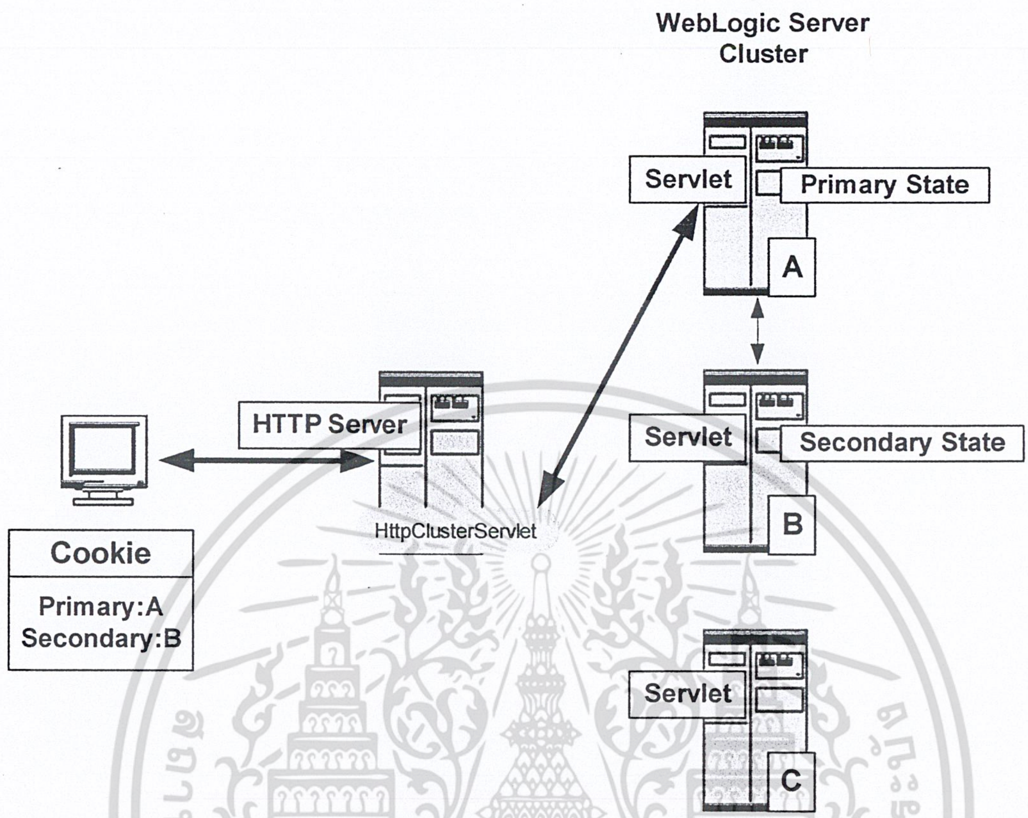
***ตารางที่ 10-1 การจัดอันดับของเซิร์ฟเวอร์ในคลัสเตอร์***

โดยใช้กฎการจัดอันดับข้างต้น primary WebLogic เซิร์ฟเวอร์ จะจัดอันดับสมาชิกในคลัสเตอร์และเลือกเซิร์ฟเวอร์ที่มีอันดับสูงสุดเป็นโฮสต์ของ secondary session state

**10.4.3 การแอ็กเซสคลัสเตอร์ Servlet และ JSP ผ่าน Proxy**

รูปที่ 10-5 แสดงถึงไคลเอ็นต์ที่แอ็กเซส Servlet ในสถาปัตยกรรม 2 เทียร์ โดยตัวอย่างนี้ใช้ WebLogic เซิร์ฟเวอร์เครื่องเดียวในการให้บริการสแตติก HTTP request เท่านั้น Servlet request ทั้งหมดจะถูกส่งต่อไปให้ WebLogic เซิร์ฟเวอร์ด้วย HttpClusterServlet

เมื่อไคลเอ็นต์ request Servlet HttpClusterServlet proxy จะส่ง request ต่อไปยัง WebLogic เซิร์ฟเวอร์คลัสเตอร์ HttpClusterServlet จะดูแลรายการของเซิร์ฟเวอร์ทั้งหมดในคลัสเตอร์รวมถึงวิธีในการ load balancing ทั้งหมด จากตัวอย่างข้างบน HttpClusterServlet จะส่งไคลเอ็นต์ request ไปยังโฮสต์ของ Servlet บน WebLogic เซิร์ฟเวอร์ A WebLogic เซิร์ฟเวอร์ A จะกลายเป็นโฮสต์ของ primary เซิร์ฟเวอร์ที่มี Servlet session ของไคลเอ็นต์




*รูปที่ 10-5 การแอ็กเซสคลัสเตอร์ Servlet และ JSP ผ่าน proxy*

ในการ failover เซอร์วิสสำหรับ Servlet session state primary เซิร์ฟเวอร์จะ replicate ไคลเอ็นต์ไปยัง secondary WebLogic เซิร์ฟเวอร์ ซึ่งตามตัวอย่างเซิร์ฟเวอร์ B ถูกเลือกให้เป็น secondary

response ของ Servlet จะถูกคืนไปให้ไคลเอ็นต์ผ่านทาง HttpClusterServlet และไคลเอ็นต์บราวเซอร์จะถูกสั่งให้เขียนคุกกี้ที่ประกอบด้วยรายการตำแหน่งของ primary และ secondary ของ Servlet session state ถ้าไคลเอ็นต์บราวเซอร์ไม่สนับสนุนคุกกี้ WebLogic เซิร์ฟเวอร์สามารถใช้ URL rewriting แทนได้ โดย URL rewriting เป็นวิธีการฝังตำแหน่งของ client session state ทั้ง primary และ secondary ลงใน URL ที่ถูกส่งระหว่างไคลเอ็นต์และ proxy เซิร์ฟเวอร์

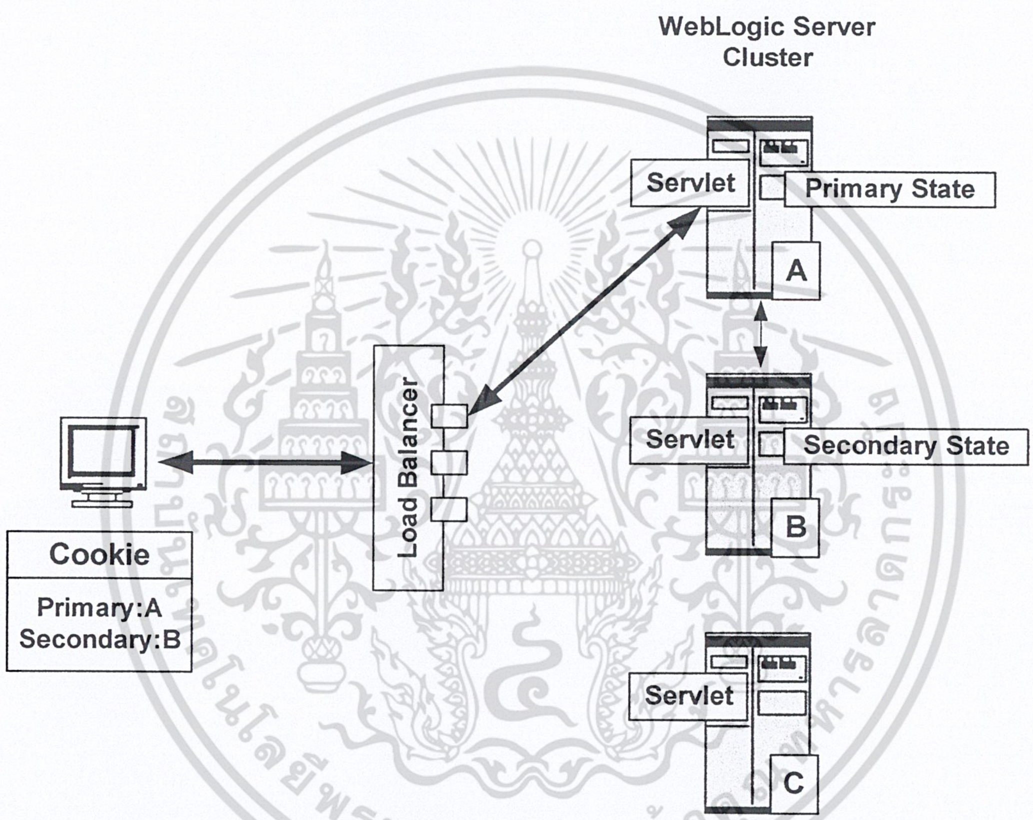
- **กระบวนการ Failover ของ Proxy**

เมื่อ primary เซิร์ฟเวอร์ขัดข้อง HttpClusterServlet จะใช้คุกกี้อินฟอร์เมชันของไคลเอ็นต์ในการกำหนดตำแหน่งของ secondary WebLogic เซิร์ฟเวอร์ที่เป็นโฮสต์ของ replica ของ session state แล้ว HttpClusterServlet จะเปลี่ยนทิศทาง HTTP request ถัดไปของไคลเอ็นต์ไปยัง secondary เซิร์ฟเวอร์โดยอัตโนมัติ failover จึง transparent ต่อไคลเอ็นต์

ภายหลังขัดข้อง WebLogic เซิร์ฟเวอร์ B จะกลายเป็น primary เซิร์ฟเวอร์และ secondary เครื่องใหม่ คือ เซิร์ฟเวอร์ C จะถูกสร้างขึ้น หลังจาก HTTP response proxy จะอัพเดตคุกกี้ของไคลเอ็นต์ที่มีผลต่อ primary และ secondary เซิร์ฟเวอร์เครื่องใหม่

**10.4.4 การแอ็กเซสคลัสเตอร์ Servlet และ JSP ผ่าน Load Balancing ฮาร์ดแวร์**

วิธีนี้มีแนวคิดในการจัดการเหมือนกับการใช้ proxy plug-in ต่างกันที่ใช้ Load Balancer ที่เป็นฮาร์ดแวร์ในการจัดการแทน proxy plug-in ดังรูปที่ 10-6




*รูปที่ 10-6 การแอ็กเซสคลัสเตอร์ Servlet และ JSP ผ่าน Load Balancing ฮาร์ดแวร์*

**10.5** การคลัสเตอร์ออบเจ็กต์

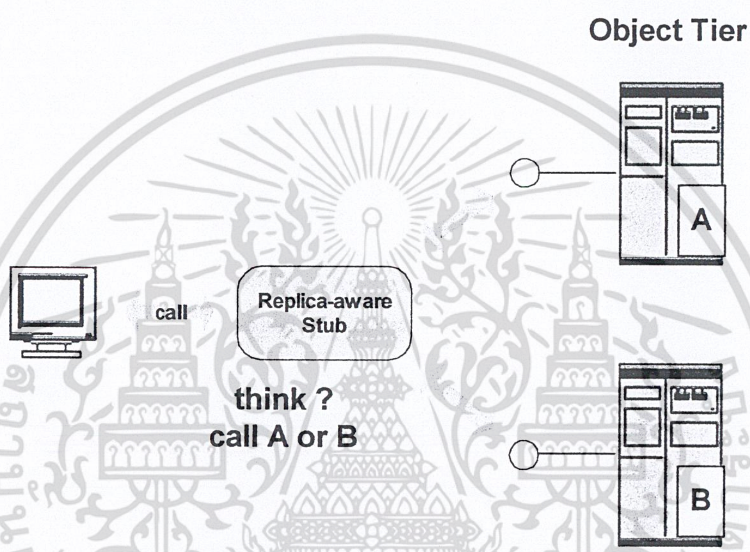
คีย์เทคโนโลยีที่เป็นพื้นฐานของการคลัสเตอร์ออบเจ็กต์ใน WebLogic เซิร์ฟเวอร์คือ replica-aware stub

**10.5.1 Replica-aware Stub**

Replica-aware stub ปรากฏต่อผู้เรียกเป็น RMI stub ธรรมดา แต่แทนที่จะแสดงถึงออบเจ็กต์เดียว จะแสดงถึง replica ทั้งหมดของออบเจ็กต์นั้นในคลัสเตอร์ replica-aware stub จะมีวิธีที่ใช้ในการกำหนดตำแหน่งของ EJB หรือ RMI คลาสบนอินสแทนซ์ของ WebLogic เซิร์ฟเวอร์ใด ๆ ที่ออบเจ็กต์ดีพลอยอยู่

เมื่อดีพลอย EJB หรือ RMI ออบเจ็กต์ในคลัสเตอร์ มันจะถูก bind ลงใน JNDI ทรี เมื่อไคลเอ็นต์แอ็กเซสคลัสเตอร์ออบเจ็กต์ แทนที่ไคลเอ็นต์จะได้รับ stub ของออบเจ็กต์แบบปกติมันจะได้รับ replica-aware stub ที่สามารถ load balancing ไปยังอินสแทนซ์ใด ๆ ของ WebLogic เซิร์ฟเวอร์ในคลัสเตอร์ ดังรูปที่ 10-7

Stub จะประกอบด้วย load balancing อัลกอริทึม แต่ละครั้งของการเรียกกับ stub สามารถใช้โหลดอัลกอริทึมในการเลือก replica ของออบเจ็กต์ที่จะเรียก ซึ่งจะ tranparent ต่อผู้เรียก ถ้ามีการขัดข้องเกิดขึ้นระหว่างการเรียก stub จะขว้าง exception และจะพยายามเรียก replica ของออบเจ็กต์อื่น ซึ่ง tranparent ต่อผู้เรียกเช่นกัน




*รูปที่ 10-7 การเรียกกับ replica-aware stub*

**10.5.2 คลัสเตอร์ EJB**

EJB ต่างจาก RMI ออบเจ็กต์ตรงที่แต่ละ EJB จะมี replica-aware stub ที่ต่างกัน 2 อัน อันหนึ่งสำหรับ EJBHome interface และอีกอันสำหรับ EJBObject interface ซึ่งหมายความว่า EJB จะสามารถ load balancing และ failover ได้ 2 ระดับคือ

- เมื่อไคลเอ็นต์ look up EJB object โดยใช้ EJBHome stub
- เมื่อไคลเอ็นต์เรียกเมธอดของ EJB โดยใช้ EJBObject stub

- **EJB Home Stub**

Bean home ทั้งหมดสามารถคลัสเตอร์ได้ เมื่อไคลเอ็นต์ look up home มันจะได้รับ replica-aware stub ที่มีอ้างอิงไปยัง home บนแต่ละเซิร์ฟเวอร์ที่ดีพลอย bean ไว้ เมื่อเมธอด *create()* หรือ *find()* ถูกเรียก replica-aware stub จะส่งการเรียกไปยัง replica หนึ่ง home replica ของเซิร์ฟเวอร์นั้นจะได้รับผลลัพธ์ของ *find()* หรือสร้างอินสแทนซ์ของ bean บนเซิร์ฟเวอร์

- **Stateless EJB**

เมื่อ home ถูกสร้างเป็น stateless session bean มันจะคืนค่า replica-aware EJBObject stub ที่สามารถส่งการเรียกไปยังเซิร์ฟเวอร์ใด ๆ ที่ bean ถูกดีพลอยอยู่ เพราะ stateless bean ไม่ได้เก็บ state ของไคลเอ็นต์ไว้ stub จึงสามารถส่งการเรียกใด ๆ ไปยังเซิร์ฟเวอร์ใดก็ได้ที่เป็นโฮสต์ของ bean แต่มันไม่สามารถกู้ทุกการขัดข้องที่เกิดขึ้นได้ มันจะ failover อัตโนมัติได้ในกรณีที่ bean ถูกเขียนเป็นเมธอดที่ idempotent เท่านั้น

- **Stateful EJB**

คลัสเตอร์ stateful session EJB ใช้ประโยชน์จาก replica-aware EJBHome stub ในการกำหนดตำแหน่งของโฮสต์ทั้งหมดของ bean โดย load balancing จะเกิดในระดับของการ look up EJB object โดยใช้ EJBHome stub เท่านั้น และจะ failover โดยใช้ replica-aware EJBObject stub ในการดูแลการกำหนดตำแหน่งของ primary และ secondary state ของ EJB ซึ่งวิธีการจัดการนั้นเหมือนกับ HTTP session state

- **Entity EJB**

มี 2 ชนิดของ entity bean ที่ต้องพิจารณาคือ entity ที่อ่านและเขียนข้อมูลและ entity ที่อ่านข้อมูลเพียงอย่างเดียว

เมื่อ home find หรือ create entity bean ที่อ่านและเขียนข้อมูล มันจะได้รับอินสแทนซ์บนโลคอลเซิร์ฟเวอร์และคืนค่า stub ที่ชี้ไปยังเซิร์ฟเวอร์นั้น load balancing และ failover จะเกิดขึ้นที่ระดับของ home เท่านั้น เพราะมันเป็นไปได้ที่จะมีหลายอินสแทนซ์ของ entity bean ในคลัสเตอร์ แต่ละอินสแทนซ์ต้องอ่านข้อมูลจากฐานข้อมูลก่อนทุก ๆ ทรานแซ็กชันและเขียนข้อมูลทุก ๆ ครั้งที่ commit

เมื่อ home find หรือ create entity bean ที่อ่านข้อมูลเพียงอย่างเดียว มันจะคืนค่า replica-aware stub stub นี้สามารถ load balancing ได้ทุก ๆ ครั้งที่มีการเรียกแต่ไม่ failover อัตโนมัติในกรณีที่การเรียกเมธอดขัดข้อง read-only bean ยังถูกแคช (cache) บนทุก ๆ เซิร์ฟเวอร์ เพื่อหลีกเลี่ยงการอ่านข้อมูลจากฐานข้อมูลด้วย

**10.5.3 การ Replicate Stateful Session Bean**

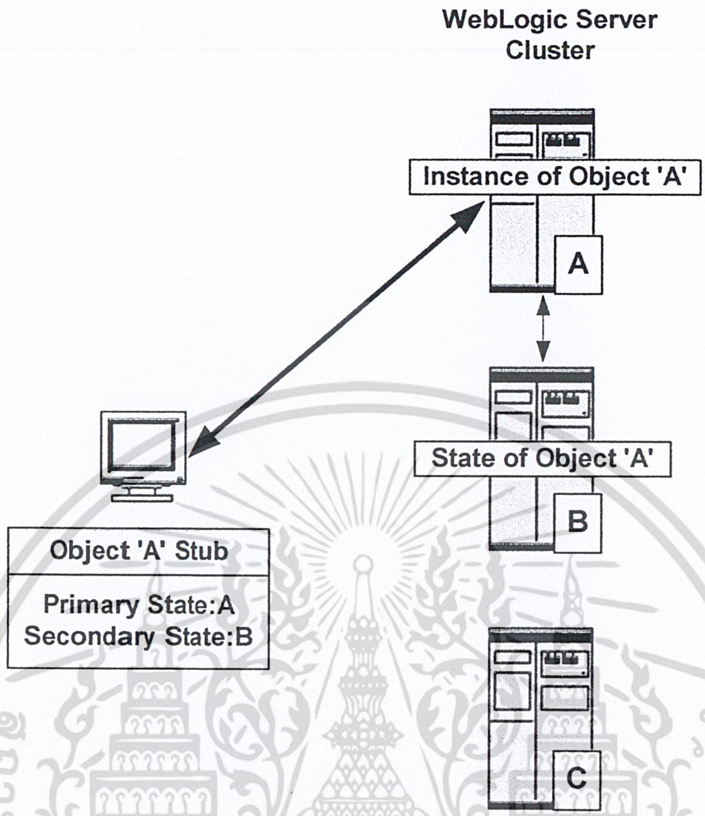
Replicate state ของ stateful session EJB จะเหมือนกับ replicate HTTP session state ดังรูปที่ 10-8

Primary เซิร์ฟเวอร์จะมีอินสแทนซ์ที่แท้จริงของ EJB ที่ไคลเอ็นต์ติดต่ออยู่ secondary เซิร์ฟเวอร์จะมีเพียง replicate state ของ EJB เท่านั้น secondary จะไม่สร้างอินสแทนซ์ของ EJB นอกจากจะเกิด failover ทำให้มีการใช้ทรัพยากรน้อยลงบน secondary

- **การเปลี่ยนแปลงของ EJB state ที่ Replicate**

เมื่อไคลเอ็นต์เปลี่ยน state ของ EJB state ที่ต่างไปจะถูก replicate บน secondary เซิร์ฟเวอร์ สำหรับ EJB ที่เกี่ยวข้องกับทรานแซ็กชัน การ replicate จะเกิดขึ้นทันทีที่ทรานแซ็กชัน commit สำหรับ EJB ที่ไม่เกี่ยวกับทรานแซ็กชัน การ replicate จะเกิดขึ้นภายหลังทุก ๆ การเรียกเมธอด

ทั้ง 2 กรณีจะมีเพียงสิ่งที่เปลี่ยนจริง ๆ เท่านั้นที่จะถูก replicate บน secondary เซิร์ฟเวอร์เพื่อลด overhead ของการประมวลผลการ replicate




*รูปที่ 10-8 การ replicate Stateful Session Bean*

- **Failover สำหรับ Stateful Session EJB**

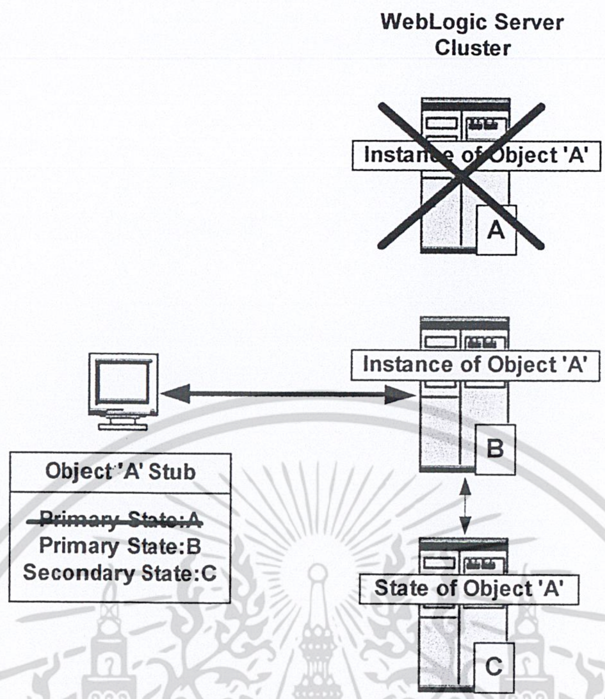
เมื่อ primary เซิร์ฟเวอร์ขัดข้อง EJB stub ของไคลเอ็นต์จะเปลี่ยนทิศทางของ request ต่อ ๆ ไปโดยอัตโนมัติไปยังอินสแทนซ์ของ secondary WebLogic เซิร์ฟเวอร์ ตรงจุดนี้ secondary เซิร์ฟเวอร์จะสร้างอินสแทนซ์ของ EJB ใหม่โดยใช้ข้อมูลของ state ที่ถูก replicate ไว้ ดังรูปที่ 10-9

**10.5.4 การเพิ่มประสิทธิภาพโดยการเรียกออบเจ็กต์บนเครื่องที่เหมาะสม**

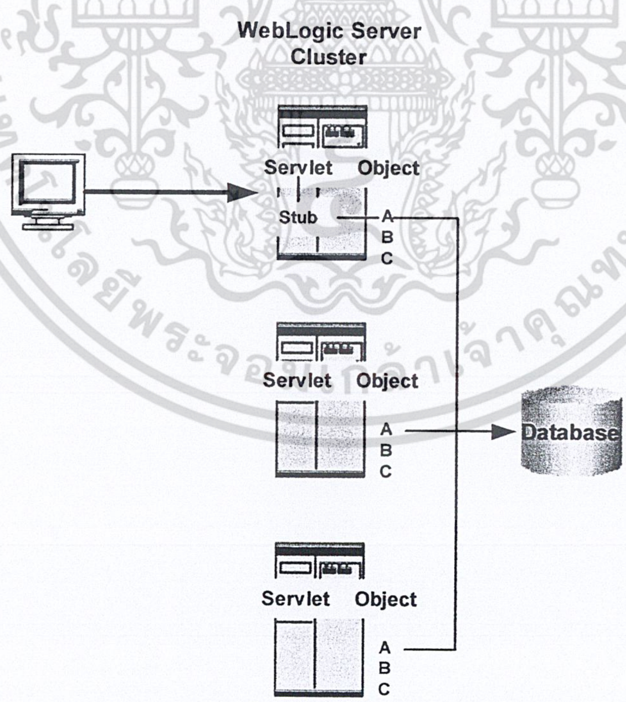
แม้ว่า replica-aware stub จะ load balancing โดยใช้อัลกอริทึม WebLogic เซิร์ฟเวอร์จะไม่ load balancing สำหรับการเรียกเมธอดของออบเจ็กต์ที่อยู่บนเครื่องเดียวกัน เนื่องจากมันจะมีประสิทธิภาพมากกว่า รูปที่ 10-10 จะอธิบายสถานการณ์นี้

จากตัวอย่างไคลเอนต์ติดต่อไปยังโฮสต์ของ Servlet ในการตอบสนองต่อไคลเอนต์ Servlet จะใช้ replica-aware stub สำหรับออบเจ็กต์ A เพราะ replica ของออบเจ็กต์ A มีอยู่บนเซิร์ฟเวอร์เดียวกัน

WebLogic เซิร์ฟเวอร์จะใช้โลคอล (ก๊อบปี้ของออบเจ็กต์ A) มากกว่าที่จะใช้ distribute ซึ่งมันจะมีประสิทธิภาพที่ดีกว่าเพราะมันช่วยลดเน็ตเวิร์ก overhead ของการเชื่อมต่อระหว่างเซิร์ฟเวอร์ในคลัสเตอร์




*รูปที่ 10-9 Failover สำหรับ Stateful Session EJB*




*รูปที่ 10-10 การเพิ่มประสิทธิภาพโดยการเรียกออบเจ็กต์บนเครื่องที่เหมาะสม*

## 10.6 การออกแบบคลัสเตอร์ WebLogic เซิร์ฟเวอร์

### 10.6.1 ปัจจัยในการออกแบบสถาปัตยกรรม

ในการออกแบบสถาปัตยกรรมของคลัสเตอร์มีปัจจัยที่ต้องคำนึงถึงดังนี้

- load balancer หรือ proxy plug-in

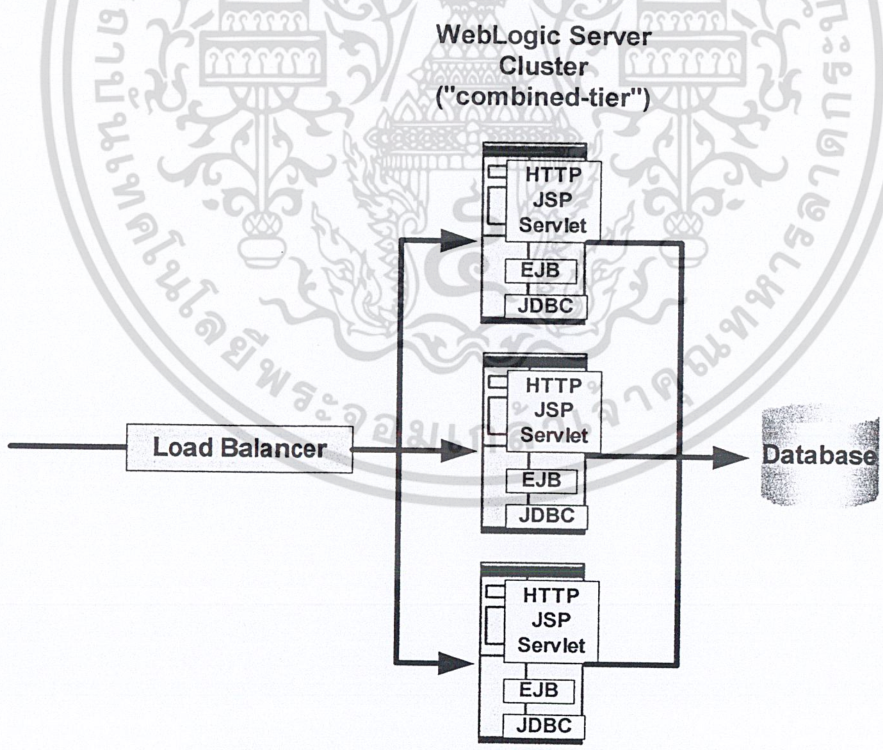
load balancer เป็นการใช้ฮาร์ดแวร์มีข้อดีคือมีความยืดหยุ่นในการ load balancing มากกว่า โดยอาจคอนฟิกให้ load balancer ตรวจสอบปริมาณงานของเซิร์ฟเวอร์ในขณะนั้น (current server load) และส่งไคลเอ็นต์ request ได้อย่างเหมาะสม

- การแบ่งเทียร์ทางกายภาพ

เป็นการรวมเทียร์ทางลอจิกเข้าด้วยกันเช่น อาจรวมเว็บเทียร์กับพรีเซนเทชันเทียร์ ในการที่จะพิจารณาการแบ่งเทียร์จะต้องคำนึงลักษณะของเว็บแอพพลิเคชันว่ามีการประมวลผลในส่วนไหนมากกว่ากัน

### 10.6.2 สถาปัตยกรรมพื้นฐานของคลัสเตอร์

สถาปัตยกรรมนี้รวมเว็บแอพพลิเคชันเทียร์และเซอร์วิสทั้งหมดไว้ในคลัสเตอร์เดียว ดังรูปที่ 10-11




*รูปที่ 10-11 สถาปัตยกรรมพื้นฐานของคลัสเตอร์*

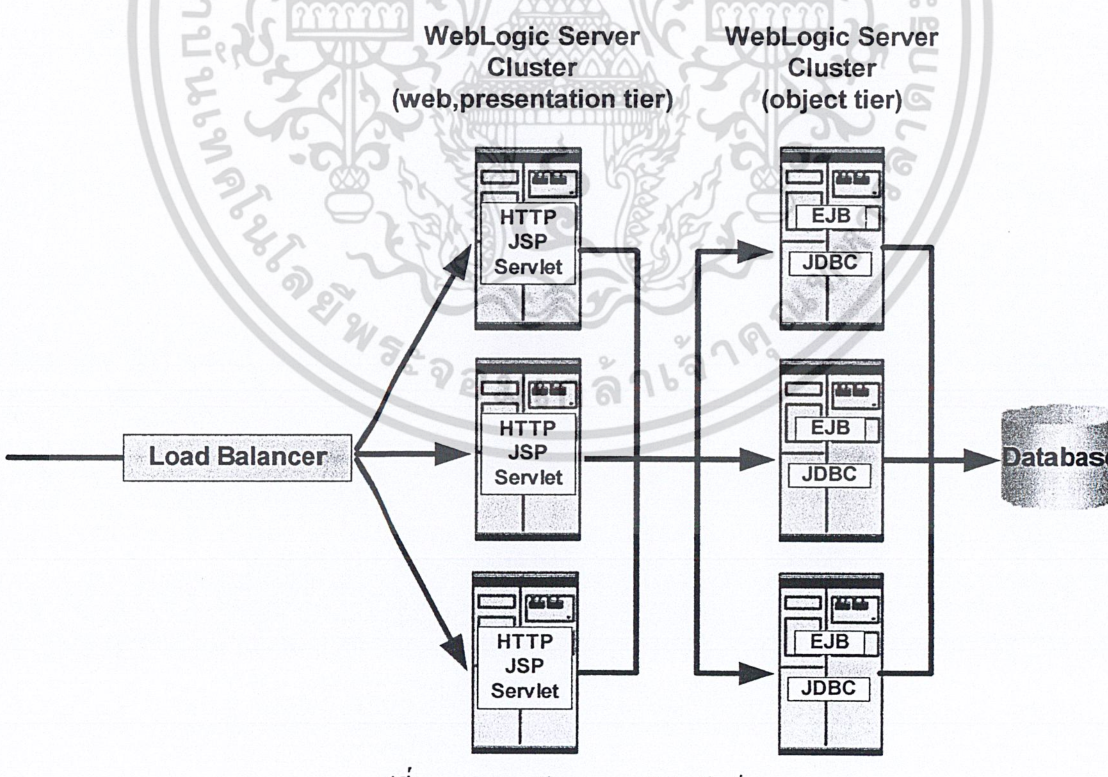
สถาปัตยกรรมพื้นฐานมีข้อดีคือง่ายต่อการบริหาร สามารถที่จะคอนฟิกเว็บแอพพลิเคชันทั้งหมด , ดีพลอยและอันดีพลอยออบเจ็กต์โดยใช้ administrator คอนโซลไม่จำเป็นต้องดูแลเว็บเซิร์ฟเวอร์ที่แยกกันและคอนฟิก WebLogic Server proxy plug-in

แม้ว่าสถาปัตยกรรมพื้นฐานจะง่าย แต่มันก็เหมาะกับเว็บแอพพลิเคชันบางประเภทเท่านั้น แต่ความสามารถทางด้าน load balancing และ failover สามารถทำได้ที่อินเทอร์เฟซระหว่างเว็บแอพพลิเคชันเทียร์เท่านั้น ทำให้ความสามารถทางด้าน load balancing และ failover ลดลง การรวมกันของเทียร์ทำให้ไม่เหมาะกับการ load balancing เมธอดที่เรียกคลัสเตอร์ EJB เพราะคลัสเตอร์ออบเจ็กต์ถูกดีพลอยบนทุกอินสแทนซ์ของ WebLogic เซิร์ฟเวอร์ในคลัสเตอร์ WebLogic เซิร์ฟเวอร์จะ optimize การเรียกเมธอดของคลัสเตอร์ EJB โดยจะเลือกอินสแทนซ์ของโลคอลออบเจ็กต์เสมอ ทำให้กลายเป็นไม่ balance

เพื่อที่จะใช้ประโยชน์ของ load balancing สำหรับการเลือกเมธอดของคลัสเตอร์ EJB จะต้องแบ่งพรีเซนเทชันและออบเจ็กต์เทียร์ของเว็บแอพพลิเคชันเป็นคลัสเตอร์ทางกายภาพที่ต่างกัน

**10.6.3 สถาปัตยกรรมมัลติเทียร์**

สถาปัตยกรรมมัลติเทียร์ที่ได้รับการแนะนำจะใช้ WebLogic เซิร์ฟเวอร์คลัสเตอร์ 2 ส่วน ส่วนหนึ่งในการให้บริการสแตติก HTTP และคลัสเตอร์ Servlet และอีกส่วนให้บริการคลัสเตอร์ EJB ดังรูปที่ 10-12




*รูปที่ 10-12 สถาปัตยกรรมมัลติเทียร์*

สถาปัตยกรรมมัลติเทียร์แบบนี้มีข้อดีดังนี้

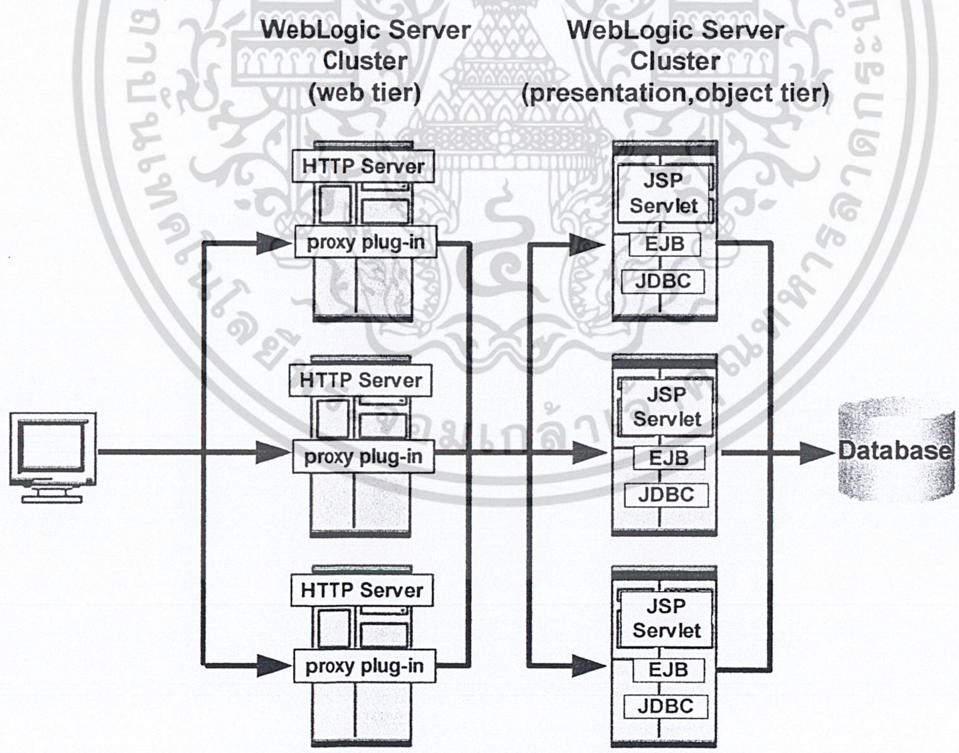
- **Load Balancing EJB เมธอด** - โฮสต์ของ Servlet และ EJB จะอยู่บนคนละคลัสเตอร์ เมธอดของ Servlet ที่เรียก EJB สามารถ load balance ผ่านเซิร์ฟเวอร์หลายตัว
- **มีความยืดหยุ่นในการปรับปรุง load balancing** - การแบ่งพรีเซนเทชันและออบเจ็กต์เทียร์ให้อยู่กันคนละคลัสเตอร์ทำให้มีทางเลือกสำหรับการ distribute โหลดของเว็บแอพพลิเคชันมากขึ้น เช่นถ้ามีการแอ็กเซส HTTP และ Servlet มากกว่า EJB จะสามารถใช้อินสแทนซ์ของ WebLogic เซิร์ฟเวอร์จำนวนมากในพรีเซนเทชันเทียร์ที่มีการแอ็กเซสเซิร์ฟเวอร์ที่เป็นโฮสต์ของ EJB ที่มีจำนวนน้อยกว่า

**10.6.4 สถาปัตยกรรมที่ใช้ Proxy**

สถาปัตยกรรมที่ใช้ proxy มีข้อเสียที่สามารถใช้ load balancing อัลกอริทึมได้เพียง round-robin เท่านั้น

- **สถาปัตยกรรม 2 เทียร์ Proxy**

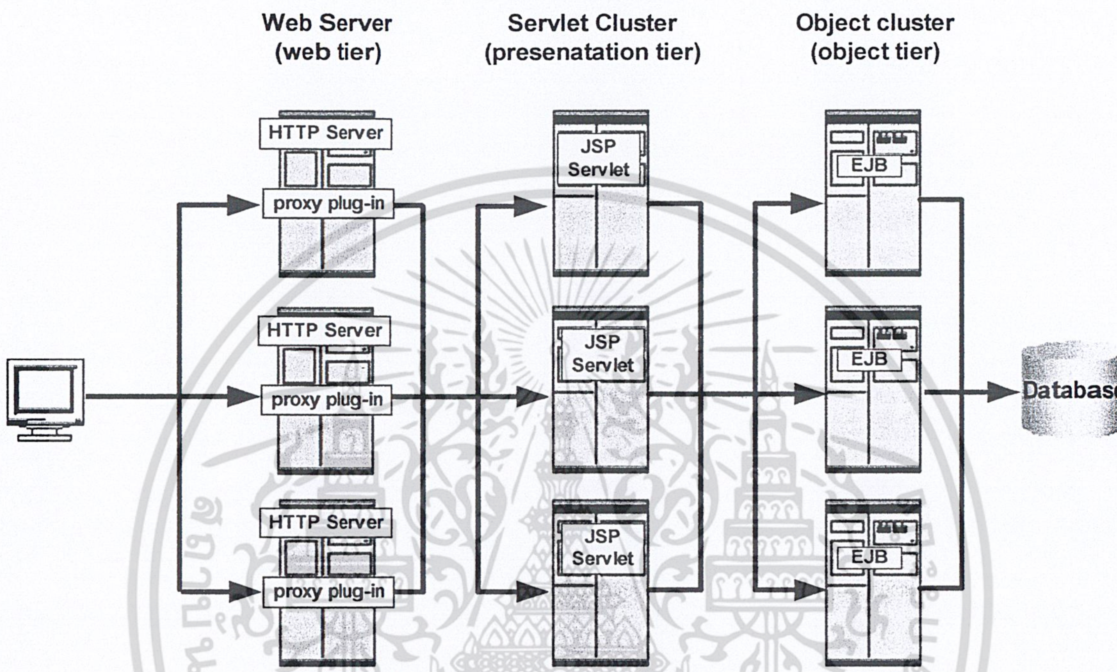
สถาปัตยกรรมนี้เหมือนกับสถาปัตยกรรมพื้นฐานเพียงแต่ใช้สแตติก HTTP เซิร์ฟเวอร์เป็นโฮสต์ของเว็บเซิร์ฟเวอร์ ดังรูปที่ 10-13




*รูปที่ 10-13 สถาปัตยกรรม 2 เทียร์ proxy*

- **สถาปัตยกรรม 3 เทียร์ Proxy**

จะเหมือนกับสถาปัตยกรรม 2 เทียร์ proxy แต่จะแยกพรีเซนเทชันเทียร์และออบเจ็กต์เทียร์ออกจากกัน ดังรูปที่ 10-14




*รูปที่ 10-14 สถาปัตยกรรม 3 เทียร์ proxy*

# บทที่ 11

# การออกแบบระบบรถโดยสารและสายการบิน

ประกอบด้วยส่วนบริการต่างๆ ดังนี้

1. การลงทะเบียน
2. การสืบค้นข้อมูลของระบบรถโดยสาร — สายการบิน
3. การจองที่นั่ง
4. การยืนยันการจอง
5. การยกเลิกการจอง
6. การคิดราคา

## 11.1 การลงทะเบียน

ระบบการจองการเดินทาง ทั้งรถโดยสารและสายการบินนั้น จะให้ผู้จอง (Customer) ได้ทำการลงทะเบียน (Register) กับทางระบบก่อน เพื่อเก็บข้อมูลเอาไว้ใช้ในการติดต่อภายหลัง ผู้จองถือเป็นตัวแทนของผู้เดินทางจริง ๆ อีกที และไม่จำเป็นต้องเป็นผู้เดินทางเองก็ได้ สิ่งสำคัญที่ผู้จองต้องให้ข้อมูลคือ mail address ซึ่งจะใช้ในการล็อกอินเข้าใช้งานบริการในระบบ

## 11.2 การสืบค้นข้อมูลของระบบรถโดยสาร — สายการบิน

ระบบรถโดยสาร มีหน้าที่ให้บริการข้อมูลเกี่ยวกับรายละเอียดของ

- บริษัทขนส่ง จะมีรายละเอียดเกี่ยวกับ ชื่อ และ รายละเอียด
- รถโดยสาร รุ่นของรถ บริษัทที่สังกัด
- สายการเดินทาง ประกอบด้วยรายการเส้นทางย่อยๆ ที่ผ่าน
- ตารางเวลาของแต่ละสายการเดินทาง
- สถานที่ที่แต่ละสายการเดินทางจะไป
- ข้อมูลการจองของลูกค้า ซึ่งแบ่งเป็น การจองที่นั่ง และการเช่าเหมาทั้งคัน
- ข้อมูลลูกค้า (Customer) ที่จะเอาไว้ติดต่อ
- ข้อมูลผู้เดินทาง
- ราคาของที่นั่งแต่ละรุ่นและแต่ละบริษัท

ระบบสายการบิน มีหน้าที่ให้บริการข้อมูลเกี่ยวกับรายละเอียดของ

- สายการบิน จะมีรายละเอียดเกี่ยวกับ ชื่อ และ รายละเอียด
- ชนิดของเครื่องบิน
- สายการเดินทาง ประกอบด้วยรายการเส้นทางย่อย ๆ ที่ผ่าน

- ตารางเวลาของแต่ละสายการเดินทาง
- สถานที่ที่แต่ละสายการเดินทางจะไป ซึ่งหมายถึงท่าอากาศยาน
- ข้อมูลการจองของลูกค้า ก็คือ การจองที่นั่ง
- ข้อมูลลูกค้า (Customer) ที่จะเอาไว้ติดต่อ
- ข้อมูลผู้เดินทาง (Traveler)
- ราคาของที่นั่งแต่ละคลาสของแต่ละสายการบิน

■ **การค้นหาข้อมูลของรถโดยสาร**

ผู้ที่เข้ามาใช้บริการสามารถหาเส้นทางการเดินทางได้ โดยกรอกข้อมูลที่จำเป็นเกี่ยวกับรายละเอียดต่าง ๆ ที่ต้องการค้น ข้อมูลที่ใช้ประกอบเป็นเงื่อนไข มีดังนี้

- สถานที่ต้นทาง
- สถานที่ปลายทาง
- ตัวเลือกการเดินทาง (ไปอย่างเดียว หรือไปกลับ)
- วัน เวลา ที่จะออกเดินทาง
- ชนิดของที่นั่ง
- บริษัทที่ต้องการ

ผลลัพธ์ที่คืนกลับมามีรายละเอียดดังนี้

- ชื่อสายการเดินรถ
- ชื่อบริษัทขนส่ง
- สถานีต้นทาง / ปลายทาง
- เวลารถออก / เวลารถถึงที่หมาย
- ราคา

■ **การค้นหาข้อมูลของสายการบิน**

ผู้ที่เข้ามาใช้บริการสามารถหาเส้นทางการเดินทางได้โดยกรอกข้อมูลที่จำเป็น เกี่ยวกับรายละเอียดต่าง ๆ ที่ต้องการค้นหา จากนั้นระบบจะทำการค้นการเส้นทางให้ และคืนผลลัพธ์กลับมาเป็นรายละเอียดต่าง ๆ ของเที่ยวการเดินทางที่ตรงกับที่กำหนดเงื่อนไขการค้นหา ข้อมูลที่ใช้ประกอบเป็นเงื่อนไข มีดังนี้

- สถานที่ต้นทาง
- สถานที่ปลายทาง
- ตัวเลือกการเดินทาง (ไปอย่างเดียว หรือไปกลับ)
- วัน เวลา ที่จะออกเดินทาง
- ชนิดของที่นั่ง

- สายการบิน

ผลลัพธ์ที่คืนกลับมามีรายละเอียดดังนี้
- ชื่อเที่ยวบิน
- ชื่อสายการบิน
- เวลาขึ้นเครื่อง/เวลาลงเครื่อง
- ระดับของที่นั่ง
- ราคาต่อที่นั่ง

**11.3 การจองที่นั่ง**

เมื่อผู้ใช้สืบค้นข้อมูลของที่นั่งที่ต้องการได้แล้ว ก็สามารถเลือกรายการที่ต้องการ เพื่อทำการจองได้ ในขั้นตอนนี้ ผู้ใช้จะต้องเป็นสมาชิกของระบบเท่านั้น โดยจะต้องระบุอีเมล์แอ็ดเดรส และรหัสผ่าน หากผู้ใช้ยังไม่ได้เป็นสมาชิก ก็จะต้องทำการลงทะเบียนก่อน การจองที่นั่งนี้เป็นการจองล่วงหน้า ผู้ใช้จะต้องทำการยืนยันการจอง และชำระเงินก่อนวันเดินทางระยะเวลาหนึ่งซึ่งอาจเปลี่ยนแปลงได้ในภายหลัง เมื่อทำการจองเสร็จสมบูรณ์แล้ว ระบบจะแจ้งหมายเลขที่นั่ง แต่ถ้าเป็นการจองทั้งคันก็จะไม่ระบุที่และกำหนดวันที่จะต้องมายืนยัน และชำระเงิน โดยข้อมูลทั้งหมดนี้ จะส่งให้กับผู้ใช้ทางอีเมล์ ในกรณีที่ผู้จองไม่มายืนยันการจอง และชำระเงินภายในระยะเวลาที่กำหนด ระบบจะยกเลิกการจองให้โดยอัตโนมัติ และจะส่งอีเมล์ไปแจ้งให้ผู้จองทราบด้วย

**11.4 การยืนยันการจอง**

ผู้ใช้บริการสามารถล็อกอินเข้าสู่ระบบ แล้วเรียกดูรายการที่นั่งที่จองไว้ได้ เมื่อต้องการยืนยันการจอง ระบบจะแสดงจำนวนเงินที่ต้องชำระ ผู้จองจะต้องระบุข้อมูลบัตรเครดิตเพื่อชำระเงิน การยืนยันการจองจะเสร็จสิ้นก็ต่อเมื่อการชำระเงินเสร็จสมบูรณ์แล้ว

**11.5 การยกเลิกการจอง**

ผู้ใช้สามารถยกเลิกรายการจองบางรายการที่ไม่ต้องการได้ โดยต้องล็อกอินเข้ามา การยกเลิกนั้นสามารถทำได้ทันทีโดยเลือกรายการที่ต้องการลบ หลังจากลบแล้ว รายการนั้นจะถูกลบออกไปจากฐานข้อมูลของบริษัททันที

**11.6 การคิดราคา**

ราคาของที่นั่ง และการจองรถโดยสารแบบทั้งคันนั้น จะแตกต่างกันออกไปตามแต่ระดับ ชนิดของที่นั่ง รวมถึงบริษัทขนส่งหรือสายการบินที่เป็นเจ้าของ ราคานั้นจะมีส่วนคงที่ และส่วนที่แปรผันตามระยะทาง โดยมีสูตรในการคิดคือ

ราคา = ส่วนคงที่ (Fee) +[ ระยะทาง (Total Distance) * ราคาต่อระยะทาง(PricePerKm) ]

**11.7 แนวคิดในการออกแบบ**

การพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์ มีแนวคิดที่ว่าการทำซอฟต์แวร์ให้เป็นชิ้นส่วนประกอบซึ่งมีความสมบูรณ์ในตัวเอง และสามารถนำชิ้นส่วนเหล่านี้กลับมาใช้ใหม่ได้ (Reusable) หรือเพิ่มขยาย (Extend) ได้ ซึ่งนำไปสู่การพัฒนาคุณภาพของซอฟต์แวร์ รวมถึงลดเวลาในการพัฒนาและการดีบักลง

ลักษณะเด่นของการพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์ คือ ผู้ที่เข้ามาเรียกใช้จะเห็นแต่เพียงอินเทอร์เฟซของผู้ให้บริการซึ่งก็คือคอมโพเน็นต์ที่เราเขียนขึ้น ดังนั้นการแก้ไข เปลี่ยนแปลงที่เกิดขึ้นกับส่วนอิมพลีเมนต์ของคอมโพเน็นต์นั้น จะไม่ส่งผลให้ไคลเอ็นต์ต้องทำการเปลี่ยนแปลงแก้ไข หรือคอมไพล์ใหม่แต่อย่างใด

**11.7.1 J2EE Design Patterns**

การออกแบบระบบต่าง ๆ นั้น เราจะพบแนวทางในการออกแบบจำนวนมากที่เป็นมาตรฐาน ซึ่งได้มีผู้คิดค้นเอาไว้แล้ว (Pattern) ซึ่งการนำ Pattern เหล่านี้มาใช้ สามารถช่วยให้ประหยัดเวลาในการคิดออกแบบ อีกทั้ง Pattern ต่าง ๆ เหล่านี้ยังได้รับการพิสูจน์มาแล้วว่าทำงานได้ดี มีหลักการและเหตุผลทำให้รูปแบบการออกแบบ มีความเป็นสากล และสามารถเข้าใจได้ในวงกว้าง ในระบบรถโดยสารและสายการบิน ได้มีการนำเอา Pattern บางตัวมาใช้ ได้แก่

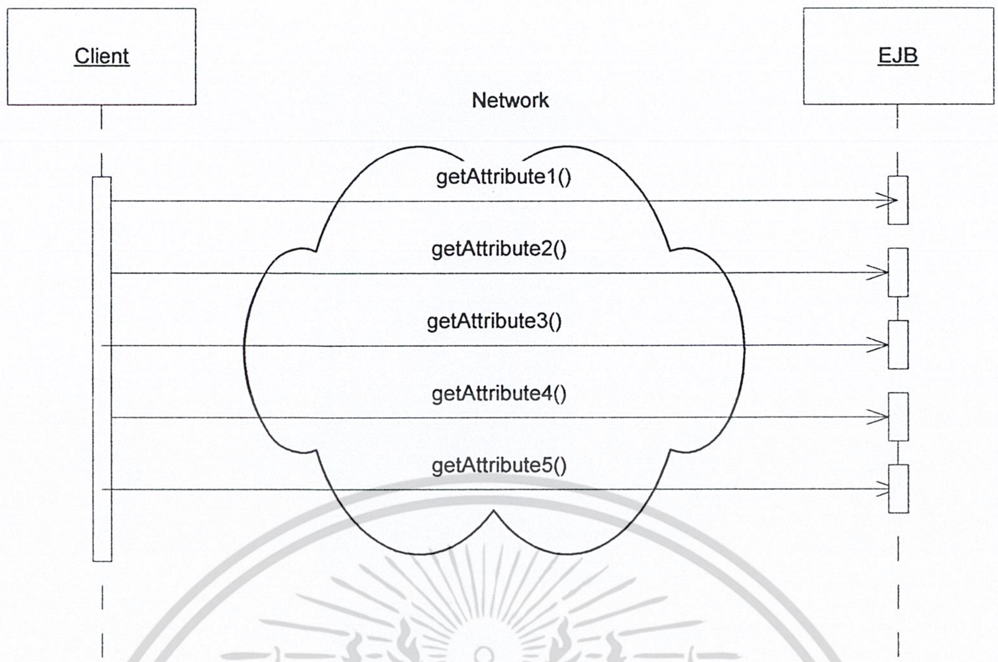
- Value Object
- Session Façade

■ **Value Object Pattern**

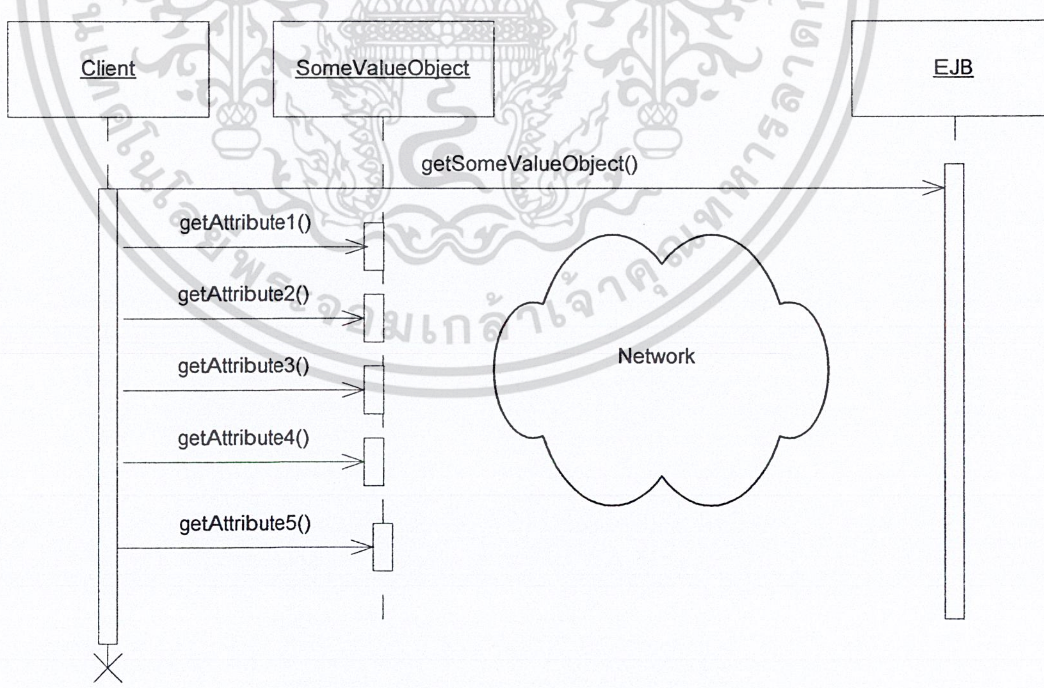
**ปัญหา** : การใช้ entity bean นั้น ส่วนใหญ่จะประกอบไปด้วย attribute จำนวนมากและมีเมธอดประเภท getter หรือ setter ที่ต้องทำกับ attribute จำนวนหลาย ๆ ตัวเหมือน ๆ กัน โดยที่การใช้เมธอดจำพวก *getXXX()* หรือ *setXXX()* 1 ครั้ง มีผลกับข้อมูลเพียง attribute เดียว ทำให้ต้องมีการเรียกเมธอดหลายครั้งเกินความจำเป็น ซึ่งจะทำให้เกิดปัญหาเมื่อเป็นการเรียกเมธอดข้ามเครือข่าย เนื่องจากการเรียกเมธอด ในแต่ละครั้งนั้นจะต้องเสีย overhead ไปกับเน็ตเวิร์กและการเรียกใช้งาน entity bean และนำมาสู่ปัญหาคอขวดด้านประสิทธิภาพในที่สุด

**วิธีการ** : สร้างคลาสช่วยขึ้นมาสนับสนุน เรียกว่า value object ซึ่งมีคุณสมบัติในการ serializes และ deserializes ซึ่งมี attribute ต่าง ๆ เหมือนกับ entity bean จากนั้นเมื่อไคลเอ็นต์ต้องการอ่านข้อมูลจาก entity bean ก็จะเรียกเมธอด *getSomeValueObject*

จากนั้นเซิร์ฟเวอร์จะทำการสร้าง value object ขึ้นมา และก๊อบปี้ attribute ต่าง ๆ ของ entity bean ใส่ไว้ใน value object ที่สร้างขึ้นมาใหม่นั้น จากนั้นถึงทำการส่ง value object กลับมายัง ไคลเอ็นต์ในรูปแบบที่ถูก serialize ส่วนไคลเอ็นต์จะสามารถดึงค่า attibute ที่ต้องการออกมาจาก value object นี้ได้โดยไม่ต้องมีการเรียกเมธอดข้ามระบบเครือข่ายอีก

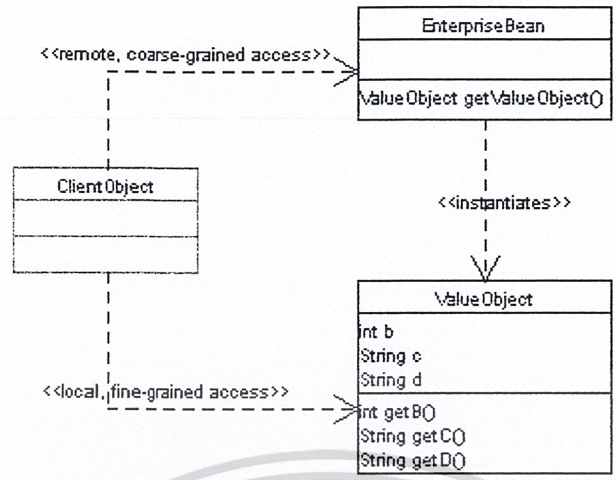



*รูปที่ 11-1 การเรียกใช้เมธอดประเภท get ของ Entity Bean โดยไม่ใช้ Value Object*




*รูปที่ 11-2 การเรียกใช้เมธอดประเภท get จาก Value Object ของ Entity Bean*

โครงสร้าง



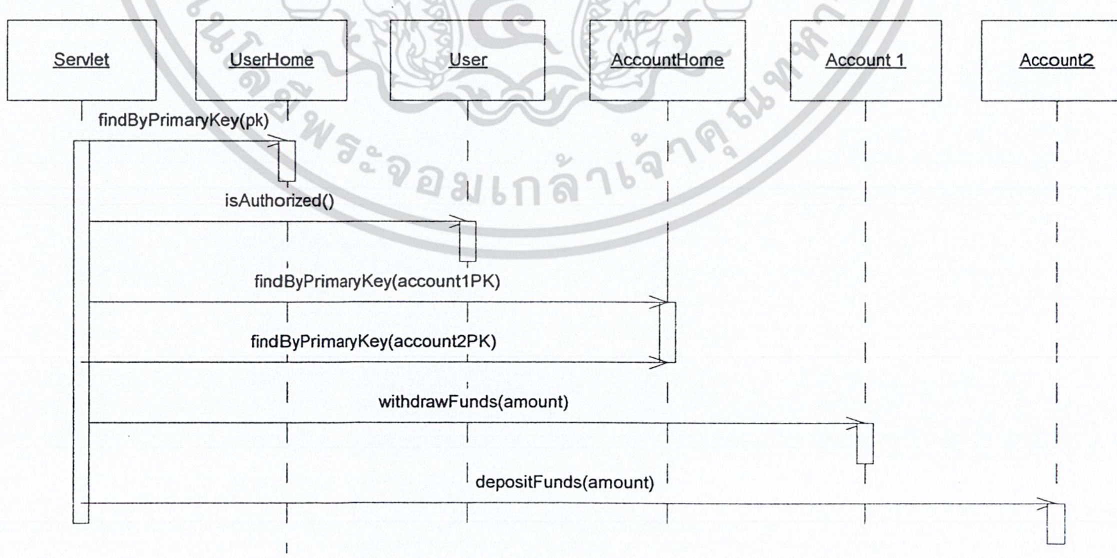

*รูปที่11- 3 โครงสร้างของ Value Object*

**ข้อดี** :

- โอนย้ายข้อมูลได้มากขึ้น ด้วยจำนวนครั้งของการเรียกเมธอดที่น้อยลง
- ลดเน็ตเวิร์กทราฟฟิกทำให้ประสิทธิภาพของระบบดีขึ้น

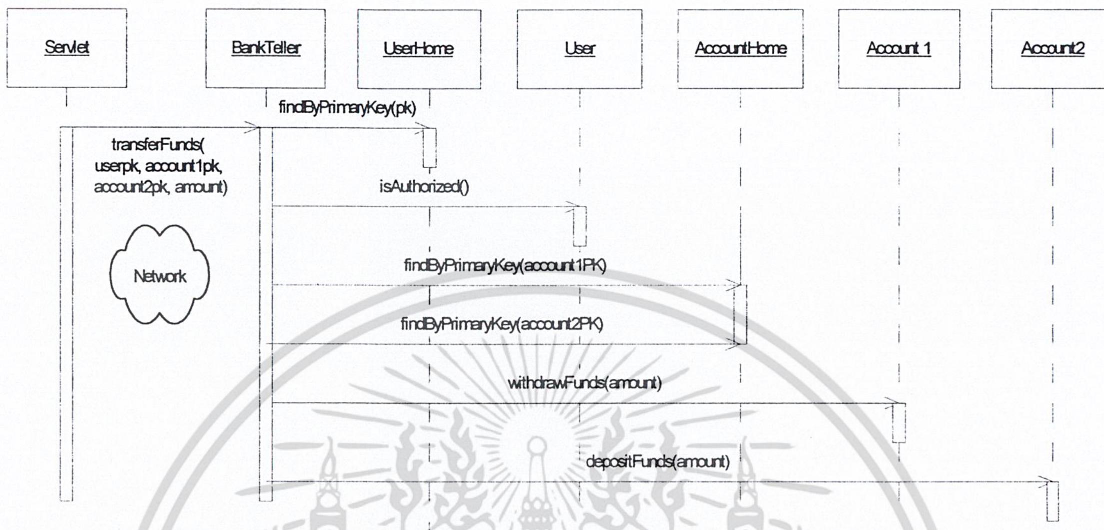
■ **Session Facade Pattern**

**ปัญหา** : ในระบบประกอบไปด้วยคอมโพเน็นต์จำนวนมาก และไคลเอ็นต์ต้องมีการเรียกใช้คอมโพเน็นต์เหล่านี้เป็นจำนวนมากแบบกระจัดกระจายในการทำงานหนึ่ง ๆ ทำให้เกิดความซับซ้อน และเกิด Tight Coupling ขึ้น และยังต้องมีการเรียกใช้งานเมธอดข้ามเครือข่ายจำนวนหลายครั้งอีกด้วย



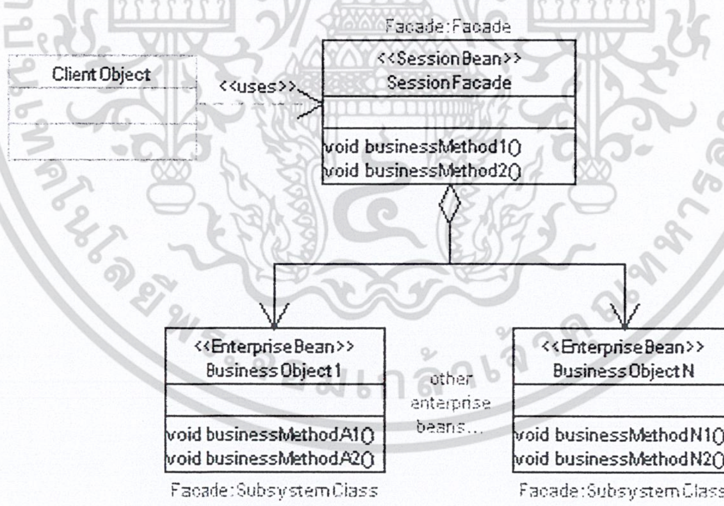

*รูปที่ 11-4 ไคลเอ็นต์เรียกใช้คอมโพเน็นต์ต่าง ๆ จากเซิร์ฟเวอร์โดยตรง*

**วิธีการ** : สร้างคอมโพเน็นต์ขึ้นมาครอบอีก 1 ชั้นที่มีอินเทอร์เฟซรวบรวมบริการที่จำเป็นไว้ทั้งหมดที่ตัวคอมโพเน็นต์นี้ โดยที่มันจะซ่อนความซับซ้อนจากการต้องติดต่อกับคอมโพเน็นต์หลาย ๆ ตัว จากไคลเอ็นต์และให้ ผู้ใช้เรียกใช้บริการผ่านคอมโพเน็นต์ตัวนี้แทน ดังรูปที่ 11-5




***รูปที่ 11-5 การเรียกใช้คอมโพเน็นต์ โดยใช้ Session Facade***

**โครงสร้าง**

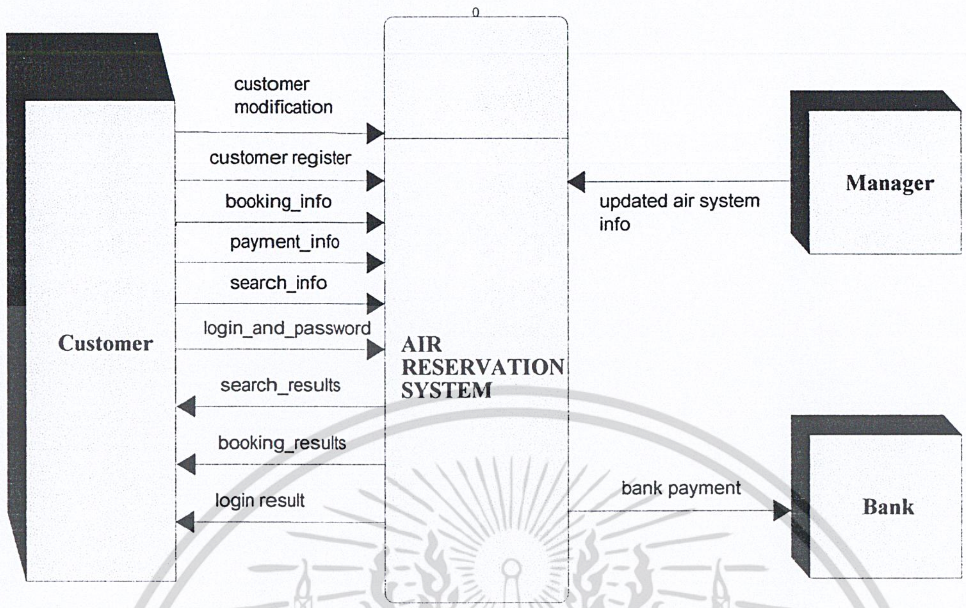



***รูปที่ 11-6 โครงสร้างของ Session Facade***

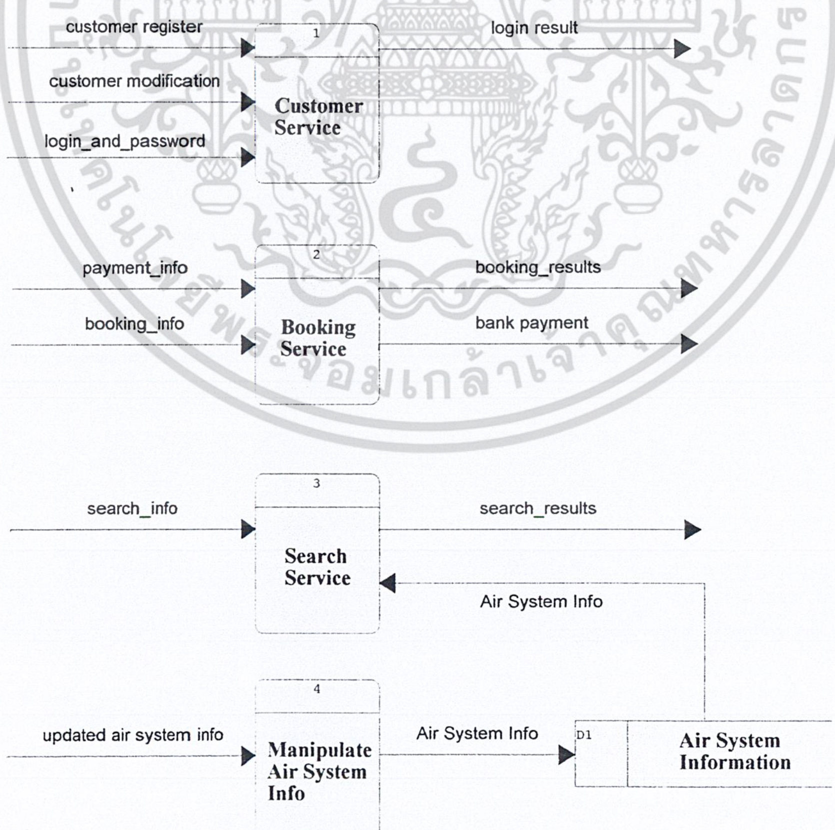
**ข้อดี :**

- ลดการเกิด Tight Coupling
- รวมการควบคุมเข้าสู่ศูนย์กลาง จัดการได้ง่ายขึ้น
- เพิ่มประสิทธิภาพจากการเรียกใช้งานที่มีการทำงานภายในมากๆแต่จำนวนครั้งการเรียกน้อย
- ลดจำนวน remote interface ที่ไคลเอ็นต์ ต้องเห็น

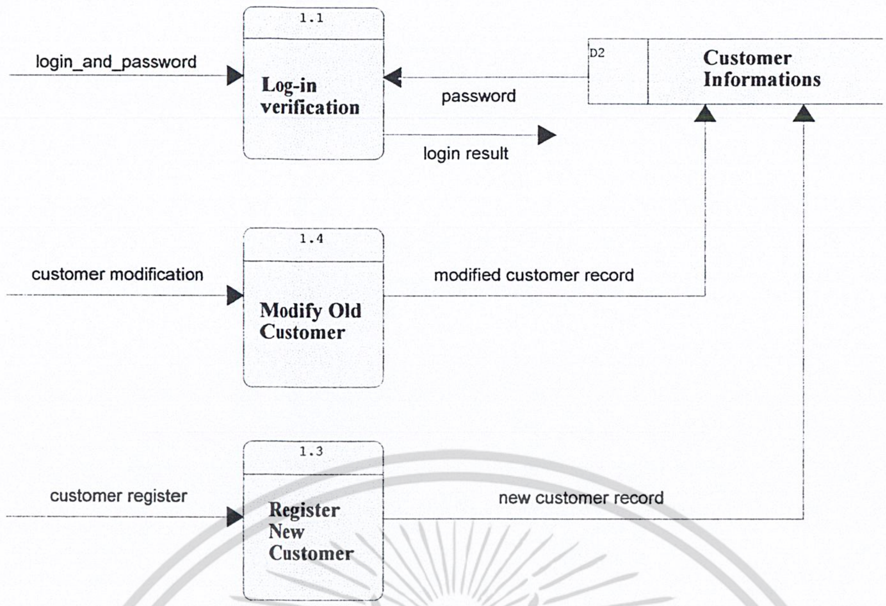
### 11.7.2 ดาต้าโฟล์วไดอะแกรมของระบบสายการบิน



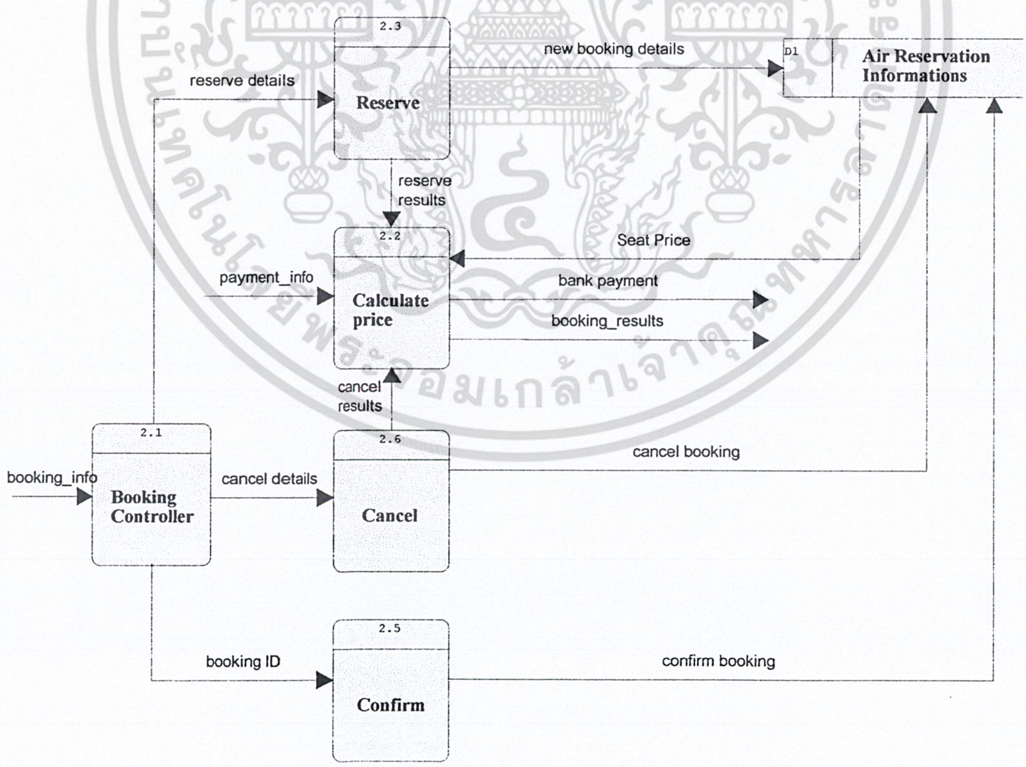

*รูปที่ 11-7 Air Reservation System Context Level*




*รูปที่ 11-8 Air Reservation System Level 1*




รูปที่ 11-9 *Air Reservation System Level 2*




รูปที่ 11-10 *Air Reservation System Level 2*

- **คำอธิบายดาต้าโฟล์วไดอะแกรม**

  ระบบสายการบิน ประกอบด้วย

**DataStore**

1. Customer Informations เก็บข้อมูลลูกค้าที่เข้ามาสั่งจอง
2. Air System Information เก็บข้อมูลสายการบินเกี่ยวกับเที่ยวบิน , ตารางเวลา , รุ่นของเครื่องบิน , สนามบินต่าง ๆ , ระดับราคา
3. Air Reservation Information เก็บข้อมูลเกี่ยวกับการจอง

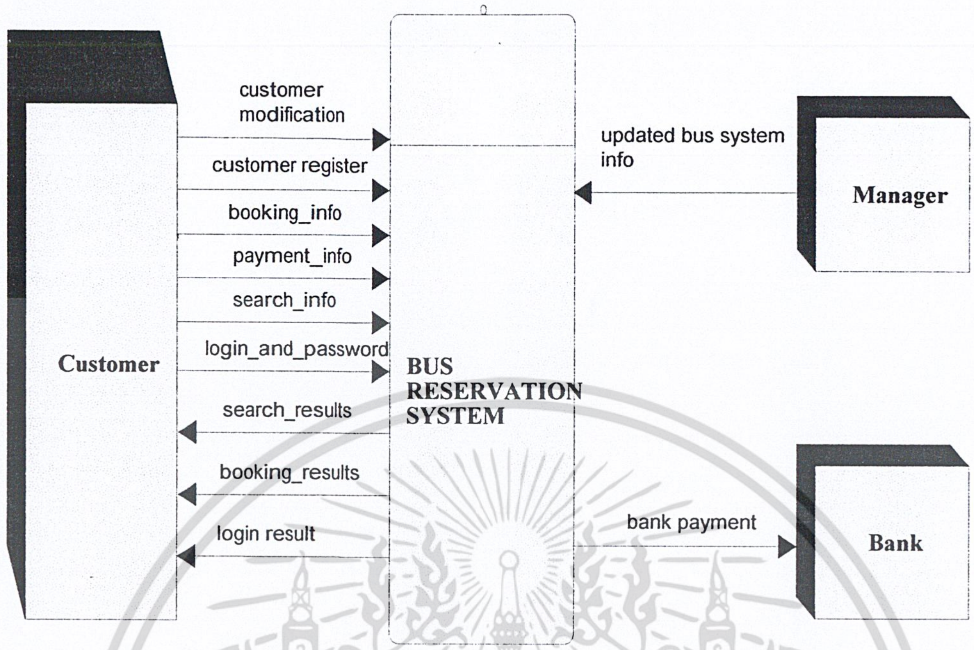
**External Entity**

1. Customer หมายถึงลูกค้าที่เข้ามาจอง
2. Manager หมายถึงผู้ดูแลระบบที่มีสิทธิในการแก้ไขข้อมูลของระบบได้ เช่น เพิ่มข้อมูลเที่ยวบิน
3. Bank คือ ระบบธนาคารที่มีหน้าที่รับการจ่ายเงินจากระบบนี้

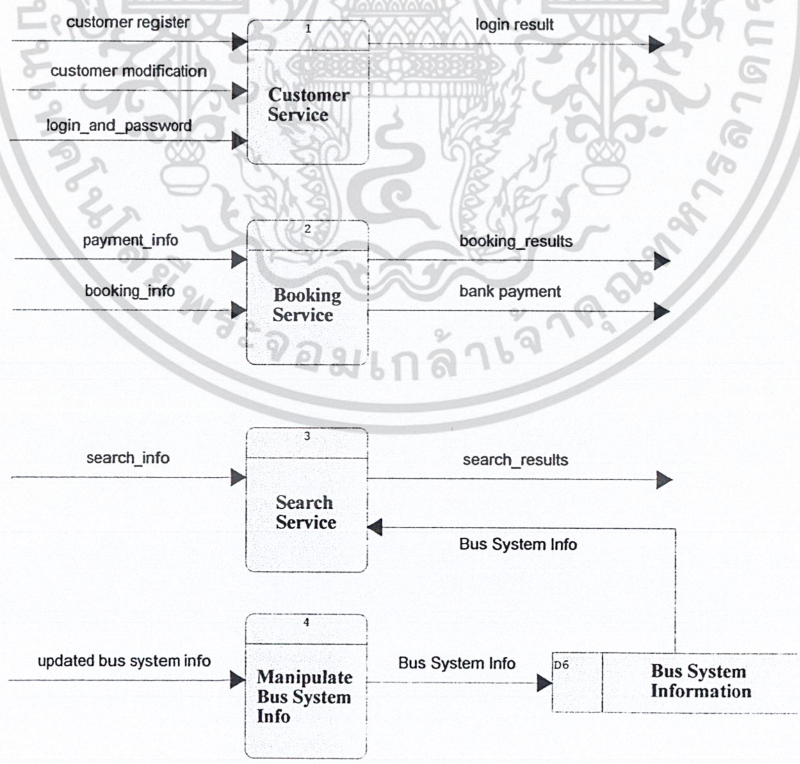
**Process**

1. Customer Service
   a. Log-in Verification ตรวจรหัสผ่านของสมาชิก
   b. Modify Old Customer แก้ไขข้อมูลสมาชิกเก่า
   c. Register New Customer ลงทะเบียนสมาชิกใหม่
2. Booking Service
   a. Reserve จอง
   b. Calculate Price คำนวณราคา
   c. Confirm ยืนยันการจอง
   d. Cancel ยกเลิกการจอง
3. Search Service ค้นหาที่นั่ง และเที่ยวบินที่ต้องการ
4. Manipulate Air System Info แก้ไขข้อมูลต่างๆในระบบ

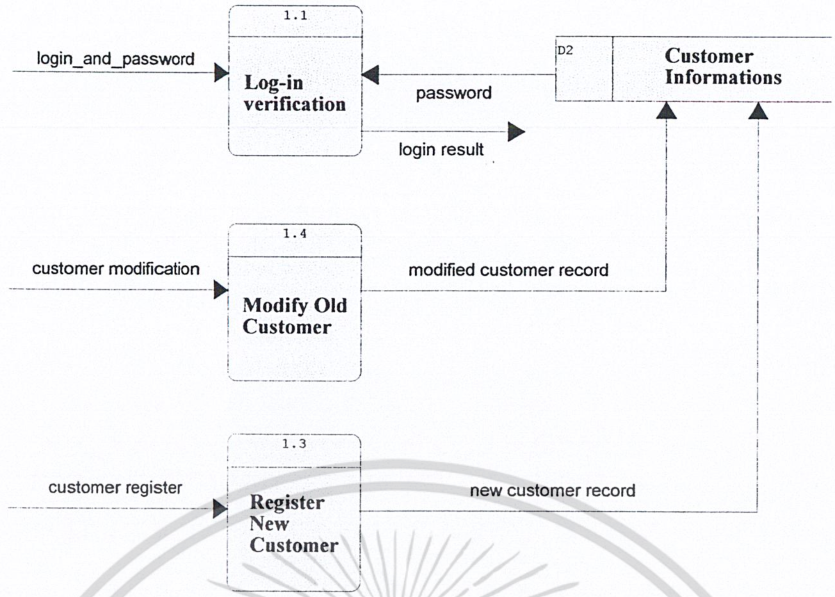
### 11.7.3 ดาต้าโฟล์วไดอะแกรมของระบบรถโดยสาร



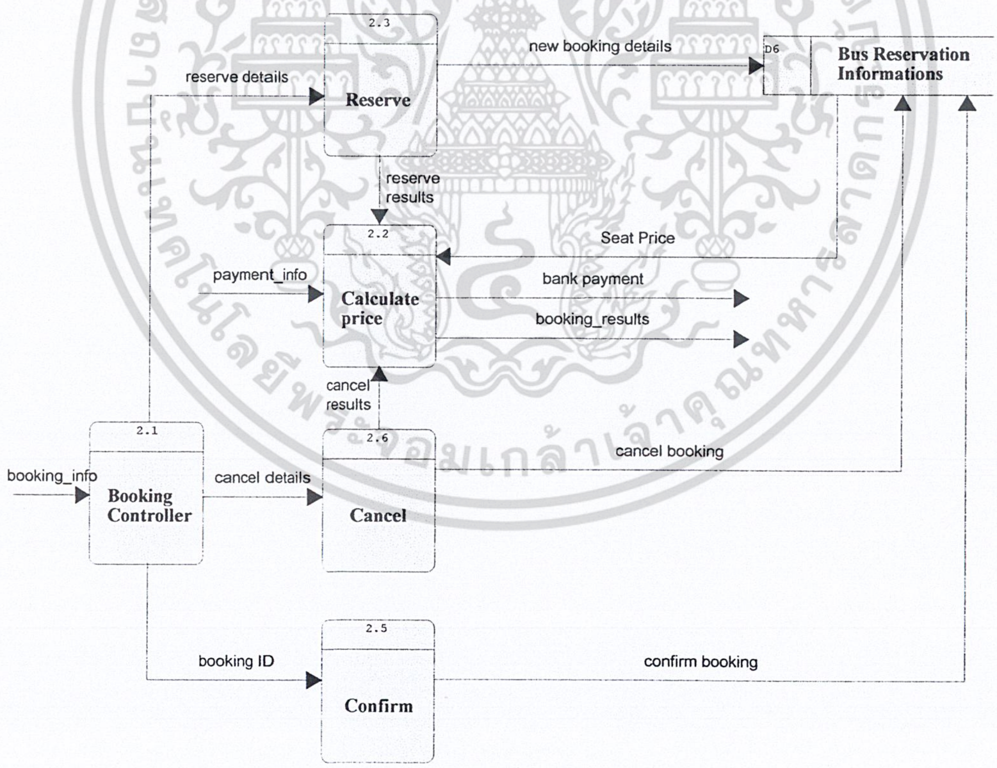

*รูปที่ 11-11 Bus Reservation System Context Level*




*รูปที่ 11-12 Bus Reservation System Level 1*




*รูปที่ 11-13 Bus Reservation System Level 2*




*รูปที่ 11-14 Bus Reservation System Level 2*

- **คำอธิบายดาต้าโฟล์วไดอะแกรม**

ระบบรถโดยสาร ประกอบด้วย

**DataStore**

1. Customer Informations เก็บข้อมูลลูกค้าที่เข้ามาสั่งจอง
2. Bus System Information เก็บข้อมูลรถโดยสาร
3. Bus Reservation Information เก็บข้อมูลเกี่ยวกับการจอง

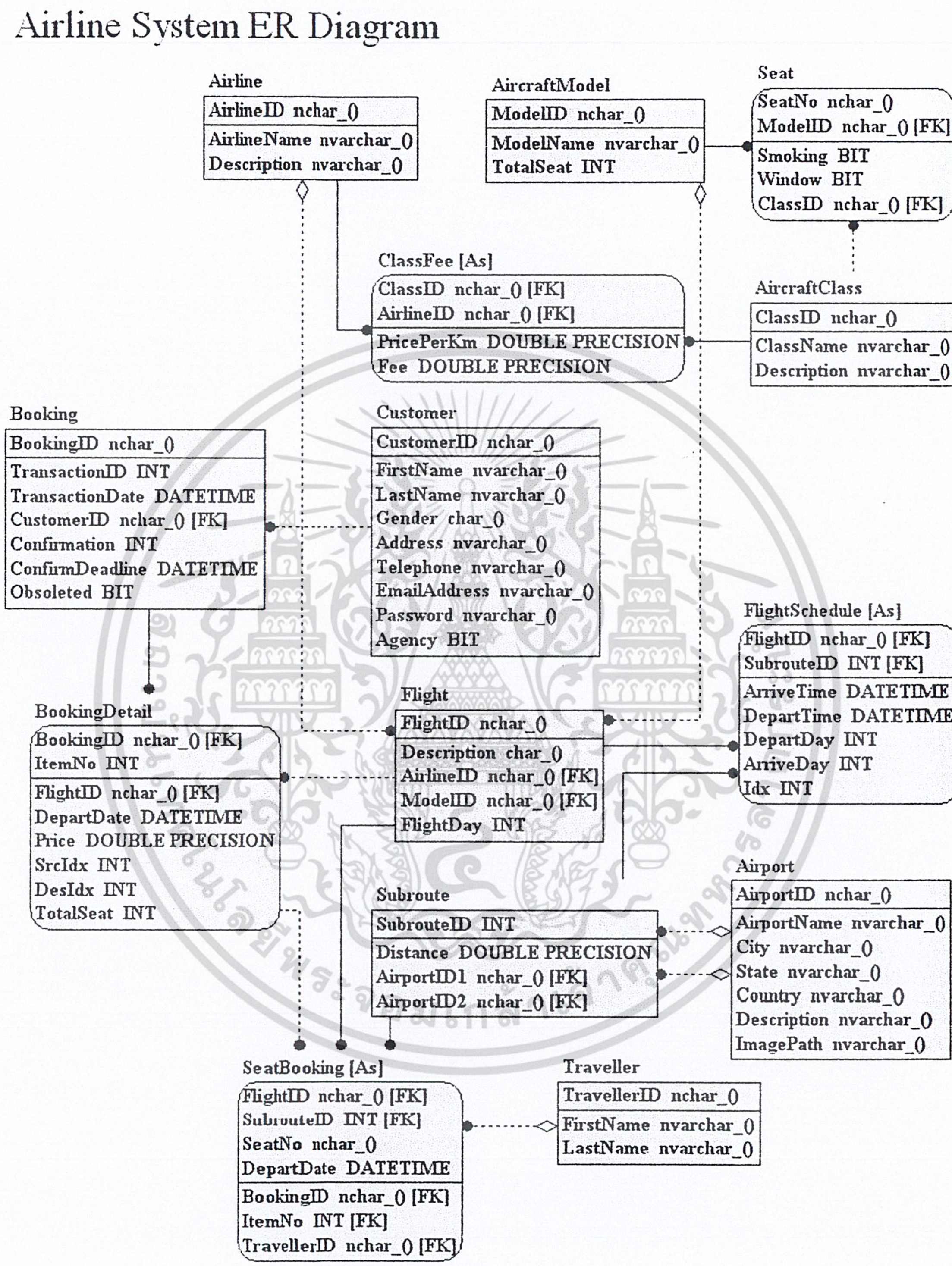
**External Entity**

1. Customer หมายถึงลูกค้าที่เข้ามาจอง
2. Manager หมายถึงผู้ดูแลระบบที่มีสิทธิในการแก้ไขข้อมูลของระบบได้ เช่น เพิ่มข้อมูลเที่ยวรถ
3. Bank คือ ระบบธนาคารที่มีหน้าที่รับการจ่ายเงินจากระบบนี้

**Process**

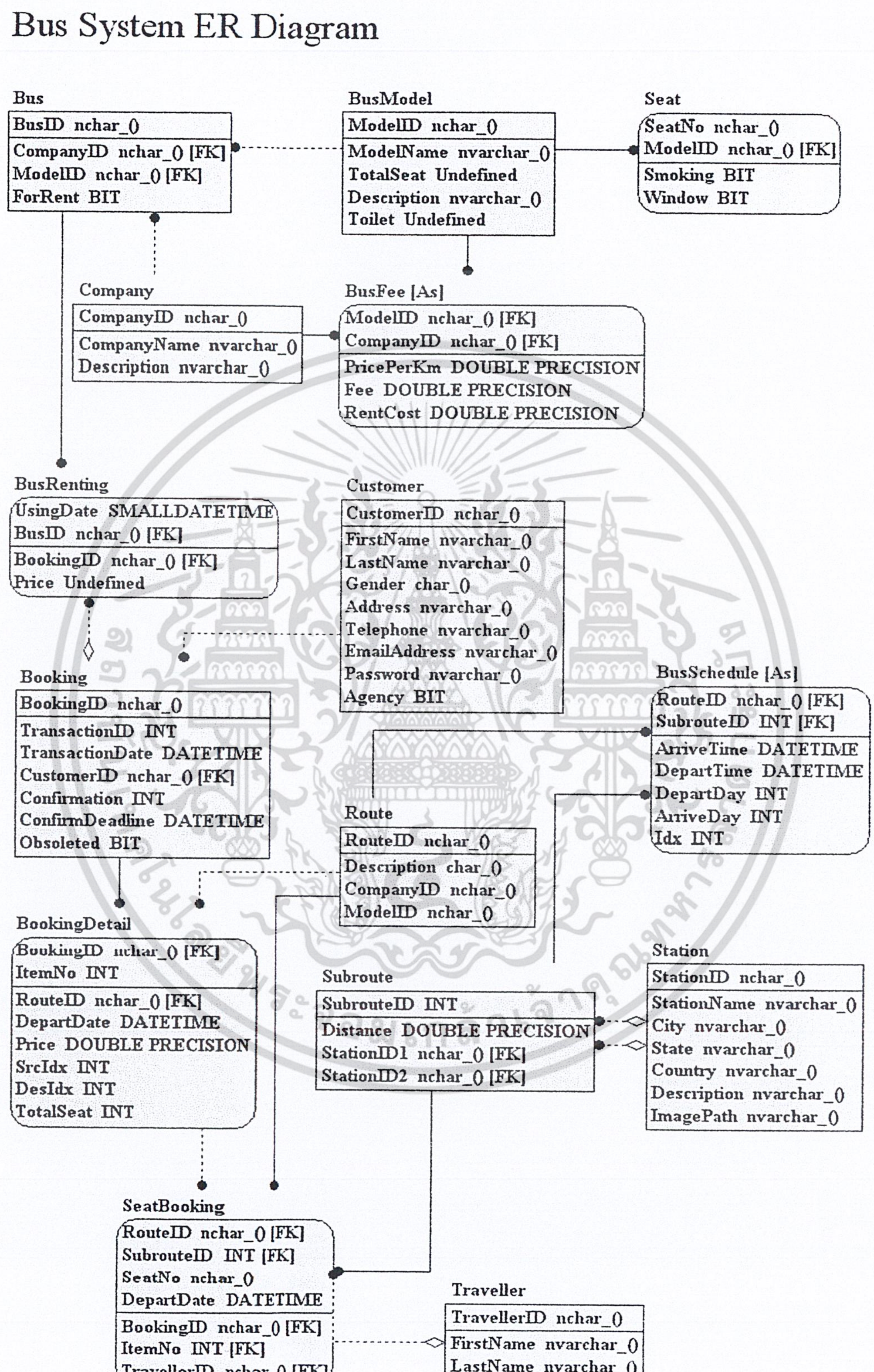
1. Customer Service
    a. Log-in Verification ตรวจรหัสผ่านของสมาชิก
    b. Modify Old Customer แก้ไขข้อมูลสมาชิกเก่า
    c. Register New Customer ลงทะเบียนสมาชิกใหม่
2. Booking Service
    d. Reserve จอง
    e. Calculate Price คำนวณราคา
    f. Confirm ยืนยันการจอง
    g. Cancel ยกเลิกการจอง
3. Search Service ค้นหาที่นั่ง และเที่ยวรถที่ต้องการ
4. Manipulate Air System Info แก้ไขข้อมูลต่างๆ ในระบบ

### 11.7.4 ER ไดอะแกรมของระบบสายการบิน



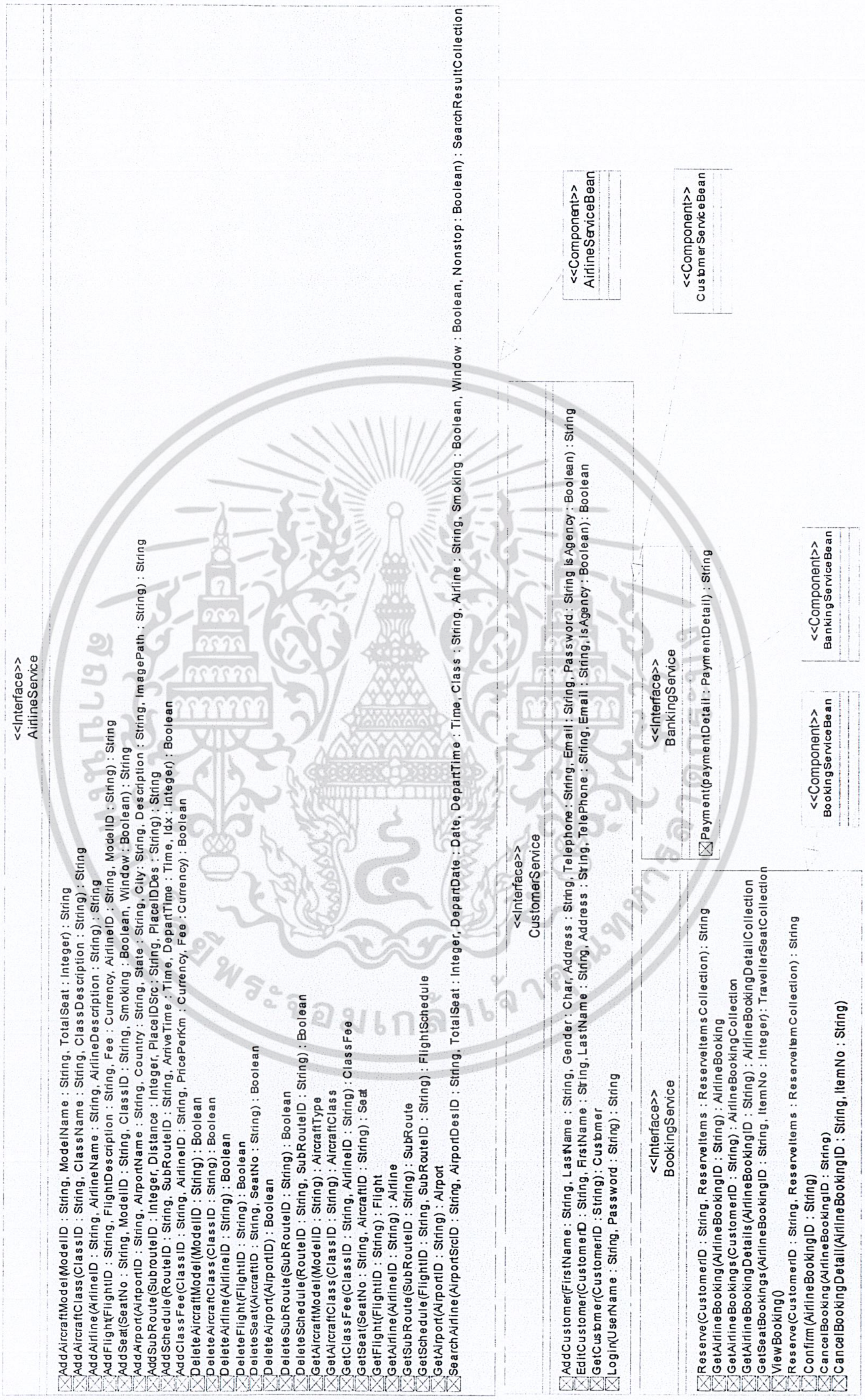

*รูปที่ 11-15 ER ไดอะแกรมของระบบสายการบิน*

### 11.7.5 ER ไดอะแกรมของระบบรถโดยสาร



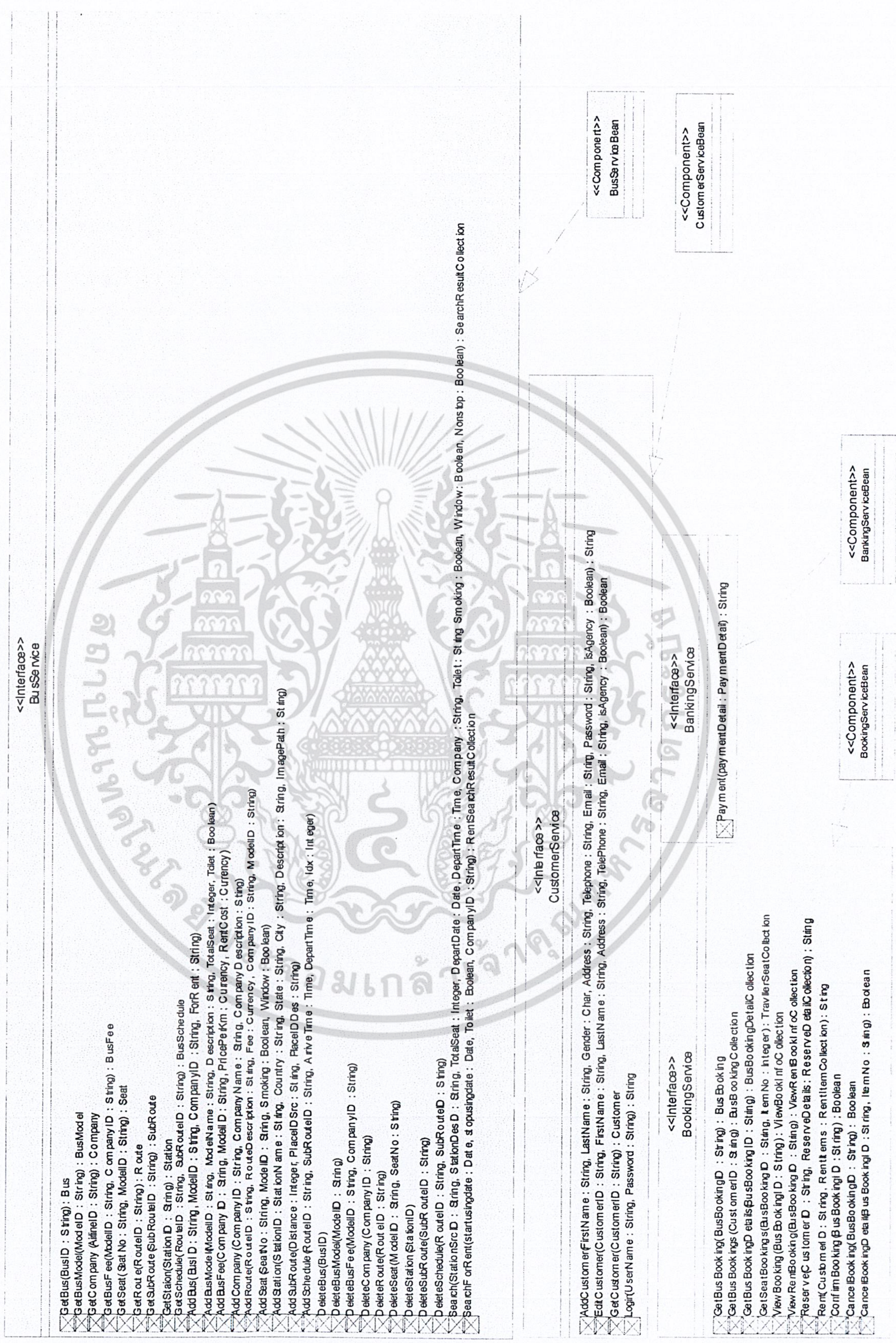

*รูปที่ 11-16 ER ไดอะแกรมของระบบรถโดยสาร*

### 11.7.6 คลาสไดอะแกรมของระบบสายการบิน



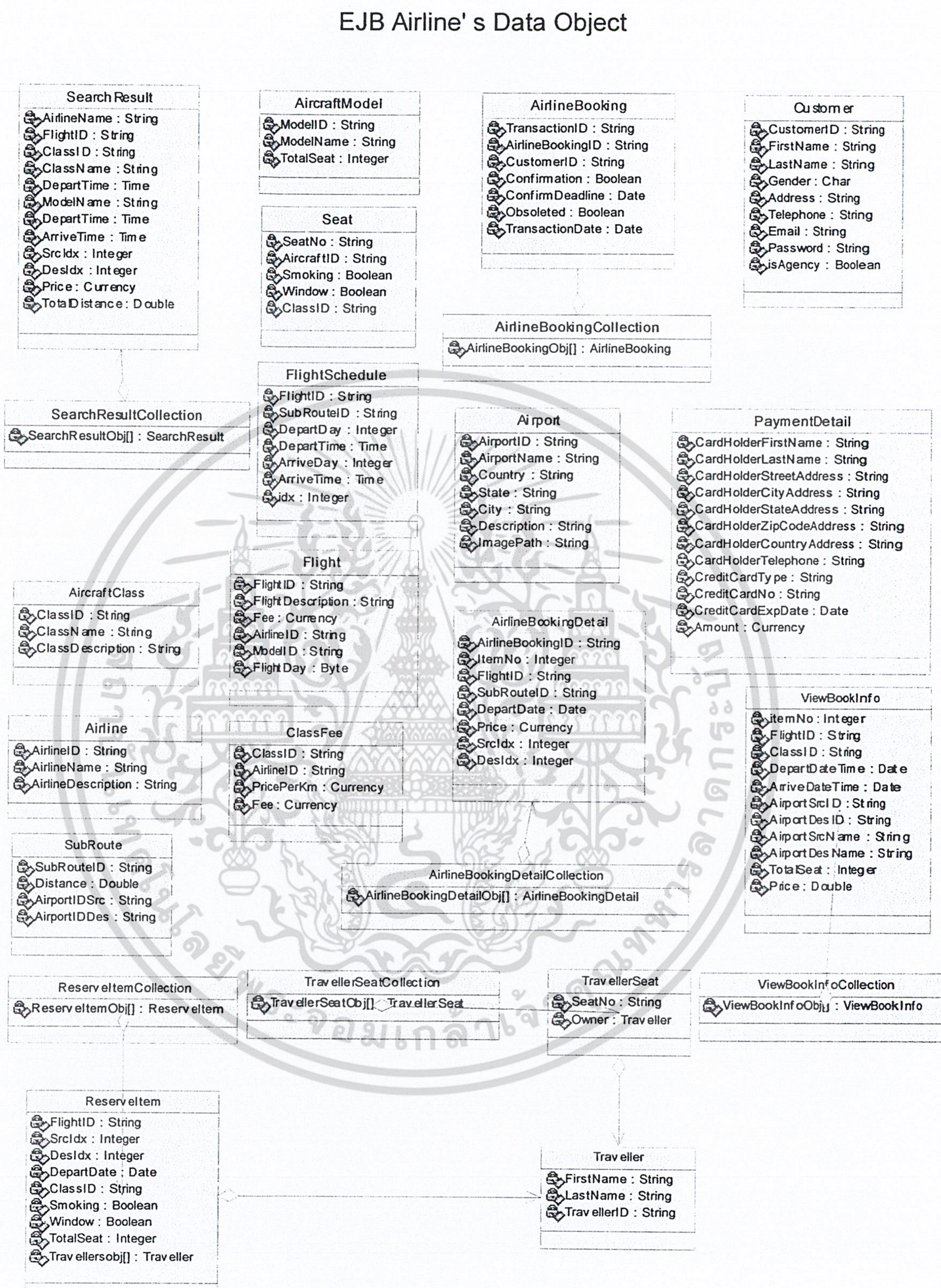

*รูปที่ 11-17 คลาสไดอะแกรมของระบบสายการบิน*

### 11.7.7 คลาสไดอะแกรมของระบบรถโดยสาร



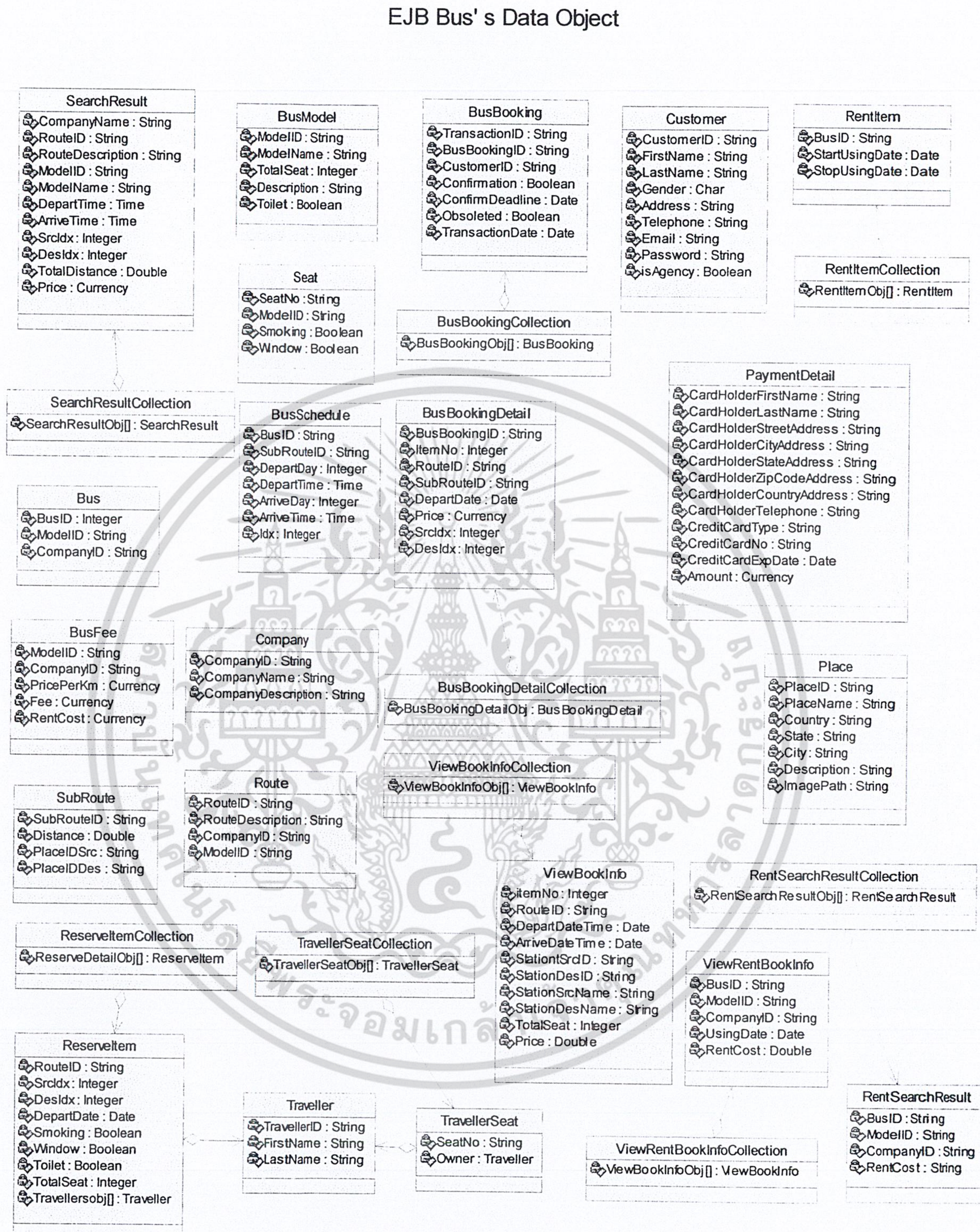

*รูปที่ 11- 18 คลาสไดอะแกรมของระบบรถโดยสาร*

### 11.7.8 คลาสไดอะแกรม (Data Object) ของระบบสายการบิน



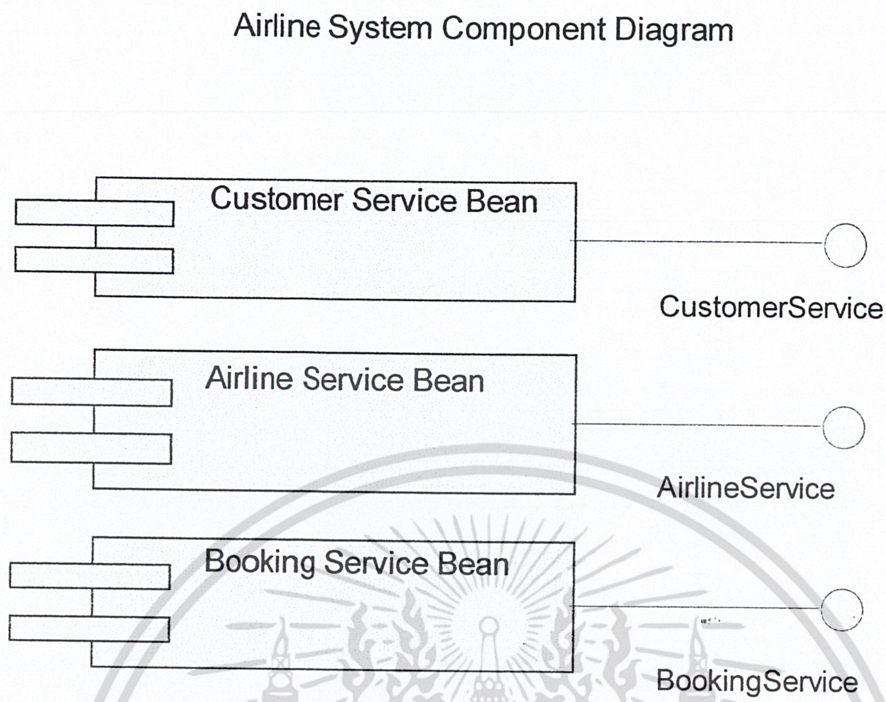

*รูปที่ 11-19 คลาสไดอะแกรม (Data Object) ของระบบสายการบิน*

### 11.7.9 คลาสไดอะแกรม ( Data Object ) ของระบบรถโดยสาร



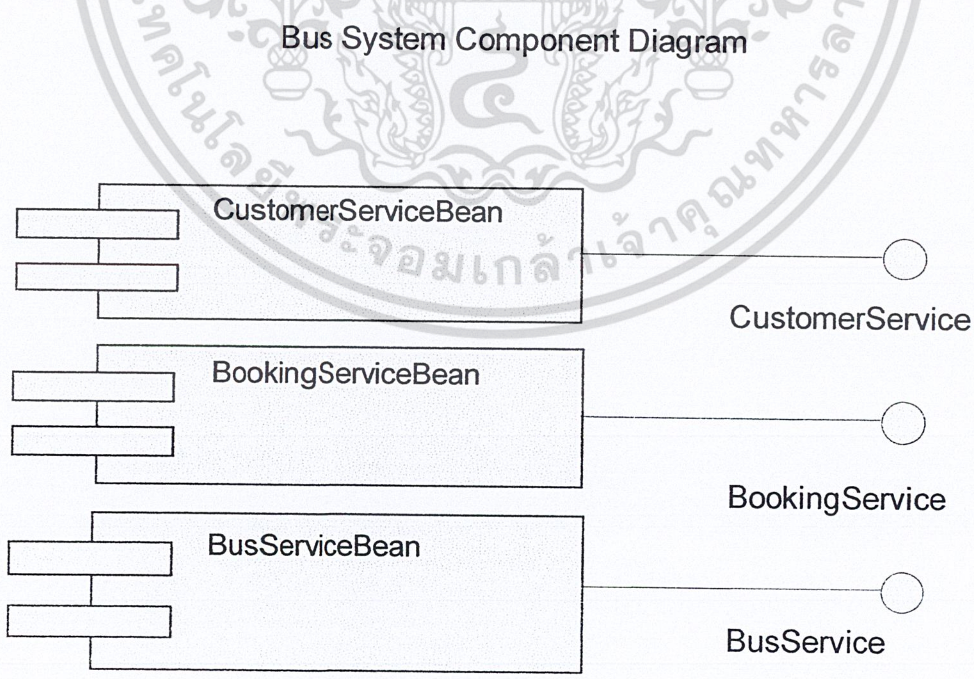

*รูปที่ 11-20 คลาสไดอะแกรม (Data Object) ของระบบรถโดยสาร*

**11.7.10 คอมโพเน็นต์ไดอะแกรมของระบบสายการบิน**




*รูปที่ 11-21 คอมโพเน็นต์ไดอะแกรมของระบบสายการบิน*

**11.7.11 คอมโพเน็นต์ไดอะแกรมของระบบรถโดยสาร**




*รูปที่ 11-22 คอมโพเน็นต์ไดอะแกรมของระบบรถโดยสาร*

## 11.8 การสร้างโปรแกรม ( Coding )

### 11.8.1 ระบบสายการบิน ( Airline System )

ระบบสายการบิน ประกอบด้วย 3 stateless session bean ดังนี้

**CustomerServiceBean**

ให้บริการเกี่ยวกับการสมัครสมาชิก การแก้ไขข้อมูลของสมาชิกโดยคอมโพเน็นต์นี้จะพัฒนาเพียงครั้งเดียว แล้วนำไปใช้ในทุกระบบ โดยการทำงานของมันจะไปแก้ไขข้อมูลกับ Customer entity bean เมธอดมีดังนี้

1. AddCustomer() ใช้สำหรับเพิ่มสมาชิกใหม่เข้าไปในระบบ โดยที่อีเมล์แอดเดรสจะต้องไม่ซ้ำกับของผู้อื่นที่มีอยู่แล้ว
2. EditCustomer() ใช้สำหรับแก้ไขข้อมูลสมาชิกเก่า ผู้ที่มีสิทธิแก้ไขได้จะต้องล็อกอินเข้าไป
3. GetCustomer() ใช้สำหรับเรียกดูข้อมูลสมาชิก โดยให้หมายเลขสมาชิกเข้าไป
4. Login() ใช้สำหรับตรวจสอบสิทธิการเข้าใช้งานระบบ โดยใช้อีเมล์แอดเดรสเป็นชื่อล็อกอิน

**BookingServiceBean**

ให้บริการเกี่ยวกับการจอง คือสามารถสั่งจอง ดูรายละเอียดการจองที่จองไปแล้ว ยืนยัน หรือ ยกเลิกการจอง

1. Reserve() ใช้สำหรับจองที่นั่ง โดยสามารถจองได้หลายรายการในครั้งเดียว แต่ละรายการก็จะมีรายละเอียดการจองต่างกันออกไป เช่น เที่ยวบิน , วันเดินทางไป-กลับ , เวลา , ลักษณะของที่นั่งที่ต้องการ , จำนวนที่นั่ง เสร็จแล้วระบบสร้างบันทึกข้อมูลการจองพร้อมรายละเอียดเฉพาะของผู้เดินทาง เช่น ชื่อ นามสกุล และจะพยายามเริ่มจองที่นั่งให้ทีละ 1 ที่จนกว่าจะครบตามกำหนด เมื่อเสร็จเรียบร้อยแล้วก็จะส่งหมายเลขการจอง (BookingID) กลับมาให้
2. Confirm() ใช้สำหรับยืนยันการจองโดยที่ ต้องเป็นไปตามกฎของบริษัท คือจะต้องยืนยันการจองพร้อมจ่ายเงินภายในช่วงจำนวนวันที่กำหนดก่อนวันเดินทางของการเดินทางที่มาถึงเร็วที่สุดใน Booking ใบนั้น ๆ เมื่อยืนยันการจองแล้ว ระบบจะเปลี่ยนสถานะ Confirmation จาก “0” เป็น “1”
3. CancelBooking() ใช้สำหรับยกเลิกการจองทุกรายการที่มีอยู่ใน Booking ซึ่งจะมีผลทำให้ระบบ ลบข้อมูลการจองออกจากฐานข้อมูลทันทีและเปิดโอกาสให้ลูกค้าคนอื่นเข้ามาจองแทนได้
4. CancelBookingDetail() ใช้สำหรับยกเลิกการเฉพาะรายการ เช่นในกรณีที่ยกเลิกการเดินทางกับเที่ยวบินบางเที่ยว

5. GetBooking() ใช้สำหรับให้ลูกค้าเข้าไปดูข้อมูลการจองที่ได้จองไปแล้วโดยระบุหมายเลขการจอง ( BookingID ) ซึ่งจะแสดงสถานะ การจองว่า Confirm แล้วหรือยัง หมดเขตการ Confirm เมื่อใด เป็นต้น
6. GetBookings() เหมือนกับ GetBooking() แต่จะแสดงทุก ๆ Booking ที่เป็นของ Customer คนนั้น
7. ViewBooking() ใช้ดูรายการจอง ทุกรายการ ภายใน Booking นั้น ๆ ซึ่งจะบอกรายละเอียดการเดินทางแบบประจำทาง เช่น เที่ยวบิน วันเดินทาง สนามบินต้นทางปลายทาง เป็นต้น

**AirlineServiceBean**

ให้บริการเกี่ยวกับการค้นหาข้อมูลสายการบินและการเรียกดูรายละเอียดข้อมูลต่างๆ ของระบบ เช่น เที่ยวบิน , สายการบิน , สนามบิน , ราคาสำหรับแต่ละระดับที่นั่ง ( Class )และสายการบิน

1. Search() ใช้สำหรับค้นหาที่นั่งตามความต้องการของลูกค้า โดยถ้าหากสามารถค้นหาเที่ยวบินที่มีลักษณะตรงตามความต้องการของลูกค้าและมีจำนวนที่นั่งเพียงพอ ก็จะส่งกลับมา เวลาต้องการจองก็นำข้อมูลที่ได้จากการค้นหานี้ไปจอง อย่างไรก็ดี ส่วนเลขที่นั่งนั้นจะรู้เมื่อได้ตกลงสั่งจองไปแล้ว และเข้าไปดูในหน้า ข้อมูลการจองและดูรายละเอียดผู้เดินทาง ซึ่งจะบอกว่า ใครได้ที่นั่งหมายเลขอะไร

**11.8.2 ระบบรถโดยสาร ( Bus System )**

ระบบรถโดยสาร ประกอบด้วย 3 stateless session bean ดังนี้

**CustomerServiceBean**

ให้บริการเกี่ยวกับการสมัครสมาชิก การแก้ไขข้อมูลของสมาชิก มีเมธอดดังนี้

1. AddCustomer() ใช้สำหรับเพิ่มสมาชิกใหม่เข้าไปในระบบ โดยที่อีเมล์แอดเดรสจะต้องไม่ซ้ำกับของผู้อื่นที่มีอยู่แล้ว
2. EditCustomer() ใช้สำหรับแก้ไขข้อมูลสมาชิกเก่า ผู้ที่มีสิทธิแก้ไขได้จะต้องล็อกอินเข้าไป
3. GetCustomer() ใช้สำหรับเรียกดูข้อมูลสมาชิก โดยให้หมายเลขสมาชิกเข้าไป
4. Login() ใช้สำหรับตรวจสอบสิทธิการเข้าใช้งานระบบ โดยใช้อีเมล์แอดเดรสเป็นชื่อล็อกอิน

**BookingServiceBean**

ให้บริการเกี่ยวกับการจอง คือสามารถสั่งจอง ดูรายละเอียดการจองที่จองไปแล้ว ยืนยัน หรือ ยกเลิกการจอง

1. Reserve() ใช้สำหรับจองที่นั่ง โดยสามารถจองได้หลายรายการในครั้งเดียว แต่ละรายการก็จะมีรายละเอียดการจองต่างกันออกไป เช่น เที่ยวรถ , วันเดินทางไป-กลับ , เวลา , ลักษณะของที่นั่งที่ต้องการ , จำนวนที่นั่ง เสร็จแล้วระบบสร้างบันทึกข้อมูลการจองพร้อมรายละเอียดเฉพาะของผู้เดินทาง เช่น ชื่อ นามสกุล และจะพยายามเริ่มจองที่นั่งให้ทีละ 1 ที่จน

กว่าจะครบตามกำหนด เมื่อเสร็จเรียบร้อยแล้วก็จะส่งหมายเลขการจอง (BookingID) กลับมาให้

2. Rent() ใช้สำหรับ การเช่า ซึ่งสามารถเช่าได้หลาย ๆ รายการที่แตกต่างกันในรายละเอียดเช่น วันเดินทางและหมายเลขรถได้ การเช่านั้นไม่ต้องการข้อมูลผู้เดินทางเหมือนการจอง
3. Confirm() ใช้สำหรับยืนยันการจองโดยที่ ต้องเป็นไปตามกฎของบริษัท คือจะต้องยืนยันการจองพร้อมจ่ายเงินภายในช่วงจำนวนวันที่กำหนดก่อนวันเดินทางของการเดินทางที่มาถึงเร็วที่สุดใน Booking ใบนั้น ๆ เมื่อยืนยันการจองแล้ว ระบบจะเปลี่ยนสถานะ Confirmation จาก “0” เป็น “1”
4. CancelBooking() ใช้สำหรับยกเลิกการจองทุกรายการที่มีอยู่ใน Booking ซึ่งจะมีผลทำให้ระบบ ลบข้อมูลการจองออกจากฐานข้อมูลทันทีและเปิดโอกาสให้ลูกค้าคนอื่นเข้ามาจองแทนได้
5. CancelBookingDetail() ใช้สำหรับยกเลิกการเฉพาะรายการ เช่นในกรณีที่ยกเลิกการเดินทางกับเที่ยวรถบางเที่ยว
6. GetBooking() ใช้สำหรับให้ลูกค้าเข้าไปดูข้อมูลการจองที่ได้จองไปแล้วโดยระบุหมายเลขการจอง (BookingID) ซึ่งจะแสดงสถานะ การจองว่า Confirm แล้วหรือยัง หมดเขตการ Confirm เมื่อใด เป็นต้น
7. GetBookings() เหมือนกับ GetBooking() แต่จะแสดงทุก ๆ Booking ที่เป็นของ Customer คนนั้น
8. ViewBooking() ใช้ดูรายการจอง ทุกรายการ ภายใน Booking นั้น ๆ ซึ่งจะบอกรายละเอียดการเดินทางแบบประจำทาง เช่น เที่ยวรถ วันเดินทาง สถานที่ต้นทางปลายทาง เป็นต้น
9. ViewRentBooking() เหมือนกับ ViewBooking() แต่เป็นข้อมูลของการเช่ารถ ซึ่งจะบอกว่าเช่ารถหมายเลขอะไร ของบริษัทใดและวันใดบ้าง

**BusServiceBean**

ให้บริการเกี่ยวกับการค้นหาข้อมูลรถโดยสารและการเรียกดูรายข้อมูลละเอียดต่างๆ ของระบบ เช่น เที่ยวรถ , บริษัท , สถานี , ราคาสำหรับแต่ละรุ่นของรถและบริษัท เป็นต้น

1. Search() ใช้สำหรับค้นหาที่นั่งตามความต้องการของลูกค้า โดยถ้าหากสามารถค้นหาเที่ยวรถที่มีลักษณะตรงตามความต้องการของลูกค้าและมีจำนวนที่นั่งเพียงพอ ก็จะส่งกลับมาเวลาต้องการจองก็นำข้อมูลที่ได้จากการค้นหานี้ไปจอง อย่างไรก็ดี ส่วนเลขที่นั่งนั้นจะรู้เมื่อได้ตกลงสั่งจองไปแล้ว และเข้าไปดูในหน้า ข้อมูลการจองและดูรายละเอียดผู้เดินทาง ซึ่งจะบอกว่า ใครได้ที่นั่งหมายเลขอะไร

2. SearchRent() ใช้สำหรับหารถโดยสารสำหรับเช่า โดยระบุบริษัทที่ต้องการ ลักษณะรถเช่น มีห้องน้ำหรือไม่ และช่วงวันที่ต้องใช้ ระบุจะค้นหาหมายเลขรถโดยสารที่ยังว่าง ไม่ถูกเช่าในช่วงวันที่กำหนด แล้วส่งกลับมาให้

### 11.8.3 SOAP เซอร์วิส

การสร้าง SOAP เซอร์วิสจะทำผ่านจาวาคลาสที่จะเรียกใช้งาน enterprise bean อีกทีหนึ่ง เนื่องจาก SOAP toolkit ไม่สามารถทำงานร่วมกันได้ในกรณีชนิดของข้อมูลเป็น date หรือ time จึงจำเป็นต้องแปลงข้อมูลชนิด date หรือ time เป็นสตริงที่ได้กำหนดรูปแบบไว้ล่วงหน้า และจำเป็นที่จะต้องแปลงข้อมูลชนิด Collection เป็นอาร์เรย์ ตัวอย่างเมธอดของจาวาคลาส เมธอดเป็นดังนี้

```
AirSeatSearchResult[] Search(String airportsrcID, String airportdesID, int totalseat, String
departdate, String departtime, String airclass, String airline, boolean smoking, boolean
window, boolean nonstoponly)
```

แต่เมธอดของ bean จะเป็นดังนี้

```
Collection Search(String airportsrcID, String airportdesID, int totalseat, java.sql.Date
departdate, java.sql.Time departtime, String airclass, String airline, boolean smoking, boolean
window, boolean nonstoponly)
```

โดย Collection จะเป็น collection ของ AirSeatSearchResult

เหตุผลของการเลือกที่จะสร้างคลาสมาครอบแทนที่จะแก้ไข signature ของ bean นั้น เพราะ bean สามารถถูกเรียกได้จากแอพพลิเคชันที่เขียนด้วยภาษาจาวา ซึ่งไม่จำเป็นต้องแปลงข้อมูล การแก้ไข signature ของ bean ให้รองรับกับ SOAP จะทำให้ต้องแปลงข้อมูลทุกครั้ง ซึ่งเป็นสิ่งที่ไม่จำเป็นถ้าแอพพลิเคชันเขียนด้วยภาษาจาวา

## 11.9 การนำคอมโพเน็นต์กลับมาใช้ใหม่ในการพัฒนาโปรแกรม

ในระบบสายการบิน และระบบรถโดยสาร มีคอมโพเน็นต์ที่สามารถใช้ร่วมกันได้ คือ CustomerService ตามหลักการพัฒนาซอฟต์แวร์เชิงคอมโพเน็นต์ ผู้พัฒนาสามารถสร้างเพียงครั้งเดียว แต่นำมาแจกจ่ายให้ผู้อื่นดีพลอยได้หลายครั้งโดยไม่ต้องเปิดเผยซอร์สโค้ด ซึ่งจะดีกว่าการนำกลับมาใช้ใหม่แบบเดิม ๆ ที่ใช้วิธีการคัดลอกระดับซอร์สโค้ด แล้วนำมาตัดแปะ ในการดีพลอยคอมโพเน็นต์ใหม่ก็จะสามารถเปลี่ยนแปลง ( Customize ) ได้บางส่วนเช่น สามารถแก้ไขฐานข้อมูลที่ใช้ ( DBMS ) โดยที่ไม่ต้องเขียนคอมโพเน็นต์ใหม่ ดังนั้นในกรณีที่เป็นคอมโพเน็นต์ที่ติดต่อกับฐานข้อมูลก็เพียงแต่ แก้ deployment descriptor เพื่อให้ระบุรายละเอียดการติดต่อฐานข้อมูลที่จะใช้ให้ถูกต้อง แล้วก็ดีพลอยใหม่เท่านั้น อย่างไรก็ดี ผู้ที่จะนำเอาคอมโพเน็นต์ไปใช้ก็ต้องสร้าง schema ของฐานข้อมูลรอรับเอาไว้ให้มีลักษณะตามที่คอมโพเน็นต์ต้องการ ผู้พัฒนาคอมโพเน็นต์ก็สามารถแจกจ่าย SQL สคริปต์สำหรับสร้าง schema ไปพร้อมกับคอมโพเน็นต์ได้

การแก้ไข deployment descriptor นั้นขึ้นอยู่กับผลิตภัณฑ์ที่ใช้อยู่เป็นหลักว่า ใช้แอพพลิเคชันเซิร์ฟเวอร์ของบริษัทใด เพราะแต่ละผลิตภัณฑ์ก็จะมีวิธีการ customize ในจุดนี้ที่ไม่เหมือนกัน ดังนั้นผู้ที่จะนำคอมโพเน็นต์ไปใช้ จะต้องรู้จักวิธีการใช้งานและโครงสร้างของ deployment descriptor ของผลิตภัณฑ์นั้น ๆ

สำหรับ EJB เมื่อผู้พัฒนาจะต้องแจกจ่ายคอมโพเน็นต์ให้ผู้ใช้ จะประกอบด้วย

- Home Interface
- Remote Interface
- Bean Implementation Class

ทั้ง 3 ชิ้น เป็นส่วนของโปรแกรมที่ถูกคอมไพล์แล้ว และส่วนมากมักจะนิยมการบรรจุมันให้อยู่ในรูปไฟล์ .jar (Java Archives) โดย .jar สามารถมีได้หลาย enterprise bean

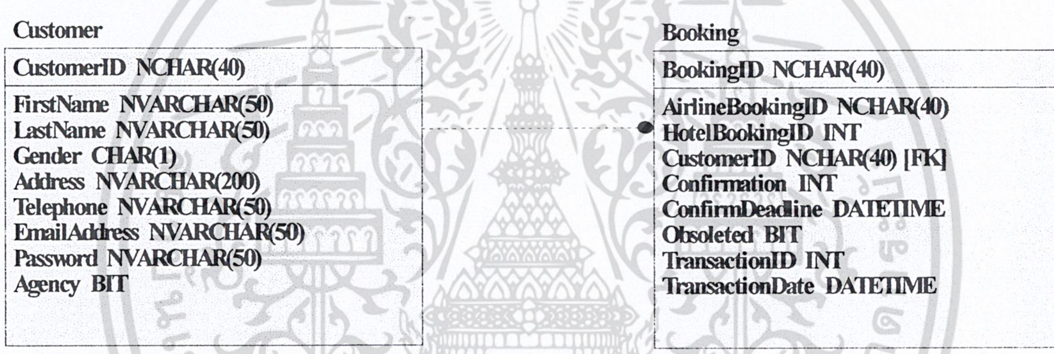
# บทที่ 12

# การออกแบบระบบทัวร์

ระบบทัวร์เป็นเว็บแอพพลิเคชันที่จะเรียกใช้คอมโพเน็นต์ของระบบสายการบิน และระบบโรงแรม ดังนั้นการออกแบบจึงมีเพียงส่วนของการเรียก SOAP เซอร์วิสของระบบโรงแรม และส่วนที่เรียกใช้คอมโพเน็นต์ของระบบสายการบินที่เป็นสถาปัตยกรรมเดียวกัน

## 12.1 ER ไดอะแกรมของระบบทัวร์

ระบบทัวร์จะมี ER ดังรูปที่ 12-1 เนื่องจากระบบทัวร์จะเก็บแค่ข้อมูลของลูกค้าและรายละเอียดการจองเท่านั้น แต่จะไม่มีรายละเอียดของระบบสายการบินและระบบโรงแรม




*รูปที่ 12-1 ER ไดอะแกรมของระบบทัวร์*

## 12.2 การเขียนโปรแกรม

ระบบทัวร์จะประกอบด้วย 2 stateless session bean ดังนี้

**CustomerServiceBean**

ให้บริการเกี่ยวกับการสมัครสมาชิก การแก้ไขข้อมูลของสมาชิก ซึ่งจะเหมือนกับระบบรถโดยสารและระบบสายการบิน

**BookingServiceBean**

ให้บริการเกี่ยวกับการจอง การยกเลิก การดูข้อมูลของรายการที่จอง และค้นหา แต่จะแตกต่างจากระบบอื่นที่ ระบบนี้จะเรียกใช้บริการจากระบบอื่น มีเมธอดดังนี้

1. Reserve() ใช้จองสายการบินและโรงแรม โดยการจองสามารถจองอย่างใดอย่างหนึ่งหรือทั้งทั้งสองอย่างก็ได้
2. GetCustomerBookings() ใช้เพื่อขอข้อมูลของการจองของลูกค้าแต่ละคน
3. GetBooking() ใช้เพื่อขอข้อมูลการจองของแต่ละรายการ
4. GetAirlineBooking() ใช้เพื่อขอข้อมูลการจองของสายการบิน
5. ViewAirlineBooking() ใช้ดูรายการจองทุกรายการของ Booking ของสายการบิน
6. GetHotelBooking() ใช้เพื่อขอข้อมูลการจองของโรงแรม
7. GetHotelBookingDetail() ใช้ดูรายการจองทุกรายการของ Booking ของโรงแรม
8. CancelBooking() ใช้ยกเลิกการจอง
9. CancelAirlineBooking() ใช้ยกเลิกการจองสายการบิน
10. CancelAirlineBookingDetail() ใช้ยกเลิกเฉพาะบางรายการของการจองสายการบิน
11. CancelHotelBooking() ใช้ยกเลิกการจองโรงแรม
12. CancelHotelBookingDetail() ใช้ยกเลิกเฉพาะบางรายการของการจองโรงแรม
13. ConfirmBooking() ใช้ยืนยันการจอง
14. SearchHotel() ใช้ค้นหาโรงแรม

# บทที่ 13

# ลักษณะของแอพพลิเคชัน

## 13.1 เว็บไซต์ OLALA AIR

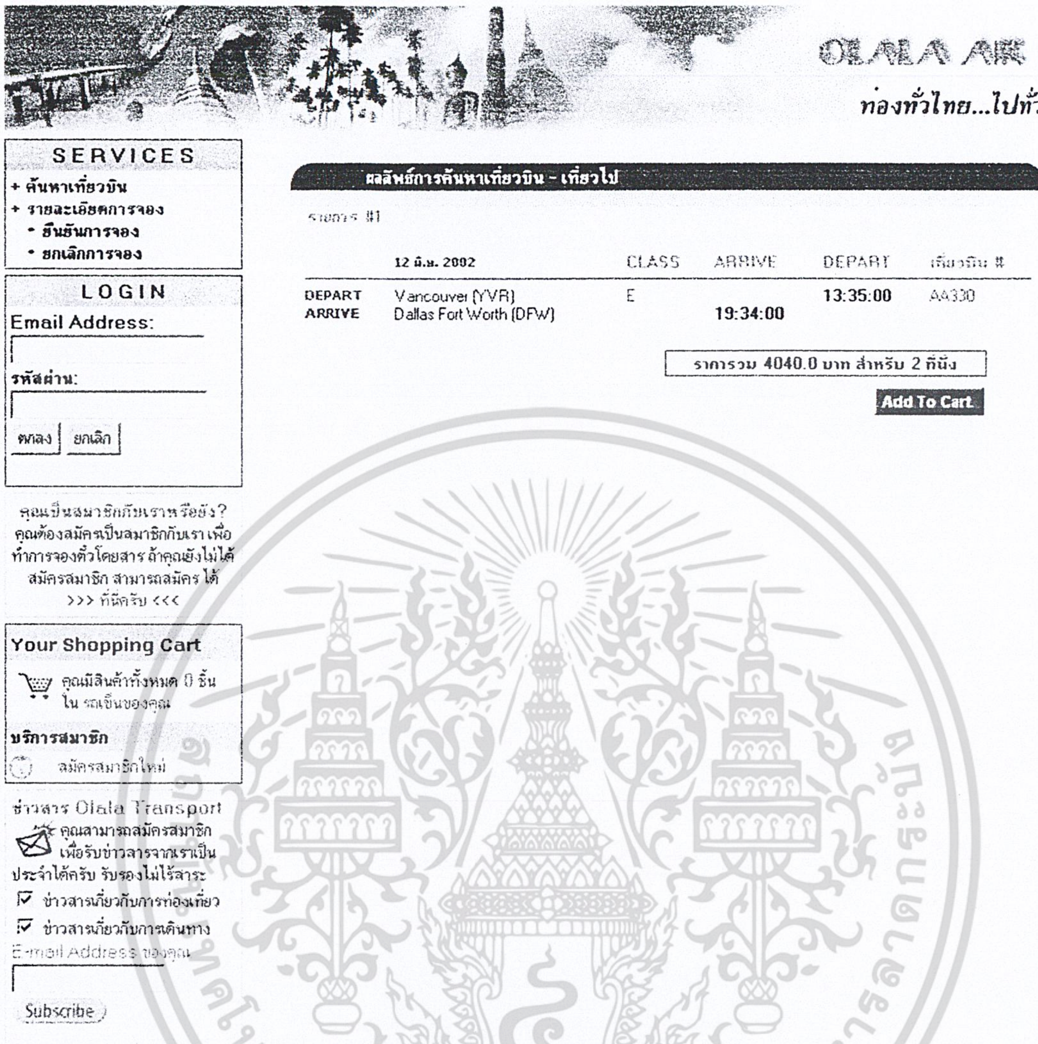
- หน้าแรก (index.jsp)


*รูปที่ 13- 1 หน้าแรกของเว็บสายการบิน*

หน้าแรกของเว็บไซต์ จะเป็นหน้าที่ผู้ใช้สามารถค้นหาข้อมูลเที่ยวบินได้ตามต้องการ โดยกรอกรายละเอียดที่จำเป็นในการค้นหาลงไป เช่น สนามบินต้นทาง สนามบินปลายทาง วันและเวลาเดินทาง และข้อมูลเกี่ยวกับประเภทของสายการบินและเที่ยวบินตามต้องการ

ผู้ใช้สามารถดูรายการของชื่อสนามบิน และรหัสสนามบินทั้งหมดได้ โดยการคลิกที่ "แสดงรหัสสนามบินทั้งหมด"

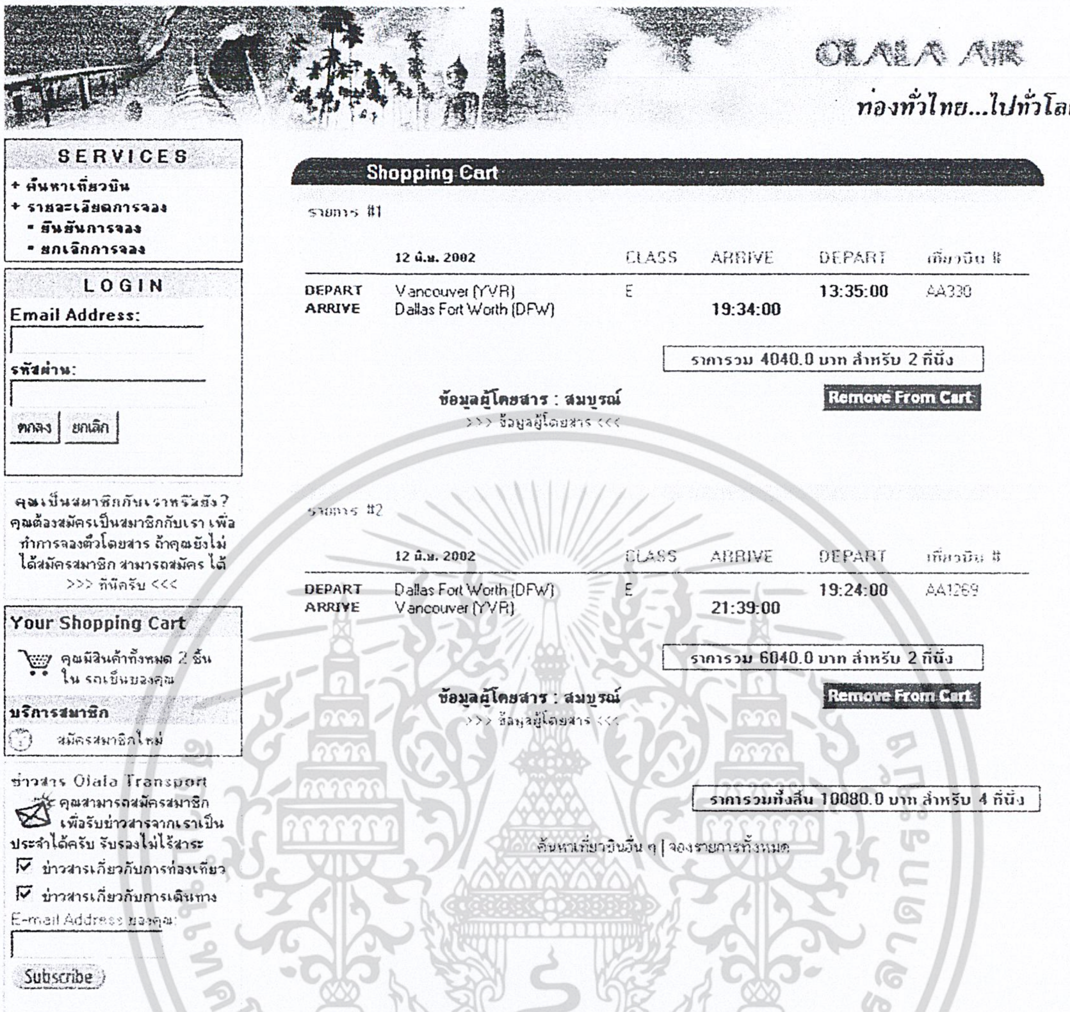
- ผลลัพธ์การค้นหา (search.jsp)




*รูปที่ 13-2 ผลลัพธ์การค้นหา*

เป็นหน้าจอแสดงผลลัพธ์การค้นหาเที่ยวบิน ซึ่งตรงตามข้อมูลที่ได้ระบุในการค้นหา โดยจะแสดงเที่ยวบิน พร้อมทั้งรายละเอียดต่าง ๆ เช่น วันเวลาเดินทาง ประเภทที่นั่ง และราคา หากลูกค้าสนใจจะทำการจองรายการใด สามารถคลิกที่ปุ่ม Add To Cart เพื่อเพิ่มรายการนั้นเข้าไปเก็บไว้ใน Shopping Cart ได้

- ข้อมูลในรถเข็น (cart.jsp)




*รูปที่ 13-3 ข้อมูลในรถเข็นของเว็บสายการบิน*

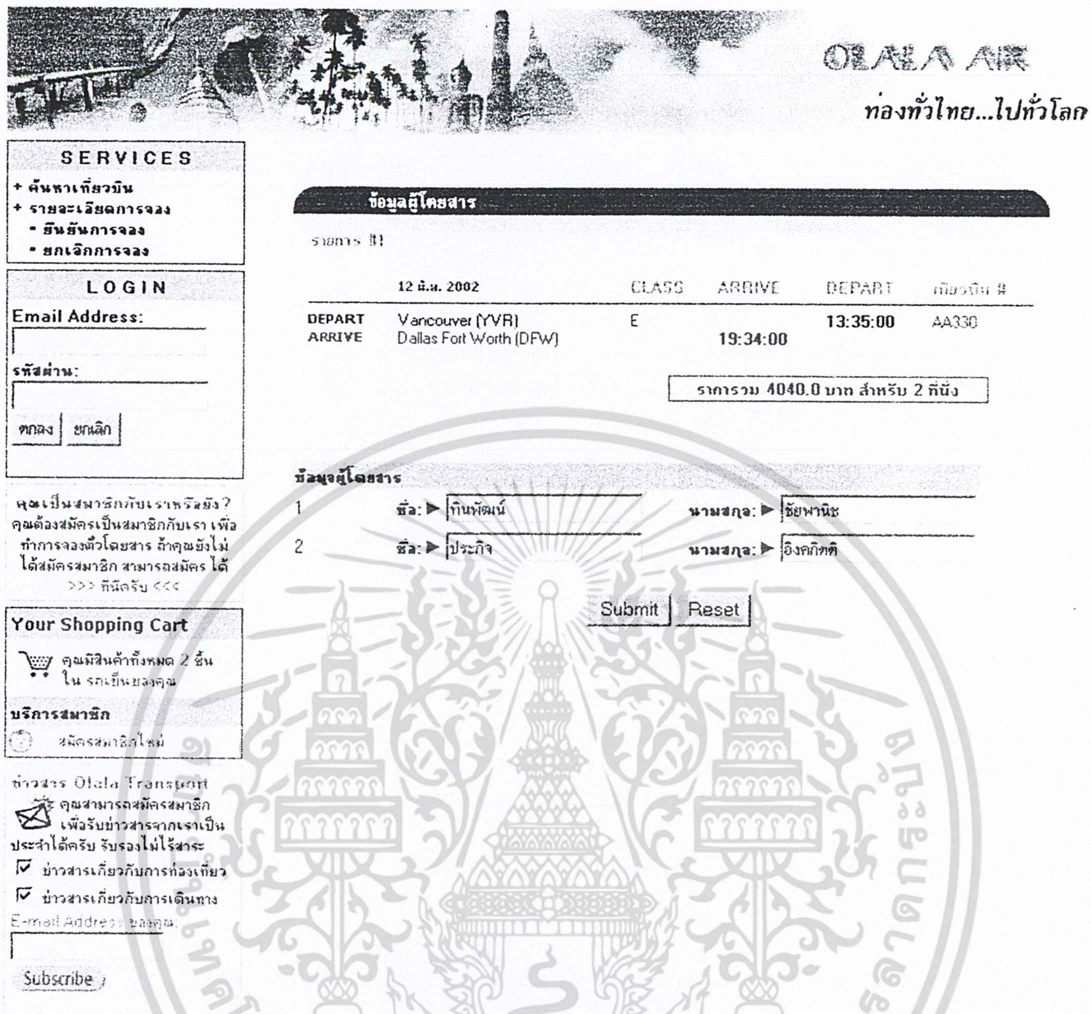
Shopping Cart จะเก็บข้อมูลเที่ยวบินที่คุณมีความสนใจจะจองเอาไว้ โดยคุณสามารถเพิ่มเที่ยวบินลงใน Shopping Cart ได้โดยการคลิกที่ปุ่ม Add To Cart ในหน้าจอผลลัพธ์การค้นหา

ผู้ใช้สามารถลบข้อมูลเที่ยวบินออกจาก Shopping Cart ได้ โดยการคลิกที่ปุ่ม Remove From Cart หากต้องการกลับไปทำการค้นหาเที่ยวบินอีกครั้ง ให้คลิกที่ปุ่ม "ค้นหาเที่ยวบินอื่น ๆ" หากต้องการทำการจองเที่ยวบินทั้งหมดที่อยู่ใน Shopping Cart ให้คลิกที่ปุ่ม "จองรายการทั้งหมด"

ก่อนที่ผู้ใช้ จะสามารถทำการจองโดยการคลิกที่ปุ่ม "จองรายการทั้งหมด" ได้ ผู้ใช้จะต้อง Login ให้เรียบร้อย และกรอกรายละเอียดผู้โดยสารสำหรับเที่ยวบิน แต่ละเที่ยวให้ครบถ้วน โดยในแต่ละรายการจะแสดงในส่วนของข้อมูลผู้โดยสารว่า ข้อมูลผู้โดยสารสำหรับรายการนั้น ๆ ครบถ้วนหรือยัง

หากผู้ใช้พยายามทำการจองรายการทั้งหมดใน Shopping Cart โดยที่ยังไม่ได้ Login เข้าสู่ระบบ หรือยังระบุข้อมูลผู้โดยสารไม่ครบถ้วน จะปรากฏข้อผิดพลาดขึ้น

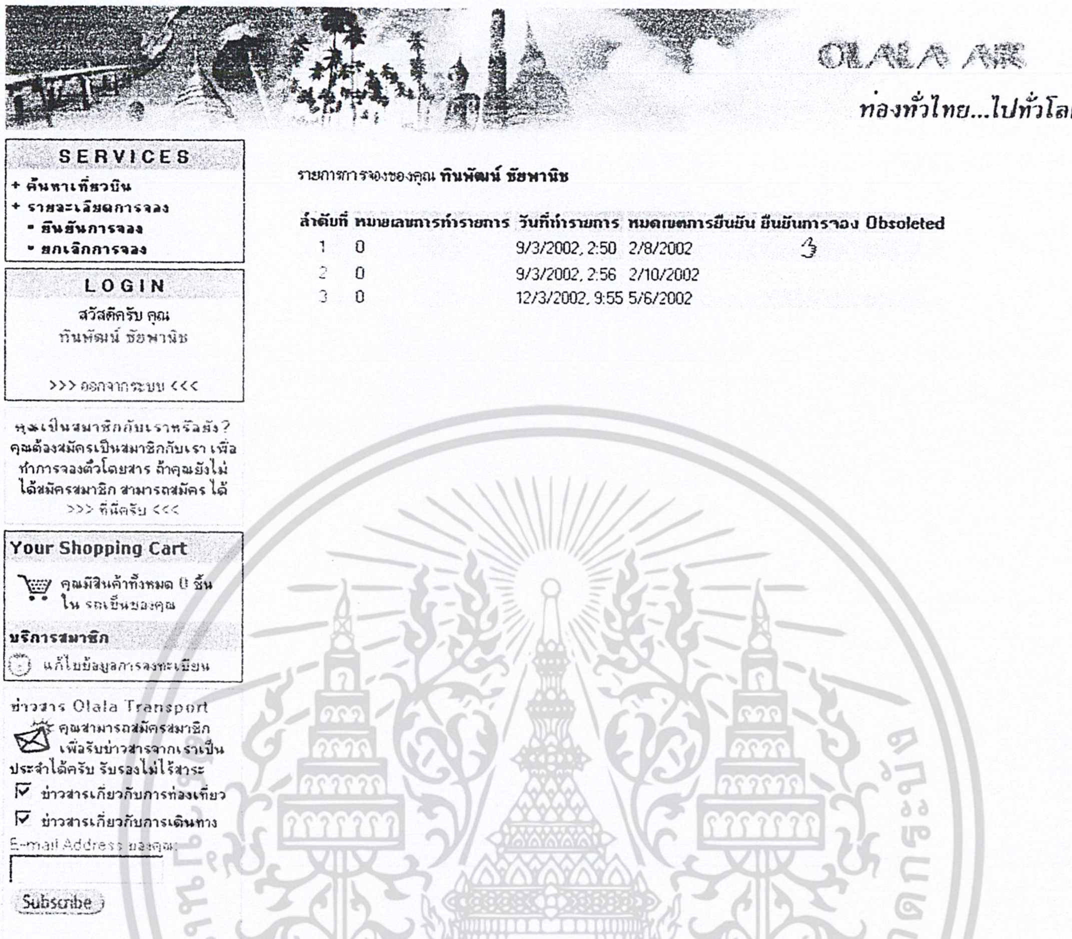
- ข้อมูลผู้โดยสาร (travellerInfo.jsp)




*รูปที่ 13-4 ข้อมูลผู้โดยสาร*

ก่อนที่จะสามารถทำการจองได้ ผู้ใช้จะต้องกรอกข้อมูลผู้โดยสารสำหรับแต่ละรายการเที่ยวบินใน Shopping Cart ให้ครบถ้วน โดยการคลิกที่ "ข้อมูลผู้โดยสาร" สำหรับแต่ละรายการ

- รายการการจอง (listBooking.jsp)

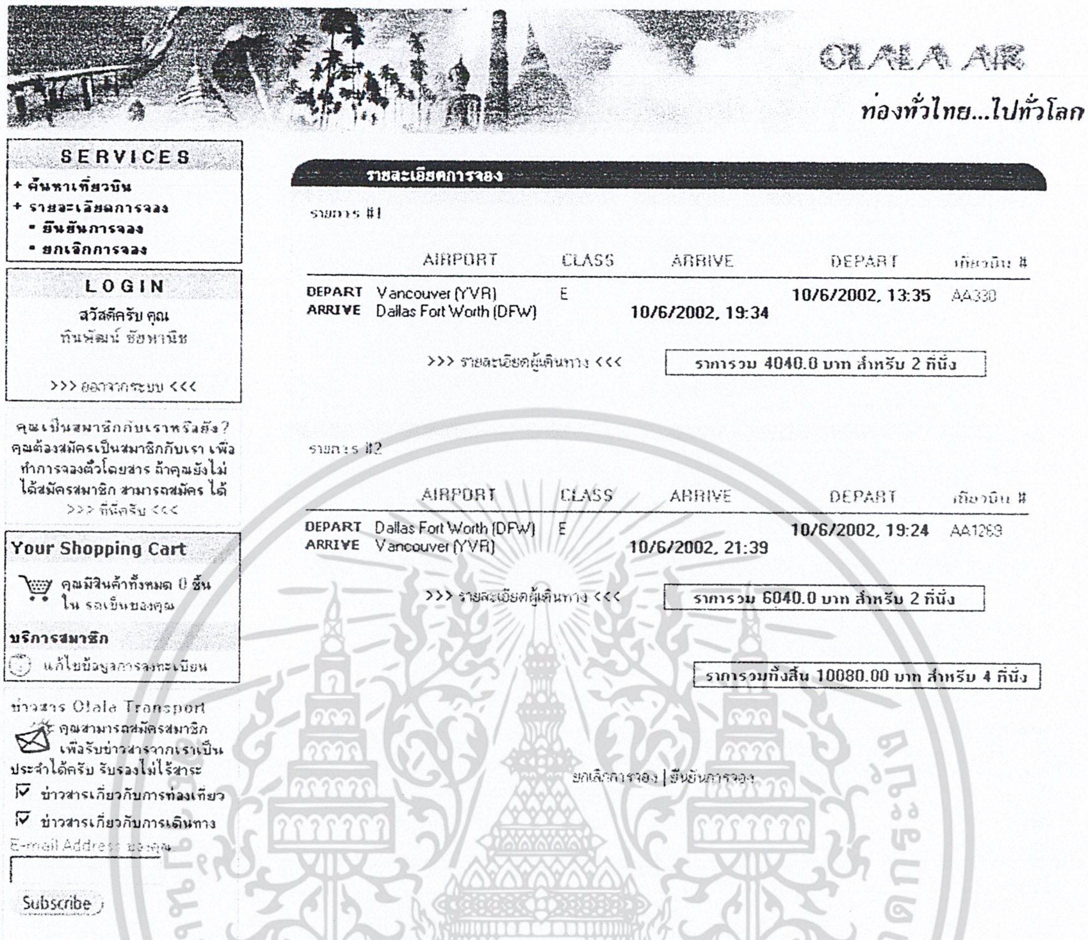



*รูปที่ 13-5 รายการการจอง*

แสดงรายการการจองทั้งหมดที่ผู้ใช้เคยทำการจองเอาไว้ โดยจะแสดงหมายเลขการทำรายการ วันที่ทำการจอง วันที่หมดเขตการยืนยัน และระบุว่า สำหรับแต่ละรายการนั้น ได้ทำการยืนยันการจองแล้วหรือยัง และเป็นรายการที่มีการเดินทางไปแล้ว (Obsoleted) หรือไม่

ผู้ใช้สามารถดูรายละเอียดทั้งหมดของการจองแต่ละครั้งได้ โดยคลิกที่หมายเลขลำดับหน้ารายการการจองที่ต้องการ ผู้ใช้ที่จะสามารถเรียกดูรายการการจอง หรือรายละเอียดของการจองแต่ละครั้งได้ จะต้อง Login เข้าสู่ระบบให้เรียบร้อยก่อน

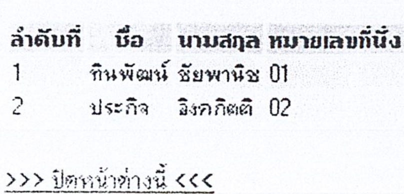
- รายละเอียดการจอง (vireBooking.jsp)




***รูปที่ 13-6 รายละเอียดการจอง***

แสดงรายละเอียดการจองสำหรับรายการใด ๆ ที่ลูกค้าเคยทำการจองเอาไว้ ลูกค้าสามารถยกเลิกการจองรายการนั้น ๆ หรือยืนยันการจอง พร้อมทั้งชำระเงินค่าโดยสารสำหรับรายการนั้น ๆ ได้จากหน้าจอนี้

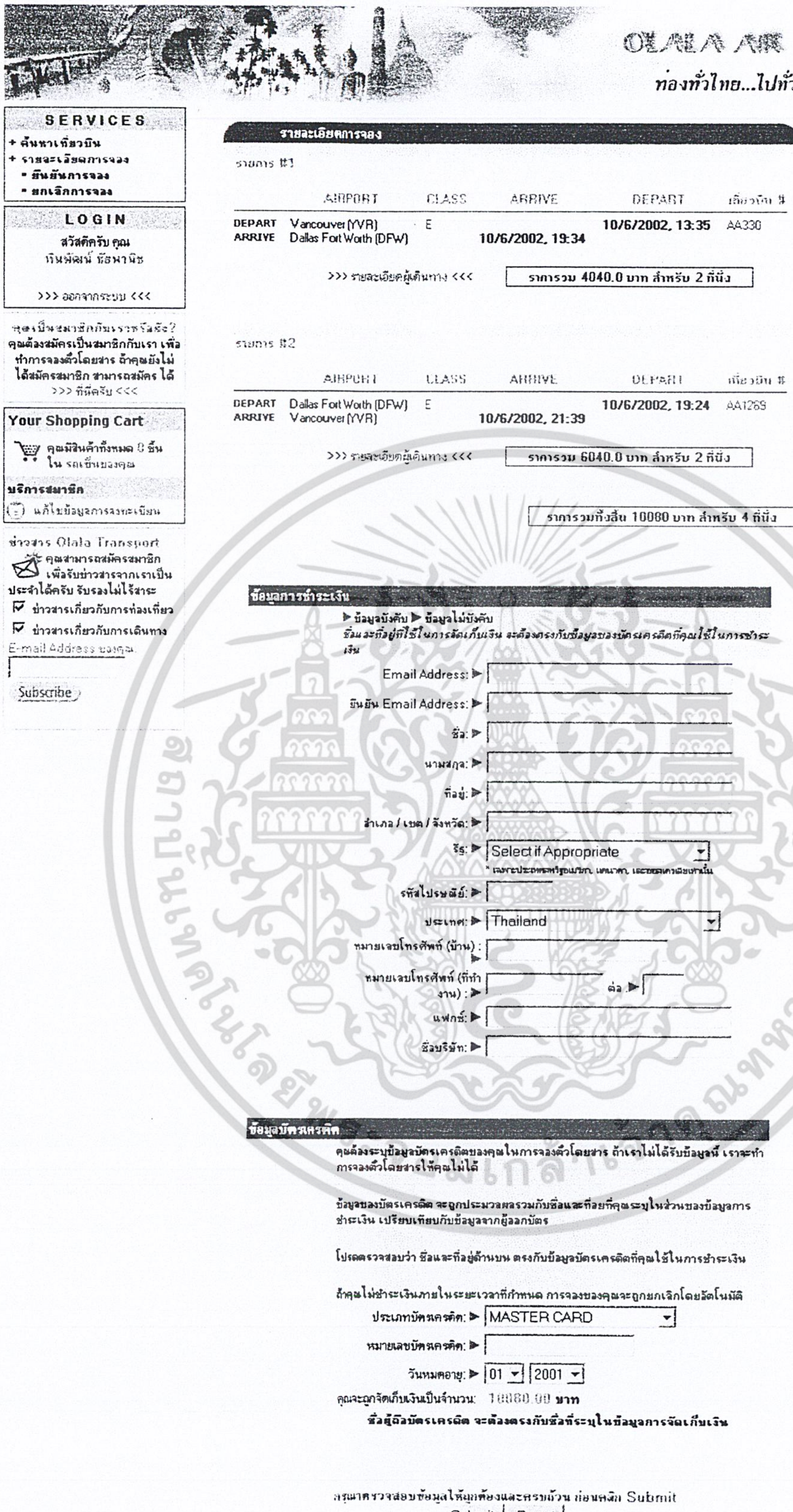
ผู้ใช้สามารถตรวจสอบรายละเอียดผู้เดินทาง สำหรับแต่ละรายการในการจองครั้งนั้น ๆ ได้ โดยการคลิกที่ "รายละเอียดผู้เดินทาง" จะปรากฏข้อมูลผู้เดินทางในรายการนั้น ๆ ขึ้นมาดังนี้



| ลำดับที่ | ชื่อ | นามสกุล | หมายเลขที่นั่ง |
|---|---|---|---|
| 1 | ทินพัฒน์ | ชัยพานิช | 01 |
| 2 | ประกิจ | ลิงคกิตติ | 02 |

>>> ปิดหน้าต่างนี้ <<<

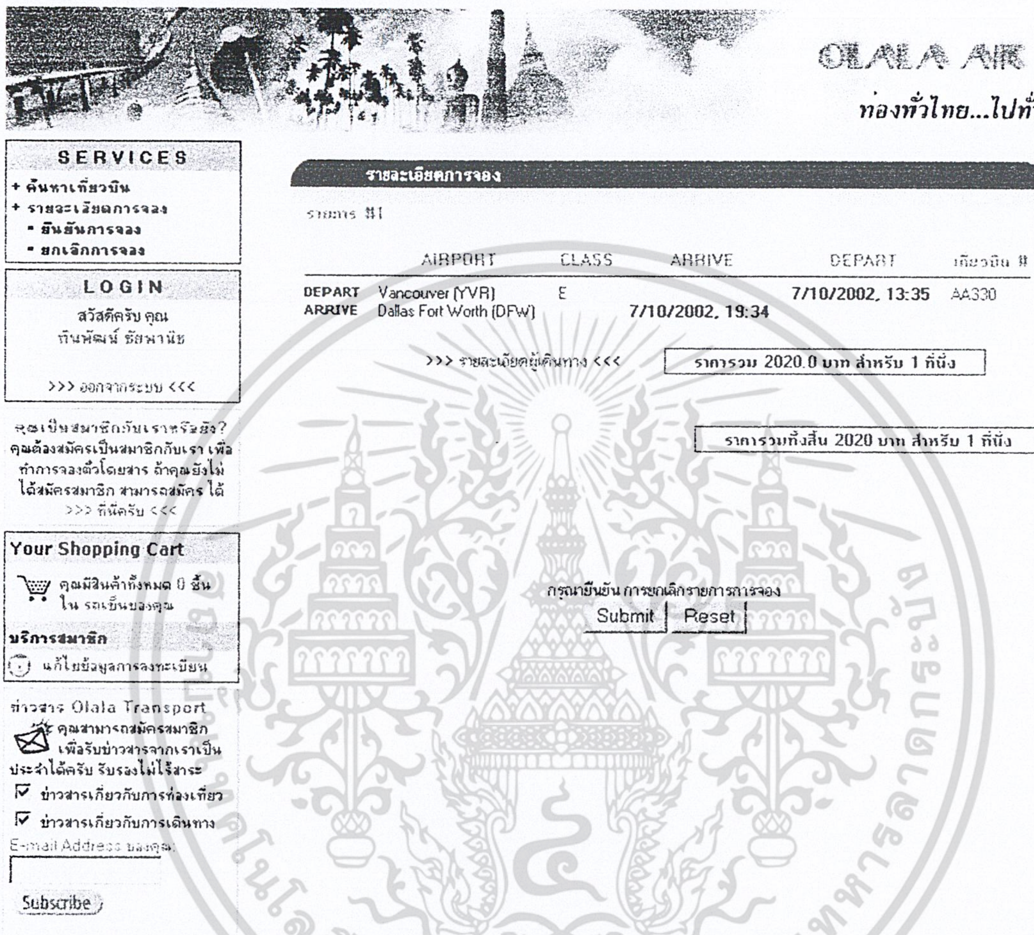
- ยืนยันการจอง (confirmBooking.jsp)




*รูปที่ 13-7 การยืนยันการจอง*

เมื่อผู้ใช้ เลือก "ยืนยันการจอง" จากหน้าจอ รายละเอียดการจองเพื่อทำการยืนยันการจองครั้งใด ๆ ผู้ใช้จะต้องระบุข้อมูลส่วนตัว รวมถึงข้อมูลเกี่ยวกับบัตรเครดิต ซึ่งใช้ในการชำระค่าโดยสาร

- ยกเลิกการจอง (cancelBooking.jsp)




*รูปที่ 13-8 การยกเลิกการจอง*

เมื่อผู้ใช้เลือก "ยกเลิกการจอง" จากหน้าจอรายละเอียดการจอง ผู้ใช้จะต้องทำการยืนยันอีกครั้งว่าต้องการยกเลิกรายการการจองนั้นจริง ๆ เมื่อผู้ใช้ทำการยกเลิกการจองไปแล้ว ข้อมูลรายการการจองนั้น ๆ จะถูกลบออกจากฐานข้อมูล และผู้ใช้คนอื่น สามารถเข้ามาจองที่นั่งเหล่านั้นแทนได้

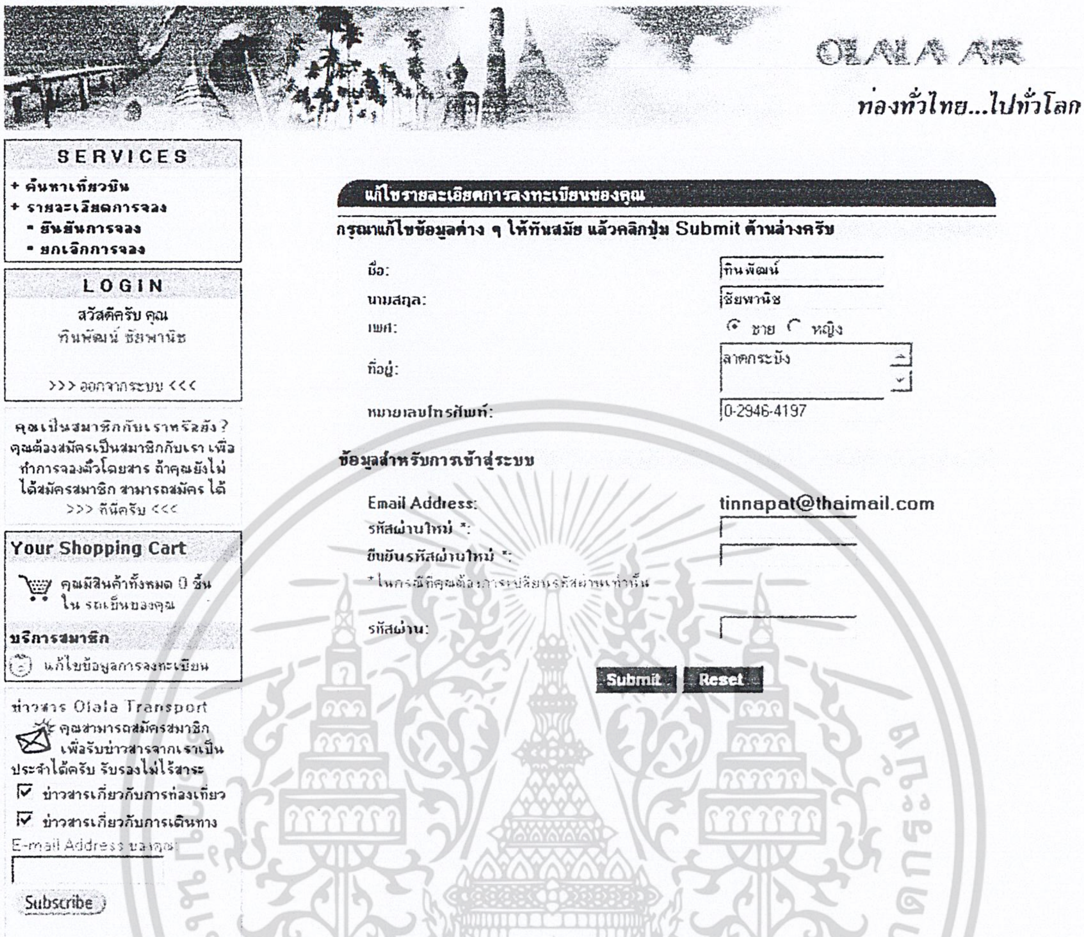
- ลงทะเบียนสมาชิกใหม่ (userRegister.jsp)


*รูปที่ 13-9 การลงทะเบียนสมาชิกใหม่*

ผู้ใช้ที่ต้องการทำการจองที่นั่งเครื่องบินโดยสารผ่านทางเว็บไซต์ OLALA AIR จะต้องลงทะเบียนสมัครเป็นสมาชิกก่อน โดยการคลิกที่ “สมัครสมาชิกใหม่” และกรอกข้อมูลต่าง ๆ ให้ถูกต้องครบถ้วน

ผู้ใช้ที่ไม่ได้เป็นสมาชิกของระบบ สามารถทำการค้นหาเที่ยวบินได้ตามปกติ เพียงแต่จะไม่สามารถทำการจองเที่ยวบินได้

- แก้ไขข้อมูลการลงทะเบียน (userRegisterEdit.jsp)

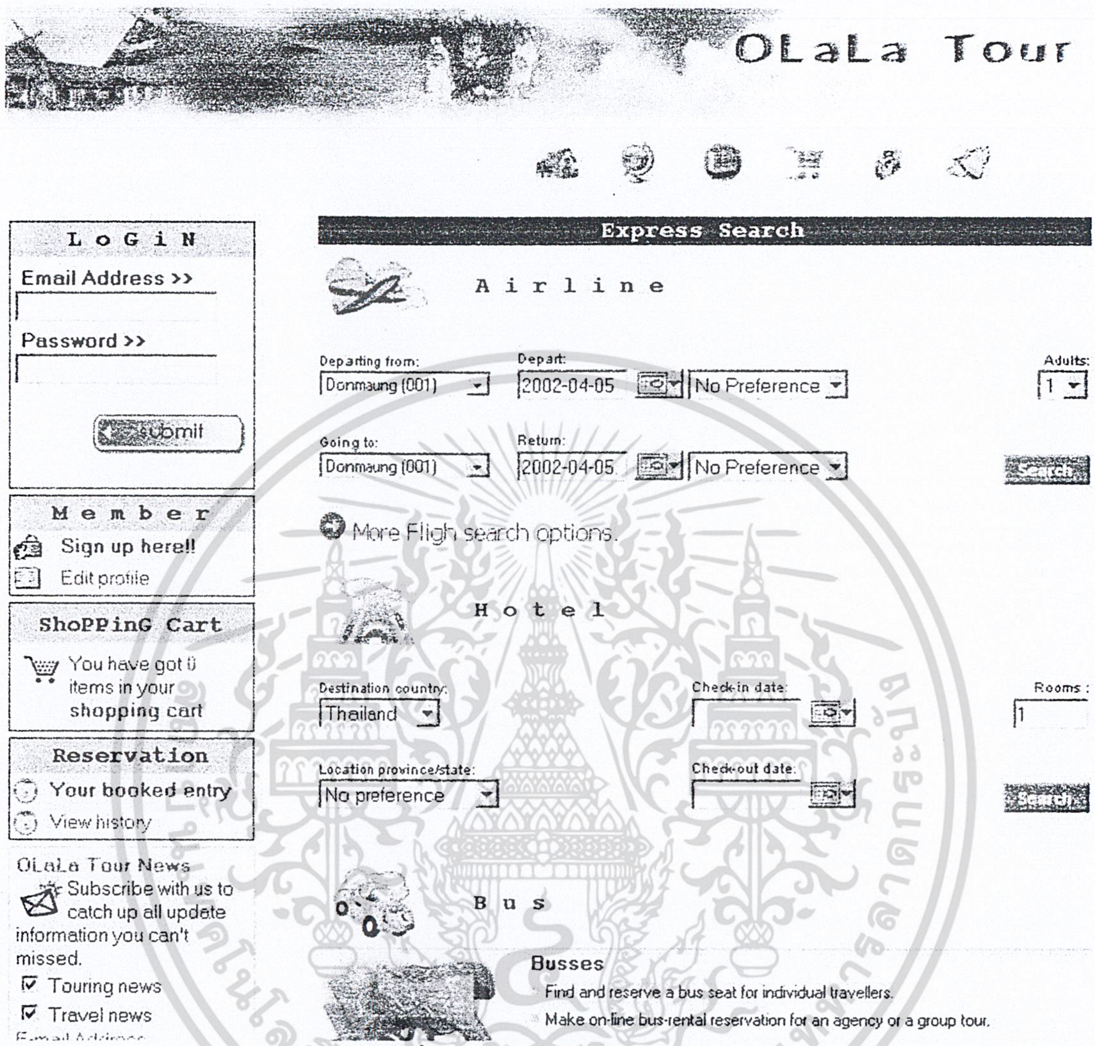



*รูปที่ 13-10 การแก้ไขข้อมูลการลงทะเบียน*

ผู้ใช้สามารถแก้ไขข้อมูลการลงทะเบียนของตนเองได้ โดยคลิกที่ “แก้ไขข้อมูลการลงทะเบียน” ซึ่งจะปรากฏขึ้นหลังจากที่ผู้ใช้ทำการ Login เข้าสู่ระบบแล้ว แต่ลูกค้าไม่สามารถแก้ไข Email Address ที่ใช้ในการลงทะเบียนได้ เนื่องจากเป็นข้อมูลที่ใช้ในการเข้าสู่ระบบ

## 13.2 เว็บไซต์ OLALA TOUR

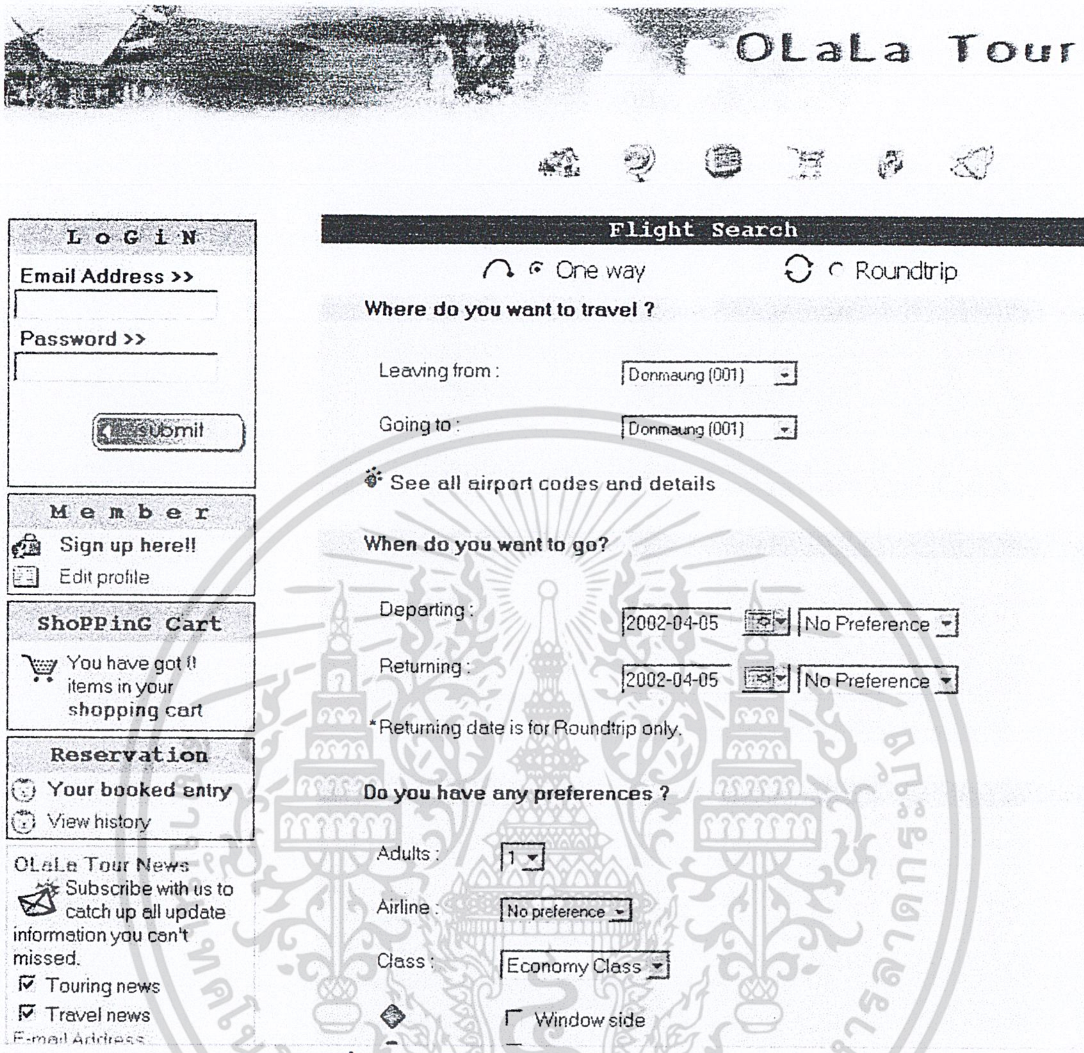
- หน้าแรก (index.jsp)




*รูปที่ 13-11 หน้าแรกของเว็บทัวร์*

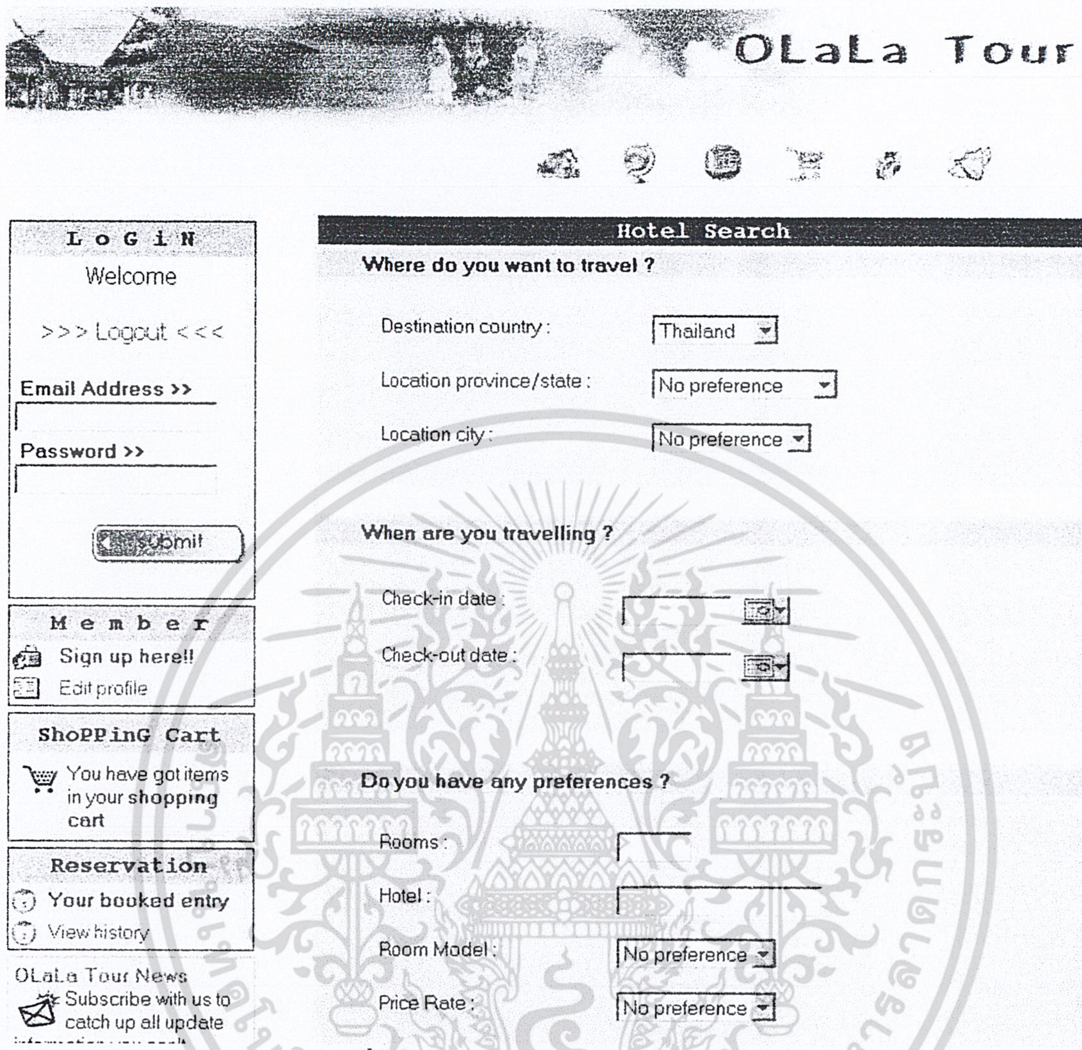
เว็บทัวร์จะประกอบด้วยระบบสายการบินและโรงแรม operation ต่าง ๆ จะเหมือนกับเว็บไซต์ OLALA AIR แต่มีส่วนของการค้นหาและจองโรงแรมเพิ่มขึ้นมา โดยระบบจองโรงแรมเป็นเซอร์วิสที่เขียนขึ้นด้วย COM+ ถ้าเซิร์ฟเวอร์ของ COM+ ไม่สามารถทำงานได้ การทำ operation เกี่ยวกับโรงแรมจะขัดข้อง

- Air Advance Search (airSearchAdv.jsp)




*รูปที่ 13-12 การค้นหาสายการบินแบบ Advance*

- Hotel Advance Search (hotelSearchAdv.jsp)




*รูปที่ 13-13 การค้นหาโรงแรมแบบ Advance*

- ข้อมูลในรถเข็น (cart.jsp)


***รูปที่ 13-14 ข้อมูลในรถเข็นของเว็บทัวร์***

# บทที่ 14
# การออกแบบสถาปัตยกรรม

## 14.1 ปัจจัยในการออกแบบสถาปัตยกรรม

การออกแบบสถาปัตยกรรมจะต้องพิจารณาถึงปัจจัยต่าง ๆ ดังนี้

- แอพพลิเคชันเซิร์ฟเวอร์ที่ใช้

เนื่องจากแต่ละแอพพลิเคชันเซิร์ฟเวอร์มีวิธีการคลัสเตอร์ต่างกันเช่น WebLogic กระจายโหลดไปยังพรีเซนเทชันเทียร์ด้วยวิธี centralize กระจายโหลดไปยังออบเจ็กต์เทียร์ด้วยวิธี distribute แต่ในผลิตภัณฑ์เช่น HP Bluestone จะเป็น centralize ทั้งหมดเป็นต้น

- แอพพลิเคชันที่จะดีพลอย

การออกแบบสถาปัตยกรรมจะต้องตอบสนองต่อความต้องการของแอพพลิเคชัน เช่น ถ้าแอพพลิเคชันมีการใช้งานพรีเซนเทชันเทียร์มากกว่าออบเจ็กต์เทียร์ ก็ควรที่จะจัดจำนวนของเซิร์ฟเวอร์ที่คลัสเตอร์ในพรีเซนเทชันเทียร์มากกว่าออบเจ็กต์เทียร์และให้เพียงพอต่อการใช้งาน

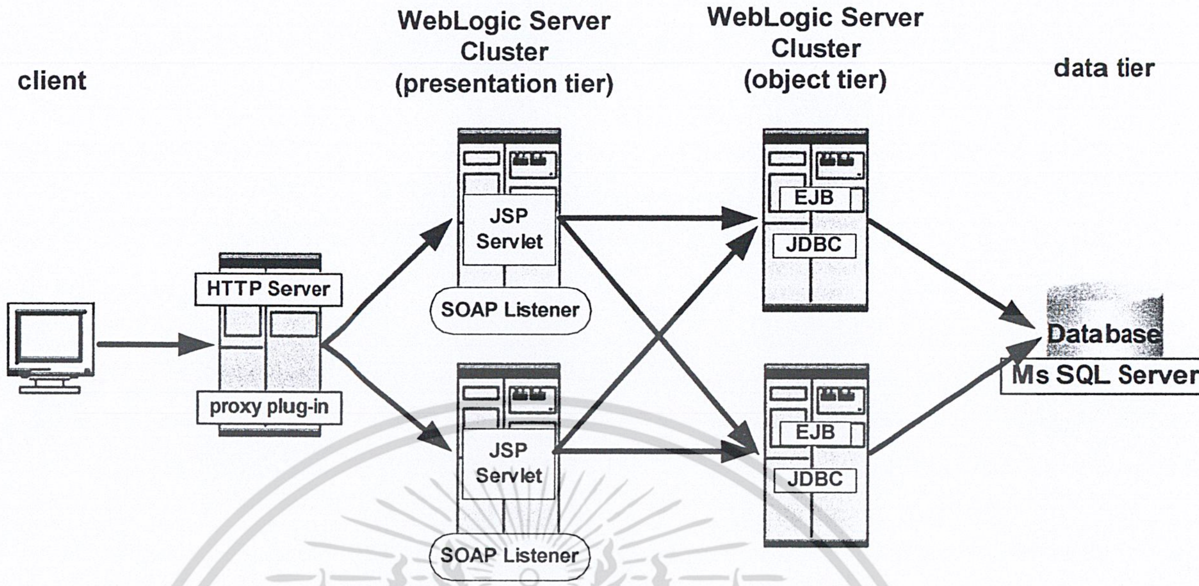
- ความต้องการฮาร์ดแวร์

ในการออกแบบจะต้องพิจาณาความเหมาะสม ในกรณีที่มีความต้องการทางฮาร์ดแวร์ เช่น load balancer ถึงแม้ว่าการใช้ฮาร์ดแวร์จะให้ประสิทธิภาพที่ดีกว่า แต่ก็มีราคาแพง และในบางกรณีสามารถหลีกเลี่ยงได้ด้วยการใช้ซอฟต์แวร์ประยุกต์กับฮาร์ดแวร์ที่มีอยู่แล้ว ถึงแม้ว่าจะมีประสิทธิภาพต่ำกว่า แต่มีราคาถูกกว่า

## 14.2 สถาปัตยกรรมที่ใช้

สถาปัตยกรรมที่ใช้จะเป็นสถาปัตยกรรมแบบ 5 เทียร์ไคลเอ็นต์/เซิร์ฟเวอร์ ดังรูปที่ 14-1 โดยมิดเดิลเทียร์จะแบ่งเป็นพรีเซนเทชันเทียร์และออบเจ็กต์เทียร์ เนื่องจากว่าแอพพลิเคชันซึ่งได้แก่ ระบบเครื่องบินและระบบรถบัสนั้นไม่มีการใช้สแตติกเพจอย่าง HTML แต่เนื่องจากการกระจายโหลดไปยังพรีเซนเทชันเทียร์นั้นต้องการฮาร์ดแวร์ที่มีความสามารถในการกระจายโหลดโดยเฉพาะ ซึ่งได้แก่ load balancer หรือ proxy plug-in ซึ่งเป็นคอมพิวเตอร์ที่ติดตั้งซอฟต์แวร์ทำหน้าที่เป็น proxy กระจายโหลดไปยังพรีเซนเทชันเทียร์

เนื่องจากมีข้อจำกัดทางด้านฮาร์ดแวร์ จึงเลือก proxy plug-in โดยในส่วน proxy plug-in นี้จะทำหน้าที่กระจายโหลดเท่านั้น แต่ก็จะเกิด single point of failure ที่ proxy plug-in หากเครื่องที่เป็น proxy plug-in ขัดข้อง ไคลเอ็นต์จะไม่สามารถส่ง request ไปยังพรีเซนเทชันเทียร์ได้เลย โดย proxy plug-in ที่เลือกใช้คือ WebLogic proxy plug-in เนื่องจากสามารถคอนฟิกได้ง่าย




*รูปที่ 14-1 สถาปัตยกรรมที่ใช้ในระบบ*

ดังนั้นสถาปัตยกรรมทั้งหมดจะประกอบด้วย

- HTTP เซิร์ฟเวอร์ใช้ WebLogic proxy plug-in จำนวน 1 เครื่อง
- พรีเซนเทชันเทียร์ใช้ BEA WebLogic Server 6.0 จำนวน 2 เครื่อง โดยจะต้องติดตั้ง Apache SOAP 2.2 ในส่วนนี้ด้วย เพื่อทำหน้าที่เป็น SOAP Listener (เซิร์ฟเวอร์) วิธีติดตั้งดูได้ในภาคผนวก ก
- ออบเจ็กต์เทียร์ใช้ BEA WebLogic Server 6.0 จำนวน 2 เครื่อง โดยจะต้องติดตั้งให้เป็น SOAP ไคลเอ็นต์ด้วย
- ฐานข้อมูลใช้ MS SQL Server 1 เครื่อง

# ภาคผนวก ก
# การติดตั้งโปรแกรม

ภาคผนวกนี้จะแสดงวิธีติดตั้งโปรแกรมต่าง ๆ ที่ใช้ในโครงงาน ประกอบด้วย

- WebLogic Server 6.0
- Apache SOAP 2.2

**1. การติดตั้ง WebLogic Server 6.0**

สามารถดาวน์โหลด WebLogic Server 6.0 (sp2) evaluate version ได้ที่ http://www.bea.com WebLogic Server รองรับหลายแพล็ตฟอร์มเช่น Window , Linux , Solaris และอื่น ๆ แต่จะอธิบายวิธีการติดตั้งเฉพาะแพล็ตฟอร์ม Window เท่านั้น (แนะนำให้ใช้ Window 2000 Advance Server)

**1.1 ความต้องการของ WebLogic Server**

- ฮาร์ดดิสก์ ประมาณ 95 MB
- แรม 128 MB (แนะนำให้ใช้ 256 MB ขึ้นไป)

**1.2 ขั้นตอนการติดตั้ง**

| หน้าต่าง | ขั้นตอน |
|---|---|
| BEA Logo | เลือกภาษาที่ต้องการให้แสดงผล |
| Welcome | คลิก Next เพื่อเริ่มขั้นตอนการ install |
| License Agreement | อ่านข้อตกลงการใช้ซอฟต์แวร์ ถ้ายอมรับให้เลือก Yes แล้ว Next ถ้าไม่ยอมรับเลือก No |
| Choose Install Set | เลือกชุดที่ต้องการจะลง ถ้าไม่ต้องการไฟล์โค้ดตัวอย่างให้เลือก Server Only ถ้าต้องการให้เลือก Server and Examples |
| Choose BEA Home Directory | ระบุไดเรกทอรี BEA Home ที่จะเป็นปลายทางของผลิตภัณฑ์ของ BEA ถ้ามีไดเรกทอรี BEA Home อยู่แล้วก็สามารถที่จะใช้ได้ หรือจะสร้างขึ้นใหม่ก็ได้ |
| Choose Product Directory | ระบุไดเรกทอรีที่ต้องการลง WebLogic Server ดีฟอลต์คือ wlserver6.0 |

| หน้าต่าง | ขั้นตอน |
|---|---|
| Default Server Configuration | ให้ใส่ชื่อโดเมน ดีฟอลต์คือ mydomain WebLogic โดเมนสามารถมีได้หลาย WebLogic Server<br>ให้ใส่ชื่อเซิร์ฟเวอร์ ดีฟอลต์คือ myserver<br>ใส่หมายเลขพอร์ต TCP/IP สำหรับเซิร์ฟเวอร์ |
| Install WebLogic Server Service (Windows systems only) | คลิก Yes ถ้าต้องการลง WebLogic Server เป็นเซอร์วิส ซึ่งเซิร์ฟเวอร์จะเปิดทุกครั้งที่เปิดเครื่อง ถ้าไม่ต้องการให้คลิก No |
| Create System Password | ให้ใส่พาสเวิร์ดที่ต้องการ พาสเวิร์ดต้องไม่น้อยกว่า 8 ตัว แต่ไม่เกิน 20 ตัว |
| Installing . . . | WebLogic Server จะเริ่ม install ให้รอจนกว่าจะผ่านหน้าต่างนี้ |
| Install Complete | คลิก Done เป็นอันจบการ install |

**1.3 แนะนำการใช้งาน WebLogic เซิร์ฟเวอร์เบื้องต้น**

**1.3.1 โครงสร้างไดเรกทอรีของ BEA Home**

จะประกอบด้วยไฟล์และไดเรกทอรีดังนี้

- ไฟล์ register.xml
- ไฟล์ license.bea
- ไฟล์ UpdateLicense (.cmd/.sh)
- ไดเรกทอรี logs
- ไดเรกทอรี utils
- ไดเรกทอรี jdk130

**1.3.2 โครงสร้างไดเรกทอรีของ WebLogic Server**

(ดีฟอลต์ไดเทกทอรีคือ wlserver6.0)

```
wlserver6.0
    bin
    config
    ext
    lib
    samples
    uninstaller
```

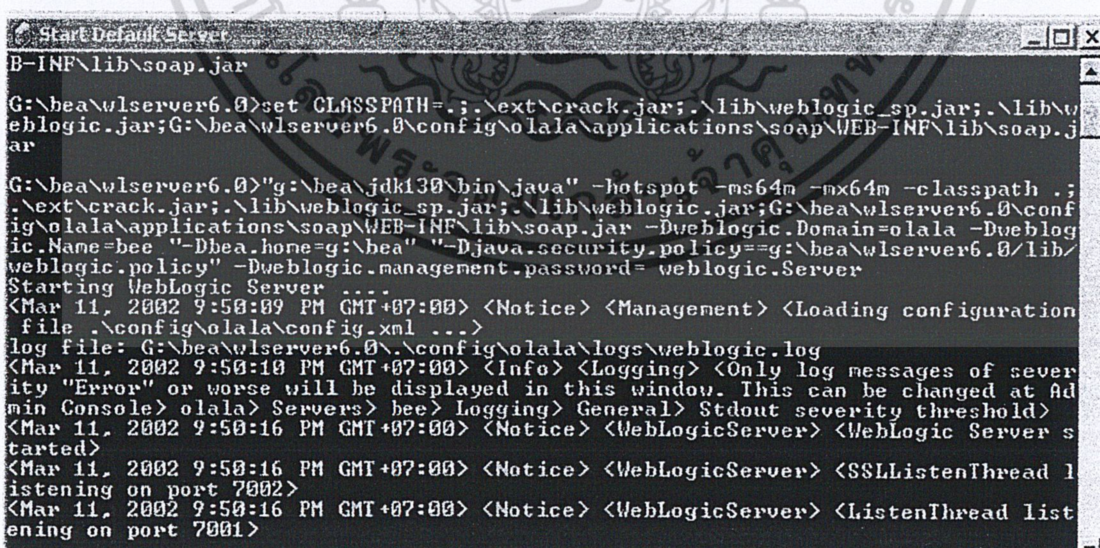
| ไดเรกทอรี | ประกอบด้วย |
|---|---|
| Bin | เอ็กซิคิวต์โปรแกรม |
| config | เป็นที่เก็บของแต่ละโดเมน แต่ละโดเมนจะเป็นซับไดเรกทอรีในไดเรกทอรีนี้ โดยชื่อของไดเรกทอรีคือชื่อของโดเมน ภายหลังจากที่ลงแล้วจะประกอบด้วยซับไดเรกทอรีดังนี้<br>*domain_name* (mydomain is the default)<br>examples<br>petstore<br>แต่ละซับไดเรกทอรีจะประกอบด้วยไฟล์ XML ที่ใช้คอนฟิกเซิร์ฟเวอร์ในแต่ละโดเมน (config.xml) |
| Ext | เป็นไดเรกทอรีที่เป็นที่อยู่ของ jar ไฟล์ที่เกี่ยวกับ XML |
| lib | ไฟล์ .jar ที่ใช้รัน WebLogic เซิร์ฟเวอร์และไลบรารีที่รองรับคอมโพเน็นต์เหล่านี้<br>- JDBC drivers<br>- Netscape Server (NS) API plug ins<br>- Apache Plug-ins<br>- Internet Server (IS) API plug-ins<br>- Other native code packages |
| samples | ตัวอย่างโค้ดและรีซอร์สที่ช่วยในการเรียนรู้การพัฒนาแอพพลิเคชันกับ WebLogic เซิร์ฟเวอร์จะประกอบด้วยซับไดเรกทอรีดังนี้<br>/examples<br>เป็นแอพพลิเคชันง่าย ๆ ที่แสดงถึงฟีเจอร์ของ WebLogic เซิร์ฟเวอร์<br>/petStore<br>แอพพลิเคชัน WebLogic Server Pet Store ซึ่งเป็นตัวอย่างอี-คอมเมิร์ชแอพพลิเคชัน<br>/eval<br>Cloudscape RDBMS ที่ใช้ในตัวอย่างและ Pet Store |
| uninstaller | โค้ดที่ใช้ในการ uninstall |
| uninstaller_servicepack | โค้ดที่ใช้ในการ uninstall เซอร์วิสแพ็ค |
| servicepacks | ก๊อบปี้ของแต่ละเซอร์วิสแพ็คที่ลง |

### 1.3.3 การสตาร์ตเซิร์ฟเวอร์

มีวิธีสตาร์ต 3 วิธีคือ

- ชอร์ตคัต ไอคอน
- สคริปต์
- คอมมานด์ไลน์

| สตาร์ตดีฟอลต์เซิร์ฟเวอร์ | ขั้นตอน |
|---|---|
| shortcut icon | เลือก Start→Programs→BEA WebLogic E-Business Platform→WebLogic Server 6.0→Start Default Server.<br>เซิร์ฟเวอร์จะเริ่มการทำงานและจะขึ้นให้ใส่พาสเวิร์ด<br>การสตาร์ตด้วยวิธีนี้เหมือนกับการรันสคริปต์ startWebLogic.cmd |
| Script startWebLogic.cmd | รันสคริปต์ที่อยู่ในไดเรกทอรี *wl_home*\config*domain_name*<br>โดย *wl_home* หมายถึงไดเรกทอรีที่ install WebLogic Server<br>*domain_name* หมายถึงโดเมนที่ได้ตั้งชื่อไว้<br>เซิร์ฟเวอร์จะเริ่มการทำงานและจะขึ้นให้ใส่พาสเวิร์ด |
| command line | ใช้คำสั่งในสคริปต์มาทำผ่านคอมมานด์ไลน์ |




*รูปที่ ก-1 ตัวอย่างการสตาร์ตดีฟอลต์เซิร์ฟเวอร์*

รูปที่ ก-1 แสดงตัวอย่างการสตาร์ตเซิร์ฟเวอร์ หากไม่ต้องการใส่พาสเวิร์ดในตอนสตาร์ต ให้สร้างไฟล์ password.ini ภายในไฟล์ให้ระบุพาสเวิร์ดและเก็บไว้ใน path เดียวกับสคริปต์ startWebLogic.cmd

การสตาร์ตเซิร์ฟเวอร์อื่นจะมีวิธีคล้าย ๆ กัน โดย Examples เซิร์ฟเวอร์เป็นเซิร์ฟเวอร์ที่คอนฟิกไว้อัตโนมัติสำหรับการทดลองแอพพลิเคชันที่มากับโค้ดตัวอย่าง และ Pet Store เซิร์ฟเวอร์เป็นเซิร์ฟเวอร์ที่คอนฟิกไว้อัตโนมัติสำหรับอี-คอมเมิร์ซแอพพลิเคชัน ซึ่งทั้ง 2 เซิร์ฟเวอร์นี้ได้มีการคอนฟิกให้ไม่ต้องใส่พาสเวิร์ดไว้เรียบร้อยแล้ว

**1.3.4 การสตาร์ตคอนโซล**

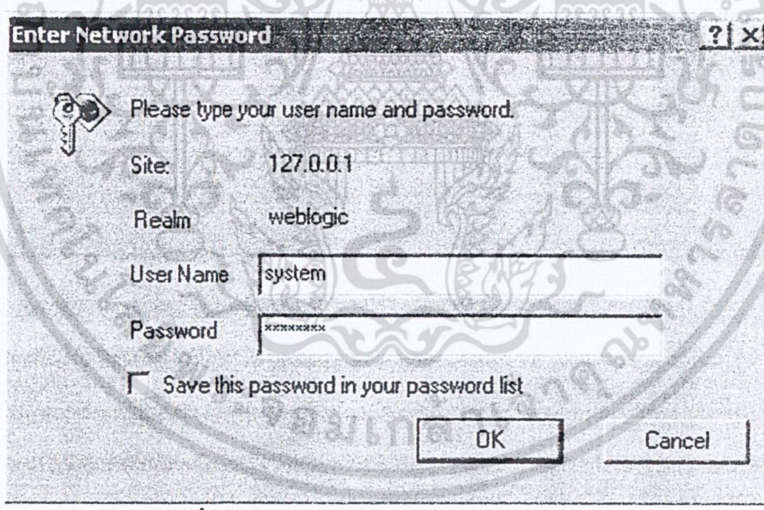
คอนโซลใช้ในการคอนฟิกเซิร์ฟเวอร์ การสตาร์ตคอนโซลจะต้องสตาร์ตเซิร์ฟเวอร์ก่อน หลังจากนั้นให้เปิดบราวเซอร์

http://*localhost*:*port*/console

โดยที่ *localhost* คือชื่อเครื่องหรือ IP

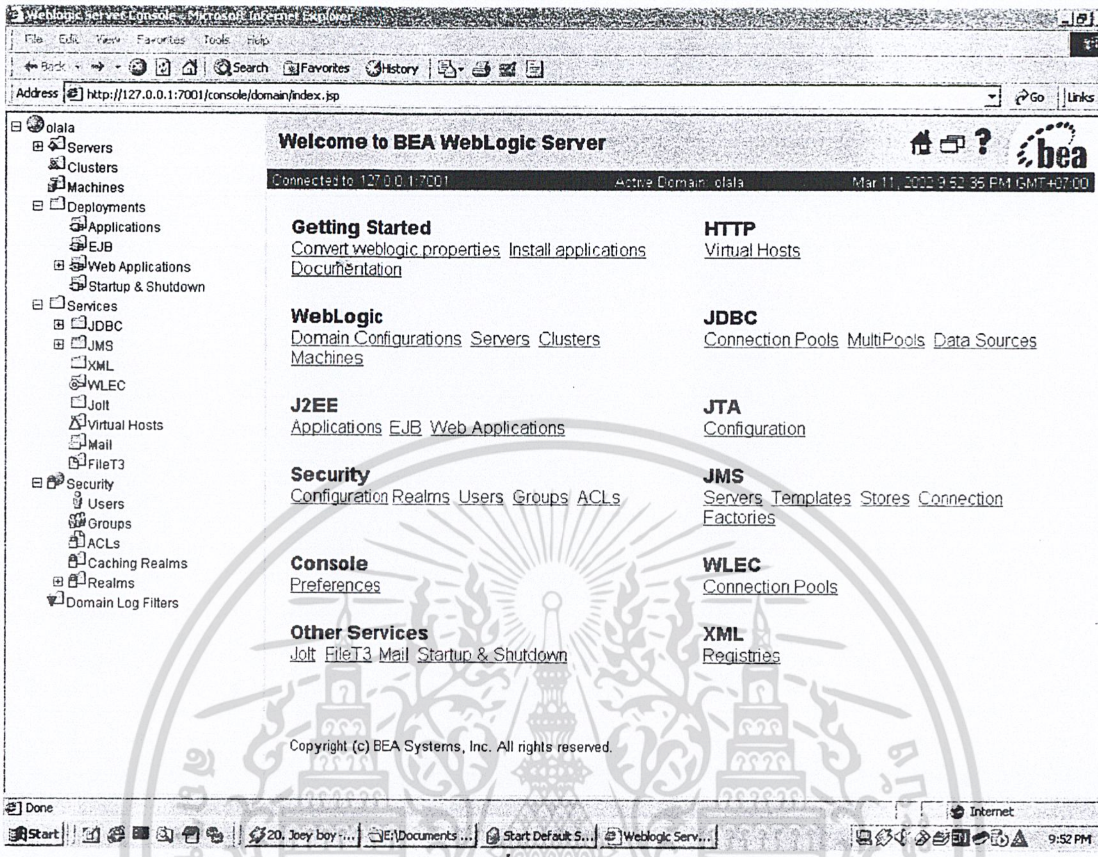
*port* คือหมายเลขพอร์ตของเซิร์ฟเวอร์ ดีฟอลต์คือ 7001

หรือใช้ชอร์ตคัต Start→Programs→BEA WebLogic E-Business Platform→WebLogic Server 6.0→Start Default Console




*รูปที่ ก-2 การล็อกอินในการเข้าใช้งานคอนโซล*

หลังจากนั้นจะให้ใส่พาสเวิร์ด โดยยูสเซอร์คือ system พาสเวิร์ดคือพาสเวิร์ดที่ใส่ในตอนลงโปรแกรม ดังรูปที่ ก-2 สำหรับ Examples เซิร์ฟเวอร์และ Pet Store เซิร์ฟเวอร์ไม่ต้องใส่พาสเวิร์ด หลังจากนั้นจะเห็นคอนโซลดังรูปที ก-3




*รูปที่ ก-3 คอนโซล*

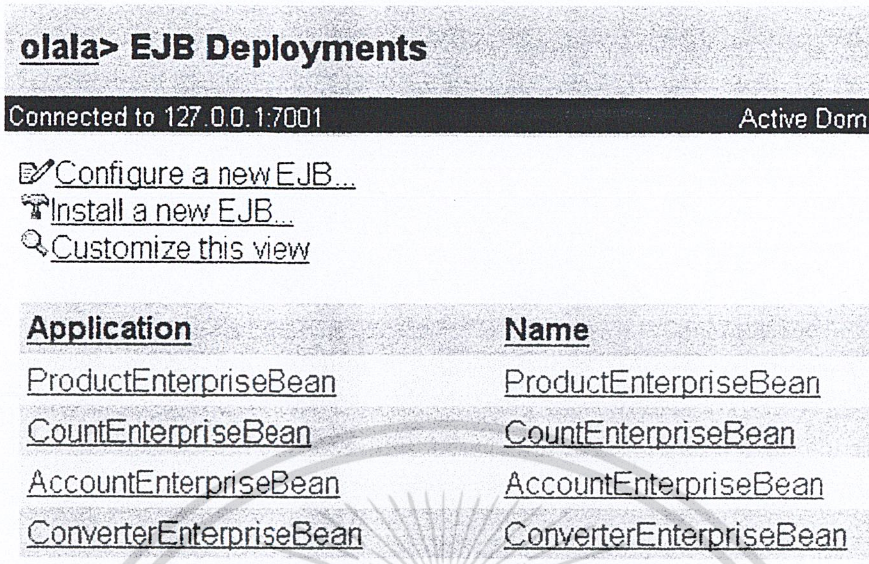
**1.3.5 การดีพลอย**

- **การดีพลอยแอพพลิเคชัน**

แอพพลิเคชันอาจจะแพ็คเก็ตเป็น .ear , .jar หรือ .war การดีพลอยแอพพลิเคชันสามารถที่จะไดนามิกดีพลอยได้ และสามารถที่จะรีดีพลอยได้ด้วย WebLogic เซิร์ฟเวอร์จะตรวจสอบ timestamp ของแอพพลิเคชันที่ ดีพลอย และรีดีพลอยให้ โดยจะต้องดีพลอยลงในไดเรกทอรี /config/*domain_name*/applications *domain_name* คือชื่อของโดเมน หรือจะดีพลอยโดยใช้คอนโซลก็ได้ การดีพลอยโดยใช้คอนโซลดังรูปที่ ก-4

ในการดีพลอย EJB ไปยัง WebLogic Server นั้น มีขั้นตอนคร่าว ๆ ดังนี้

1. สร้าง deployment descriptor ของ EJB ที่ต้องการนำไปดีพลอย
2. สร้าง EJB container class
3. โหลด EJB class เข้าไปยัง WebLogic เซิร์ฟเวอร์
4. ดีพลอย EJB ลงไปยัง WebLogic Server ในตอนเริ่มต้น หรือแบบไดนามิก

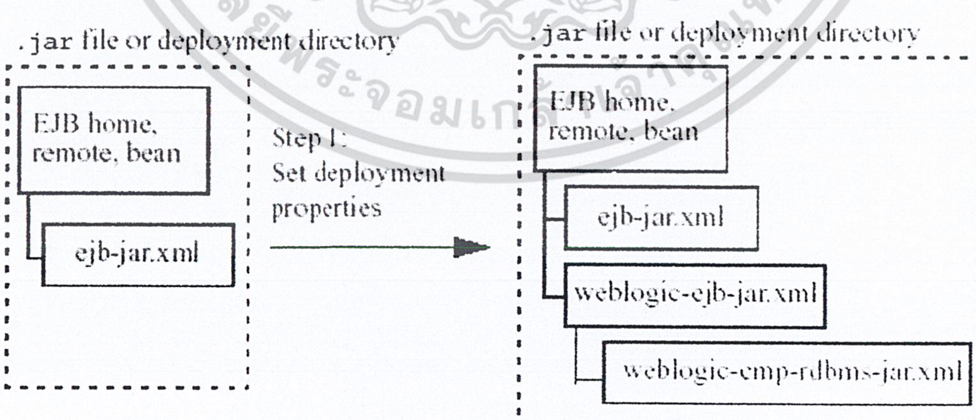



*รูปที่ ก-4 การดีพลอยโดยใช้คอนโซล*

- **การสร้าง Deployment Descriptor**

ขั้นตอนในการดีพลอยคอมโพเน็นต์นั้น เริ่มต้นจากไฟล์ .jar หรือ deployment directory ที่ประกอบไปด้วยส่วนอินเทอร์เฟซและอิมพลีเมนต์ของคอมโพเน็นต์ EJB ที่ถูกคอมไพล์แล้ว และจะต้องมีการจัดวางไดเรกทอรีที่ถูกต้องตามโครงสร้างของแพ็กเก็จในภาษาจาวา

นอกจากนี้แล้ว ผู้พัฒนาคอมโพเน็นต์ ควรจะต้องแนบไฟล์ ejb-jar.xml ซึ่งเป็นไฟล์ที่ระบุถึงลักษณะและคุณสมบัติต่าง ๆ ของคอมโพเน็นต์นี้มาด้วย โดยไฟล์ ejb-jar.xml และไฟล์ XML อื่น ๆ ซึ่งจำเป็นจะต้องอยู่ในซับไดเรกทอรีบนสุดที่มีชื่อว่า META-INF ในไฟล์ .jar หรือ deployment directory




*รูปที่ ก-5 ไฟล์ Deployment Descriptor ที่จำเป็น*

ในการดีพลอยคอมโพเน็นต์นี้ไปยัง WebLogic เซิร์ฟเวอร์นั้น การมีเพียงไฟล์ ejb-jar.xml เพียงไฟล์เดียวนั้นไม่เพียงพอ อันดับแรก เราต้องสร้างและตั้งค่าต่าง ๆ ที่เกี่ยวข้องกับ WebLogic ไว้ในไฟล์ weblogic-ejb-jar.xml และนำไฟล์นี้เข้าไปไว้ใน deployment โดยที่ไฟล์ weblogic-ejb-jar จะระบุถึงการทำ caching, clustering และ performance behavior โดยที่ไฟล์นี้เป็นไฟล์ที่จำเป็นสำหรับทุก ๆ EJB ที่จะนำมาดีพลอยบน WebLogic เซิร์ฟเวอร์

ถ้าเป็นการดีพลอย entity bean ที่ใช้ container-managed persistent จะต้องเพิ่ม deployment file สำหรับจัดการ persistent ของ EJB ด้วย โดยไฟล์ descriptor เกี่ยวกับ persistent service ต่าง ๆ บนฐานข้อมูลเชิงสัมพันธ์ของ WebLogic เซิร์ฟเวอร์นั้นอาจมีชื่อว่า weblogic-cmp-rdbms-jar.xml หรืออาจมีชื่อว่าอะไรก็ได้ แต่ต้องตรงกับที่ระบุใน weblogic-ejb-jar.xml โดยเราจะต้องมีไฟล์นี้สำหรับทุก ๆ entity bean ที่ใช้ container-managed persistent บนฐานข้อมูลเชิงสัมพันธ์

ในกรณีที่เราใช้บริการจัดการ persistent ของผู้ผลิตอื่น ชื่อไฟล์และรายละเอียดภายในไฟล์ descriptor อาจแตกต่างไปจากไฟล์ weblogic-cmp-rdbms-jar.xml

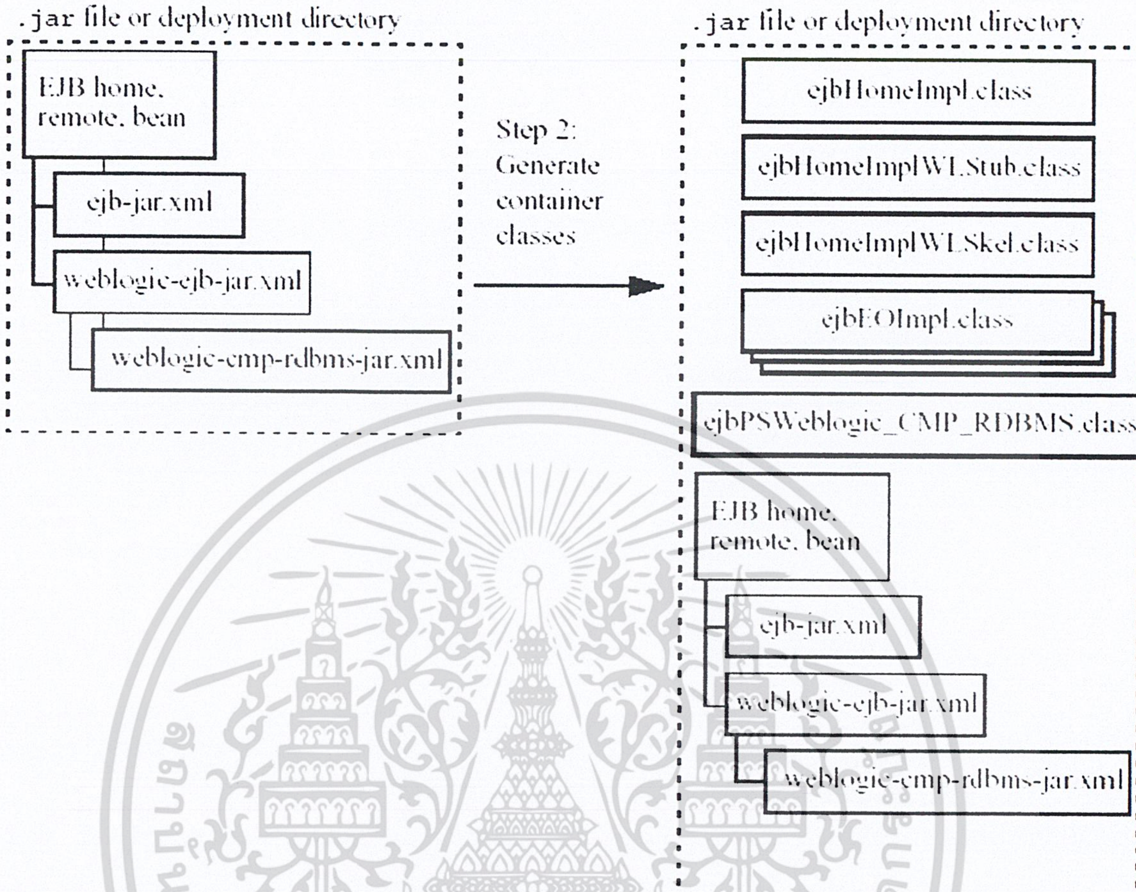
- สร้าง **EJB Container Class**

หลังจากที่คอมไพล์ EJB คลาสต่าง ๆ และเพิ่มไฟล์ XML deployment ของ WebLogic เข้าไปแล้วตามที่ระบุในหัวข้อ "การสร้าง Deployment Descriptor" จะต้องสร้าง container class ซึ่งใช้ในการเข้าถึง EJB โดยภายใน container class จะประกอบไปด้วยทั้งส่วนของ internal representation ของ EJB และส่วนอิมพลีเมนต์ของ external interface (home และ remote interface) ที่ไคลเอ็นต์เรียกใช้

โปรแกรม *ejbc* คอมไพเลอร์จะทำการสร้าง container class ตาม deployment properties ซึ่งถูกระบุไว้ใน XML deployment descriptor ต่าง ๆ ของ WebLogic ยกตัวอย่างเช่น ถ้าระบุว่า EJB ของเราจะถูกใช้งานใน cluster โปรแกรม ejbc จะทำการสร้างคลาสแบบ cluster-aware ซึ่งจะนำไปใช้ในการดีพลอย รูปที่ ก-6 แสดงไฟล์คอนเทนเนอร์คลาส

- **โหลด EJB Class เข้าไปยัง WebLogic Server**

classloader ของ WebLogic นั้น จะมีลักษณะเป็นลำดับชั้น โดยเมื่อเริ่มการทำงานของ WebLogic เซิร์ฟเวอร์ ตัว classloader ของ java system จะเริ่มทำงาน และจะเป็น parent ของ classloader อื่น ๆ ทั้งหมดที่ WebLogic สร้างขึ้น เมื่อ WebLogic เซิร์ฟเวอร์ทำการ deploy แอพพลิเคชันหนึ่ง ๆ มันจะทำการสร้าง classloader ขึ้นมาใหม่ 2 อันด้วยกัน สำหรับ EJB และเว็บแอพพลิเคชัน โดย EJB classloader จะเป็น child ของ java system classloader และเว็บแอพพลิเคชัน classloader จะเป็น child ของ EJB classloader




***รูปที่ ก-6 ไฟล์ Container Class***

- **การดีพลอย EJB ในตอนเริ่มการทำงานของ WebLogic**

ในการดีพลอย EJB โดยอัตโนมัติเมื่อเริ่มการทำงานของ WebLogic มีขั้นตอนดังนี้

1. ทำตามขั้นตอนในหัวข้อ "การสร้าง Deployment Descriptor" และ "การสร้าง EJB Container Class"
2. นำไฟล์ .jar ที่ได้จากการคอมไพล์ด้วยโปรแกรม ejbc คอมไพเลอร์ไปไว้ยังไดเรกทอรี wlserver6.0/config/*domain_name*/application
3. เมื่อ WebLogic เซิร์ฟเวอร์เริ่มทำงาน WebLogic เซิร์ฟเวอร์จะทำการดีพลอยไฟล์ .jar หรือ deployment directory ที่อยู่ในตำแหน่ง wlserver6.0/config/*domain_name*/application ให้โดยอัตโนมัติ

- **การดูรายการของ EJB ที่ถูกดีพลอยแล้ว**

ในการดูรายการของ EJB ที่ถูกดีพลอยลงไปใน WebLogic เซิร์ฟเวอร์ให้ทำตามขั้นตอนดังนี้

*จาก Command Line :*

1. พิมพ์คำสั่ง

   % java weblogic.deploy list *password*

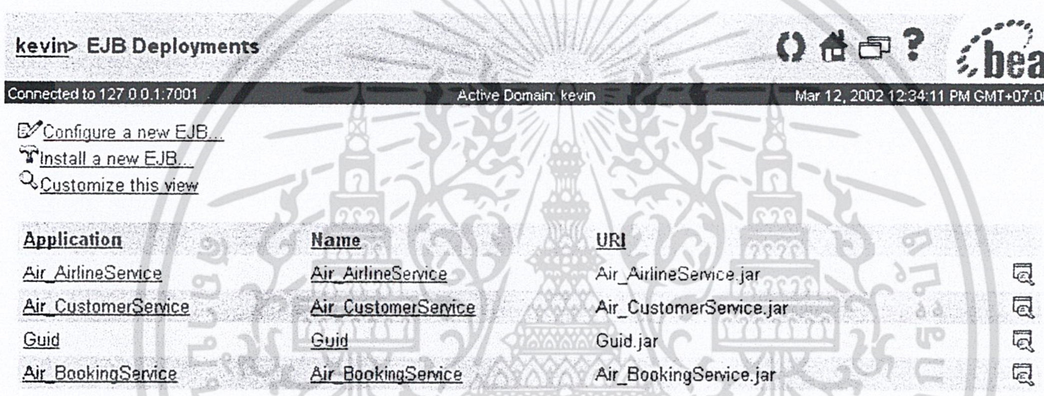
โดยที่ *password* คือรหัสผ่าน System Account ของเซิร์ฟเวอร์

2. ในการดูรายการของ EJB ที่ถูกดีพลอยไปยังรีโมตเซิร์ฟเวอร์ ให้ระบุตัวเลือก port และ host ดังนี้

   % java weblogic.deploy -port *port_number* -host *host_name* list *password*

*จาก Administration Console ของ WebLogic*

1. เลือก EJB deployment ในส่วนของ J2EE ทางขวามือของหน้าจอ
2. จะปรากฏรายการของ EJB ที่ถูกดีพลอยทั้งหมดขึ้น




***รูปที่ ก-7 หน้าจอคอนโซลสำหรับดู EJB ที่ถูกดีพลอย***

- **ดีพลอย EJB ใหม่ในสภาพแวดล้อมที่กำลังทำงานอยู่**

  ใช้คำสั่ง :

  % java weblogic.deploy -port *port_number* -host *host_name* deploy *password* *name source*

  เมื่อ :

  *Name* คือ ชื่อที่ตั้งให้กับ EJB deployment unit นี้

  *Source* คือ ตำแหน่งของไฟล์ .jar ที่ต้องการดีพลอย หรือตำแหน่งของ deployment directory

  ตัวอย่าง :

  % java weblogic.deploy -port 7001 -host localhost deploy weblogicpwd CMP_example c:\weblogic\myserver\unjarred\containerManaged\

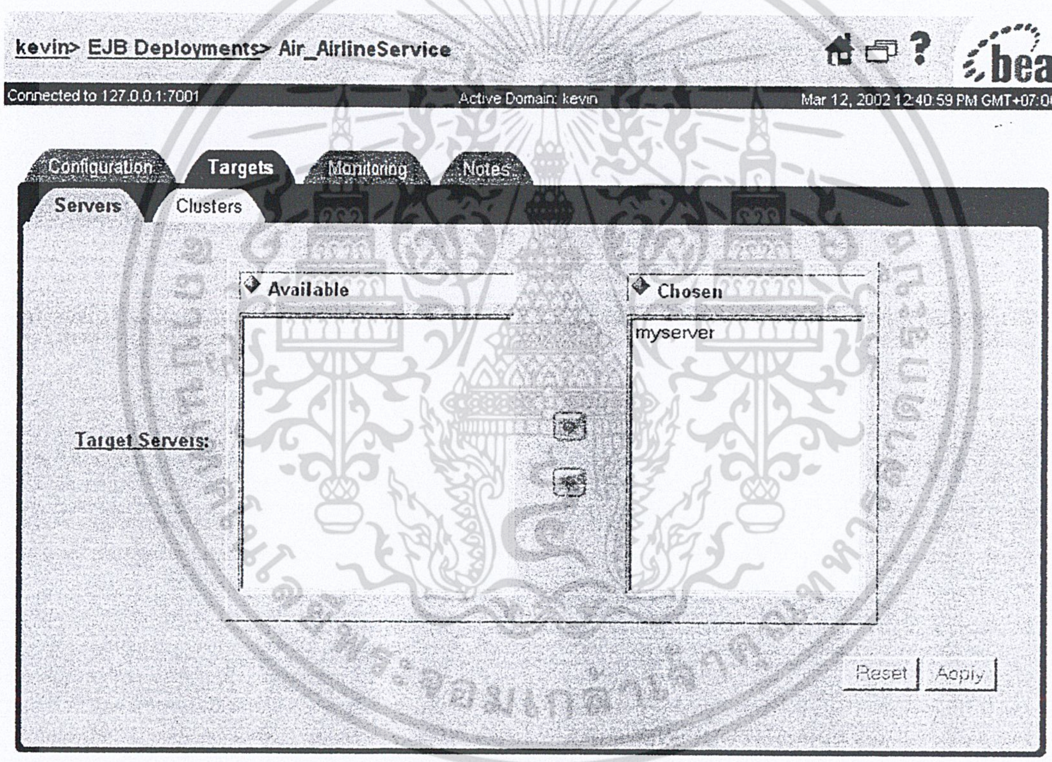
- **การ Undeploy EJB ที่ถูกดีพลอยอยู่**

*จาก Command Line :*

% java weblogic.deploy -port 7001 -host localhost undeploy weblogicpwd CMP_example

*จาก Administration Console ของ WebLogic Server :*

1. เลือก EJB deployment ในส่วนของ J2EE ทางขวามือของหน้าจอ
2. คลิกเลือก EJB ที่ต้องการ undeploy
3. เลือก Target tab และ Undeploy EJB โดยการย้าย Server มายังคอลัมน์ Available




***รูปที่ ก-8 การ undeploy EJB***

- **การอัพเดต EJB**

*จาก Command Line :*

ใช้พารามิเตอร์ update และระบุชื่อ EJB deployment

% java weblogic.deploy -port 7001 -host localhost update weblogicpwd CMP_example

*จาก Administration Console ของ WebLogic Server :*

1. เลือก EJB deployment ในส่วนของ J2EE ทางขวามือของหน้าจอ
2. คลิกเลือก EJB ที่ต้องการ Update
3. เลือก Target tab และ Update EJB โดยการย้าย Server มายังคอลัมน์ Chosen

เราสามารถอัพเดตได้เพียง EJB implementation class เท่านั้น โดยไม่รวมถึง public interface หรือ support class ต่าง ๆ ที่เป็นแบบ public

- **การดีพลอยเว็บแอพพลิเคชัน**

เว็บแอพพลิเคชันสามารถประกอบด้วยทรัพยากรเช่น Servlet , JSP , JSP แท็กไลบรารี และทรัพยากรแบบสแตติกเช่น HTML และรูปภาพ เว็บแอพพลิเคชันจะใช้โครงสร้างของไดเรกทอรีตามมาตรฐานที่ระบุใน J2EE specfication โดยการดีพลอยสามารถที่จะดีพลอยลงในไดเรกทอรีของเว็บแอพพลิเคชันโดยตรง หรือแพ็กเก็จเป็น .war ไฟล์ก็ได้

โครงสร้างของเว็บแอพพลิเคชันจะเป็นดังนี้

**WebApplicationName/**

จะประกอบด้วยสแตติกไฟล์เช่น HTML และ JSP

/WEB-INF/web.xml

เป็น deployment descriptor ที่ใช้คอนฟิกเว็บแอพพลิเคชัน

/WEB-INF/weblogic.xml

เป็น deployment descriptor ของ WebLogic ที่ใช้กำหนดการแม็ปรีซอร์สใน web.xml กับรีซอร์สที่อยู่ใน WebLogic Server ไฟล์นี้จะใช้ในการกำหนด JSP และ HTTP session attribute

/WEB-INF/classes

ประกอบด้วยคลาสที่อยู่บนฝั่งเซิร์ฟเวอร์เช่น HTTP servlet และยูทิลิตี้คลาส

/WEB-INF/lib

ประกอบด้วย .jar ไฟล์ที่ใช้ในเว็บแอพพลิเคชัน

WebApplicationName จะอยู่ในไดเรกทอรี config/*domain_name*/applications/ ปกติแล้วจะมี DefaultWebApp_*servername* (servername คือชื่อของเซิร์ฟเวอร์ที่ตั้งในตอน install ดีฟอลต์คือ myserver) เป็นดีฟอลต์เว็บแอพพลิเคชันอยู่แล้ว

**2. การติดตั้ง Apache SOAP 2.2**

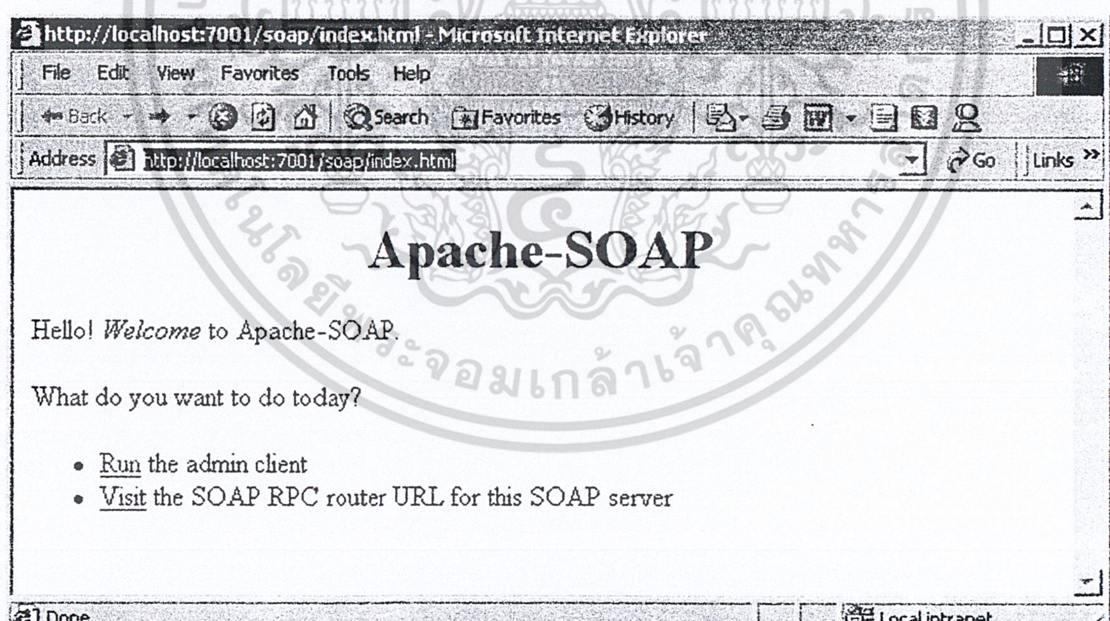
สามารถดาวน์โหลด Apache SOAP 2.2 ได้ที่ http://xml.apache.org/soap

การติดตั้ง Apache SOAP 2.2 มีวิธีติดตั้ง 2 กรณีคือติดตั้งกับเซิร์ฟเวอร์เพื่อให้เป็น SOAP listener โดยจะอินทิเกรตเข้ากับ WebLogic เซิร์ฟเวอร์และติดตั้งกับไคลเอ็นต์ โดยจะติดตั้งเฉพาะ jar ไฟล์ เพื่อให้สามารถใช้ SOAP API ในการติดต่อขอบริการได้

**2.1 การติดตั้ง SOAP Listener**

มีขั้นตอนดังนี้

1) ติดตั้ง SOAP เว็บแอพพลิเคชัน – คลาย Zip ออกแล้วก๊อบปี้โฟลเดอร์จาก path ./webapps ไปไว้ในไดเรกทอรี *wl_home/domain_name/*application/soap
2) ก๊อบปี้ soap.jar – สร้างไดเรกทอรี *wl_home/domain_name/*application/soap/WEB-INF/lib แล้วก๊อบปี้ soap.jar ไปไว้ในไดเรกทอรีนั้น
3) ติดตั้ง xerces เวอร์ชันใหม่ – WebLogic ได้รวม xerces มาแล้ว ถ้าต้องการ xerces เวอร์ชันอื่นให้เซต classpath ให้ xerces.jar อยู่ก่อนหน้า weblogic.jar (xerces เป็นชื่อของจาวา parser ที่รองรับ namespace)
4) ทดสอบโดยเปิดบราวเซอร์ URL เท่ากับ http://localhost:7001/soap/index.html จะได้ผลดังรูป ก-9




***รูปที่ ก-9 index ของ SOAP เว็บแอพพลิเคชัน***

ทดสอบโดยคลิก Run แล้วทดสอบ operation ต่าง ๆ ถ้าเกิด error ขึ้นที่คอนโซลของ WebLogic ที่แจ้งว่าหาไฟล์ DeployedServices.ds ไม่เจอ error นี้จะหายไปเมื่อได้มีการดีพลอยเซอร์วิสแล้ว หากไม่เกิด error อื่นอีกแสดงว่าติดตั้งถูกต้องแล้ว

**2.2 การติดตั้ง SOAP ไคลเอ็นต์**

จะต้องเซต classpath ชี้ไปยังไฟล์ดังนี้

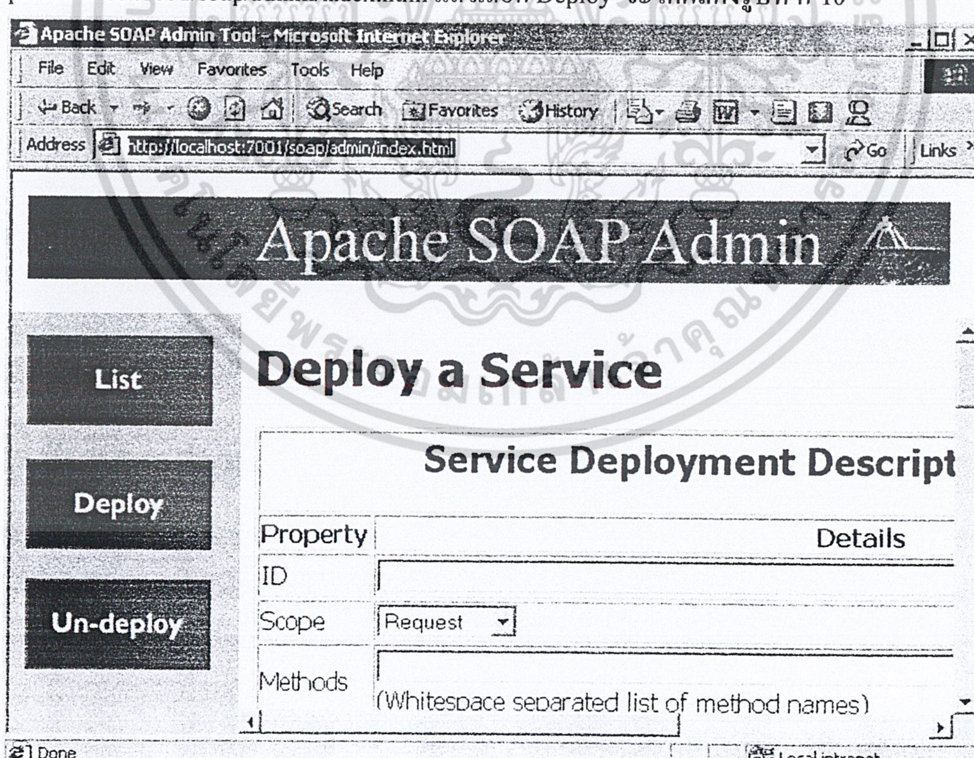
- soap.jar
- mail.jar หาได้ที่ http://java.sun.com/products/javamail
- activation.jar หาได้ที่ http://java.sun.com/products/beans/glasgow/jaf.html
- jaxp.jar หาได้ที่ http://java.sun.com/xml
- xerces.jar หาได้ที่ http://xml.apache.org

โดยต้องเซตให้ xerces.jar อยู่หน้าจาวา parser อื่น ๆ

**2.3 แนะนำการใช้งาน Apache SOAP เบื้องต้น**

**2.3.1 การดีพลอยเซอร์วิส**

การดีพลอยเซอร์วิสทำได้ 2 วิธีคือดีพลอยผ่านเว็บกับดีพลอยด้วยสคริปต์ การดีพลอยผ่านเว็บ URL คือ http://localhost:7001/soap/admin/index.html แล้วเลือก Deploy จะได้ผลดังรูปที่ ก-10




*รูปที่ ก-10 การดีพลอย SOAP ผ่านทางเว็บ*

การดีพลอยโดยสคริปต์เป็นการดีพลอยแบบคอมมานด์ไลน์ หลังจากเซต classpath ให้ถูกต้องแล้ว รันคอมมานด์ไลน์ด้วยคำสั่งนี้

java org.apache.soap.server.ServiceManagerClient http://localhost:7001/soap/servlet/rpcrouter deploy DeploymentDescriptor.xml

### 2.3.2 Deployment Descriptor

Deployment descriptor ที่แสดงเซอร์วิสที่อิมพลีเมนต์กับจาวาคลาสมาตรฐานจะเป็นไปตามนี้

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:service-urn" [type="message"]
[checkMustUnderstands="true|false"]>
  <isd:provider type="java"
                scope="Request | Session | Application"
                methods="exposed-methods">
    <isd:java class="implementing-class" [static="true|false"]/>
  </isd:provider>

<isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

***service-urn*** เป็น URN ที่ต้องการให้เป็นเซอร์วิส

***exposed-methods*** เป็นรายการของเมธอดที่ให้บริการ โดยแบ่งด้วย space

***implementing-class*** เป็นชื่อเต็มของคลาส (เช่น packetname.classname)

ใน <service> element มี optional attribute ที่ชื่อ *type* ซึ่งจะมีค่าเท่ากับ message ถ้าเซอร์วิสเป็น document-oriented แทนที่จะเป็น RPC invoke service และออปชัน attribute ที่ชื่อ *checkMustUnderstands* ซึ่ง value จะถูกเซตเป็น true หรือ false ขึ้นอยู่กับว่าต้องการให้เซิร์ฟเวอร์โยน Fault ถ้า SOAP header ใน request ถูกกำหนดให้เป็น MustUnderstand หรือไม่

ใน <java> element มีออปชัน attribute ชื่อ *static* ที่อาจถูกเซตเป็น true หรือ false ขึ้นอยู่กับว่า เมธอดนั้นเป็นสแตติกหรือไม่

<provider> element มี *scope* attribute ที่แสดงช่วงชีวิตของอินสแทนซ์ของคลาสที่อิมพลีเมนต์ 'Request' หมายถึงออบเจ็กต์จะถูก remove ภายหลัง request ทำเสร็จแล้ว 'Session' จะหมายถึงออบเจ็กต์จะอยู่ตลอดช่วงชีวิตของ HTTP session และ 'Application' หมายถึงออบเจ็กต์จะคงอยู่จนกระทั่ง Servlet ที่ให้บริการจะ terminate

Apache SOAP นั้นรองรับเซอร์วิสที่อิมพลีเมนต์กับ EJB แต่ในโครงงานนี้จะใช้จาวาคลาสมาตรฐานมาครอบ EJB อีกที เนื่องจากจำเป็นต้องทำ interop กับ .NET และมีปัญหาในการแมปชนิดของข้อมูลเช่น

date , time จึงใช้จาวาคลาสมาครอบแล้วแปลงข้อมูลจาก date หรือ time เป็นสตริง ทำให้ไม่ต้องเปลี่ยน signature ของ EJB ที่เป็น date หรือ time

# ภาคผนวก ข
# วิธีคลัสเตอร์ WebLogic เซิร์ฟเวอร์

**1.พื้นฐานการคลัสเตอร์ของ WebLogic เซิร์ฟเวอร์**

การคอนฟิกคลัสเตอร์กับ WebLogic เซิร์ฟเวอร์เป็นการแก้ไขไฟล์ XML (config.xml) ให้ระบุถึงข้อมูลในการคลัสเตอร์ ซึ่งทำได้ 3 วิธี

- ใช้ Administrator คอนโซล
- เขียนโปรแกรมแก้ไขคอนฟิกไฟล์ด้วย API ที่ WebLogic ให้มา
- ใช้คอมมานด์ไลน์ยูทิลิตี้

ในเอกสารนี้จะใช้วิธี Administrator คอนโซล เนื่องจากเป็นวิธีที่ทำได้ง่าย

**1.1 Administrator Server**

ภายในคลัสเตอร์หนึ่งจะต้องประกอบด้วย Administrator เซิร์ฟเวอร์หนึ่งเครื่อง และเซิร์ฟเวอร์ในคลัสเตอร์อื่น ๆ Administrator เซิร์ฟเวอร์จะเก็บข้อมูลการคลัสเตอร์ไว้ เมื่อเซิร์ฟเวอร์เครื่องอื่นในคลัสเตอร์ สตาร์ตขึ้นมาก็จะเข้ามาขอข้อมูลจาก Administrator เซิร์ฟเวอร์ รวมถึงข้อมูลเซอร์วิสที่ให้บริการด้วย ทำให้การ maintain ทำได้ง่ายเนื่องจากทำที่ Administrator เซิร์ฟเวอร์ที่เดียว แต่ก็ทำให้เกิด single point of failure ขึ้นคือ ถ้า Administrator เซิร์ฟเวอร์ขัดข้อง เครื่องอื่นในคลัสเตอร์จะไม่สามารถสตาร์ตได้ แต่จะไม่มีผลในกรณีที่เซิร์ฟเวอร์ในคลัสเตอร์ได้สตาร์ตก่อนที่ Administrator เซิร์ฟเวอร์จะขัดข้อง

**1.2 ความต้องการในการคลัสเตอร์**

ในการคลัสเตอร์จะต้องเป็นไปตามความต้องการนี้

- ต้องเป็น WebLogic เซิร์ฟเวอร์เวอร์ชันเดียวกัน
- จะต้องมีโดเมนเดียวกัน

**1.3 ขั้นตอนการคลัสเตอร์**

การคอนฟิกจะทำที่ Administrator เซิร์ฟเวอร์ดังนี้

1) **สร้างอินสแทนซ์ของ WebLogic เซิร์ฟเวอร์**

สตาร์ตคอนโซล คลิกที่ Servers node ที่อยู่ทางด้านซ้าย เลือก Create a new Server ใส่ข้อมูลดังนี้

| Attribute | Description |
|---|---|
| Name | ชื่อของเซิร์ฟเวอร์ |
| Listen Port | หมายเลขพอร์ต ทุกเซิร์ฟเวอร์ในคลัสเตอร์ต้องใช้หมายเลขพอร์ตเหมือนกัน |
| Listen Address | ชื่อ DNS หรือ IP แอดเดรสที่ bind กับเซิร์ฟเวอร์ |

ให้สร้างอินสแทนซ์ของทุก ๆ เซิร์ฟเวอร์ในคลัสเตอร์บน Administrator เซิร์ฟเวอร์

**2) สร้างคลัสเตอร์**

สตาร์ตคอนโซล แล้วคลิกที่ Clusters node ที่อยู่ทางด้านซ้าย เลือก Create a new Cluster ให้ใส่ข้อมูลดังนี้

**Configuration → General**

| Attribute | Description |
|---|---|
| Name | ชื่อของคลัสเตอร์. |
| Cluster Address | เป็นแอดเดรสที่ไคลเอ็นต์จะใช้ในการ connect ไปยังคลัสเตอร์ แอดเดรสอาจเป็นชื่อของโฮสต์ DNS ที่แมปไปยังหลาย ๆ IP แอดเดรสหรือ เป็นลิสต์ของ IP แอดเดรสที่แบ่งโดยใช้คอมมา (อนุโลมให้ใช้ในการทดสอบ ไม่ควรนำไปใช้ในระบบจริง) ดูวิธีคอนฟิก DNS ได้ในส่วนถัดไป |
| Default Load Algorithm | อัลกอริทึมที่ใช้ในการกระจายโหลด |

**Configuration → Multicast**

| Attribute | Description |
|---|---|
| Multicast Address | มัลติคาสต์แอดเดรสที่ใช้การติดต่อระหว่างเซิร์ฟเวอร์ในคลัสเตอร์ จะอยู่ในช่วง 224.0.0.1 to 239.255.255.255 |
| Multicast Send Delay | เวลาที่ใช้ดีเลย์ในการส่งเมสเซจ |
| Multicast TTL | ค่ามัลติคาสต์ Time To Alive |

**Server**

| Attribute | Description |
|---|---|
| Choose Servers for this Cluster | ให้เลือกเซิร์ฟเวอร์ที่ต้องการนำมาคลัสเตอร์ |

**3) คอนฟิก Replication Group**

เปิด Administrator คอนโซล เลือก Servers node แล้วเลือกเซิร์ฟเวอร์ที่ต้องการจะคอนฟิก เลือกแท็บ Cluster ให้ใส่ชื่อของ Replication Group ซึ่งเป็นชื่อกลุ่มของการ replicate และ Preferred Secondary Group ซึ่งเป็นกลุ่มที่ต้องการใช้เป็นโฮสต์ในการ replicate

**4) สตาร์ต WebLogic เซิร์ฟเวอร์คลัสเตอร์**

กระบวนการในการสตาร์ตคลัสเตอร์เป็นดังนี้

- สตาร์ต Administrator เซิร์ฟเวอร์
- สตาร์ตเซิร์ฟเวอร์แต่ละตัวในคลัสเตอร์ (ทำที่เซิร์ฟเวอร์ที่ต้องการคลัสเตอร์) โดยเซตให้ Administrator เซิร์ฟเวอร์เป็นผู้ควบคุมด้วยสคริปต์ ดังนี้

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m -classpath %CLASSPATH%
-Dweblogic.Domain=olala -Dweblogic.Name=olala01 -Dbea.home=d:\bea
-Djava.security.policy==d:\bea\wlserver6.0/lib/weblogic.policy
-Dweblogic.management.server=172.16.102.11:7001
-Dweblogic.management.username=system
-Dweblogic.management.password=systempassword weblogic.Server
```

โดยแต่ละส่วนหมายถึง

- weblogic.Name ใช้ระบุถึงอินสแทนซ์ของคลัสเตอร์ที่ต้องการสตาร์ต
- weblogic.management.server ระบุถึงโฮสต์และหมายเลขพอร์ตของ Administrator เซิร์ฟเวอร์ที่เก็บข้อมูลการคลัสเตอร์อยู่
- weblogic.management.username ระบุชื่อของยูสเซอร์ที่ติดต่อกับ Administrator เซิร์ฟเวอร์
- weblogic.management.password ระบุถึงพาสเวิร์ดของยูสเซอร์

**2. การคอนฟิก DNS**

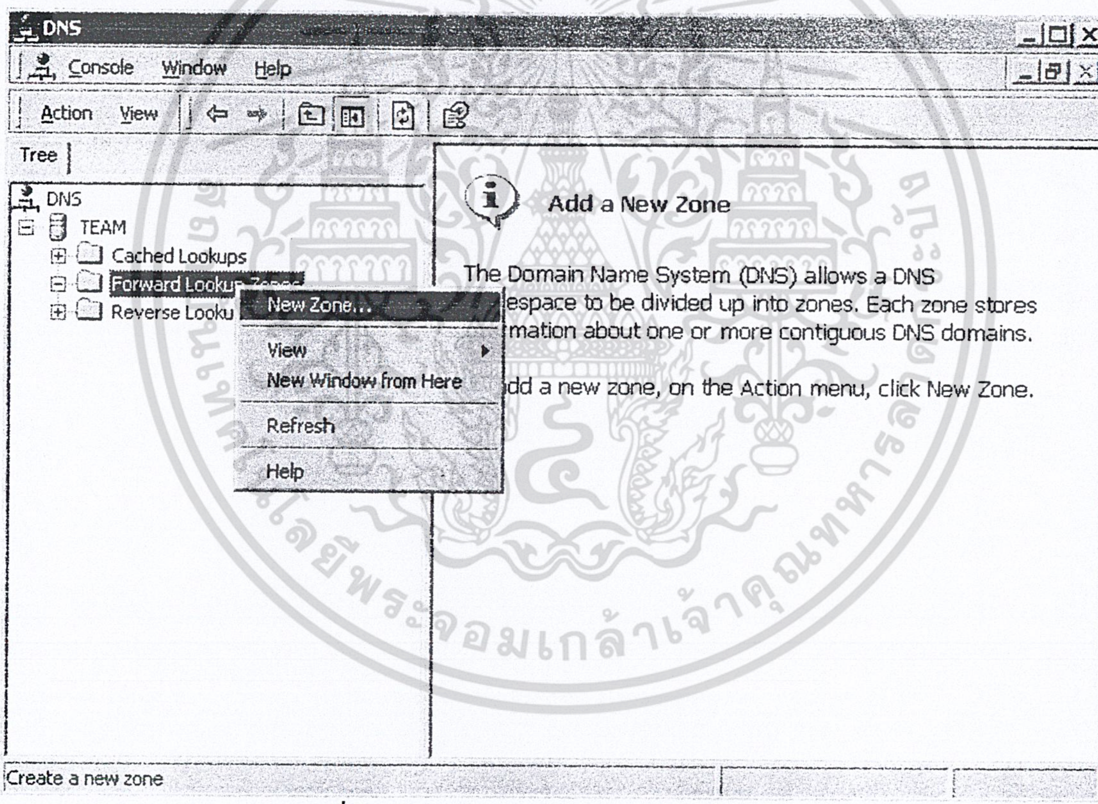
สมมติให้ Cluster address คือ mycluster ที่ประกอบด้วยเซิร์ฟเวอร์คือ cluster1 (161.246.5.247) , cluster 2 (161.246.5.248) และ cluster3 (161.246.5.249) จะต้องคอนฟิก DNS เซิร์ฟเวอร์ให้รู้จักชื่อ mycluster และแมปไปยังเซิร์ฟเวอร์ทั้ง 3 ได้

ตัวอย่างของการคอนฟิกเป็นการคอนฟิกโดยใช้ DNS เซิร์ฟเวอร์ของ Window 2000 Advance Server

**2.1 การคอนฟิก DNS เซิร์ฟเวอร์**

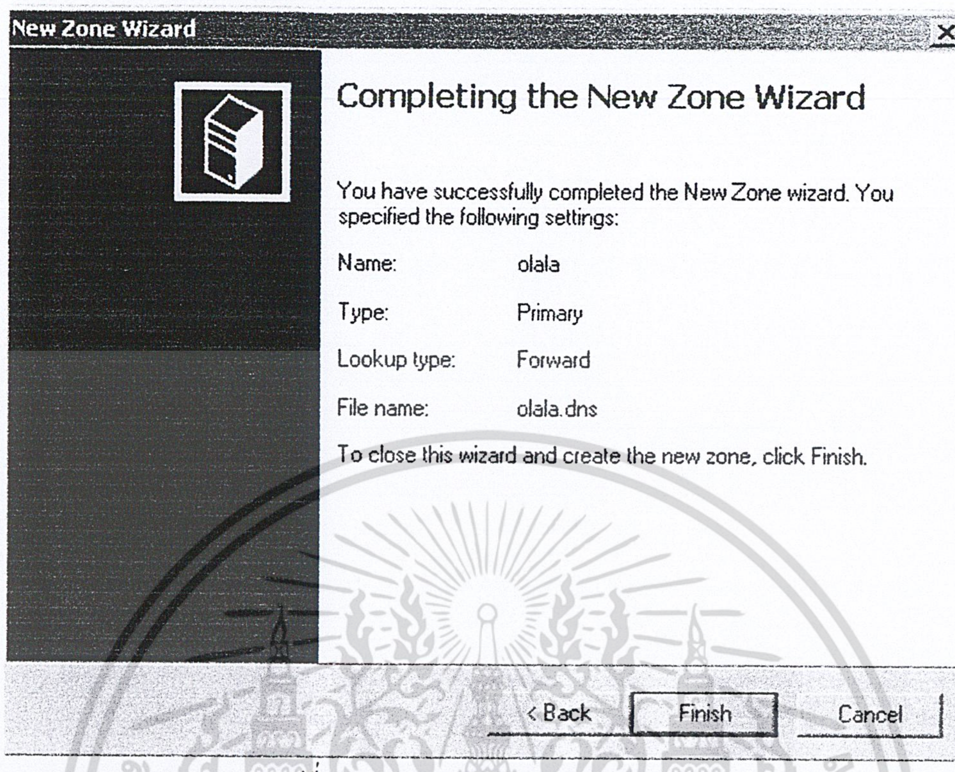
มีขั้นตอนดังนี้

- ให้ล็อกอินด้วยสิทธิ์ Administrator แล้วไปที่ Administrative Tools -> DNS
- เลือก Forward Lookup Zones -> New Zone...จะมี wizard ขึ้นมา ให้ Next และเลือก Standard primary แล้วให้ชื่อของ zone ให้ใส่ชื่อ สมมติให้ชื่อ olala ให้ Next ไปจนจบ wizard

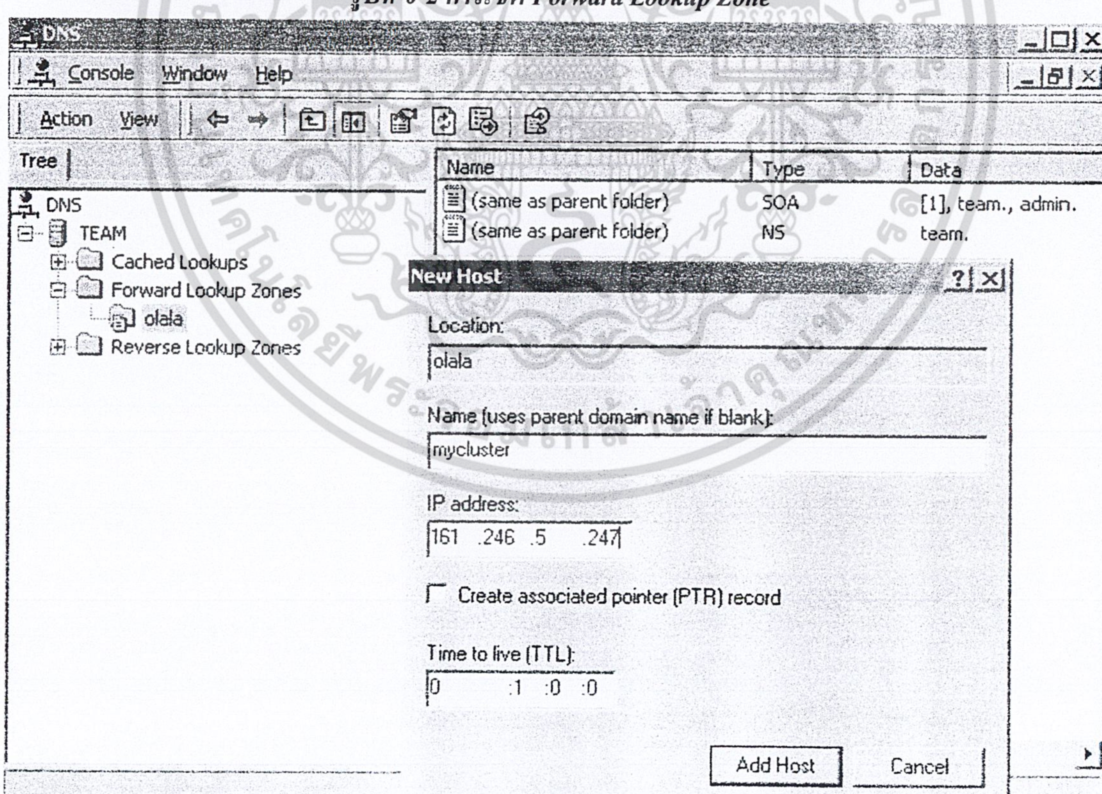



***รูปที่ ข-1 Forward Lookup Zones -> New Zone...***

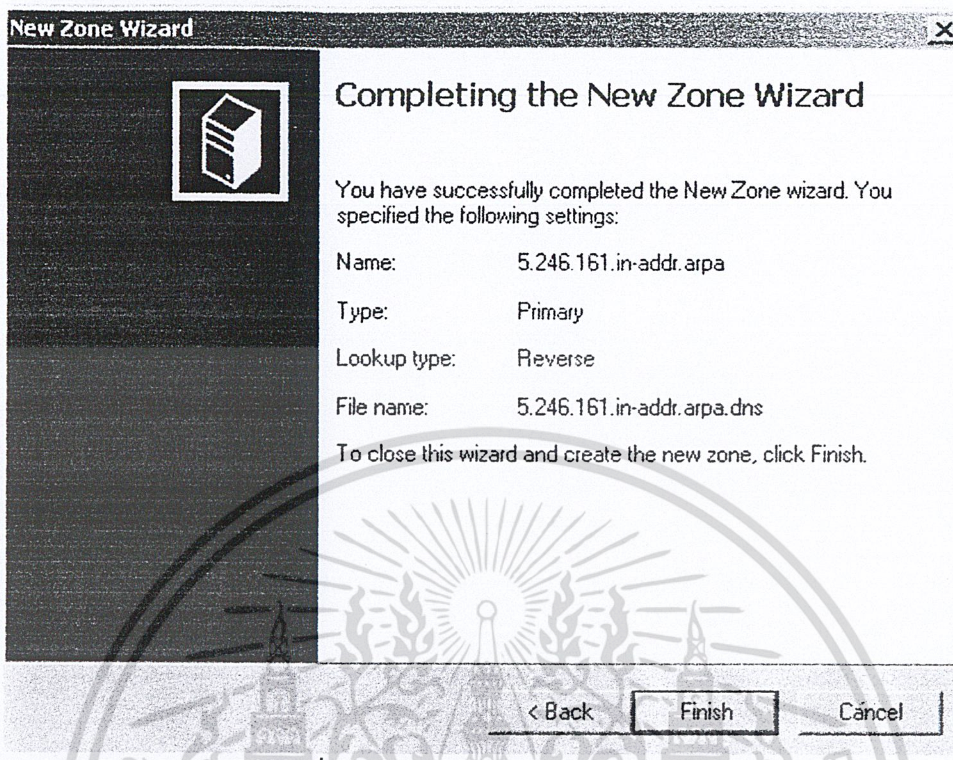
- คลิกที่ซับโฟลเดอร์ olala แล้วเลือก New Host... ให้ใส่ชื่อ mycluster และระบุ IP แอดเดรสไปยัง IP ของ cluster1 ทำขั้นตอนนี้จนครบทุกเซิร์ฟเวอร์ในคลัสเตอร์
- เลือก Reserve Lookup Zones -> New Zone.. จะมี wizard ขึ้นมา ให้ Next และเลือก Standard primary แล้วใส่ค่า Network ID เท่ากับ 161.246.5 ให้ Next ไปจนจบ wizard



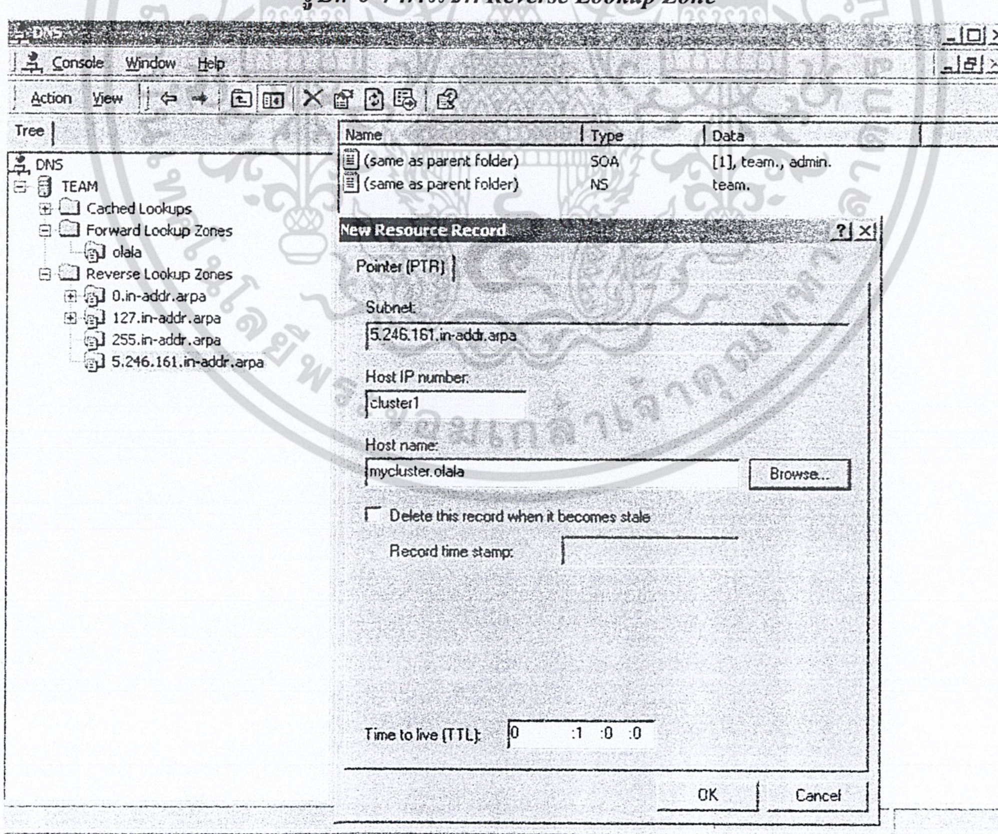

รูปที่ ข-2 การเซต *Forward Lookup Zone*




รูปที่ ข-3 *New Host…*




*รูปที่ ข-4 การเซต Reverse Lookup Zone*



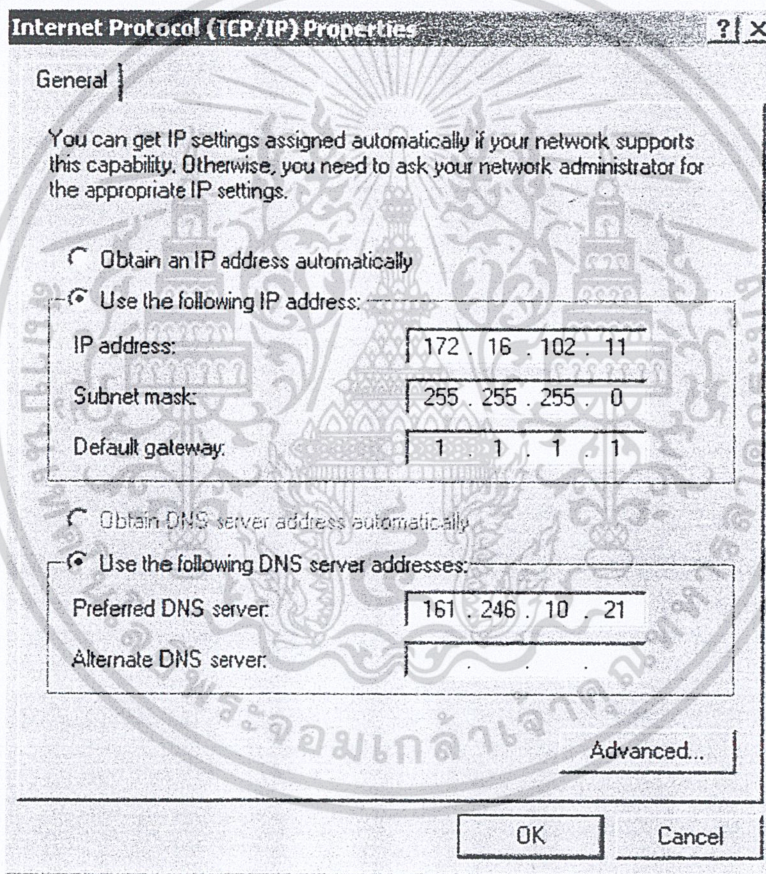

*รูปที่ ข-5 New Pointer…*

- คลิกที่ซับโฟลเดอร์ที่เป็นชื่อของไฟล์ที่ได้สร้าง Reserve Lookup Zones ไว้ แล้วเลือก New Pointer… ใส่ค่า Host IP number เป็น cluster1 แล้ว Browse ในส่วนของ Host name แล้วคลิกเลือกจนพบ mycluster ที่มี IP ตรงกัน ทำขั้นตอนนี้จนครบทุกเซิร์ฟเวอร์ในคลัสเตอร์

**2.2 การคอนฟิก DNS เซิร์ฟเวอร์ให้กับเซิร์ฟเวอร์ในคลัสเตอร์**

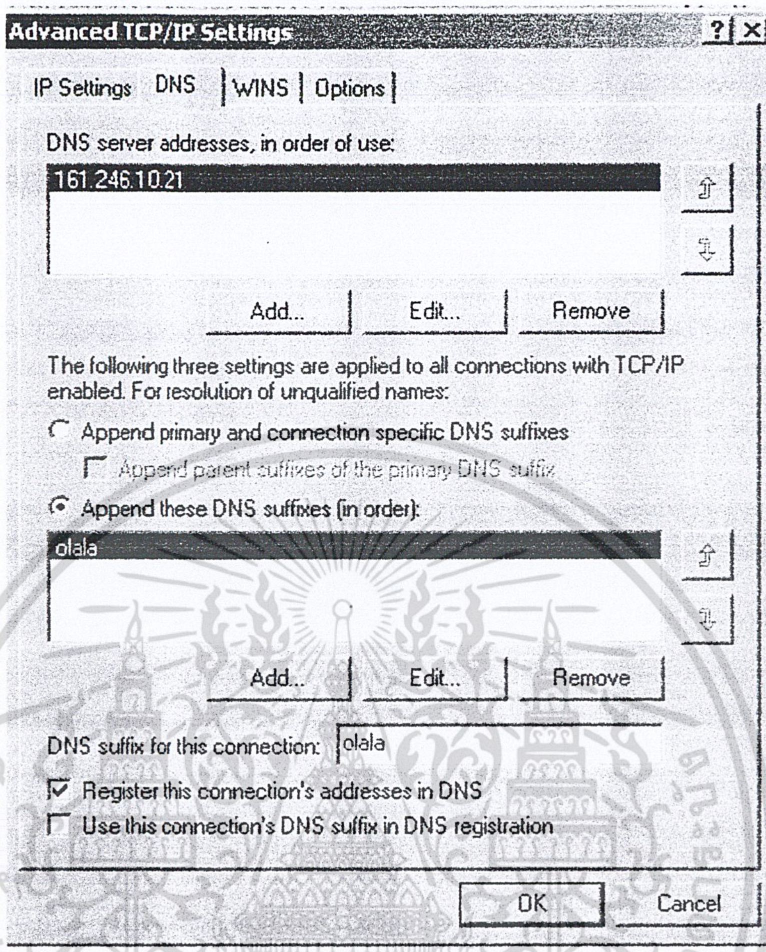
เมื่อคอนฟิก DNS เซิร์ฟเวอร์แล้วจะต้องคอนฟิกให้เซิร์ฟเวอร์ในคลัสเตอร์ bind เซิร์ฟเวอร์กับ DNS เซิร์ฟเวอร์ดังนี้

- ในส่วนของการเซตเน็ตเวิร์กในวินโดว์ของ Internet Protocal (TCP/IP) Properties ให้เซตค่า Preferred DNS Server เป็น DNS เซิร์ฟเวอร์ที่ได้คอนฟิกไว้




*รูปที่ ข-6 การเซต Preferred DNS server*

- คลิก Advanced… แล้วเลือกแท็บ DNS เลือก radio button ที่ชื่อ Append these DNS suffixes แล้วคลิก Add ใส่ค่า Domain suffix เป็น olala
- ใส่ค่าในส่วนของ DNS suffix for this connection เป็น olala
- คลิก OK ให้หมดแล้ว restart




***รูปที่ ข-7 การเซต DNS suffix***

เมื่อวินโดว์สตาร์ตขึ้นมามันจะ bind ชื่อ cluster1.olala (ในกรณีเซิร์ฟเวอร์คือ cluster1) กับ DNS เซิร์ฟเวอร์

**2.3 เหตุผลของการใช้ DNS**

ในการคลัสเตอร์ออบเจ็กต์เทียร์จะต้องใช้ DNS เซิร์ฟเวอร์ พิจารณาการเรียกใช้ EJB ในส่วนของการ look up ดังนี้

```
Properties p = new Properties();
String url="t3://161.246.5.247:7001"
p.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
p.put(Context.PROVIDER_URL, url);
```

การระบุ url ในลักษณะนี้จะทำให้เกิด single point of failure ขึ้นคือถ้าเซิร์ฟเวอร์ 161.246.5.247 ขัดข้องจะไม่สามารถทำงานได้เลย ถึงแม้เซิร์ฟเวอร์อื่นในคลัสเตอร์จะยังสามารถทำงานได้ก็ตาม

แต่ถ้าระบุ url เป็น “t3://mycluster:7001” การ initial context จะขอลิสต์ของเซิร์ฟเวอร์จาก DNS เซิร์ฟเวอร์ก่อน แล้วจึงเลือกเซิร์ฟเวอร์ในลิสต์นั้นมา look up สมมติ DNS เซิร์ฟเวอร์คืนค่ามาดังนี้

161.246.5.247 , 161.246.5.248 , 161.246.5.249

ลิสต์ของเซิร์ฟเวอร์จะมีเฉพาะเซิร์ฟเวอร์ที่ทำงานอยู่เท่านั้น กรณีนี้ไคลเอ็นต์จะ look up กับ 161.246.5.247 แต่ถ้า 161.246.5.247 ทำงานอยู่ แต่ WebLogic เซิร์ฟเวอร์ไม่ได้เปิดไว้หรือขัดข้อง การ look up จะขัดข้อง ไคลเอ็นต์จะ look up กับ 161.246.5.248 ซึ่งอยู่ลำดับถัดไป

และ DNS เซิร์ฟเวอร์สามารถ Round-robin ได้คือ เมื่อมีการขอลิสต์จาก DNS เซิร์ฟเวอร์ครั้งแรกอาจจะคืนค่าเป็น 161.246.5.247 , 161.246.5.248 , 161.246.5.249 ครั้งต่อไปจะเป็น 161.246.5.248 , 161.246.5.249 , 161.246.5.247 และถัดไปจะเป็น 161.246.5.249 , 161.246.5.247 , 161.246.5.248 แต่จากการทดสอบพบว่าในการขอลิสต์จาก DNS เซิร์ฟเวอร์จะไม่ Round-robin ทุกครั้งที่ initial context เนื่องจากได้มีการแคชค่าเหล่านี้ไว้ เพื่อลดการขอลิสต์จาก DNS เซิร์ฟเวอร์

**3. เซต Proxy Plug-in**

เนื่องจากการคลัสเตอร์พรีเซนเทชันเทียร์นั้นต้องการ load balancer หรือ proxy plug-in ที่จะใช้ในการกระจายโหลดให้กับคลัสเตอร์พรีเซนเทชันเทียร์

การเซต WebLogic proxy plug-in (HttpClusterServlet) ทำได้โดยการแก้ไข web.xml ของเว็บแอพพลิเคชันดังตัวอย่างข้างล่างนี้

```
<!-- DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
//DTD Web Application 1.2//EN"
 "file:///weblogic/dev/myserver/servlet2.2/WEB-INF/web-jar.dtd"
-->
<web-app>
<servlet>
  <servlet-name>HttpClusterServlet</servlet-name>
    <servlet-class>
      weblogic.servlet.internal.HttpClusterServlet
```

```
  </servlet-class>
 <init-param>
  <param-name>defaultServers</param-name>
  <param-value>
    myserver1:7736:7737|myserver2:7736:7737|myserver:7736:7737
  </param-value>
 </init-param>
 <init-param>
  <param-name>DebugConfigInfo</param-name>
  <param-value>ON</param-value>
 </init-param>
</servlet>
<servlet-mapping>
 <servlet-name>HttpClusterServlet</servlet-name>
 <url-pattern>/</url-pattern>
</servlet-mapping>
<servlet-mapping>
 <servlet-name>HttpClusterServlet</servlet-name>
 <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
 <servlet-name>HttpClusterServlet</servlet-name>
 <url-pattern>*.htm</url-pattern>
</servlet-mapping>
<servlet-mapping>
 <servlet-name>HttpClusterServlet</servlet-name>
 <url-pattern>*.html</url-pattern>
</servlet-mapping>
</web-app>
```

พารามิเตอร์แต่ละตัวมีความหมายดังนี้

| <param-name> | <param-value> |
|---|---|
| defaultServers | เป็นส่วนที่ต้องมีระบุถึงสมาชิกในคลัสเตอร์ที่จะกระจายโหลดไปให้ อยู่ในรูปแบบดังนี้ hostname1:HTTP port1:HTTPS port1\|hostname2:HTTP port2:HTTPS port2 โดยถ้าเซต secureProxy เป็น ON จะต้องระบุ HTTPS พอร์ต ด้วย |
| secureProxy | ON/OFF ถ้าเป็น ON จะยอมให้ใช้ SSL ระหว่าง HttpClusterServlet และสมาชิกในคลัสเตอร์ |
| DebugConfigInfo | ON/OFF ถ้าเป็น ON จะสามารถ query HttpClusterServlet เพื่อดีบักได้โดยจะเพิ่มพารามิเตอร์ ?_WebLogicBridgeConfig ให้กับทุก ๆ request |
| connectionTimeout | กำหนดเวลา timeout เป็นมิลลิวินาที ถ้าเกิน HttpClusterServlet จะโยน java.io.InterruptedIOException ออกมา |
| numOfRetries | จำนวนครั้งที่ HttpClusterServlet จะพยายามทำ connection ที่ขัดข้อง |

**4. เซตการ Replicate State ของ HTTP Session**

เซตพารามิเตอร์ PersistentStoreType ที่อยู่ในไฟล์ weblogic.xml (อยู่ในซับไดเรกทอรี web-inf) ที่อยู่ในเว็บแอพพลิเคชันเป็น replicated (ดีฟอลต์คือ memory)

```
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web Application 6.0//EN"
"http://www.bea.com/servers/wls600/dtd/weblogic-web-jar.dtd">
<weblogic-web-app>
  <session-descriptor>
    <session-param>
      <param-name>PersistentStoreType</param-name>
      <param-value>replicated</param-value>
    </session-param>
  </session-descriptor>
</weblogic-web-app>
```

# ภาคผนวก ค

# JDBC

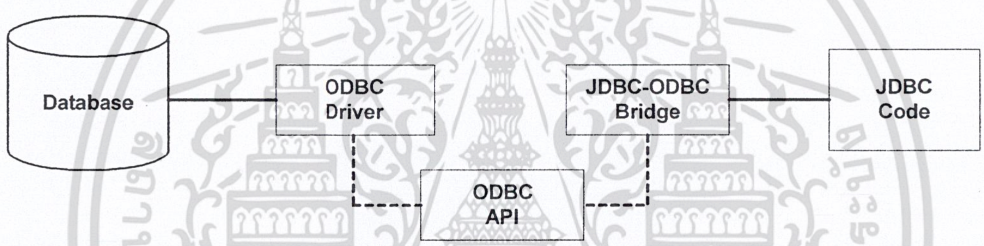
ในส่วนนี้จะอธิบายถึงความรู้พื้นฐานเกี่ยวกับ JDBC และวิธีการเซต Connection Pool และ Data Source กับ WebLogic เซิร์ฟเวอร์

## 1. ชนิดของ JDBC ไดรเวอร์

ไดรเวอร์ของ JDBC มี 4 ชนิดคือ

### 1.1 Type 1

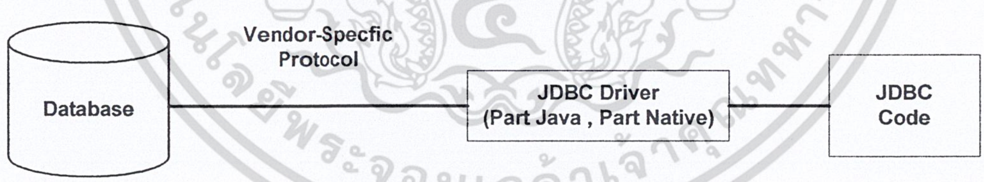
ไดรเวอร์นี้คือ JDBC-ODBC bridge ที่มากับ Java 2 Platform การเข้าถึง ODBC ผ่านทาง JDBC นี้มีประสิทธิภาพต่ำ




*รูปที่ ค-1 JDBC ไดรเวอร์ Type 1*

### 1.2 Type 2

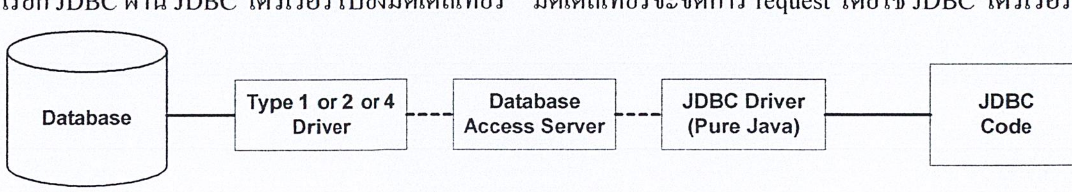
type 2 จะใช้จาวาไดรเวอร์ติดต่อกับ API ของฐานข้อมูล




*รูปที่ ค-2 JDBC ไดรเวอร์ Type 2*

### 1.3 Type 3

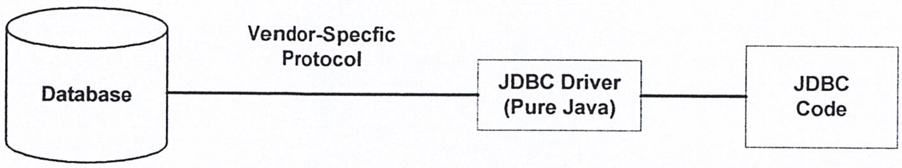
ไดรเวอร์แบบนี้จะใช้จาวาไดรเวอร์ทั้งหมดและจาวามิดเดิลเทียร์เพื่อเข้าถึงข้อมูล จาวาโปรแกรมจะเรียก JDBC ผ่าน JDBC ไดรเวอร์ไปยังมิดเดิลเทียร์ มิดเดิลเทียร์จะจัดการ request โดยใช้ JDBC ไดรเวอร์อื่น




*รูปที่ ค-3 JDBC ไดรเวอร์ Type 3*

### 1.4 Type 4

type 4 เป็นจาวาไดรเวอร์ทั้งหมดที่ทำงานโดยตรงกับฐานข้อมูล




*รูปที่ ค-4 JDBC ไดรเวอร์ Type 4*

## 2.ไดรเวอร์ที่มากับ WebLogic เซิร์ฟเวอร์

จะกล่าวถึงเฉพาะของ Oracle 8i และ MS SQL server 2000

ไดรเวอร์ของ Oracle 8i มี 2 ชนิดคือ type 2 และ type 4 ส่วนของ MS SQL server จะมีเฉพาะ type 4 ดังตารางนี้

| ฐานข้อมูล | Type | JDBC Class | JDBC URL |
|---|---|---|---|
| **Oracle 8i** | 2 | weblogic.jdbc.oci.Driver | jdbc:weblogic:oracle |
| **Oralce 8i** | 4 | oracle.jdbc.driver.OracleDriver | jdbc:oracle:thin:@*Host*:*Port*:*Database* |
| **MS SQL server 2000** | 4 | weblogic.jdbc.mssqlserver4.Driver | jdbc:weblogic:mssqlserver4 |

## 3. การเซต Connection Pool

การเซต Connection Pool มีขั้นตอนดังนี้

- สตาร์ตคอนโซล
- เลือก *domain_name*/Services/JDBC/Connection Pools ในเฟรมทางด้านซ้าย แล้วคลิก Create a new JDBC Connection Pool... ที่อยู่ในเฟรมทางด้านขวา
- เซตค่า Name เช่น OlalaPool
- เซตไดรเวอร์ URL เช่น jdbc:weblogic:mssqlserver4
- เซตไดรเวอร์คลาส เช่น weblogic.jdbc.mssqlserver4.Driver (ต้องตรงกับไดรเวอร์ URL)
- เซตค่าคุณสมบัติ เช่น user=sa password=password server=olalaDB@Olala37:1433
- เซตค่า connection เช่น Initial Capacity = 2 , Maximum Capacity =10 เป็นต้น
- เซต Target ไปยังเซิร์ฟเวอร์ที่ต้องการ

**4. การเซต Data Source**

การเซต Data Source มีขั้นตอนดังนี้

- สตาร์ตคอนโซล
- เซต Connection Pool
- เลือก *domain_name*/Services/JDBC/Data Sources ในเฟรมทางด้านซ้าย แล้วคลิก Create a new JDBC Data Source… ที่อยู่ในเฟรมทางด้านขวา
- เซตค่า Name เช่น OlalaDB
- เซตค่า JNDI name เช่น OlalaDB
- เซตค่า Pool name เช่น OlalaPool
- เซต Target ไปยังเซิร์ฟเวอร์ที่ต้องการ

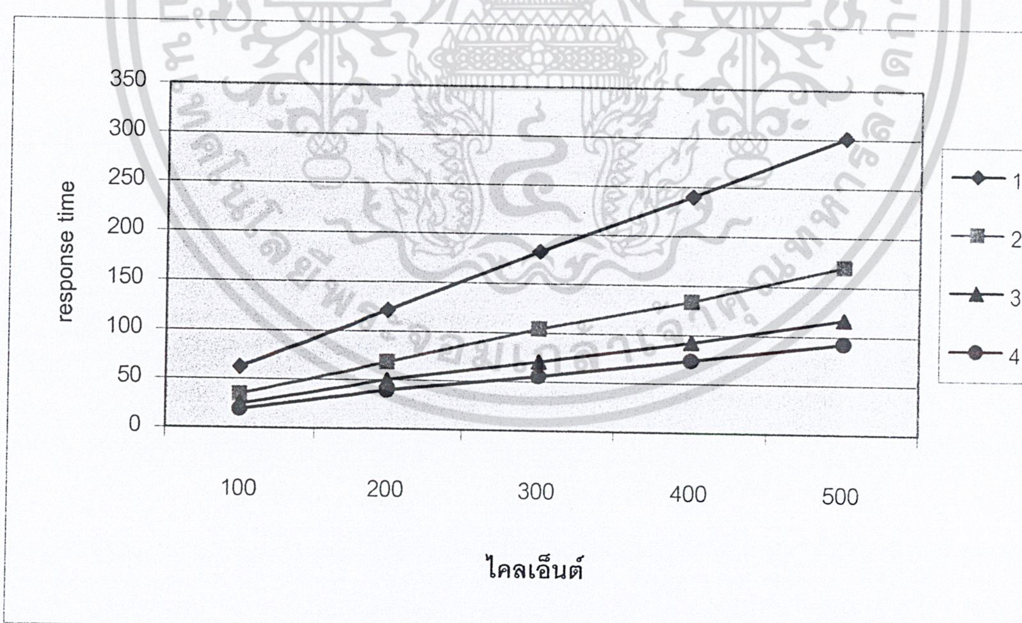
# ภาคผนวก ง

# ทดสอบประสิทธิภาพของ WebLogic เซิร์ฟเวอร์คลัสเตอร์

การทดสอบนี้เป็นการทดสอบการคลัสเตอร์ของพรีเซนเทชันเทียร์ของ WebLogic เซิร์ฟเวอร์คลัสเตอร์ โดยใช้ไคลเอ็นต์จำนวน 100 ไคลเอ็นต์และเพิ่มขึ้นครั้งละ 100 จนถึง 500 และใช้เซิร์ฟเวอร์ 1 เครื่อง โดยเพิ่มเซิร์ฟเวอร์ทีละเครื่องจนถึง 4 เครื่อง ได้ผลดังนี้ (response time มีหน่วยเป็นวินาที)

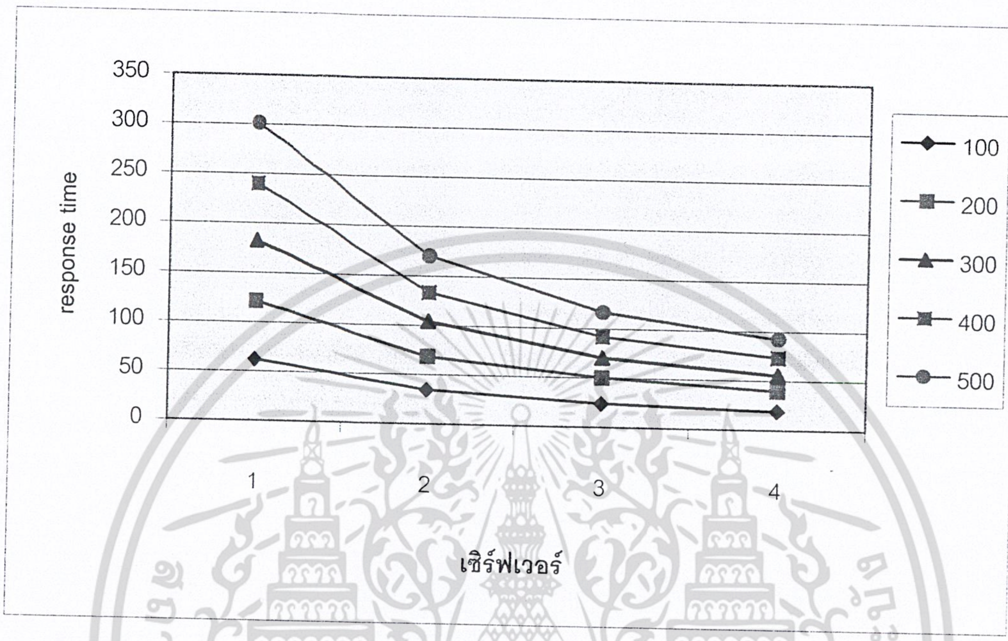
| เซิร์ฟเวอร์ \ ไคลเอ็นต์ | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|
| 1 | 61.941 | 121.01 | 182.127 | 239.99 | 300.86 |
| 2 | 34.5 | 68.06 | 103.92 | 133.22 | 170.02 |
| 3 | 23.94 | 49.92 | 70.19 | 91.72 | 116.43 |
| 4 | 18.918 | 39.174 | 55.45 | 73.527 | 92.078 |

เมื่อนำมาเขียนเป็นกราฟจะได้ดังนี้




กราฟนี้แสดงให้เห็นถึงการเพิ่มขึ้นของไคลเอ็นต์จะทำให้ response time เพิ่มขึ้นอย่างเป็นเชิงเส้น ส่วนกราฟถัดไปแสดงให้เห็นถึงการลดลงของ response time เมื่อเพิ่มเซิร์ฟเวอร์เข้าไป โดยการลดลงเกือบจะเป็น scale ยกเว้นในกรณีที่เพิ่มเป็น 4 เครื่อง เนื่องจากเซิร์ฟเวอร์ที่ใช้ทดสอบมีประสิทธิภาพไม่เท่ากัน ลำดับ

การเพิ่มเซิร์ฟเวอร์จะเป็นดังนี้ 700 MHz , 700 MHz , 750 MHz และ 550 MHz ซึ่งจะเห็นว่าเซิร์ฟเวอร์ตัวสุดท้ายมีประสิทธิภาพต่างจากตัวอื่นมากเกินไป ทำให้การลดลงไม่เป็น scale




รายละเอียดของเครื่องที่ใช้ทดสอบ

| | **CPU** | **RAM** |
|---|---|---|
| 1. | 700 MHz. | 384 MB. |
| 2. | 700 MHz. | 384 MB. |
| 3. | 750 MHz. | 384 MB. |
| 4. | 550 MHz. | 384 MB. |

โดยอัลกอริทึมของการ load balance คือ round-robin และไคลเอ็นต์หนึ่งจะ request ทั้งหมด 10 เพจเรียงกันไป เพจที่ให้ประมวลผลเป็นการคำนวณจำนวนของจำนวนเฉพาะระหว่าง 1-20000 โค้ดดังนี้

**FindPrime.jsp**

```
<html>
<head>
<title>
prime
</title>
```

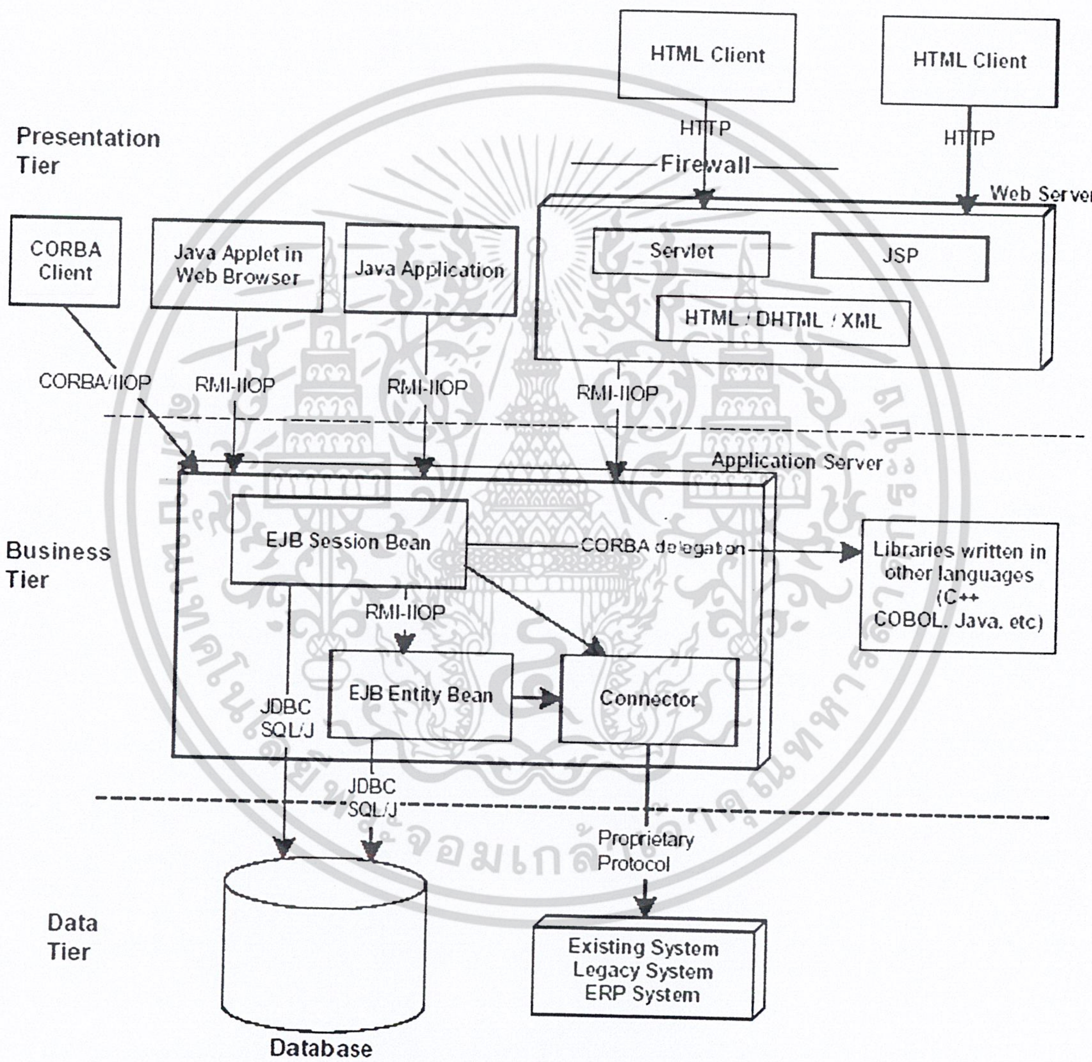
```
</head>


<body>
<%
 int length=20000;
 int countprime=0;
 for (int i=2;i<length;i++)
 {
  int half=i/2;
  boolean prime=true;
  for (int j=2;j<half;j++)
  {
   if ((i%j==0)&&(i!=j))
   {
    // not prime
    prime=false;
    break;
   }
  }
  if (prime)
  {
   countprime++;
  }
 }
 System.out.println("prime");
%>
<h2> <%= countprime %> </h2>
</body>
</html>
```

# ภาคผนวก จ

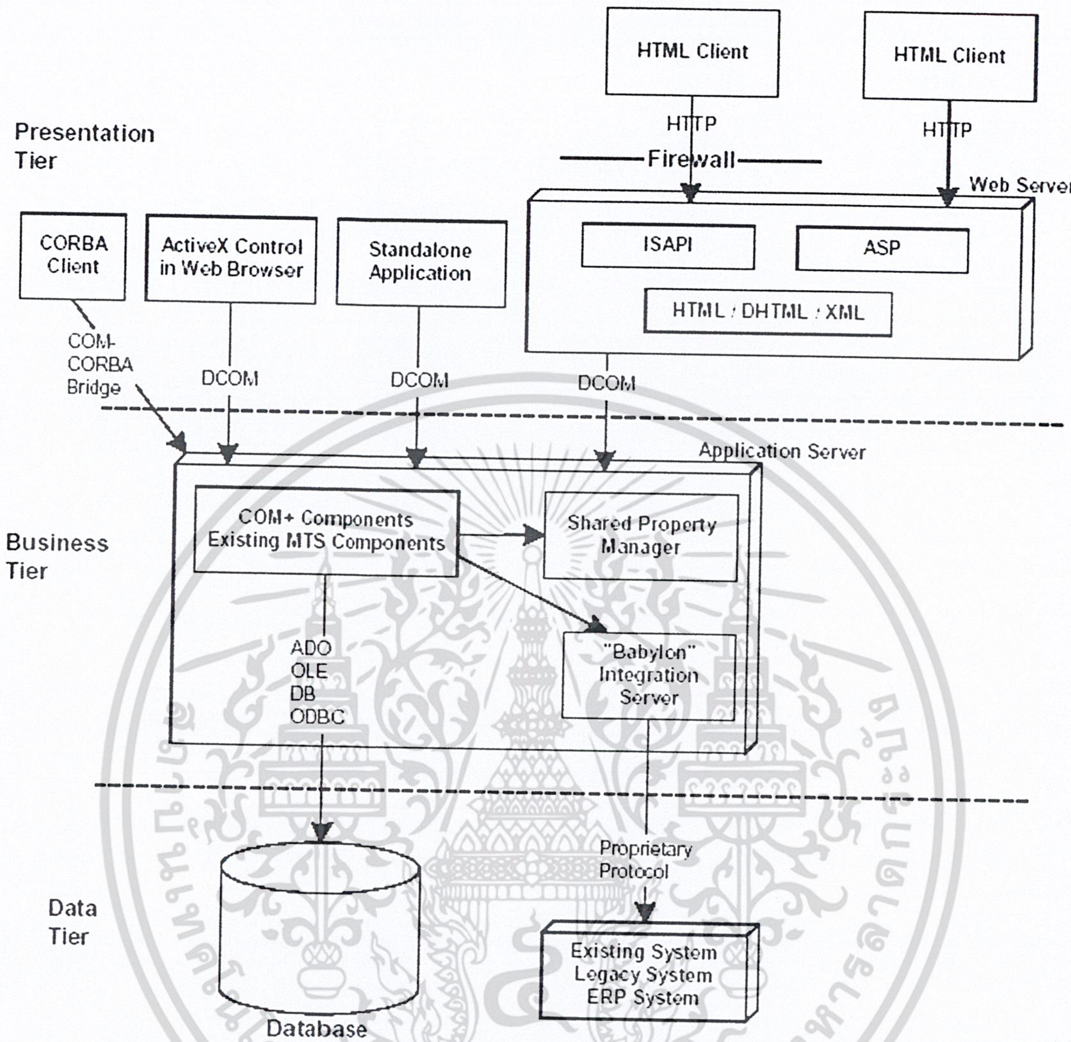
## เปรียบเทียบระหว่าง EJB และ COM+

สถาปัตยกรรมของ J2EE และ Window DNA มีโครงสร้างที่คล้ายกันมาก โครงสร้างของ J2EE เป็นดังรูปที่ จ-1




*รูปที่ จ-1 โครงสร้างของ J2EE*

ส่วนโครงสร้างของ Window DNA ดังรูปที่ จ-2




*รูปที่ จ-2 โครงสร้างของ Window DNA*

ใน J2EE มี EJB เป็นเทคโนโลยีคอมโพเน็นต์บนฝั่งเซิร์ฟเวอร์ ส่วน Window DNA มี COM+ โดยจะเปรียบเทียบทั้ง 2 เทคโนโลยีในประเด็นต่าง ๆ ดังนี้

- **ภาษาที่ใช้ในการพัฒนา**

  COM+ สามารถพัฒนาได้จากหลายภาษา แต่ EJB จะต้องพัฒนาด้วยภาษาจาวาเท่านั้น

- **แพล็ตฟอร์มที่รองรับ**

  COM+ รองรับแพล็ตฟอร์ม window 2000 เท่านั้น แต่ EJB ซึ่งเขียนด้วยภาษาจาวาจะรองรับหลายแพล็ตฟอร์ม นอกจากนั้น COM+ ยังรองรับเซิร์ฟเวอร์ที่มี CPU แค่ 32 ตัวเท่านั้น แต่ EJB สามารถรองรับได้มากกว่านั้น

- **ชนิดของคอมโพเน็นต์ที่รองรับ**

COM+ ไม่รองรับแนวคิดของ entity bean ซึ่งเป็นคอมโพเน็นต์ของข้อมูล แต่จะใช้วิธีติดต่อกับฐานข้อมูลโดยตรง ทำให้การเปลี่ยนแปลงฐานข้อมูลมีผลกระทบต่อคอมโพเน็นต์ที่ทำหน้าที่จัดการข้อมูล

- **Stateless และ Stateful**

COM+ ไม่รองรับ Stateful ทำให้ในกรณีที่ต้องมีการเก็บข้อมูลของไคลเอ็นต์ต้องเก็บลงดาต้าเบสแทน ซึ่งเสียเวลากว่าการเก็บข้อมูลเหล่านั้นไว้ในเมโมรี เช่นในรถเข็นแบบออนไลน์ โดยมีอีกวิธีหนึ่งในการจัดการกับข้อมูลนั้นคือเก็บเป็นคุกกี้ ทำให้ overhead ของเน็ตเวิร์กสูง ถึงแม้ stateful จะดูมีประสิทธิภาพมากกว่า แต่ถ้าเซิร์ฟเวอร์ที่เก็บข้อมูลเกิดขัดข้องขึ้นก็จะทำให้ข้อมูลหายไป จึงเหมาะกับแอพพลิเคชันที่ไม่ต้องการความแน่นอนของข้อมูลมากนัก

- **การจัดการข้อมูลที่ persistent**

กับ container-managed persistent จะช่วยลดการเขียนโค้ดได้มาก โดยการใช้การกำหนดค่าแทน ส่วน COM+ ต้องโค้ดจัดการกับข้อมูลเองทั้งหมดและยังยากต่อการแก้ไขกว่า container-managed persistent ที่เก็บค่าที่กำหนดไว้ใน deployment descriptor แต่ก็มีข้อเสียที่ต้องการการเรียนรู้ในส่วนของการแมปค่าใน descriptor เนื่องจากแต่ละผลิตภัณฑ์มีการจัดการไม่เหมือนกัน จึงไม่ portability ทั้งหมดและต้องการการเรียนรู้ในทุก ๆ ผลิตภัณฑ์

- **Transaction**

EJB ไม่รองรับ nested transaction ซึ่งจำเป็นในแอพพลิเคชันบางประเภท ทั้ง 2 เทคโนโลยีรองรับการประกาศทรานแซ็กชันเหมือนกัน แต่ COM+ จะประกาศทรานแซ็กชันในระดับอินเทอร์เฟซ ส่วน EJB จะประกาศทรานแซ็กชันได้ในระดับเมธอด

- **ประสิทธิภาพ**

ภาษาจาวาจะช้ากว่าประมาณ 3-10 เท่า

# ภาคผนวก ฉ
# เปรียบเทียบการคลัสเตอร์

การเปรียบเทียบจะแบ่งเป็น 2 ส่วนคือเปรียบเทียบพรีเซนเทชันเทียร์และออบเจ็กต์เทียร์

## 1. เปรียบเทียบพรีเซนเทชันเทียร์

COM+ จะใช้ Network Load-Balancing ส่วน EJB ไม่มีการกำหนดวิธีการคลัสเตอร์ไว้ วิธีการคลัสเตอร์จึงขึ้นอยู่กับผู้ผลิตแอพพลิเคชันเซิร์ฟเวอร์ โดยในส่วนนี้จะเปรียบเทียบโดยใช้ WebLogic เซิร์ฟเวอร์

- **สถาปัตยกรรม**

การกระจายโหลดไปยังพรีเซนเทชันเทียร์ของ NLB จะเป็นแบบ distributed โดยใช้ IP sharing ส่วน WebLogic จะเป็นแบบ centralize ที่รองรับการกระจายโหลดโดยใช้ proxy หรือใช้ฮาร์ดแวร์ ทั้ง 2 สถาปัตยกรรมมีข้อดีและข้อเสียต่างกัน distributed นั้นอาจจะมีปัญหาในกรณีที่เพิ่มเซิร์ฟเวอร์เข้าไปมาก ๆ เพราะมันจะต้องการการติดต่อกันในการวิเคราะห์การกระจายโหลด โดย NLB จะรองรับการคลัสเตอร์ได้สูงสุด 32 เครื่อง ส่วน centralize อาจเกิดปัญหาคอขวดได้ แต่ปัญหาคอขวดสามารถที่จะแก้ได้ด้วยฮาร์ดแวร์ที่มีความสามารถสูง แต่ก็มีราคาแพงด้วย

- **Balance**

NLB จะ balance request WebLogic จะ balance user เนื่องจากมันมีการ replicate state ของ HTTP session ไว้ ทำให้การ balance จะไม่ scale โดยอาจจะมีไคลเอ็นต์จำนวนมากที่ยังทำงานอยู่กับเซิร์ฟเวอร์นั้น

- **Failover**

NLB ไม่รองรับ failover แต่ WebLogic จะรองรับโดยการ replicate state ของไคลเอ็นต์ไว้

- **Single Point Of Failure**

WebLogic จะมี single point of failure ในกรณีที่ใช้ proxy

## 2. เปรียบเทียบออบเจ็กต์เทียร์

COM+ จะใช้ Component Load-Balancing

- **สถาปัตยกรรม**

CLB จะเป็นแบบ centralize WebLogic จะเป็น distributed

- **Balance**

CLB จะ balance โดยกระจายการสร้างออบเจ็กต์ แต่ EJB จะ balance ในระดับเมธอด โดยขึ้นอยู่กับชนิดของ bean

- **Failover**

CLB ไม่สามารถ failover ได้ แต่ WebLogic สามารถ failover ได้ในกรณีที่เป็น stateless session bean ที่เป็น idempotent และจะ replicate state ของ stateful session bean

นอกเหนือจากคุณสมบัติเหล่านี้แล้ว สิ่งที่ต้องพิจารณาในการที่จะเลือกใช้คือ ราคาและลักษณะของแอพพลิเคชันว่าต้องการความถูกต้องมากแค่ไหน ต้องการการขยายมากขนาดไหน เป็นต้น

# บรรณานุกรม


[1] Ed Roman : "*Mastering Enterprise JavaBeans*" , Wiley Computer Publishing , 1999

[2] Ed Roman , Scott Ambler , Tyler Jewell : "*Mastering Enterprise JavaBeans Second Edition*" , Wiley Computer Publishing , 2002

[3] Danny Ayers , Hans Bergsten , Michael Bogovich , Jason Diamond , Matthew Ferris , Marc Fleury , Ari Halberstadt , Paul Houle , Piroz Mohseni , Andrew Patzer , Ron Phillips , Sing Li , Krishna Vedati , Mark Wilcox and Stefan Zeiger : "*Professional Java Server Programming*" , Wrox Press Ltd , 1999

[4] Sun Microsystems : "*The Java™ 2 Enterprise Edition Developer's Guide*" , 2000

[5] Deepak Alur , John Crupi , Dan Malks : "*core J2EE Patterns Best Practices and Design Strategies*" , Prentice Hall PTR , 2001

[6] BEA Systems , Inc : "*Using WebLogic Server Clusters*" , 2001

[7] BEA Systems , Inc : "*Programming WebLogic Enterprise JavaBeans*" , 2001

[8] BEA Systems , Inc : "*BEA WebLogic Server Administration Guide*" , 2001

[9] Patrick Cauldwell, Rajesh Chawla, Vivek Chopra, Gary Damschen, Chris Dix, Tony Hong, Francis Norton, Uche Ogbuji, Glenn Olander, Mark A. Richman, Kristy Saunders, and Zoran Zaev : "*Professional XML Web Services*" , Wrox Press Ltd , 2001

[10] Alan W. Brown : "*Large-Scale , Component-Based Development*" , Prentice Hall PTR , 2000

# แหล่งข้อมูลเพิ่มเติม

[1] http://www.theserverside.com

[2] http://www.bea.com

[3] http://java.sun.com

[4] http://www.ibm.com

[5] http://www.perfectxml.com

[6] http://xml.apache.com

[7] http://www.w3schools.com