การโปรแกรมเชิงวัตถุแบบกระจาย

CORBA-Distributed object environment

โดย

นาย ทศพร เส็งกิ่ง 37013295

นาย วรวิทย์ หมื่นศรี 37013306

นาย วีระพงษ์ แซ่โง้ว 37013310

อาจารย์ที่ปรึกษา

อาจารย์ อภิเนตร อูนากูล

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปริญญานิพนธ์ปีการศึกษา 2539

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การโปรแกรมเชิงวัตถุแบบกระจาย (CORBA-Distributed object environment)

ผู้จัดทำ

1. นายทศพร เส็งกิ่ง เลขประจำตัวนักศึกษา 37013295
2. นายวรวิทย์ หมื่นศรี เลขประจำตัวนักศึกษา 37013306
3. นายวีระพงษ์ แซ่โง้ว เลขประจำตัวนักศึกษา 37013310

.....อาจารย์ที่ปรึกษา

(อาจารย์ อภิเนตร อูนากูล)

การโปรแกรมเชิงวัตถุแบบกระจาย

นายทศพร เส็งกิ่ง
นายวรวิทย์ หมื่นศรี
นายวีระพงษ์ แซ่โง้ว
อาจารย์อภิเนตร อูนากูล อาจารย์ที่ปรึกษา
ปีการศึกษา 2539

บทคัดย่อ

ในวิทยานิพนธ์ฉบับนี้เรียบเรียงขึ้นจาก การศึกษาเทคโลยีของการทำออปเจคแบบกระจาย (Distributed Object Technology) โดยใช้มาตรฐานการสร้างออปเจคของ OMG (Object Management Group) และออปเจคบัสแบบกระจาย CORBA (Common Object Request Broker Architecture) รุ่นที่ 2.0 สภาพแวดล้อมการทำงานในระบบ ออปเจคแบบกระจายมีข้อดีนอกเหนือจากการโปรแกรมมิ่งแบบออปเจคทั่วไปคือ สามารถนำออปเจคแบบกระจายมาช่วยในการรวมระบบ (System integration) ได้ง่ายขึ้น และยังสามารถนำมาใช้พัฒนาระบบแอพพลิเคชั่นขนาดใหญ่ได้เป็นอย่างดี ระบบออปเจคแบบกระจายจะมีภาษากลางที่ใช้ช่วยในการทำให้ออปเจคต่างๆ สามารถอธิบายถึงคุณสมบัติของตัวเองเพื่อเป็นข้อมูลในการทำงานร่วมกับออปเจคอื่นเรียกว่าภาษา IDL (Interface Definition Language) โดยทุกออปเจคที่เป็นสมาชิกในระบบแบบกระจายจะต้องมี IDL บอกถึงคุณสมบัติของออปเจคทุกๆออปเจค ในระบบออปเจคแบบกระจายจะมีแกนหลักที่มีการให้การบริการที่เป็นพื้นฐานเพื่อให้เกิดสภาวะแวดล้อมในการทำงานของออปเจคได้เรียกว่า ORB (Object Request Broker) และจะมีมาตรฐานในการเชื่อมต่อกับ ORB ตัวอื่นโดยใช้โปรโตคอล IIOP เพื่อขยายขอบเขตการทำงานของออปเจคแบบกระจายให้มีขนาดใหญ่ขึ้น ในวิทยานิพนธ์ได้ยกตัวอย่างการพัฒนาแอพพลิเคชั่นโดยเริ่มตั้งแต่ขั้นตอนการวิเคราะห์ออกแบบไปจนถึงการสร้างออกมาเป็นแอพพลิเคชั่นใช้งาน สำหรับแอพพลิเคชั่นที่ยกมาเป็นตัวอย่างในที่นี้คือ ระบบ POS (Point of Service) ของซุปเปอร์มาร์เกตขนาดกลาง ข้อดีที่สำคัญอีกประการของเทคโนโลยีแบบกระจายคือจะช่วยลดต้นทุนในการซ่อมบำรุงซอฟแวร์ เพราะโครงสร้างของซอฟต์แวร์ที่ถูกพัฒนาขึ้นมาแต่ละส่วนเป็นอิสระต่อกัน การปรับปรุงแก้ไขออปเจคจะกระทำภายในออปเจคดังนั้นจึงไม่มีผลกระทบกับออปเจคอื่นในระบบ

CORBA Distrubuted Object Environment

Tossaporn SengKing
Voravit Munsri
Weerapong Saengow
Apinetr Unakul Advisor
1996

Abstract

This thesis is a system study of distributed object technology by using OMG standard with CORBA (Common Object Request Broker Architecture) version 2.0. The environment of the distributed object is better than the primitive object oriented programming (OOP) because it can be used for helping System Integration at the high level and easily used for developing the enterprise application. The distributed object has a simple language called IDL (Interface Definition Language) that objects in the system used for self describing. The core of the distributed object has services that used by objects called ORB (Object Request Broker). The ORB has the protocols IIOP (Internet Inter-ORB Protocol) for spanning objects to other ORBs. At the last part of the thesis, there is a case study of POS (Point of Service). It presents a process of developing software by using the distributed object technology step by step starting from analysis to design of application software. The focus is on the POS system used in a medium-size supermarket. It has been found an advantage of the distributed object technology that helps reduce the cost of software maintenance as each software's structure has been developed independently.

สารบัญ

สารบัญ

สารบัญ

สารบัญรูปภาพ

สารบัญรูปภาพ

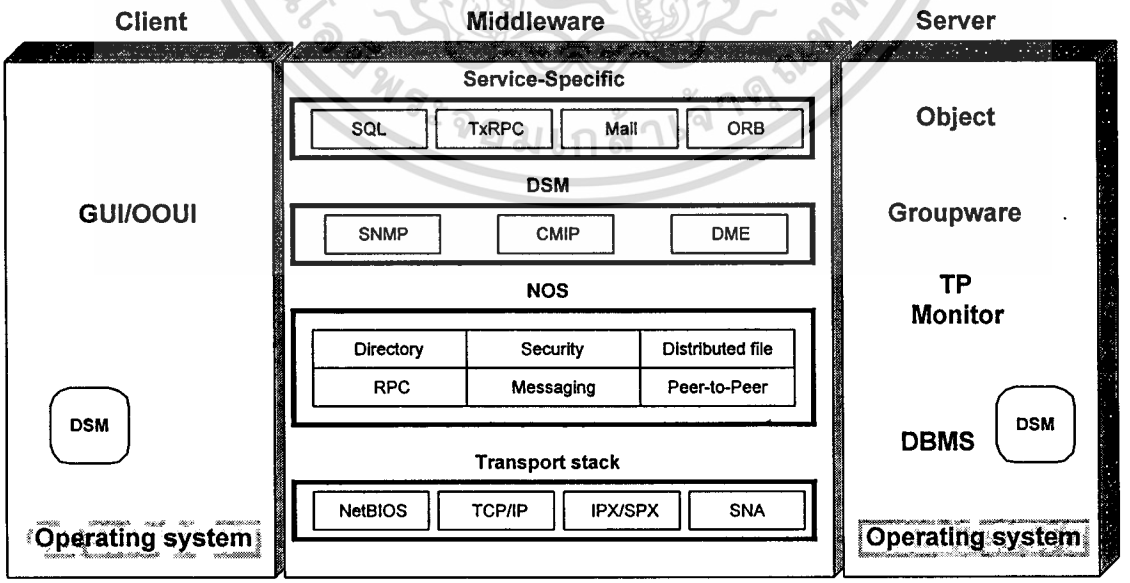
หน้า

บทที่ 1
บทนำ

ระบบเน็ตเวิร์กโดยทั่วไปที่ใช้อยู่ในปัจจุบันเป็น LAN (Local Area Network) แบบอีเทอเน็ตท์ (Ethernet) ที่เชื่อมต่อกับเซิร์ฟเวอร์ (Server) ข้อมูลทั้งหมดก็จะเก็บอยู่ในระบบ LAN เช่นกัน ระบบแบบนี้ให้ความสะดวกสบายเพียงพอต่อการทำงานในระดับหนึ่ง แต่ในปัจจุบันการเติบโตของระบบเน็ตเวิร์ก เป็นไปอย่างรวดเร็วและมีการเชื่อมต่อ ระหว่าง เน็ตเวิร์ก กับ เน็ตเวิร์ก เพื่อทำงานร่วมกัน ติดต่อสื่อสารกัน แอพพลิเคชั่น (Application) ที่ทำงานร่วมกันระหว่างเน็ตเวิร์กต่างๆ ต้องมีมาตรฐานในการทำงานและแลกเปลี่ยนข้อมูลเข้าด้วยกัน

1.1 รูปแบบของไคลเอ็นต์/เซิร์ฟเวอร์ยุคใหม่

ปัจจุบันมีเทคโนโลยีที่สนับสนุนการรวมกันของระบบไคลเอ็นต์/เซิร์ฟเวอร์บน อีเทอเน็ตท์ให้ทำงานร่วมกัน แต่ยังไม่มีเทคโนโลยีตัวใดที่เป็นมาตรฐานกำหนดแน่นอนลงไป ว่าระบบไคลเอ็นต์/เซิร์ฟเวอร์ จะต้องใช้เทคโนโลยีแบบใด เทคโนโลยีที่กล่าวถึงนี้ปัจจุบันแบ่งออกเป็น 4 ประเภทคือ ระบบฐานข้อมูล SQL ,ระบบการจัดการทรานส์แอคชั่น (TP Monitor), ระบบกรุปแวร์(Groupware) และระบบออบเจคแบบกระจาย (Distributed object technology) รูปแบบของไคลเอ็นต์/เซิร์ฟเวอร์ (Client/Server) จะมีลักษณะทั่วไป ดังรูปที่ 1.1



รูปที่ 1.1 รูปแบบทั่วไปของ ไคลเอ็นต์/เซิร์ฟเวอร์

1.1.1 ส่วนประกอบของไคลเอ็นต์/เซิร์ฟเวอร์

ไคลเอ็นต์/เซิร์ฟเวอร์ ประกอบด้วยส่วนสำคัญ 3 ส่วนคือ ไคลเอ็นต์,เซิร์ฟเวอร์,มิดเดิลแวร์ (Middleware) ในส่วนของไคลเอ็นต์จะทำหน้าที่รันแอพพลิเคชั่นของไคลเอ็นต์ เป็นส่วนที่มี GUI(Graphic User Interface) หรือ OOUI (object-oriented user interface) และDSM(distributed system management) ในรูปแบบใดรูปแบบหนึ่ง

ส่วนของมิดเดิลแวร์มันเป็นส่วนที่อยู่ตรงกลางระหว่างเครื่องหมายทับ(/) มันทำงานทั้งในส่วนของไคลเอ็นต์และเซิร์ฟเวอร์ของแอพพลิเคชั่น เราแบ่งมิดเดิลแวร์ออกเป็น 4 ประเภทด้วยกันคือ ทรานส์พอร์ตสแต็ก, NOS (network operating system), DSM และมิดเดิลแวร์สำหรับบริการแบบใดแบบหนึ่งโดยเฉพาะ

NOS กับทรานพอร์ตสแต็กให้พื้นฐานในการสื่อสารทั่วไปแก่มิดเดิลแวร์ทั้งหมด ส่วน DSM จะทำงานบนทุกๆ โหนดของเน็ตเวิร์กแบบไคลเอ็นต์/เซิร์ฟเวอร์ DSMต้องมีมิดเดิลแวร์เป็นของตัวเองและทำงานบน NOS เพื่อที่จะทำการแลกเปลี่ยนข้อความระหว่างสถานีต่างๆ ส่วนมิดเดิลแวร์สำหรับบริการแบบใดแบบหนึ่งโดยเฉพาะ จะขึ้นอยู่กับรูปแบบของแอพพลิเคชั่น แอพพลิเคชั่นฐานข้อมูลใช้มิดเดิลแวร์ในรูปแบบของ SQL พร้อมกับ ODBC (Open Database Connectivity), DRDA (Distributed Relational Database Architecture), RDA (Remote Database Access), Oracle Glue และ CLI (call-level interface) ของ X/Open

ระบบจัดการทรานส์แอคชั่นใช้รูปแบบของมิดเดิลแวร์ประเภทเพียร์-ทู-เพียร์ (peer-to-peer) หรือ RPC (remote procedure call) ส่วนแอพพลิเคชั่นบนกรุปแวร์ส่วนใหญ่จะใช้อีเมล์(E-mail) ส่วนออปเจคแบบกระจายจะใช้ ORB (Object Request Broker) เป็นแกนหลักในการทำงาน

ในส่วนของเซิร์ฟเวอร์ ทำหน้าที่รันแอพพลิเคชั่นที่จัดการกับทรัพยากรต่างๆ ที่ใช้ร่วมกัน และเป็นส่วนที่แสดงให้เห็นถึงรูปแบบของแอพพลิเคชั่นทั้ง 4 รูปแบบในลักษณะการทำงานของไคลเอ็นต์/เซิร์ฟเวอร์ นั่นคือ ฐานข้อมูล SQL, ระบบจัดการทรานส์แอคชั่น,กรุปแวร์ และ ออปเจคแบบกระจาย ในด้านของเซิร์ฟเวอร์ก็ยังบรรจุส่วนของ DSM เอาไว้ด้วย

ส่วนประกอบทั้งสามส่วนนี้ อาจจะทำงานอยู่บนคอมพิวเตอร์เครื่องเดียวกันเลยก็ได้เพราะโหนดใดๆ สามารถจะเป็นได้ทั้งไคลเอนต์และเซิร์ฟเวอร์ การโต้ตอบระหว่างเซิร์ฟเวอร์กับเซิร์ฟเวอร์ ส่วนใหญ่อยู่ในรูปแบบของไคลเอนต์/เซิร์ฟเวอร์ เซิร์ฟเวอร์จะเป็นไคลเอนต์ของเซิร์ฟเวอร์ตัวอื่นๆ อย่างไรก็ตามการโต้ตอบระหว่างเซิร์ฟเวอร์กับเซิร์ฟเวอร์ ต้องมีมิดเดิลแวร์พิเศษสำหรับเซิร์ฟเวอร์โดยเฉพาะ

1.1.2 ออปเจคแบบกระจาย

ในหัวข้อของวิทยานิพนธ์ฉบับนี้จะทำการศึกษาระบบไคลเอ็นต์/เซิร์ฟเวอร์ที่ใช้เทคโนโลยีของออปเจคแบบกระจายและทำการพัฒนาโปรแกรมตัวอย่างที่จำลองการทำงานของระบบขายปลีก(Point of Service) หรือ POS โดยแสดงขั้นตอนของการพัฒนาเริ่มจากการวิเคราะห์และออกแบบ การเขียน IDL (Interface Definition Language) และการสร้างออปเจคบนไคลเอ็นต์/เซิร์ฟเวอร์ ส่วนมาตรฐานที่ใช้ในการสร้างออปเจค

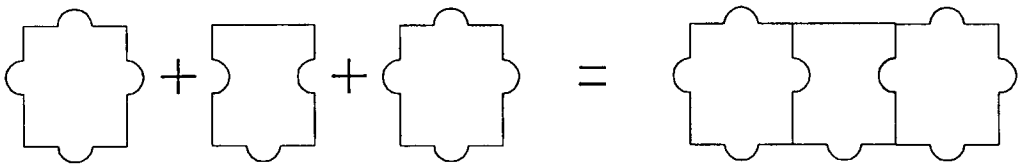
แบบกระจายจะใช้ CORBA(Common Object Request Broker) เวอร์ชั่น 2.0 ซึ่งออกเป็นมาตรฐานที่กำหนดโดย OMG(Object Management Group)

การใช้เทคโนโลยีออปเจคแบบกระจายจะเป็นวิธีสะดวกที่สุดในการทำไคลเอนต์/เซิร์ฟเวอร์ แอพพลิเคชั่นเพราะว่าสามารถปกปิดข้อมูล(encapsulatedata) และขั้นตอนวิธีการทำงาน(business logic)ไว้ภายในออปเจคและ สามารถกระจายไปยังเครือข่ายของคอมพิวเตอร์ที่มีสภาพแวดล้อมการทำงานแตกต่างกัน (platfrom)และ มีความสามารถในการทำงานร่วมกับแอพพลิเคชั่นของเดิมได้

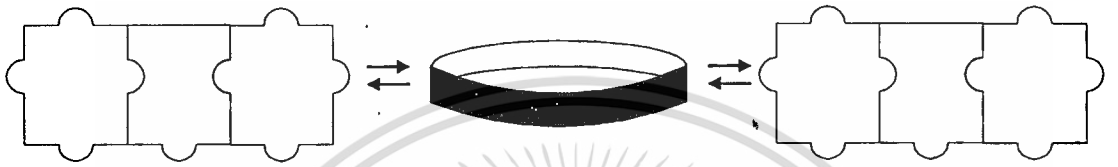
OMG คือองค์การอิสระที่ทำหน้าที่กำหนด อินเตอร์เฟส(Interface)ระหว่างคอมโพเนนท์ (component)ต่างๆและ ออปเจคบัส(Object Bus) OMG ไม่ได้กำหนดว่าจะทำการสร้างอย่างไรแต่ได้กำหนดรูปแบบการเขียน IDLซึ่งไม่เกี่ยวข้องกับภาษาโปรแกรมมิ่ง IDLจะเป็นตัวอธิบายว่าในคอมโพเนนท์มี การให้บริการอะไรอยู่บ้างรวมทั้งเมธอด(method),ตัวแปรของคอมโพเนนท์,การจัดการเกี่ยวกับข้อผิดพลาดที่เกิดขึ้น (error handles) และความสัมพันธ์ในการสืบทอด(Inheritance relationship)กับคอมโพเนนท์อื่น

IDLจะเป็นตัวเชื่อมไคลเอนต์/เซิร์ฟเวอร์ให้ทำงานเข้าด้วยกันข้อดีของIDL คือง่ายในการที่จะใช้ครอบแอพพลิเคชั่นของเดิมโดยเราไม่ต้องเขียน แอพพลิเคชั่นขึ้นมาใหม่ซึ่งสิ่งนี้เป็นข้อดีของเทคโนโลยีออปเจคแบบกระจายดังรูปที่ 1.2 ในออปเจคบัสจะมีORBซึ่งทำให้ไคลเอนต์สามารถเรียกเมธอดบนรีโมทออปเจค (remote object) ในแบบสแตติก(statically) หรือไดนามิค(Dynamically) โดยถ้าการเชื่อมต่อของคอมโพเนนท์มีอยู่แล้วเราสามารถเชื่อมโปรแกรมของเรากับ IDL สตับ(Stub) เพื่อที่จะเรียกใช้งานเมธอดที่ต้องการ แต่ถ้าไม่มีอินเตอร์เฟสที่ต้องการอยู่ไคลเอนต์สามารถที่จะสร้างอินเตอร์เฟสได้ในขณะช่วงเวลารันไทม์(Runtime) โดยการทำงานร่วมกันกับที่ใส่อินเตอร์เฟส(Interface Repository)ตามข้อกำหนดของ OMG

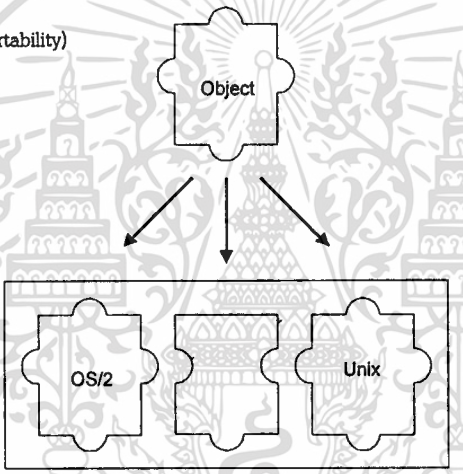
1. การทำงานแบบตัวต่อ (Pug and Play)



2. การทำงานร่วมกัน(Interoperability)

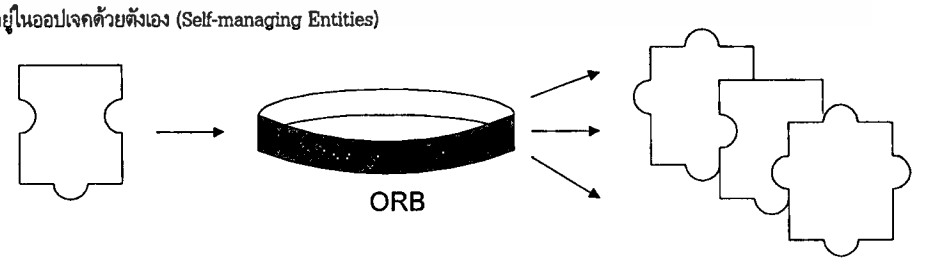


3. ทำงานต่างสภาวะแวดล้อม (Portability)



4. ทำงานร่วมกับซอฟแวร์ที่มีอยู่ (Coexistence)

5. จัดการสิ่งที่อยู่ในออปเจคด้วยตัวเอง (Self-managing Entities)



รูปที่ 1.2 แสดงข้อดีของการใช้เทคโนโลยีออปเจคแบบกระจาย

บทที่ 2
ทฤษฎีการทำงานของระบบออปเจคแบบกระจาย

O M G ได้กำหนดลักษณะของบัสทั่วไป(Global Bus) สำหรับการกระจายคอมโพเนนท์เรียกว่า CORBA และได้กำหนด รายละเอียดต่างๆ ของ IDL (Interface Definition Language) ซึ่งเป็นตัวกำหนดขอบเขตว่าจะอินเตอร์เฟสกันอย่างไร คอมโพเนนท์ที่เขียนไว้ใน IDL ควรที่จะสามารถทำการแมปปิ้ง (mapping)ข้ามภาษา(Language), ทูล(tools), โอเอส(OS) และเน็ตเวิร์ก ส่วนบริษัทMicrosoft ได้สร้าง COM (Common Object Model หรือ Network OLE) มาอีกหนึ่งโมเดลเพื่อจะนำออกมาให้เป็นมาตรฐานในการทำการกระจายออปเจคเช่นกัน

ผลิตภัณฑ์ของ CORBA มีออกมาแล้วในปัจจุบันและได้ใช้กันในวงการอุตสาหกรรมซอฟต์แวร์แล้ว เช่น

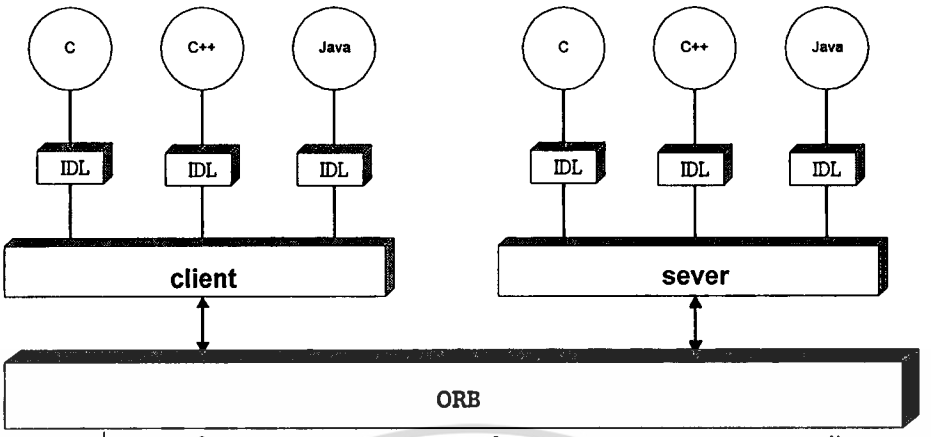
SOM	ของบริษัท IBM
ORB Plus	ของบริษัท HP
Object Broker	ของบริษัท Digital
Orbix	ของบริษัท Iona
COM	ของบริษัท Microsoft
CORBAPlus	ของบริษัท Exper Soft

2.1 ออปเจคแบบกระจายของในระบบ CORBA

CORBA ออปเจค คือ ชิ้นส่วนของแพคเกจซอฟต์แวร์ที่กระจายไปบนเครือข่ายในลักษณะของไบนารี่ คอมโพแนนท์ที่สามารถทำให้รีโมท์ไคลแอนท์สามารถเรียกใช้งานเมธอดต่างๆได้ ภาษา และ คอมไพเลอร์ ที่ใช้สร้าง เซิร์ฟเวอร์ออปเจค(Server Object) จะเป็นลักษณะทรานแพรเร้นท์(transparent) กับ ไคลเอนต์ โดยไคลเอนต์ไม่ต้องรู้ว่าออปเจคแบบกระจาย(Distributed Object) อยู่ที่ไหนหรือ ทำงานบนโอเอสใด สิ่งที่ไคลเอนต์ต้องการรู้คือ อินเตอร์เฟสกับเซอร์วิสของออปเจคเซิร์ฟเวอร์(Server Object Publish) ข้อมูลทุกอย่างที่เป็นข่าวสารเกี่ยวกับการทำออปเจคแบบกระจายจะอยู่ใน IDL

OMG IDL เป็นการประกาศ(Declarative)ของค่าต่างๆที่ต้องใช้เฉยๆ ไม่มีส่วนของเครื่องมือที่ช่วยในการทำงาน(tool) เราสามารถใช้ IDL เพื่อที่จะกำหนด API (Application Program Interface) อย่างย่อๆ และ ควบคุมข้อผิดพลาดต่างๆ (error handling)

การใช้ภาษา IDL เป็นตัวกำหนด จะทำให้ OS และ ภาษาโปรแกรมมิ่ง(Program Language) เกิดอินเตอร์เฟสกันได้ไม่ขึ้นกับ OS ใดๆ หรือ ภาษาใดภาษาหนึ่ง (independent interface)กับบริการ(Service) ทั้งหมดของออปเจค และ คอมโพเนนท์ต่างๆ ที่อยู่บน CORBA bus



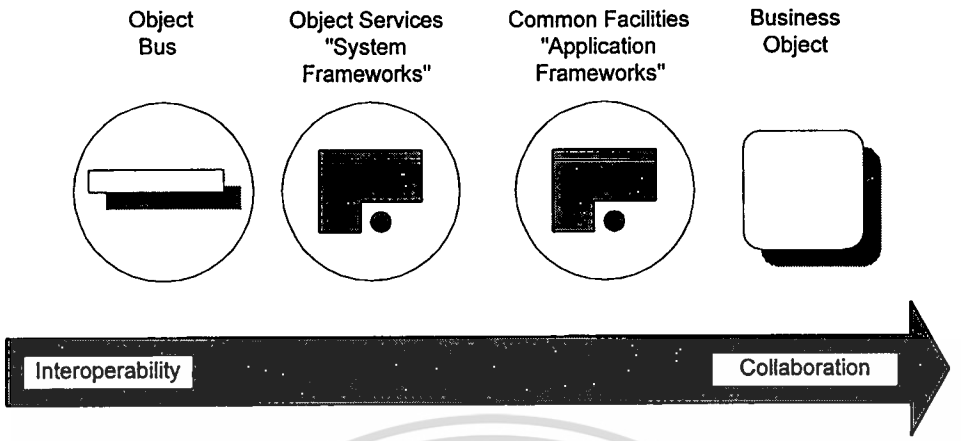
รูปที่ 2.1 การใช้ CORBA IDL สามารถทำให้ Client/Server ทำงานร่วมกันได้

เราสามารถใช้ IDL กำหนดสิ่งต่างๆในออปเจคดังต่อไปนี้

- คุณสมบัติของ คอมโพเนนท์ (Component's attribute) กับ คลาส(Class) ที่เป็นบรรพบุรุษของออปเจค
- การจัดการกับข้อผิดพลาดที่เกิดขึ้นในขณะการทำงาน(Exception)
- ชนิดของ เหตุการณ์ (event) จะเกิดขึ้นระหว่างการทำงาน
- เมธอดที่อินเตอร์เฟสสนับสนุน(Support) รวมทั้ง อินพุท(Input),เอ้าพุท(Output), พารามิเตอร์ (Parameter) และ ชนิดของข้อมูล(data type) ที่อินเตอร์เฟสเหล่านั้นใช้งาน

2.2 ขั้นตอนการแปลงจาก CORBA คอมโพเนนท์ ไปเป็นออปเจคในชีวิตประจำวัน(Business Object)

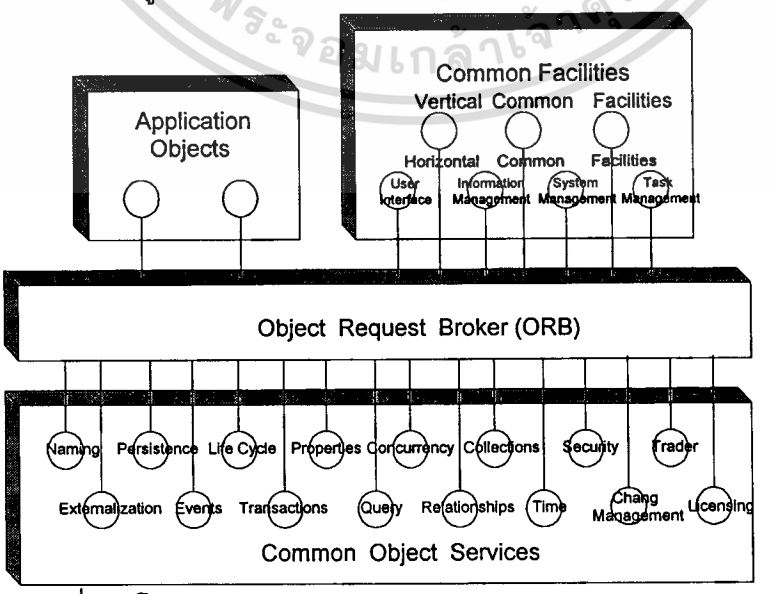
CORBA ออปเจค คือ คอมโพเนนท์ เพราะว่ามันคือกลุ่มของออปเจคแบบกระจาย(Package Distributed Object) ที่เป็นส่วนหนึ่ง(Unit) ของงาน และ สามารถกระจายออกเป็นคอมโพเนนท์ต่างๆ CORBA Distributed Object เป็นโครงสร้างพื้นฐานที่จะทำให้ง่ายสำหรับทำให้คอมโพเนนท์มีความเป็นอิสระ มีการจัดการจัดการข้อมูลในตัวเอง(Selfmemaging) และความสามารถในการทำงานร่วมกันกับออปเจคอื่นๆ เทคโนโลยีการโปรแกรม CORBA แบบกระจาย(CORBA's Distributed Technology) ทำให้เราเชื่อมข้อมูลที่ซับซ้อนระหว่างไคลเอ็นและเซิร์ฟเวอร์(complex Client/Server information) เข้าด้วยกันโดยง่าย และให้คอมโพเนนท์ ออปเจค(Component Object) สามารถแก้ไข(modified) โดยปราศจากการกระทบกับออปเจคอื่นๆ ภาพของไคลเอ็น/เซิร์ฟเวอร์แอพพลิเคชั่น (Client/Server Application) จะมาเป็นรูปแบบของการทำงานร่วมกันระหว่างคอมโพเนนท์ ต่างๆ ในขั้นตอนการแปลงจากออปเจคที่พบอยู่ในชีวิตประจำวันไปเป็นรูปแบบของคอมโพแนนท์ต่างๆแสดงดังรูป 2.2



รูปที่ 2.2 ขั้นตอนการแปลงจาก CORBA คอมโพเนนท์ ไปเป็นออปเจคในชีวิตประจำวัน

ขั้นตอนที่ระดับออปเจคบัสคือ ORB จะรับผิดชอบหน้าที่ในการทำให้ออปเจคสามารถทำการขอใช้บริการโดยไม่ต้องคำนึงถึงว่าจะได้รับการบริการจากที่ใด และออปเจคที่ต้องการอยู่ที่ใด(transparently) ทางฝั่งไคลเอ็นต์ไม่ต้องรู้ถึงขั้นตอนที่ใช้ในการติดต่อสื่อสารว่ากระทำอย่างไร ทำหน้าที่เพียงขอใช้บริการและรอรับการให้บริการเท่านั้น

ขั้นตอนถัดมาคือส่วนที่เป็นการให้บริการขั้นพื้นฐานของออปเจค(object service) เช่น การให้บริการชื่อ(naming) , การเก็บรักษาออปเจค(persistence) , การสร้างและทำลาย (life cycle) ,ตรวจจับเหตุการณ์(events) , ทรานแอคชั่น(transactions) ,การใช้ทรัพยากรร่วมกัน(concurrency), ระบบรักษาความปลอดภัย(security) และในขั้นตอนสุดท้ายคือการให้บริการการทำงานร่วมกันของออปเจคในแนวนอนและแนวตั้ง ในระดับของแอพพลิเคชั่นเฟรมเวอร์ค(application-level framework) สิ่งที่ได้จากขั้นตอนสุดท้ายคือเป็น ออปเจคในชีวิตประจำวันที่ถูกสร้างในรูปแบบของแอฟริเคชั่นเราสามารถแสดงลักษณะการจัดการระบบออปเจคใน CORBA 2.0 เป็นลักษณะดังรูปที่ 2.3

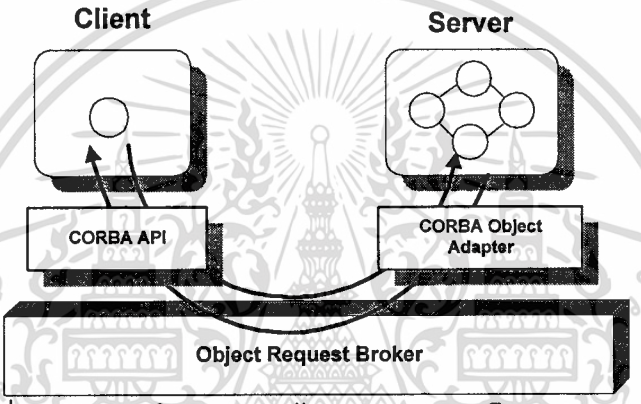


รูปที่ 2.3 โครงสร้างการจัดการกับออปเจคแบบกระจายของ CORBA 2.0

2.3 โครงสร้างของ CORBA 2.0 ORB และการทำงานของ ORB

การทำงานระหว่างไคลเอ็นต์/เซิร์ฟเวอร์บน ORB

CORBA 2.0 Object Request Broker (ORB) คือมิดเดิลแวร์ ที่ทำให้เกิดความสัมพันธ์(Relationship) ระหว่างไคลเอ็นต์/เซิร์ฟเวอร์ และออปเจคต่างๆ ORB จะทำให้ไคลเอ็น ออปเจค (Client Object) สามารถทำการเรียกขอการใช้งานโดยไม่คำนึงถึงสถานที่(transparently invoke) กับหนึ่งเมธอดบนหนึ่งเซิร์ฟเวอร์ ออปเจคใดๆ ที่อยู่บนเครื่องเดียวกัน หรือข้ามเน็ตเวิร์ก ORB มีหน้าที่รับและ รับผิดชอบในการค้นหาออปเจค ซึ่งสามารถทำงานได้ตามคำขอร้อง การผ่านพารามิเตอร์(Pass Parameter) การเรียกเมธอด และการรีเทิน (Return) ผลลัพธ์

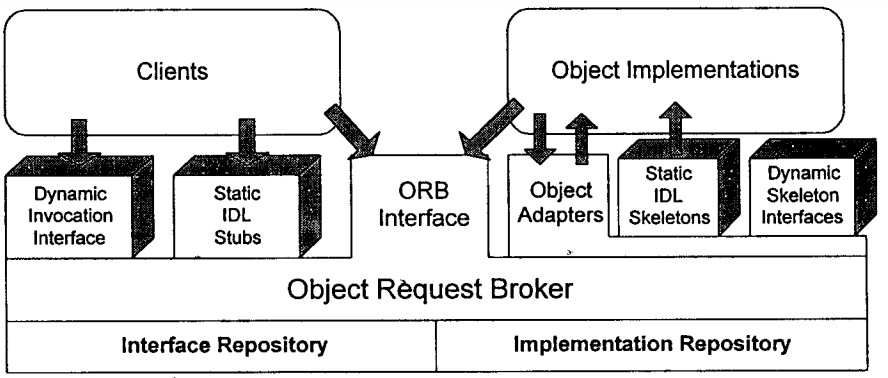


รูปที่ 2.4 การเรียกใช้งานระหว่างไคลเอ็น/เซิร์ฟเวอร์โดยผ่าน ORB

ไคลเอ็นต์ไม่ต้องสนใจว่าออปเจคอยู่ที่ไหน ลักษณะภาษาโปรแกรมมิ่งของไคลเอ็นต์และระบบปฏิบัติการ ของทั้งทางฝั่งไคลเอ็นต์/เซิร์ฟเวอร์ ORB จะมีหน้าที่ใช้ทำควบคุมความสอดคล้องกัน(Coordinate)ในการทำงานระหว่างสองออปเจค ออปเจคบน ORB สามารถเป็นได้ทั้งไคลเอ็นต์ หรือ เซิร์ฟเวอร์ ขึ้นอยู่กับว่าตอนนี้มันใช้งานเป็นอะไร

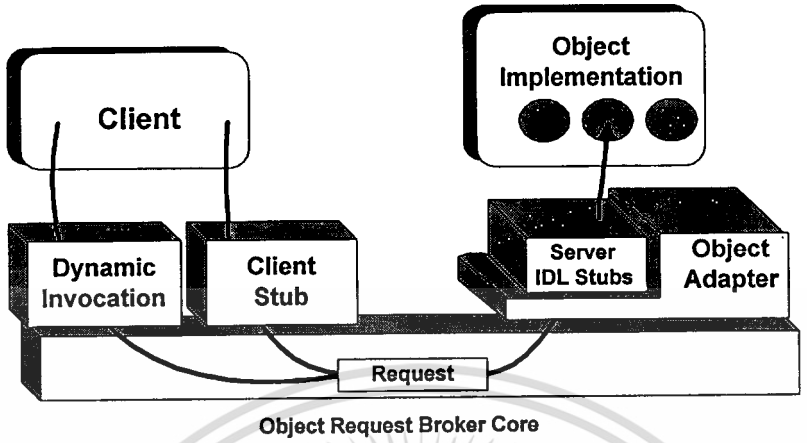
2.3.1 โครงสร้างโดยละเอียดของ CORBA 2.0 ORB

OMG ได้รับเอาข้อดีของ HyperDesk และ Digital มาเป็น Dynamic API และได้เอาข้อดีของ Sun และ HP มาเป็นแอฟริเคชั่นโปรแกรมมิ่งอินเตอร์เฟสแบบสแตติก (static APIs)



รูปที่ 2.5 โครงสร้างโดยละเอียดของ CORBA 2.0 ORB

2.3.2 การเรียกใช้งานเมธอดของ CORBA (CORBA method Invocation)



รูปที่ 2.6 การเรียกใช้งานแบบสแตติกและไดนามิกของ CORBA

ภาพที่ 2.6 แสดงถึง 2 ลักษณะการทำงานของไคลเอ็นต์/เซิร์ฟเวอร์ ซึ่ง CORBA ORB ได้สนับสนุน คือ สแตติก(Static) และ ไดนามิกส์ไคลเอ็นต์(Dynamic Client) กระทำโดยร้องขอ(Request)เข้า(accress) ไปยังออปเจค เรฟเฟอะเร้นท์(Object reference หรือ ออปเจค ID) และทำการเรียกเมธอดที่กระทำบริการ

ไคลเอ็นจะเห็นการติดต่อระหว่างออปเจค(Object interfaces) ที่เป็นเหมือนลักษณะของภาษาแมปปิ้ง (language mapping) หรือบินดิ้ง (binding) ซึ่งจะทำในระดับของการโปรแกรม (Program level) ที่ไคลเอ็นต์โปรแกรมสามารถที่จะทำงานโดยปราศจากการเปลี่ยนแปลงใดๆ บน ORB ซึ่งจะมีการทำงานที่สนับสนุนการทำบินดิ้ง(language binding)และการเรียกใช้งานออปเจคใดๆ จะทำผ่านทางอินเตอร์เฟสที่ถูกสร้างขึ้นเท่านั้น

สแตติก อินเตอร์เฟส (Static Interface) คือ การสร้างอินเตอร์เฟสในรูปแบบของสตับ (stub) โดย IDL คอมไพล์เลอร์ (compiler) เป็นตัวสร้างสตับใหม่การที่โปรแกรมรู้จักอินเตอร์เฟสขณะเวลาทำการคอมไพล์ (Compile time) และสแตติกอินเตอร์เฟสในลักษณะสตับ(Static stub interface) จะกำหนดขอบเขตการทำงานขอบเขตการทำงานของออปเจคที่เวลาคอมไพล์เช่นกัน

ในทางกลับกันไดนามิกส์เมธอดอินโวเคชั่น (dynamic method invocation) จะมีความยืดหยุ่นในการใช้งานมากกว่าซึ่งจะยอมให้เรา เพิ่มเติมคลาส(classes) ใหม่ๆ เข้าไปในระบบโดยไม่ต้องเปลี่ยนไคลเอ็นโค้ด(Client Code) แต่โดยส่วนมากแล้ว แอพพลิเคชั่น ไม่ต้องการความยืดหยุ่นมาก(flexible)โดยส่วนมากจะใช้งานกับแบบคงที่(static) มากกว่า

ข้อดีของสแตติก(static) ที่เหนือกว่าไดนามิกส์(dynamic) คือ

1. ง่ายต่อการโปรแกรม
2. ถูกตรวจสอบโดย คอมไพล์ตั้งแต่เวลาคอมไพล์ (Build time) แล้ว
3. สามารถตรวจสอบการทำงานโดยการไล่คำสั่ง(Code)

2.3.3 การทำงานทางด้านเซิร์ฟเวอร์ของ CORBA (The Server side of CORBA)

ในการสร้างออปเจคหนึ่งจาก ORB ทางด้านเซิร์ฟเวอร์นั้นจะต้องมีโครงสร้างพื้นฐานในบริการที่จะทำการลงทะเบียนชนิดของแอพพลิเคชั่น (Registed Application Classes)

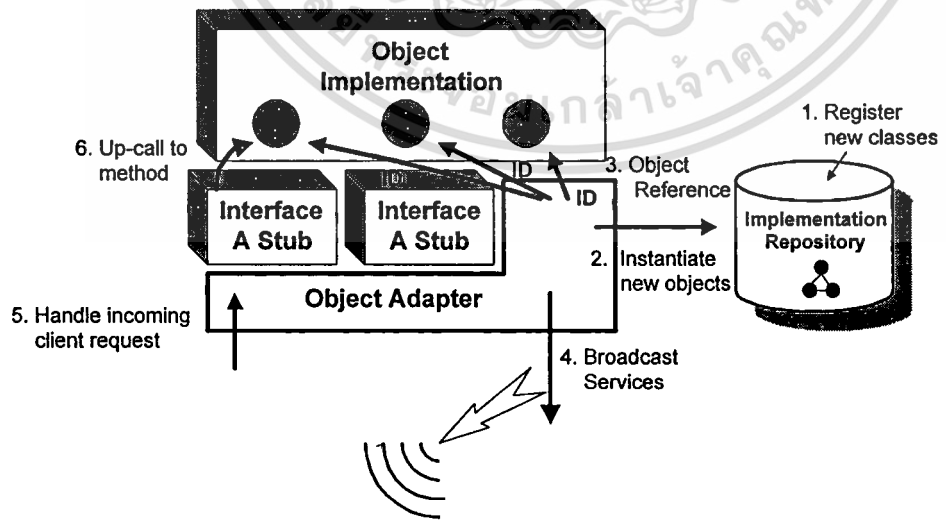
ขั้นตอนการทำงานทางด้านเซิร์ฟเวอร์มีดังนี้

1. ออปเจคที่เกิดขึ้นทันทีทันใด โดยทำการสร้างออปเจคจากการเรียกใช้บริการ ORB
2. ให้ ID กับออปเจคนั้นๆ
3. บอกว่ามันมีออปเจคอะไรอยู่บ้าง โดยทำการลงทะเบียนโดยใช้เนมเซอร์วิส(name service) ของ CORBA
4. ให้บริการหรือเมธอดต่างๆ เมื่อไคลเอ็นต์ต้องการ (ทางฝั่งไคลเอ็นต์จะเป็นผู้ทำการขอมา)
5. จัดการกับพวกการทำงานพร้อมๆกันไป(Concurrent) สำหรับบริการต่างๆ ภายในเซิร์ฟเวอร์

นอกจากนี้ยังมีส่วนของการจัดการทรานแอคชั่น(transaction management), การบริหารโหลด (load balance) และการทำระบบรักษาความปลอดภัยด้วย เพราะว่าเราต้องการให้โปรแกรมหนึ่ง ทำตามกฎของคลาส ไลบรารี่(Class Libraries)และเปลี่ยนรูปออปเจค(Transform Object)ทั้งหลายไปยังเซิร์ฟเวอร์ต่างๆ (Multiserver) ซึ่งการทำแบบนี้เหมือนกับ ทีพี มอนิเตอร์(TP Monitor) สิ่งที่ทำงานหน้าที่นี้คือออปเจคอแดปเตอร์ (Object Adapter)ของ CORBA

2.3.4 การทำงานของ ออปเจค อแดปเตอร์(Object Adapter หรือ OA)

OA คือกลไกหลักสำหรับการสร้างออปเจค เพื่อจะให้ออปเจคใช้งานบริการต่างๆ ใน ORB ได้



รูปที่ 2.7 โครงสร้างของออปเจคอแดปเตอร์ (Object Adapter)

จากรูปที่ 2.7 แสดงให้เห็นว่า OA ให้เอ็นไวรอนเม้นท์(environment)ทั้งหมดในการทำงานของบริการแอพพลิเคชั่นมีขั้นตอนต่างๆดังนี้

1. ลงทะเบียนคลาสใน อิมพลีเม้นเตชั่น รีโพสิโตรี่(Register sever Class with the Implementation Repository)เราสามารถมองได้ว่า อิมพลีเม้นเตชั่น รีโพสิโตรี่ เป็นที่เก็บข้อมูลไว้ใช้งานตลอดเวลาของตัวจัดการออปเจค อแดปเตอร์ (Object Adapter manager) เมื่อสร้างประเภทของออปเจค (Object Classes) ขึ้นมาก็จะถูกนำมาเก็บใน อิมพลีเม้นเตชั่น รีโพสิโตรี่ และจะถูกลงทะเบียนไว้ (Registered) เพื่อเรียกใช้งาน

2. ขั้นตอนการสร้างออปเจคใหม่ขึ้นมาขณะเวลารันไทม์(Instantiate new objet at runtime) OA จะมีหน้าที่สร้างออปเจค จากอิมพลีเม้นเตชั่น คลาส(implementation class) จำนวนของออปเจคที่ถูกสร้างขึ้นจะขึ้นอยู่กับความคับคั่งจากโหลด(traffic load) ที่เกิดขึ้นจากไคลเอ็นต์ ออปเจคอแดปเตอร์จะมีหน้าที่รับผิดชอบในการรักษาความสมดุลย์ (balance) การให้ออปเจคกับความต้องการของไคล์เอ็นต์ที่เข้ามา

3.สร้างและจัดการเกี่ยวกับออปเจคเรฟเฟอร์เรนท์(Generates and manages object references) OA จะกำหนดการอ้างถึงเรฟเฟอร์เรนท์ไอดี (reference ID) ให้กับ ออปเจคที่ถูกสร้างขึ้นมาใหม่ และมีหน้าที่รับผิดชอบในการหาแมปปิ้งระหว่างออปเจคที่มันสร้างขึ้นมาใหม่ กับออปเจคเรฟเฟอร์เร้นท์ตัวอื่นๆที่กำหนดไว้ก่อนหน้านี้

4. OA จะต้องประกาศเกี่ยวกับเซอร์วิสต่างๆ ของออปเจคที่สร้างขึ้นมาใหม่ให้ ORB ทราบว่ามันมีบริการอะไรอยู่ในตัว

5. รอรับการเรียกการร้องขอบริการจากทางไคลเอ็นต์(Handles incoming client calls) OA จะทำงานอยู่บนชั้นบนของแกนหลักของ ORB เป็นลักษณะของการสื่อสารแบบเข้าก่อนออกทีหลัง (Communication stack) ทำหน้าที่รับการร้องขอ และส่งต่อให้กับอินเตอร์เฟสสตับและ สตับ(stub) มีหน้าที่แปลความหมายของค่าตัวแปรต่างๆ ที่ผ่านเข้ามาแล้วแปลความหมาย จากนั้นจึงส่งผ่านมันขึ้นไปเป็นรูปแบบของการอ้างถึงออปเจคเพื่อเรียกใช้บริการจากออปเจคที่กำหนดไว้ในออปเจคเรฟเฟอร์เรนท์

6. ทำให้เกิดเส้นทางเดินขึ้นเพื่อจับจองเมธอด(Routes the up-call to the appropriate method) ในออปเจค ออปเจคอแดปเตอร์ทำหน้าที่ในการเรียกเมธอดเดสคริปชั่น(method described)ในสตับหรือ สเกล ลิตอน(skeleton) และการทำงานของออปเจค อแดปเตอร์อาจรวมถึงการกระตุ้น การสร้างออปเจค และมันยังสามารถตรวจสอบการอนุญาติการเข้ามาของคำร้องขอ (Request) ต่างๆจากไคลเอ็นต์

2.3.5 มาตรฐานการทำงานของ BOA (Basic Object Adapter)

ในออปเจค อแดปเตอร์หนึ่งๆ กำหนดว่าออปเจคจะถูกกระตุ้นอย่างไร มันสามารถทำโดยการสร้างโปรเซส(Process)ใหม่,สร้างเส้นทางเธด(thread)ใหม่ภายในโปรเซส หรือใช้โปรเซสที่มีอยู่แล้ว เซิร์ฟเวอร์สามารถที่จะสนับสนุนในการใช้งานให้ OA ทำงานตามคำร้องขอหลายๆแบบ เช่นออปเจค ดาต้าเบส (Object Database หรือ ODBMS) อาจต้องการที่จะลงทะเบียนออปเจคในลักษณะที่เป็น ไฟน-เกรน(fine-grain)ทั้งออปเจค โดยไม่ต้องการให้ OA ทำการวิเคราะห์การเรียกทุกๆครั้ง เมื่อเรียกใช้ออปเจคเว้นในกรณีนี้ OA จะไม่ต้องทำหน้าที่คอยดูแลการทำงานของออปเจคในขั้นตอนการทำงานต่างๆ ODBMS อาจจะต้องการใช้งาน

บางประเภทที่ออปเจค อแดปเตอร์ไม่จำเป็นต้องการที่จะเชื่อมต่อกับ ORB การใช้งานในกรณีอื่นๆที่นอกเหนือจากนี้ จะทำให้เกิดความหลากหลายในการใช้งานมากเกินไปดังนั้น OMG ได้กำหนด BOA(Basic Object Adapter) ซึ่งเป็นมาตรฐานในการใช้งานใน ORB ทั่วไปดังนี้

CORBA ต้องการที่จะให้ BOA อแดปเตอร์ มีอยู่บนทุกๆ ORB ออปเจคที่ถูกสร้างขึ้นมานั้นควรที่จะทำงานอยู่บน ORB ใดๆก็ได้และสนับสนุน การทำภาษาผูกผัน (Language bindings) ในมาตรฐานของ CORBA ต้องการฟังก์ชั่นในการทำงานดังนี้ใน BOA

1. มีที่เก็บเครื่องมือมีอิมพีเมนต์ ริโพสิเตอรี (Implement Repository) 1 ตัวเพื่อที่จะทำให้เราติดตั้งและลงทะเบียนออปเจคที่เราสร้างขึ้น และยังให้ข่าวสารที่อธิบายว่าออปเจคนั้นทำอะไร

2. มีกลไกในการสร้าง และการแปลการอ้างถึงออปเจคเรฟเฟอร์เร็นท์(Object Reference), การกระตุ้นการสร้างออปเจคและการหยุดสร้างออปเจค การเรียกและการผ่านค่าพารามิเตอร์ (passing parameter) ไปยังออปเจค

3. กลไกในการขออนุญาติเข้าใช้งานจากตัวไคลเอ็นต์ (authenticating) BOA ไม่ได้กำหนดรูปแบบของการทำระบบรักษาความปลอดภัย แต่ BOA จะทำการรับรองว่าทุกๆออปเจค และเมธอดที่ถูกเรียกจะต้องทำได้อย่างถูกต้อง

4. ทำการเรียกเมธอดโดยผ่านทางสตับ

5.เป็นตัวกระตุ้นการสร้างออปเจคและเป็นตัวบอกให้ออปเจคเลิกทำงาน (Activation and deactivation of implement object)

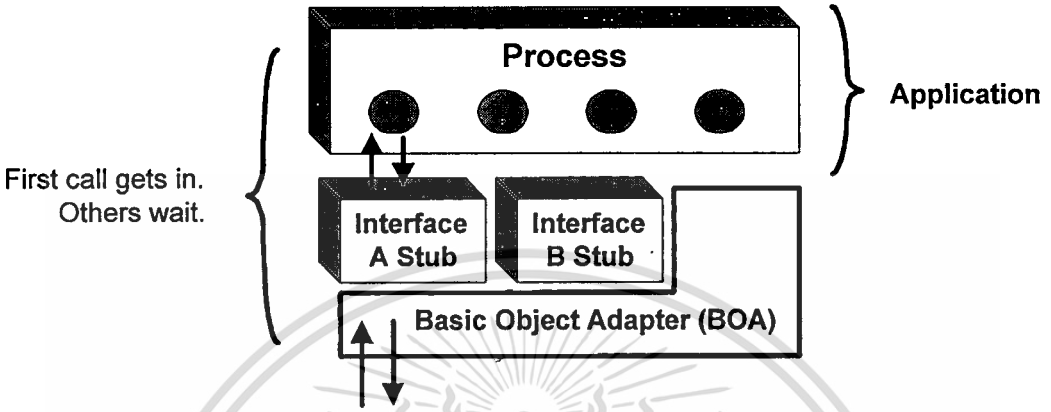
BOA จะสนับสนุนการทำงานของ ออปเจค-ออร์เรนท์เต็ด แอปพลิเคชั่น(Object-Oriented application) แต่จะไม่ระบุลงไปว่าเมธอดจะอยู่ในรูปแบบของแพคเกจ หรือจะอยู่ที่ไหน เพราะสิ่งนี้ควรที่จะทำให้กระจ่างชัดเจนผ่านการเรียกใช้งานอินเตอร์เฟสแบบไดนามิก หรือ การเรียกโดยระบบ(System call) ตั้งแต่ตอนเริ่มทำงานซึ่งจะเป็นตัวกำหนดที่อยู่ของเมธอดว่าจะอยู่ในที่ไหน

BOA จะมีลักษณะการสร้างออปเจคอยู่ 4 ลักษณะด้วยกันคือ แบบมีการใช้ทรัพยากรร่วมกัน(share server), ไม่มีการใช้ทรัพยากรร่วมกัน(unshared server),แบบหนึ่งเมธอดต่อหนึ่งเซิร์ฟเวอร์ (server-per-method) และการทำงานแบบคงที่(persistent server)

2.3.5.1 การทำงานแบบใช้ทรัพยากรร่วมกัน (BOA Share Server)

ในลักษณะของแชร์เซิร์ฟเวอร์จะมีรูปแบบการทำงานดังนี้ ออปเจคต่างๆอาจจะอยู่ในโปรแกรม (เช่น โปรเซส) เดียวกัน BOA จะกระตุ้นในเซิร์ฟเวอร์ ในครั้งแรกที่มีการขอให้สร้างออปเจคขึ้นมา (รูปที่2.8) และหลังจากนั้น เซิร์ฟเวอร์จะกำหนดการทำงานเบื้องต้นโดยตัวมันเอง (initialized) BOA จะทำงานโดยการเรียกซิสเต็มคอล impl_is_ready จากนั้นคำร้องขอจะส่งเข้ามาที่เซิร์ฟเวอร์ โปรเซส (server process) BOA จะไม่กระตุ้น เซิร์ฟเวอร์ โปรเซส(server process) อื่นๆ เพื่อที่จะสร้างออปเจคใหม่ขึ้นมา เซิร์ฟเวอร์จะเลือก คำ

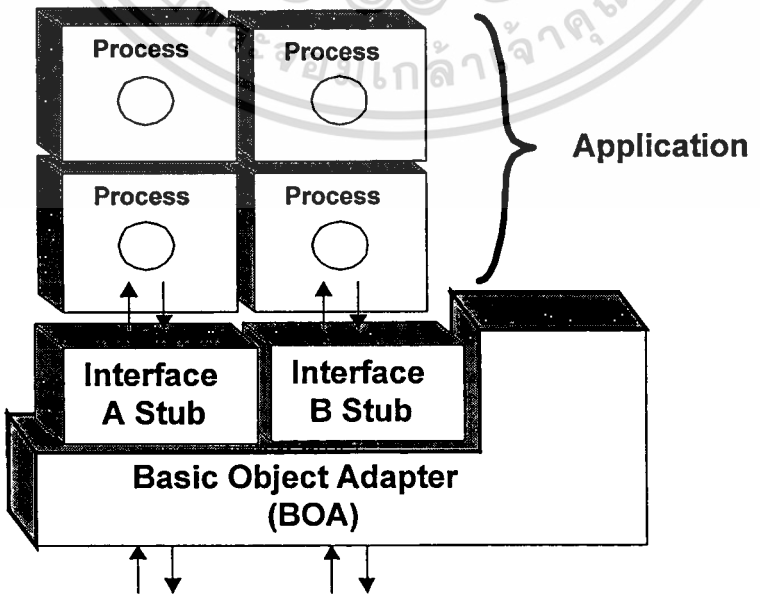
ร้องขอใช้บริการมาหนึ่งตัวและ BOA จะประกาศโดยการเรียกซิสเต็มคอล deactivate_obj เมื่อมันทำที่ คำร้องขอเสร็จแล้ว และเมื่อโปรเซสนั้นพร้อมที่จะหยุดการทำงาน BOA จะประกาศโดยการเรียก ซิสเต็มคอล deactive_impl



รูปที่ 2.8 การให้การบริการ การใช้ทรัพยากรร่วมกันของ BOA

2.3.5.2 การทำงานแบบไม่ใช้ทรัพยากรร่วมกัน (BOA Unshare Server)

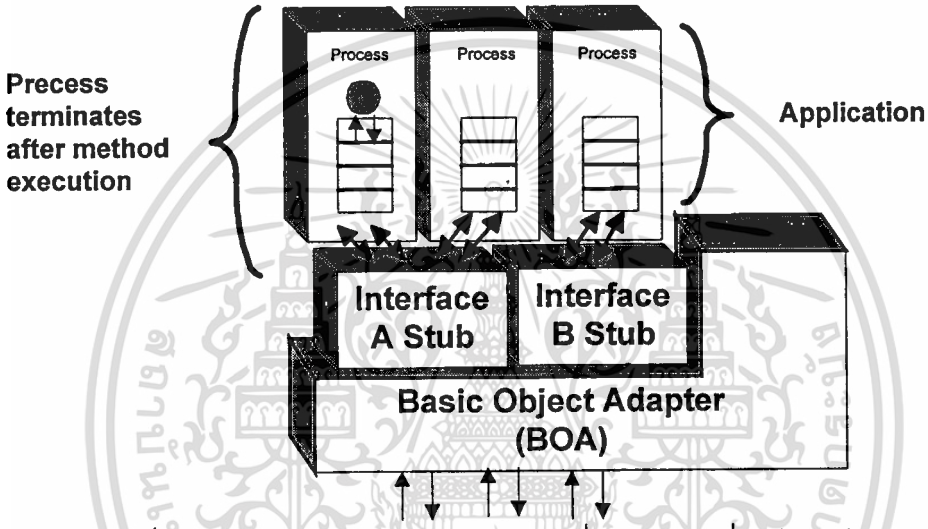
ในกรณีของแบบไม่ใช้ทรัพยากรร่วมกัน มีรูปแบบการทำงานดังนี้ แต่ละออปเจคจะอยู่บนเซิร์ฟเวอร์ต่างๆ หลายเซิร์ฟเวอร์ และเมื่อมีการขอคำร้องขอเข้ามาใหม่มันก็จะทำการสร้างออปเจคใหม่ขึ้นมา เมื่อสร้างออปเจคแล้วจะเรียกซิสเต็มคอล Obj_is_ready เซิร์ฟเวอร์ตัวอื่นก็จะสร้างออปเจคขึ้นมาใหม่อีก ถึงแม้ว่าออปเจคนั้นถูกสร้างใน เซิร์ฟเวอร์ตัวอื่นอยู่แล้ว ออปเจคบนเซิร์ฟเวอร์จะทำงานไปจนกระทั่งมันถูกสั่งให้เลิกทำงาน โดยเรียกซิสเต็มคอล deactive_obj



รูปที่ 2.9 การทำงานของ BOA ในกรณีที่เป็นแบบไม่ใช้ทรัพยากรร่วมกัน.

2.3.5.3 การทำงานแบบหนึ่งเมธอดต่อหนึ่งเซิร์ฟเวอร์ (BOA Server-per-method)

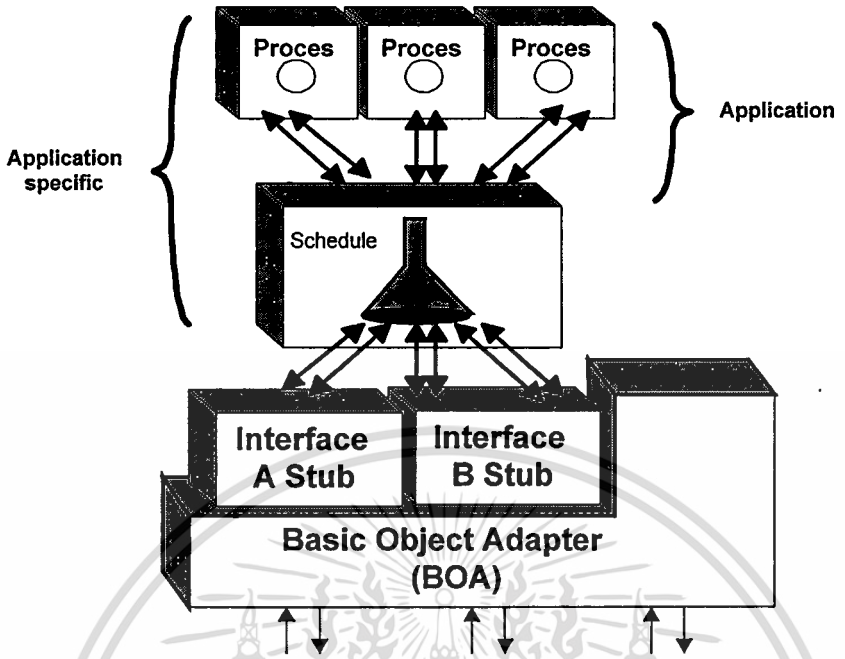
ในการทำงานแบบหนึ่งเมธอดต่อหนึ่งเซิร์ฟเวอร์มีรูปแบบการทำงานดังนี้เซิร์ฟเวอร์จะทำงานทุกครั้งที่มีการขอคำร้องขอเข้ามาในเซิร์ฟเวอร์ แต่จะทำงานในช่วงเวลาที่มีการเรียกใช้เมธอดของออปเจคเท่านั้น ดังรูป 2.10 เซริฟเวอร์จะหาว่าโปรเซสต่างๆบนเซิร์ฟเวอร์ที่ทำงานกับออปเจคที่เหมือนกัน หรือมีเมธอดเหมือนกัน อาจจะทำงานแบบเดียวกัน ในเวลาเดียวกันก็ได้ เซิร์ฟเวอร์จะเริ่มทำงานเมื่อมีการขอใช้บริการเข้ามาจากไคลเอ็นท์ ดังนั้นเมื่อมีการสร้างโปรเซส ใหม่ก็ไม่จำเป็นที่จะต้องแจ้งไปยัง BOA เพราะว่า BOA ได้ทำการสร้างโปรเซสไว้อยู่แล้วสำหรับทุกๆ การร้องขอที่เข้ามา



รูปที่ 2.10 การทำงานของ BOA ในลักษณะหนึ่งเมธอดต่อหนึ่งเซิร์ฟเวอร์.

2.3.5.4 การทำงานแบบคงที่ (BOA Persistent Server)

ในการทำงานแบบคงที่มีรูปแบบการทำงานดังนี้ เซิร์ฟเวอร์ จะถูกกระตุ้นโดย BOA ตัวอื่นดังรูป 2.11 BOA อาจจะเริ่มทำงานที่ เซิร์ฟเวอร์แอพพลิเคชั่น ซึ่ง BOA ประกาศว่ามันพร้อมที่จะทำงานแล้วโดยการเรียกซิสเต็มคอล impl_is_ready BOA จะคอยดูแลผลที่ตามมาทั้งหมดกับคำร้องขอต่างๆ ที่เกิดขึ้นตามมามันจะกระตุ้นให้แต่ละออปเจค และเมธอด ทำงานโดยการเรียกเป็นโปรเซสเดียว แต่ถ้าไม่มีการสร้างอะไรขึ้นมา (โปรเซสและ ออปเจค) BOA ก็จะส่งข้อผิดพลาดกลับมา ไปยังผู้ขอคำร้องขอนั้น

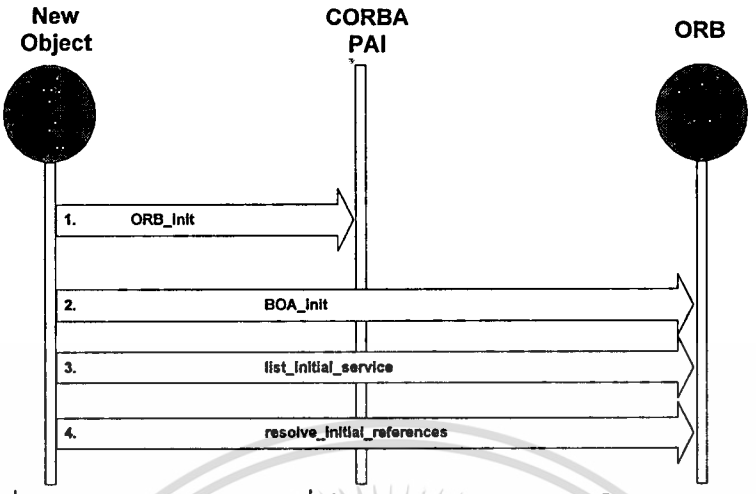


รูปที่ 2.11 การกระตุ้นแบบ BOA Persistent Server

2.4 การเริ่มการทำงานของ CORBA 2.0 และ การค้นหาคอมโพเนนท์ใน ORB

ใน CORBA 2.0 ได้กำหนดว่าออปเจคสามารถทำการ กำหนดค่าเริ่มต้นแอพพลิเคชั่นโปรแกรมอินเตอร์เฟส(initialization APIs) ตัวเองในขณะช่วงบูตสแต็ป (Bootstrap) ก่อนที่มันจะเข้าไปอยู่ใน สภาวะแวดล้อมของ ORB ซึ่งแอพพลิเคชั่นจะทำให้ คอมโพเน้นเห็น ORB,BOA,ที่เก็บอินเตอร์เฟส(Interface Repository) และกลุ่มของฟังก์ชั่นบริการพื้นฐานของออปเจค(Object Services) ต่างๆ

ขั้นตอนต่างๆ ของการทำงานภายใน ORB แสดงเป็นขั้นตอนได้ดังนี้



รูปที่ 2.12 CORBA 2.0 ทำการเริ่มให้บริการ หรือ การหาคอมโพเนนท์บน ORB

1.รับออปเจคเรฟเฟอร์เรนท์จาก ORB ใช้ CORBA API เพื่อที่จะเรียกซิสเต็มคอล ORB_init ซึ่งเป็นรูปแบบของการแสดงการมาของออปเจคเข้าสู่ ORB และจะได้รับการอ้างถึงออปเจคไปยัง ORB pseudo-object (คือออปเจคที่ถูกสร้างโดยตรงจาก ORB แต่เราสามารถเรียกได้โดยตรงมันเหมือนกับออปเจคอื่นๆ จริงๆแล้ว ORB ก็คือ pseudo-object อันหนึ่งซึ่งให้อินเตอร์เฟสไปยัง บริการต่างๆ ของมัน)

หมายเหตุ การเรียกใช้งานแอพพลิเคชั่นโปรแกรมอินเตอร์เฟส (API call) นั้นไม่ใช่การทำเมธอดการร้องขอ (method invocation) ซึ่งในการทำเมธอดการร้องขอเราต้องทำการ บูตสแต็ป ตัวเราเอง(Object) เพื่อที่จะเข้าไปอยู่ในสภาวะแวดล้อมของ CORBA

2.รับค่าตัวชี้ไปที่ OA ของ ORB เราจะต้องเรียกเมธอด BOA_init บน ORB Pseudo-object เพื่อที่จะบอก BOA ว่าเราได้รับการอ้างถึงออปเจคแล้ว(BOA คือ Pseudo-object อันหนึ่ง)

3.สามารถรับรู้ได้ว่ามีบริการอะไรอยู่บ้างในออปเจคเราสามารถที่จะเรียกเมธอดlist_intial_services บน ORB pseudo-object เพื่อที่จะได้รับรายการที่บอกว่าบน ORB นี้มีบริการอะไรบ้างเช่น ตัวอย่างของการใช้ที่เก็บอินเตอร์เฟส (Interface Repository) และบริการในเรื่องชื่อ (naming service) ซึ่งจะทำให้เราทราบถึงรายการต่างๆ ของออปเจคบน ORB

4.ได้รับการอ้างอิงถึงออปเจคที่เราต้องการ เราสามารถเรียกเมธอด resolve_initial_references เพื่อที่จะได้รับการอ้างอิงถึงออปเจคที่เราต้องการ แต่ต้องทำให้มันเป็นสมาชิกใน ORB ก่อน

2.5 การเชื่อมต่อระหว่าง CORBA 2.0 ORB ขนาดใหญ่ (CORBA 2.0 The Intergalactic ORB)

CORBA 2.0 ได้เพิ่มความสามารถของโปรโตคอล IIOP (Internet Inter-ORB Protocol) ซึ่ง IIOP โดยพื้นฐานแล้วมันคือ TCP/IP รวมกับรูปแบบของข่าวสารที่ CORBA ได้กำหนดขึ้นเพื่อแลกเปลี่ยนกันเป็นโปรโตคอล (protocol) พื้นฐานหลักของโปรโตคอล(backbone protocol) ทุกๆ ORB ที่เป็นคอลบ้า-คอมไพล

แอ้นต์ (CORBA-Compliant) จะต้องมี IIOP หรือจะต้องให้ "ฮาร์ฟ-บริดจ์" (half-bridge) ไปสู่ตัว ORB ของมัน

หมายเหตุ เราเรียกฮาร์ฟบริดจ์ เพราะว่า IIOP คือมาตรฐานของ CORBA ORB ดังนั้นคุณสมบัติของ ORB สามารถที่จะต่อกับ ORB ตัวอื่นๆ ได้โดยการส่งคำร้องขอผ่านทาง ฮาร์ฟ-บริดจ์ และ จากทาง IIOP แบล็คโบน(backbone)

2.5.1 โครงสร้างการเชื่อต่อของ CORBA 2.0 (The Inter-ORB Architecture)

2.5.1.1 โปรโตคอล GIOP(General Inter-ORB Protocol)

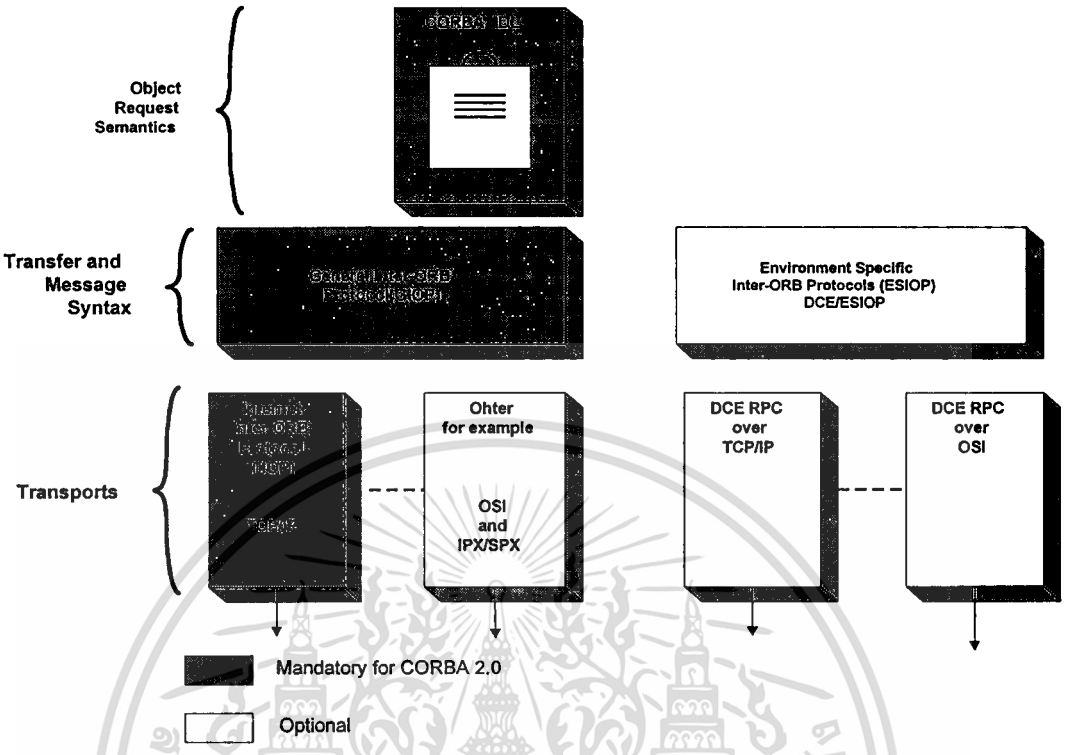
เป็นกลุ่มรูปแบบของข่าวสาร และข้อมูลทั่วไปที่ ORBs ใช้สื่อสารกัน GIOP ได้ถูกกำหนดขึ้นเพื่อการต่อกันของ ORB-to-ORB มันถูกออกแบบในการทำงานโดยตรงบนโปรโตคอลการต่อเพื่อการสื่อสาร (connection-oriented transport protocol) GIOP กำหนด 7 ข้อความ(message) ในการทำ ประกาศคำร้องขอ/ตอบรับคำร้องขอ(Request/reply) ไม่มีรูปแบบของการทำการเจรจา (negotiation) ในกรณีส่วนใหญ่ไคลเอ็นต์ส่ง คำร้องขอไปยังออปเจคทันทีทันใดหลังจากการสื่อสารได้ก่อตั้งขึ้น Common Data Representation (CDR) จะแมปเอาชนิดของข้อมูล ที่กำหนดไว้ใน IDL ไปเป็นรูปแบบของการส่งข่าวสารผ่านเน็ตเวิร์ก (flat network message) CDR ยังรักษารูปแบบของรูปแบบ (platform) ที่มีอยู่แล้วเช่น ไบต์ ออเดอะริ่ง(byte Ordering) เข้ามาอย่างไรก็คงไว้อย่างเดิม(no byte swapping is needed) และการจัดเรียงหน่วยความจำ

2.5.1.2 โปรโตคอล IIOP (Internet Inter-ORB Protocol)

เป็นการกำหนดว่าจะแลกเปลี่ยนข่าวสารบน TCP/IP เน็ตเวิร์กกันอย่างไร IIOP จะทำให้มันสามารถใช้อินเตอร์เน็ตเป็นแกนหลัก (Backbone) ได้ ORB จะใช้เป็นเส้นทางในการเชื่อมต่อกับ ORB ตัวอื่น มันออกแบบมาให้ใช้งานง่ายและสามารถทำให้เกิด "เอ้า ออฟ เดอะ บ็อก"(out of the box) ในการทำงานสำหรับORB โดยใช้ TCP/IP เป็นพื้นฐาน GIOP อาจจะถูกแมปไปในการส่งข้อมูล(transport) แบบต่างๆในอนาคต

CORBA 2.0 คอมแพททริเบิ้ล(Compatible) นั้นจะต้องสนับสนุน GIOP บน TCP/IP (หรือต่อทางฮาร์ฟ-บริดจ์)

หมายเหตุ ทั้ง IIOP และ DCE/ESIOP จะมีการอรรถธิบายข้อมูล (context data) ซึ่งมีความสัมพันธ์กับการติดต่อ (transaction) หรือบริการในระบบรักษาความปลอดภัย (security service) ต่างๆอยู่แล้ว ORB จะต้องทำการผ่านค่าคำร้องขอเหล่านี้โดยที่แอพพลิเคชั่นโดยเราไม่ต้องเข้าไปยุ่งเกี่ยว และที่ดีกว่านั้นข่าวสารชนิดนี้สามารถผ่านข้าม ORB ต่างๆ ผ่านทางบริดจ์(Bridge)



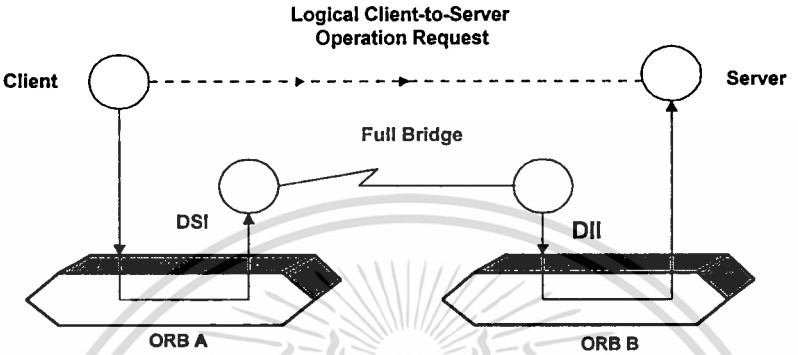
รูปที่ 2.13 โครงสร้างของการเชื่อมต่อ ORB ของ CORBA 2.0

2.5.1.3 โปรโตคอล ESIOP(Environment-Specific Inter-ORB Protocol)

จะถูกใช้สำหรับทำให้เกิด "เอ้า ออฟ เดอะ บ็อก" ในการทำงานร่วมกันบนเน็ตเวิร์ก CORBA 2.0 กำหนดให้ DCE เป็นรูปแบบอันแรกในหลายๆทางเลือกของESIOP สิ่งที่เหมือนกับ GIOP คือDCE/ESIOP จะใช้ GIOP CDR ในการแสดงชนิดข้อมูลของไอดีแอล(IDL data type) บน DCE RPC สิ่งนี้หมายความว่า DCE IDL ไม่ต้องการ OMG IDL และ CDR type ในการแมป ไปโดยตรงบน DCE's native Network Data Representation (NDR) DCE ESIOP ปัจจุบันจะให้สภาพแวดล้อมที่คงทนสำหรับการทำงานของ ORBมันได้รวมเอาข้อดีของแคปลอส เซอคูริตี้ (Kerberos Security),เซล(Cell),โกลบอล ไดเรคทอรี(Gobal directory),ดิสทริบิ้วเต็ดไทม์ Distribute time, และ ออเธนทิเคทเต็ด(authenticated) RPC DCE ได้ทำให้เราส่งข้อมูลที่มีจำนวนมาก อย่างมีประสิทธิภาพและสนับสนุนโปรโตคอลหลายๆแบบรวมถึง TCP/IP สิ่งสุดท้ายคือ DCE สามารถทำให้เราสามารถใช้การเชื่อมต่อ(Connection) และในโปรโตคอลที่ปราศจากการเชื่อมต่อ (Conectionless protocol) สำหรับใช้ในการสื่อสารของ ORB

CORBA 2.0 มีความต้องการสร้างสะพานการเชื่อมต่อ (Bridge) ระหว่าง ORB-to-ORB ดังรูป 2.14 แสดงการสร้างสตับ-ฟรี บริดจ์ (stub-free bridge) มี Dynamic Skeleton Interface (DSI) ใช้ใน

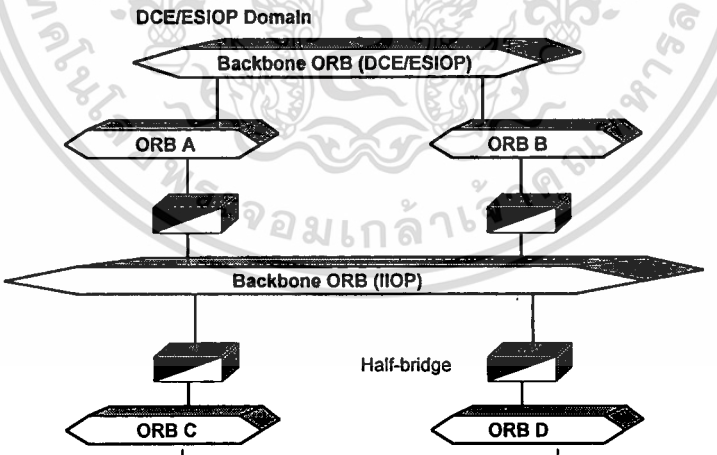
การรับเอาคำร้องขอไปยังปลายทางของ ORB ออปเจคการทำ เลท-บรินดิ้ง เทคโนโลยี(Late-binding technology) คือการยอมให้สร้างทางเข้าทางออก (gateway) ไปยัง non-CORBA Object bus เช่น OLE/COM ของ Microsoft เราจะเรียกว่า CORBA/COM



รูป 2.14 การสร้างบริดจ์โดยใช้รูปแบบของการเรียกแบบ DSI

2.5.2 การเชื่อมต่อกันหลายๆORB (ORB Federated ORBs)

เราสามารถที่จะใช้ inter-ORB bridge และ IIOP เพื่อที่จะทำการสร้างการเชื่อมโยงเป็นลักษณะของรวมกันเข้า(federation) ของ ORB ดังรูป 2.15 แสดงถึง IIOP แบล็คโบน ตัวหนึ่งกับการส่งออปเจคข้ามไปหา ORB ตัวอื่นๆทางฮาร์ฟ-บริดจ์



รูป 2.15 การเชื่อมต่อระหว่าง ORB จาก Vender ที่แตกต่างกัน

หมายเหตุ เราสามารถที่จะใช้ DCE ESIOP ได้โดยการที่แยกORB ไปเป็นโดเมน (domain) ที่ให้ผู้บริหารระบบต้องการตามลักษณะของผู้สร้าง ORB,เน็ตเวิร์ก โปรโตคอล,ความคับคั่งในการโหลด, รูปแบบของบริการ, และระบบรักษาความปลอดภัยที่เกี่ยวข้อง ซึ่งการใช้งานในแต่ละส่วนอาจจะไม่สอดคล้องกัน ดังนั้นเราอาจจะทำไฟรวอล(firewall) บนแบล็คโบนORBทางฮาร์ฟบริดจ์ในCORBA 2.0 ได้สนับสนุนความหลากลายของรูปแบบในการรวมและผสมกันที่เราจะใช้ใน IIOP สำหรับทำโกบอลแบล็คโบน (global backbone)

บทที่ 3

การสร้าง การลงทะเบียน และการเรียกใช้ CORBA ออปเจค

(Creating Registering and Invoking of CORBA object)

เทคโนโลยีทางด้านออปเจคแบบกระจาย (Distributed Object) ทำให้ออปเจคสามารถอยู่กระจัดกระจายในที่ต่างๆ บนเน็ทเวิร์ค โปรแกรมประยุกต์จะเรียกใช้ออปเจคข้ามเน็ทเวิร์คเสมือนว่า ออปเจค นั้นอยู่บนเครื่องคอมพิวเตอร์เครื่องเดียวกัน โดยการเรียกใช้ หรือ การเชื่อมต่อกันของออปเจคนั้น จะต้องมีสื่อกลางในการเชื่อมต่อถึงทำให้ออปเจคสามารถใช้งานร่วมกันได้ ในปัจจุบันมีอยู่หลายเทคโนโลยีที่ใช้ในการเชื่อมต่อระหว่างออปเจคกับโปรแกรมผู้รับบริการ (Client program) เช่น DCOM ของบริษัทไมโครซอฟท์ (Microsoft corporation), OpenDOC ของ CI Lab และ CORBA ของกลุ่มโอเอ็มจี (OMG หรือ Object Mananment Group) ทั้งสามเทคโนโลยีเป็นเทคโนโลยีทางด้านออปเจคแบบกระจายที่นำมาใช้เป็นสื่อกลางการเชื่อมต่อได้ โปรเจคนี้เลือกใช้เทคโนโลยี CORBA ซึ่งมี ORB เป็นสื่อกลางในการเชื่อมต่อ ทำให้โปรแกรมผู้รับบริการ เรียกใช้ฟังก์ชั่น (function) และเมดธอด(method) ที่อยู่บนออปเจคอื่นได้

CORBA เป็นมาตราฐานที่กำหนดมาตราฐานทางด้านออปเจคแบบการจาย มีเซอร์วิส (Service) ให้เรียกใช้ 11 เซอร์วิสเกี่ยวกับออปเจคโอเร้นเต็ด และ คอมโพแน้น และอีก 5 เซอร์วิสเกี่ยวกับการจัดการโพเซส แต่ปัจจุบันบริษัทผู้ผลิตตัว ORB ที่รองรับมาตราฐาน CORBA นั้นยังไม่สามารถสร้างผลิตภัณฑ์ที่มีเซอร์วิสได้ครบทุกเซอร์วิสตามที่มาตราฐานกำหนด

ผลิตภัณฑ์ PowerBroker CORBAplus for Virutal C++ ของ บริษัทเอ็คเปอร์ซอฟต์ (Expersoft Corporation) เป็นผลิตภัณฑ์ที่อยู่บนมาตราฐาน CORBA 2.0 compliant ที่ถูกเลือกมาใช้ในการพัฒนาโปรแกรมตัวอย่างสำหรับโปรเจคนี้ การพัฒนาโปรแกรมจะใช้ Virutal C++ 4.0 ในการแปลภาษา (Compile) โปรแกรมและลิงค์ (link) C++ library ที่เกี่ยวกับการสื่อสาร, การสร้างออปเจค, การลงทะเบียนออปเจค, การลบออปเจค,การเคลื่อนย้ายออปเจค และ การเรียกใช้ออปเจคของ PowerBroker CORBAplus เข้าไปด้วย

3.1 โปรแกรม และเครื่องคอมพิวเตอร์ที่ใช้ในการสร้างและพัฒนาโปรแกรม (Software and Hardware Requirement)

3.1.1 เครื่องคอมพิวเตอร์ CPU pentium 75 RAM 32 Mbyte 2 เครื่อง

3.1.2 ระบบเน็ทเวิร์คที่ใช้โปรโตคอล ทีซีพี/ไอพี (Protocal TCP/IP) ในการติดต่อสื่อสาร

3.1.3 ระบบปฏิบัติการ(operating system หรือ OS) ใช้ วินโดวส์ เอ็นที เวคร์สเตชัน (Window NT workstation) และ วินโดวส์ เอ็นที เซฟเวอร์(Window NT server)

3.1.4 ตัวแปลภาษา IDL(IDL Compiler) ใช้ PowerBroker CORBAplus for Virutal C++ ในการแปลภาษา IDL

3.1.5 ตัวแปลภาษา Virtual C++ 4.0 ใช้สำหรับแปลโปรแกรมประยุกต์ (Application Program) ให้เป็นภาษาเครื่อง

3.2 ขั้นตอนในการสร้างโปรแกรมประยุกต์โดยใช้ CORBA ในการกระจายออปเจค

โปรแกรมประยุกต์แบบออปเจคกระจาย(CORBA Application or Distributed Application)มีส่วนประกอบ 2 ส่วน ส่วนแรก คือ โปรแกรมผู้ให้บริการ (Server Program) และส่วนที่สอง คือ โปรแกรมผู้รับบริการ (Client Program) ออปเจคของ CORBA (CORBA Object) ถูกออกแบบและสร้างขึ้นโดยโปรแกรมผู้ให้บริการ และในส่วนของโปรแกรมผู้รับบริการจะต้องสร้างตัวชี้ชี้ไปยังออปเจคของผู้ให้บริการ เพื่อขอใช้บริการของออปเจคนั้น โดยขั้นตอนการสร้างออปเจค CORBA และการเรียกใช้ฟังก์ชั่นของออปเจคมีดังต่อไปนี้

3.2.1 กำหนดอินเตอร์เฟสของบริการต่าง ๆ ที่มีในออปเจค

เป็นการกำหนดฟังก์ชั่น (Method) หรือ ตัวแปร (Data Member) ของออปเจค ฟังก์ชั่นและตัวแปรเหล่านี้ได้จากขั้นตอนการวิเคราะห์และการออกแบบ(Object Analysis and Design) จากปัญหาจริง การกำหนดอินเตอร์เฟสจะต้องระบุอินเตอร์เฟสทั้งหมดที่จะให้โปรแกรมผู้รับบริการเรียกใช้ได้ โดยใช้ไวยกรณ์ (Syntax) ของภาษา IDL และเขียนอินเตอร์เฟสเหล่านั้นลงแฟ้มที่มีนามสกุลเป็น ".IDL"

ตัวอย่าง IDL ที่ระบุถึงอินเตอร์เฟสของฟังก์ชั่น ของออปเจค "Depot" ซึ่งเป็นออปเจคของสำนักงานคลังสินค้า(Central Depot) ทำหน้าที่ค้นหาข้อมูลของสินค้าให้กับผู้รับบริการ เมื่อมีการสอบถามเข้ามาโดยการส่งรหัสสินค้า และ จำนวนของสินค้ามาให้ ฟังก์ชันจะส่งข้อมูลของสินค้ากลับไปให้ผู้รับบริการ โดยมีฟังก์ชั่น "FindItemInfo" ทำหน้าที่ค้นหาข้อมูลของสินค้าในฐานข้อมูลมาให้ อินเตอร์เฟสทั้งหมดของออปเจค "Depot" เก็บอยู่ในแฟ้มชื่อ "Central.idl"

Central.idl

```
#ifndef CENTRAL_IDL
#define CENTRAL_IDL

  interface Depot {
    void FindItemInfo(
                in AStoreId    StoreId,
                in Barcode     Item,
                in long        Quantity,
                inout ItemInfo  IInfo)
     raises (BarcodeNotFound);
    };

#endif
```

รูปที่ 3.1 แฟ้มIDL ของออปเจค Depot

ไวยกรณ์ของภาษา IDL จะมีลักษณะคล้ายกับไวยกรณ์ของภาษา C++ โดยคำว่า

Interface เป็นการระบุชื่อออปเจค

Void FindItemInfo เป็นชื่อฟังก์ชั่นที่ใช้ค้นหาข้อมูลสินค้า ของออปเจค "Depot" โดยไม่มีค่าคืนกลับ

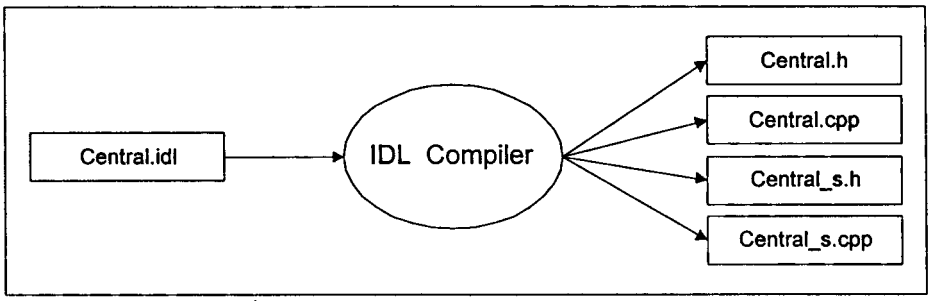
In ใช้นำหน้าตัวแปร เพื่อแสดงว่าค่าของตัวแปรนั้นถูกส่งค่ามาจากโปรแกรมผู้รับบริการ มาให้โปรแกรมผู้ให้บริการ

Inout ใช้นำหน้าตัวแปรเพื่อแสดงว่าค่าของตัวแปร ถูกส่งจากโปรแกรมผู้รับบริการมาให้โปรแกรมผู้ให้บริการ หรือ ส่งจากโปรแกรมผู้ให้บริการกลับไปให้โปรแกรมผู้รับบริการ

3.2.2 การแปลภาษา IDL ในแฟ้ม .IDL เป็นคำสั่งภาษา C++

หลังจากกำหนดอินเตอร์เฟสของออปเจคลงในแฟ้ม .IDL แล้ว ขั้นตอนต่อไป คือการแปลแฟ้ม .IDL ให้เป็นคำสั่งภาษา C++ ด้วยโปรแกรม CORBAplus IDL Compiler (IDLC)

โปรแกรม IDLC เป็นโปรแกรมแปลภาษา IDL ที่ทำหน้าที่แปลภาษา IDL ให้เป็นภาษา C++ โดยการสร้างแฟ้มโปรแกรมภาษา C++ ขึ้นอัตโนมัติ 4 แฟ้ม โดยแต่ละแฟ้มการนำไปใช้งานจะแตกต่างกัน



รูปที่ 3.2 แผนผังการทำงานของตัวแปลภาษา IDL

Central.h เป็นแฟ้มอินเตอร์เฟสเฮดเดอร์(Interface header file) ประกอบด้วยชนิดข้อมูล (Type)ของภาษา C++ ที่แปลมาจากชนิดข้อมูลที่ระบุไว้ใน IDL และยังเป็นคลาสพื้นฐาน(base class) ของทุกๆ อินเตอร์เฟสที่ระบุใน IDL ซึ่งชนิดข้อมูลของแฟ้มอินเตอร์เฟสเฮดเดอร์นี้ จะถูกนำไปใช้ โดยออปเจคอิมพลีเมนท์เทชั่น (Implementation Object) และ โดยโปรแกรมผู้รับบริการ โปรแกรมผู้รับบริการ จะใช้แฟ้มเฮดเดอร์นี้ เป็น stub สำหรับอ้างถึงออปเจคอิมพลีเมนท์เทชั่น

Central.cpp เป็นแฟ้มอินเตอร์เฟสอิมพลีเมนท์เทชั่น (Interface Implementation file)ประกอบด้วย อิมพลีเมนท์เทชั่นของชนิดข้อมูลที่ประกาศไว้ในแฟ้มอินเตอร์เฟสเฮดเดอร์ จะถูก ใช้โดยรวมกับโปรแกรมผู้รับบริการ และ ออปเจคอิมพลีเมนท์เทชั่น

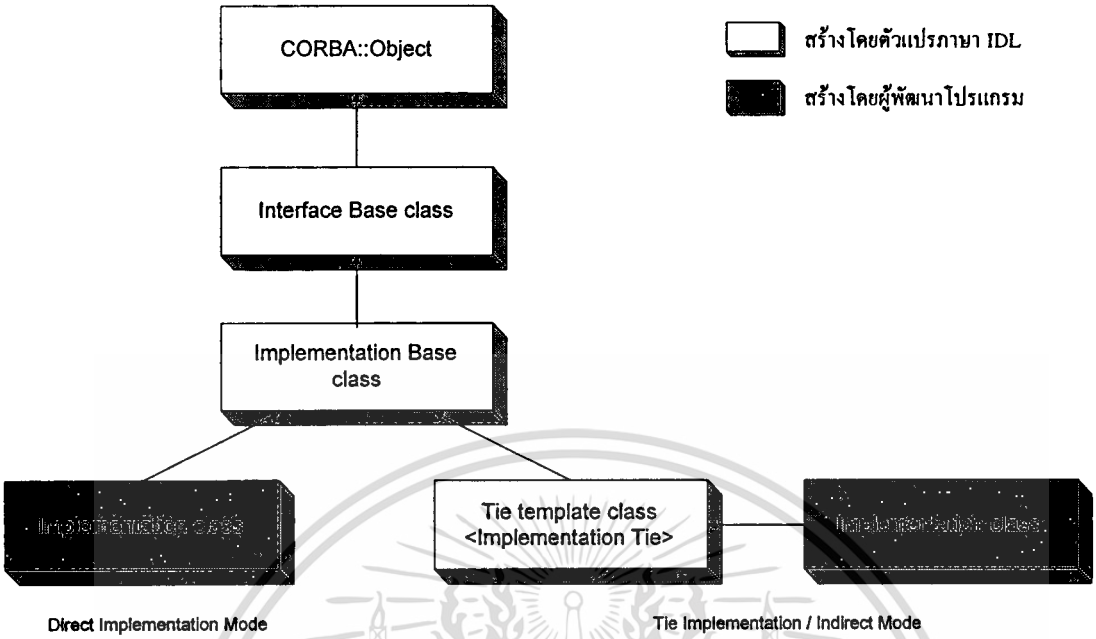
Central_s.h เป็นแฟ้มเฮดเดอร์พื้นฐาน ใช้สำหรับสร้างออปเจคอิมพลีเมนท์เทชั่น เท่านั้น การสร้าง ออปเจคอิมพลีเมนท์เทชั่น จะต้องมีการรวมแฟ้มเฮดเดอร์พื้นฐานนี้เข้าไว้ในแฟ้มของออปเจคอิมพลีเมนท์เทชั่นด้วย

Central_s.cppเป็นแฟ้มที่ประกอบด้วย ออปเจคอิมพลีเมนท์เทชั่นของคลาสพื้นฐาน จะถูกแปลและลิงค์ รวมกับออปเจคอิมพลีเมนท์เทชั่น ที่ผู้พัฒนาโปรแกรมเขียนขึ้น

3.2.3 การเขียนคำสั่ง (Code) และข้อมูล (Data) จริงของออปเจค

เป็นการเขียนคำสั่งและข้อมูลจริงที่ใช้แทนตัวออปเจค (Implementation Object) คำสั่งจะกำหนดหน้าที่การทำงานของฟังก์ชั่นต่าง ๆ ของออปเจคที่โปรแกรมผู้รับบริการสามารถเรียกใช้ได้ การเขียนคำสั่ง และ ข้อมูลแทนออปเจคทำได้สองทาง คือ ทางตรง (Direct) และ ทางอ้อม (Indirect)

คำสั่งภาษา C++ ที่ถูกสร้างโดยตัวแปลภาษา IDL จะมี คลาสของออปเจคหลัก ๆ สองคลาส คือ คลาสอินเตอร์เฟสพื้นฐาน (Interface Base class) หรือ คลาสสตรับ (Stub class) ที่สืบทอดมาจากคลาส CORBA::object ประกอบด้วยฟังก์ชั่นเสมือนบริสุทธ์ (pure virtual function) ของทุก ๆ ฟังก์ชั่นที่ระบุไว้ใน IDL และ คลาสอิมพลีเมนท์เทชั่นพื้นฐาน (Implementation Base class) หรือ คลาสสคีรีทอน (Skeleton class) ที่สืบทอดมาจากคลาสอินเตอร์เฟสพื้นฐานอีกทอดหนึ่ง การเขียนคำสั่งที่ใช้แทนตัวออปเจคต้องเขียนคลาสอิมพลีเมนท์เทชั่น (Implementation class) ขึ้นใหม่



รูปที่ 3.3 ความสัมพันธ์ระหว่างคลาส จากตัวแปรภาษา IDL กับ คลาสอิมพลีเมนต์เทชั่น

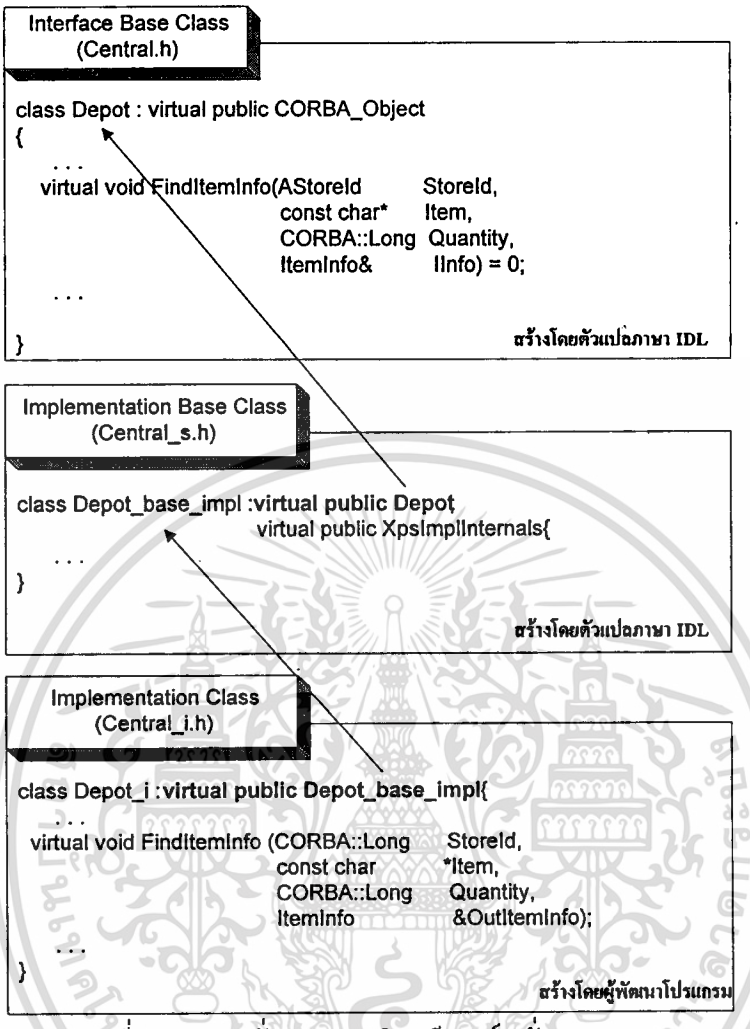
การเขียนคลาสอิมพลีเมนท์เทชั่น

3.2.3.1 ทางตรง เป็นการเขียนคลาสอิมพลีเมนท์เทชั่นขึ้นใหม่ที่สืบทอดโดยตรงจากมาจากคลาสอิมพลีเมนท์เทชั่นพื้นฐาน จะต้องเขียนคำสั่งสำหรับทุกฟังก์ชั่นที่ระบุไว้ใน IDL โดยกำหนดฟังก์ชั่นให้เป็นฟังก์ชั่นเสมือน (virtual function)

3.2.3.2 ทางอ้อม คลาสอิมพลีเมนท์เทชั่นไม่ต้องสืบทอดมาจากคลาสอิมพลีเมนท์เทชั่นพื้นฐาน แต่จะต้องมีการเขียนคำสั่งสำหรับทุกฟังก์ชั่นที่ระบุใน IDL เหมือนกับการเขียนแบบทางตรง การสืบทอดของคลาสอิมพลีเมนท์เทชั่น จะสืบทอดมาจากคลาส Tie template ที่สืบทอดมาจากคลาสอิมพลีเมนท์เทชั่นพื้นฐานอีกทีหนึ่ง Tie template ถูกสร้างโดย ตัวแปลภาษา IDL

การเขียนคลาสอิมพลีเมนท์เทชั่นของโปรเจคนี้จะใช้การเขียนอิมพลีเมนท์เทชั่นแบบทางตรง

คลาสอิมพลีเมนท์เทชั่นที่ได้จะเป็น CORBA ออปเจค หรือ ออปเจคแบบกระจาย โดยมีแฟ้มอินเตอร์เฟส เฮดเดอร์เป็น "Central.h" เมื่อโปรแกรมผู้รับบริการต้องการเรียกใช้ออปเจคนี้จะต้องรวม (Include) แฟ้มนี้เข้าไว้ในแฟ้มหลักของโปรแกรมผู้รับบริการด้วย ในการใช้งานจริงโปรแกรมผู้รับบริการจะยังไม่สามารถเรียกใช้ออปเจคนี้ได้จนกว่า โปรแกรมผู้ให้บริการจะมีการลงทะเบียนออปเจคบน BOA (Basic Object Adapter) ก่อนถึงจะเรียกใช้ออปเจคได้



รูปที่ 3.4 การคำสั่งของคลาสอิมพลีเมนท์เทชั่นแบบทางตรง

3.3 การลงทะเบียนออปเจค และ การสร้างตัวชี้ออปเจค (Register and Resolveing object)

การลงทะเบียนออปเจค คือการนำเอาออปเจค CORBA มาประกาศไว้บน BOA เพื่อทำให้โปรแกรมผู้รับบริการสามารที่จะเรียกใช้งานออปเจคนั้นได้ การประกาศออปเจคจะทำทางด้านโปรแกรมผู้ให้บริการ เริ่มต้นโปรแกรมด้วยการสร้างสภาวะแวดล้อมของ ORB(ORB environment) และ BOA เพื่อใช้ในการลงทะเบียนออปเจค จากนั้นสร้างตัวชี้ออปเจค เป็นการสร้างสิ่งที่ผู้รับบริการจะใช้ติดต่อกับออปเจค CORBA เพื่อเรียกใช้ฟังก์ชั่นภายในออปเจค การลงทะเบียน และ การสร้างตัวชี้ออปเจคทำได้หลายวิธี แต่จะแสดงให้เห็นสองวิธีง่าย ๆ คือ

3.3.1 การลงทะเบียน และการสร้างตัวชี้ออปเจคที่ต้องระบุตำแหน่งที่อยู่ของออปเจคที่แน่นอน

การลงทะเบียน

โปรแกรมผู้ให้บริการจะลงทะเบียน โดยเริ่มจากการสร้างสภาวะแวดล้อมของ ORB และ สภาวะแวดล้อมของ BOA กำหนดช่องทางติดต่อ (port) ที่แน่นอนด้วยคำสั่ง

```
int argc_central = 3;
char * argv_central[3] = {"Central","-pbport","5555"};   //กำหนดช่องทางติดต่อ 5555
CORBA::ORB_ptr  orb   =  CORBA::ORB_init(argc_central, argv_central);
CORBA::BOA_prt  aboa =  orb->BOA_init(argc_central_argv_central);
XpsBOA*              pboa =  XpsBOA::narrow(aboa);
```

จากนั้นสร้างออปเจคอิมพลีเมนท์เทชั่น จากคลาสอิมพลีเมนท์เทชั่น และ ลงทะเบียนออปเจคบน BOA โดยกำหนดชื่อที่จะใช้เรียกตอนค้นหาออปเจคไว้ด้วย

```
Depot_prt  Dp  =  new  Depot_i( );  //สร้าง ออปเจคอิมพลีเมนท์เทชั่น
pboa->obj_is_read(Dp,nil);              //ลงทะเบียนออปเจค
pboa->reigsterr Alias(Dp,"depot");    //กำหนดชื่อที่ใช้เรียกแทนออปเจค
```

Central_Main.cpp

```
#include "Depot_i.h"
#include <pbroker/corba/orb.h>
#include <pbroker/corba/xpsboa.h>
#include <pbroker/pberr.h>
#include <pbroker/winsvc/winsvc.h>

        . . .

BOOL CCentral_MainApp::InitInstance()
{
        . . .

int   argc_central = 3;
char * argv_central[3] = {"central","-pbport","5555"};

CORBA::ORB_ptr   orb  = CORBA_ORB::_nil();
CORBA::BOA_ptr   aboa = CORBA_BOA::_nil();
XpsBOA* pboa = XpsBOA::_nil();
XpsEventService    eventService;

orb = CORBA::ORB_init(argc_central, argv_central);
aboa = orb->BOA_init(argc_central, argv_central);
pboa = XpsBOA::_narrow(aboa);

Depot_ptr Dp = new Depot_i(); // create instance
//register instance without implementation
pboa->obj_is_ready(Dp, nil);
//register alias forr instance to use URRLs with
//initialization serrrvice
pboa -> registerAlias(Dp,"depot");
     . . .

eventService.mainloop();
return FALSE;
}
```

รูปที่ 3.5 ตัวอย่างการลงทะเบียนที่ต้องระบุตำแหน่งที่อยู่ของออปเจคที่แน่นอน

การสร้างตัวชี้ออปเจค

โปรแกรมผู้รับบริการจะต้องรู้หมายเลข IP ของเครื่องที่ให้บริการออปเจค หรือเครื่องที่ถือออปเจค และจะต้องรู้ชื่อจริงของออปเจคที่จะเรียกใช้ ถึงจะสามารถเรียกออปเจคมาใช้ได้ และถ้ามีมากกว่า 1 ออปเจค แต่ละออปเจคอยู่ต่างเครื่องกันต้องระบุหมายเลข IP ของเครื่องที่ออปเจคนั้นอยู่ทั้งหมดทุกออปเจค ฉะนั้น ถ้ามีออปเจคใหม่เกิดขึ้นผู้พัฒนาโปรแกรมจะต้องรู้ว่า ออปเจคนั้นอยู่ที่ไหน มีชื่อจริงว่าอะไร และชื่อที่เรียกแทน (ชื่อเล่น) ว่าอะไร ถึงจะนำออปเจคมาใช้งานได้

การสร้างตัวชี้ออปเจคเริ่มจากการสร้างสภาวะแวดล้อมของ ORB กำหนดที่อยู่ ช่องทางการติดต่อ และชื่อของออปเจคด้วยคำสั่ง

```
init argc_central = 4;
char * argv_central[4] = {"Store","-pbinit","Depot_i","iiop://161.246.2.31:5555/depot"};
CORBA::ORB_prt  orb  =  CORBA::ORB_init(argc_central,argv_central);
```

จากนั้นทำการค้นหาออปเจคโดยกำหนดจากชื่อของออปเจคที่ต้องการเรียกใช้ ด้วยคำสั่ง

```
CORBA::object_ptr  obj  =  orb->Resolve_initial_referrences("Depot_i");
Depot_ptr  DP_ptr      =  Depot::_narrow(obj);
```

3.3.2 การลงทะเบียนและการสร้างตัวชี้ออปเจค โดยใช้บริการ Naming ของ CORBA

บริการ Naming (Naming Service) เป็นบริการของ CORBA ที่ช่วยให้การสร้างตัวชี้ออปเจคไม่ต้องระบุหมายเลข IP ของเครื่องที่ออปเจคอิมพลีเมนท์เทชั่นอยู่ แต่จะต้องรู้ชื่อที่โปรแกรมผู้ให้บริการใช้เรียกแทนออปเจคในการประกาศออปเจค การใช้บริการ Naming เป็นบริการที่ทำให้การค้าหาออปเจคเป็นไปได้สะดวกขึ้น จะต้องรันโปรแกรม "pbnamed.exe" (โปรแกรมที่มาพร้อมกับผลิตภัณฑ์ PowerBroker CORBAplus) เป็นโปรแกรมแบบฝังตัวสำหรับให้บริการแก่ออปเจคที่จะมาลงทะเบียนใหม่ โปรแกรมนี้จะทำการสร้างออปเจคชื่อ NameService ไว้สำหรับเก็บข้อมูลของออปเจคที่มีการลงทะเบียนบน BOA ไว้ ทุกออปเจคที่ลงทะเบียนบน BOA จะติดประกาศการลงทะเบียนบนออปเจค NameService ด้วยฟังก์ชั่น 'rebind' โดยระบุชื่อที่ใช้แทนออปเจค และตัวชี้ที่ชี้ไปยังออปเจคไปให้ เมื่อโปรแกรมผู้รับบริการต้องการสร้างตัวชี้ออปเจค จะมาค้นหาตัวชี้ออปเจคในออปเจค NameService ด้วยฟังก์ชั่น "resolve" โดยระบุชื่อของออปเจค ถ้าชื่อของออปเจคมีอยู่ ฟังก์ชั่นจะคืนค่าตัวชี้ออปเจคกลับคืนมาให้ โปรแกรมผู้รับบริการถึงจะนำตัวชี้ไปเรียกใช้ฟังก์ชั่นภายในของออปเจคนั้นได้

การลงทะเบียนออปเจค

การใช้ บริการ Naming ไม่ต้องระบุหมายเลข IP ของเครื่องที่ออปเจคอยู่ แต่จะต้องระบุหมายเลข IP ของเครื่องที่รันโปรแกรม "pbnamed.exe" หรือบริการ Naming และช่องทางติดต่อ ชื่อที่ใช้แทนออปเจค NameService คือชื่อ "NameServiceRoot" ช่องทางติดต่อเบอร์ 6004

ขั้นตอนแรก ของการลงทะเบียนโดยใช้บริการ Naming คือ การสร้างตัวชี้ออปเจคชี้ไปยัง ออปเจค NameService โดยระบุหมายเลข IP , ชื่อออปเจค NameService , ช่องทางการติดต่อ และ ชื่อที่ใช้เรียกแทนด้วยคำสั่ง

```
init argc_central = 4;
char * argv_central[4] = {"Central","-pbinit","NameService","iiop://161.246.2.31:6004/
                          NameServiceRoot"};
CORBA::ORB_ptr  orb   =  CORBA::ORB_init(argc_central, argv_central);
CORBA::BOA_prt  aboa =  orb->BOA_init(argc_central_argv_central);
XpsBOA*          pboa =  XpsBOA::narrow(aboa);
CORBA::Object_ptr  obj =  orb->resolve_initial_references("NameService");
CosNaming::NamingContext_ptr pNC = CosNaming::NamingContext::narrow(obj);
```

ขั้นตอนที่สอง ลงทะเบียนออปเจคอิมพลีเมนท์เทชั่น บน BOA โดยไม่ต้องกำหนดชื่อที่ใช้แทนออปเจคไว้บน BOA แต่จะกำหนดชื่อตอนติดประกาศเท่านั้น

```
Depot_ptr  Dp = new Depot_i( );
pboa->obj_is_ready(Dp,nil);
```

ขั้นตอนที่สาม ประกาศออปเจคอิมพลีเมนท์เทชั่น ในออปเจค NameService โดยการกำหนดชื่อที่ใช้แทนออปเจค และตัวชี้ออปเจคด้วยคำสั่ง rebind เพื่อทำให้โปรแกรมผู้รับบริการสามารถเข้ามาค้นหาได้ว่ามีออปเจคใดลงทะเบียนไว้และอยู่ที่ไหน

```
CosNaming::NameComponent    ncStore;
CosNaming::Name             StoreName(1);
ncStore.id                  = (const char *) "depot";      // กำหนดที่ที่ติดประกาศ
StoreName(0)                = ncStore;
pNc->rebind(StoreName,Dp);
```

การสร้างตัวชี้ออปเจค

โปรแกรมผู้รับบริการไม่ต้องรู้ที่อยู่ของออปเจคแต่จะต้องรู้ชื่อออปเจคที่ผู้ให้บริการใช้แทนตัวออปเจค เพราะมีการประกาศออปเจคด้วยชื่อที่ใช้แทนออปเจค และ ตัวชี้ที่ชี้ไปยังออปเจคใน ออปเจค NameService อยู่แล้ว ฉะนั้นการสร้างตัวชี้ออปเจคทำได้โดย

ขั้นตอนแรก คือสร้างตัวชี้ชี้ไปยังออปเจค NameService ด้วยคำสั่ง

```
init argc_central = 4;
char * argv_central[4] = {"Store","-pbinit","NameService","iiop://161.246.2.31:6004/
                          NameServiceRoot"};
CORBA::ORB_ptr  orb  =  CORBA::ORB_init(argc_central, argv_central);
CORBA::BOA_prt  aboa =  orb->BOA_init(argc_central_argv_central);
XpsBOA*         pboa =  XpsBOA::narrow(aboa);
CORBA::Object_ptr  obj =  orb->resolve_initial_references("NameService");
CosNaming::NamingContext_ptr pNC = CosNaming::NamingContext::narrow(obj);
```

ขั้นตอนที่สอง คือการค้นหาออปเจคที่ต้องการในออปเจค NameService ด้วยการระบุชื่อของออปเจคด้วยคำสั่ง resolve ตัวออปเจค NameService จะคืนค่าที่ใช้อ้างออปเจคอิมพลีเมนท์เทชั่นกลับมาให้ผู้รับบริการเพื่อที่จะเอามาสร้างตัวชี้ที่ชี้ไปยังออปเจคอิมพลีเมนท์เทชั่น

```
CORBA::Object_ptr                StoreObj;
CosNaming::NameComponent         ncStore;
CosNaming::Name                  StoreName(1);
ncStore.id                       = (const char *) "Depot";
StoreName(0)                     = ncStore;
StoreObj                         = pNC->resolve(StoreName); // จะคืนค่าที่ใช้อ้างออปเจค
Depot_ptr  DP_ptr                = Depot::_narrow(StoreObj);   // สร้างตัวชี้ออปเจค
```

3.4 การเรียกใช้บริการของออปเจค

การเรียกใช้บริการของออปเจค ทางด้านโปรแกรมผู้รับบริการต้องมีการรวมแฟ้มอินเตอร์เฟสเฮดเดอร์ของออปเจคอิมพลีเมนท์เทชั่น เข้าไว้ในแฟ้มหลักของโปรแกรมผู้รับบริการ และมีการประกาศตัวแปรพอยเตอร์ของออปเจค สำหรับถือค่าตัวชี้ของออปเจค

Store_Main.h

```
#include  "central.h"
        . . .
class CStore_Main {
private:
        Depot_ptr m_DPptr;
        . . .
}
```

รูปที่ 3.6 การประกาศตัวแปรสำหรับถือที่ใช้อ้างถึงออปเจค Depot

เมื่อเริ่มโปรแกรมจะมีการสร้างตัวชี้ชี้ไปยังออปเจคอิมพลีเมนท์เทชั่น เก็บตัวแปรไว้ใน m_DPptr และเมื่อต้องการเรียกใช้ฟังก์ชั่นสมาชิก (member function) ของออปเจค Depot ทำได้โดยการอ้างตัวแปรพอยเตอร์ ชี้ไปยังฟังก์ชั่นสมาชิกนั้น เหมือนกับการอ้างอิงตัวแปรพอยเตอร์ของออปเจคทั่ว ๆ ไป

```
m_DPptr->FindItemInfo(StoreNumber,
                      m_InBarcodeEdit,
                      m_InQtyEdit,
                      Information);
```

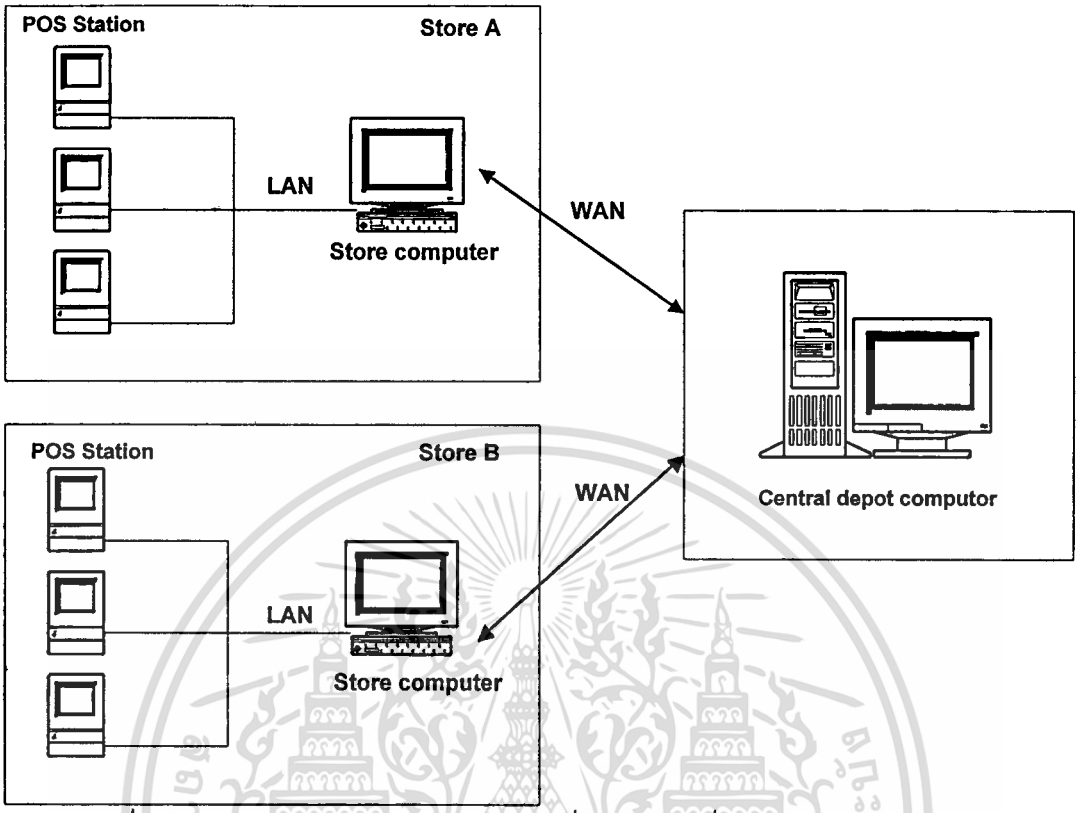
บทที่ 4
การวิเคราะห์ ออกแบบออปเจค และการพัฒนาโปรแกรมระบบขายปลีกที่เป็นสาขาย่อย

"ระบบขายปลีกที่เป็นสาขาย่อย" ถูกนำมาพัฒนาเป็นโปรแกรมตัวอย่างของโปรเจคนี้ เพื่อแสดงให้เห็นว่าเทคโนโลยีออปเจคแบบกระจาย แต่ละออปเจคสามารถอยู่กระจัดกระจายบนเน็ทเวิร์ค และเมื่อต้องการใช้สามารถเชื่อมออปเจคถึงกันได้โดยใช้ตัวกลางเชื่อมต่อทำให้ออปเจคเสมือนอยู่บนเครื่องเดียวกัน โปรแกรมผู้รับบริการแต่ละโปรแกรมสามารถใช้ออปเจคร่วมกัน (share) หรือสร้างออปเจคขึ้นใหม่ (pump) ได้ และการสนับสนุนการนำออปเจคที่สร้างไว้แล้วกลับมาใช้ใหม่ (Reuse) อีกด้วย นอกจากนี้ยังเป็นการศึกษาขั้นตอนการวิเคราะห์ และออกแบบออปเจค (Object Analyse and Design) จากปัญหาจริง

4.1 ขอบเขตโดยรวมของปัญหาตัวอย่าง ระบบขายปลีกที่เป็นสาขาย่อย(Point-Of-Sale)

ร้านขายปลีกที่เป็นสาขาย่อยๆ ต้องการเปลี่ยนแปลงวิธีการขายและ วิธีการจัดการข้อมูลในร้านให้เป็นระบบการเก็บเงินแบบฉลาด(intelligent cash registers) ที่เรียกว่า พอยต์ออฟเซล(Point-of-Sale) หรือโพสต์(POS) เครื่องโพสต์ เหล่านี้ จะมีตัวอ่านแถบรหัส(Barcode Reader) และ เครื่องพิมพ์(Printer) อยู่ด้วย เครื่องโพสต์ แต่ละตัวจะต่ออยู่กับ เครื่องคอมพิวเตอร์ร้านค้า(Store Computer) ตัวเดียว ที่ทำหน้าที่เก็บข้อมูลทั่วไปของเครื่องโพสต์ทุกเครื่องในร้านค้า แต่ละร้านค้าย่อยจะมีคอมพิวเตอร์ร้านค้าเพียงตัวเดียวเท่านั้น และเครื่องคอมพิวเตอร์ร้านค้าแต่ละร้าน จะต่อเข้ากับเครื่องคอมพิวเตอร์ของสำนักงานคลังสินค้า (Central Depot Computer) ข้อมูลของเครื่องโพสต์ทุกเครื่องในระบบ สามารถถูกเปลี่ยนแปลงแก้ไขได้จากเครื่องคอมพิวเตอร์ของสำนักงานคลังสินค้า เพื่อเปลี่ยนแปลงข้อมูลบางตัวเช่น ราคาสินค้า, ประเภทของภาษีสินค้า, ต้นทุนของสินค้า และจำนวนสินค้าที่คงเหลือในคลังสินค้า(Stock)

เครื่องคอมพิวเตอร์ร้านค้าจะรับข้อมูลการขายสินค้าจากเครื่องโพสต์ส่งไปยังเครื่องสำนักงานคลังสินค้าเพื่อสอบถามข้อมูลของสินค้าที่กำลังจะขาย(เช่น ราคาขาย, ประเภทภาษีสินค้า) และเพื่อเป็นการแก้ไข (Update) จำนวนสินค้าคงคลัง นอกจากนั้นเครื่องคอมพิวเตอร์ร้านค้ายังรับข้อมูลของสินค้าที่จะขาย (เช่น ราคาต้นทุนของสินค้า, ประเภทภาษี) จากสำนักงานคลังสินค้า นำมาคำนวณหาราคาสินค้าที่คิดภาษีได้, ราคาขาย ส่งให้กับเครื่องโพสต์ที่สอบถามมา



รูปที่ 4.1 แผนผังแสดงความสัมพันธ์ระหว่างเครื่องโพสต์, เครื่องคอมพิวเตอร์ร้านค้า และ เครื่องคอมพิวเตอร์สำนักงานคลังสินค้า

4.2รายละเอียดของปัญหา

4.2.1 ในร้านค้ามีเครื่องโพสต์ต่ออยู่กับเครื่องคอมพิวเตอร์ร้านค้า โดยพนักงานเก็บเงิน (Cashier) จะเป็นผู้เปิดเครื่องโพสต์และเข้าสู่ระบบ(Login) เครื่องโพสต์ถึงพร้อมใช้งาน เครื่องจะพร้อมใช้งานไปตลอดจนกว่าจะออกจากระบบ(Logout) หรือจนกว่าพนักงานเก็บเงินจะปิดเครื่องโพสต์ เครื่องคอมพิวเตอร์ร้านค้าจะรับรายการขายที่เกิดขึ้นกับเครื่องโพสต์ ก่อนส่งต่อไปที่เครื่องคอมพิวเตอร์สำนักงานคลังสินค้า เพื่อที่จะเก็บเป็นบัญชีรายการที่เกิดขึ้นทั้งหมด และตอบสนองความต้องการในการถามถึงต้นทุนราคาสินค้า ส่วนเรื่องการคำนวณภาษีของสินค้าแต่ละชิ้น จะคำนวณที่เครื่องคอมพิวเตอร์ร้านค้า

4.2.2 แต่ละเครื่องโพสต์ ถูกต่อเข้ากับเครื่องอ่านแถบรหัส และแผงคีย์(keypad) ผลที่ได้จากเครื่องอ่านแถบรหัส จะถูกส่งไปให้เครื่องโพสต์ แผงคีย์สามารถเลือกคำสั่งได้ 5 คำสั่งดังนี้

4.2.2.1 เข้าสู่ระบบ สำหรับพนักงานเก็บเงินแต่ละคน

4.2.2.2 พิมพ์ใบเสร็จ(Slip) แสดงการขายทั้งหมดของเครื่องโพสต์ ตั้งแต่เข้าสู่ระบบครั้งล่าสุด

4.2.2.3 พิมพ์ใบเสร็จ แสดงการขายทั้งหมดของร้านค้าตั้งแต่เครื่องคอมพิวเตอร์ร้านค้าเริ่มทำงาน

4.2.2.4 ใส่จำนวนของสินค้าที่จะอ่านครั้งต่อไปในกรณีที่มีการขายสินค้าชนิดเดียวกันมีจำนวนมากกว่า 1 ชิ้น ถ้าไม่ใส่จำนวนแสดงว่าการอ่านรหัสสินค้าครั้งต่อไปจำนวนของสินค้าคือ 1 ชิ้น

4.2.2.5 รวมรายการซื้อของลูกค้าแต่ละราย พิมพ์ใบเสร็จให้กับลูกค้า จะมีเพียงรายการสินค้าและจำนวนที่ถูกต้องเท่านั้นที่ถูกเก็บข้อมูล และพิมพ์ให้ลูกค้า ส่วนจำนวนอื่น หรือตัวเลือกอื่นๆ เช่น รายงาน และ การเข้าสู่ระบบ ไม่ต้องนำมาพิมพ์

4.2.3 ร้านค้ามีหลายๆสินค้า และแต่ละสินค้ามีแถบรหัสเป็นของตนเองซึ่งไม่เหมือนกัน ลูกค้าจะนำตะกร้าที่มีสินค้ามาที่พนักงานเก็บเงินที่อยู่ที่เครื่องโพสต์ พนักงานเก็บเงินสามารถที่จะใส่จำนวนของสินค้าได้ เพื่อลดการอ่านรหัสสินค้าชนิดเดียวที่ถูกซื้อเป็นจำนวนมากกว่า 1 ชิ้น เครื่องโพสต์ จะพิมพ์ผลรวมของการขายทั้งหมดเป็นใบเสร็จให้กับลูกค้า พนักงานเก็บเงินจะฉีกกระดาษพิมพ์ออก และส่งให้ลูกค้า พนักงานเก็บเงินนำสินค้าใส่ถุงและรับเงิน และ ทอนเงิน จากลูกค้าพร้อมกล่าวขอบคุณ

4.2.4 เพื่อที่จะพิมพ์ใบเสร็จแต่ละรายการของใบเสร็จ เครื่องโพสต์จะส่งรายการขายบอกไปยังเครื่องคอมพิวเตอร์คลังสินค้าเกี่ยวกับบัญชีรายการแต่ละครั้ง เครื่องโพสต์แต่ละตัวจะต้องส่งรหัสสินค้า และ จำนวนสินค้าไปยังเครื่องคอมพิวเตอร์ร้านค้า ซึ่งจะผ่านค่าเหล่านี้ไปยังเครื่องคอมพิวเตอร์คลังสินค้า เครื่องคอมพิวเตอร์ร้านค้าจะรับราคาต้นทุนของสินค้า, ประเภทภาษี, ชื่อสินค้า จากนั้นเครื่องคอมพิวเตอร์ร้านค้าจะเก็บค่าเอาไว้ ส่งกลับไปให้ เครื่องโพสต์เพื่อที่จะพิมพ์ลงในใบเสร็จอีกที เครื่องคอมพิวเตอร์ร้านค้า ยังคำนวณราคาขายที่บวกกำไรแล้วของสินค้าแต่ละชนิดส่งไปยังเครื่องโพสต์ จากนั้นเครื่องโพสต์จะคำนวณจำนวนของราคาสินค้าทั้งหมด และพิมพ์ออกมาเป็นใบเสร็จให้กับลูกค้า

4.2.5 เครื่องคอมพิวเตอร์ร้านค้า จะเก็บการทำงานทั้งหมดของการขายสินค้าตลอดทั้งวันจากเครื่องโพสต์ ผู้จัดการของร้านค้า สามารถที่จะถามถึงรายงานประจำวัน และสามารถที่จะรายงานการขายสินค้าตั้งแต่เครื่องคอมพิวเตอร์ร้านค้าถูกเปิด (ไม่จำเป็นเสมอไปที่เครื่องโพสต์สามารถเก็บข้อมูลเองได้)

4.2.6 เครื่องโพสต์เก็บบันทึกราคาขายรวมและราคาขายรวมภาษีตั้งแต่เวลาเริ่มแรกเข้าสู่ระบบ และพนักงานเก็บเงินสามารถถามถึงราคาของสินค้าที่ขายไปทั้งหมด ยกเว้นขณะกำลังขายสินค้า

4.2.7 การคำนวณภาษีจะคำนวณที่เครื่องคอมพิวเตอร์ร้านค้า เพื่อที่จะให้อำนาจการตัดสินใจอยู่ที่ระดับนี้ ผู้จัดการสาขาสามารถตั้งภาษีขึ้นมาเป็นมาตรฐานเพื่อที่จะใช้คำนวณหาภาษีกับสินค้าที่สามารถคิดภาษีได้ ภาษีต่อสินค้าอาจเป็น 0% สำหรับสินค้าที่งดเว้นภาษี หรืออาจคิดเต็มราคา ภาษีจะถูกคำนวณตอนที่สินค้ารวมเสร็จแล้ว ไม่คิดแยกเป็นชิ้น ๆ เพื่อหลีกเลี่ยงค่าผิดพลาด

4.2.8 สำหรับสินค้าแต่ละชนิดจะมีทั้งมีภาษี หรือ ไม่มีภาษี และผลรวมราคาของสินค้าที่คิดภาษีได้ทั้งหมดจะถูกคูณด้วย ค่าคงที่ที่เป็ นเปอร์เซ็นต์ เพื่อคิดเป็นภาษี ในชีวิตจริงแต่ละร้านค้าอาจมีการคำนวณภาษี แต่การคำนวณภาษีจะแตกต่างกันไปเหตุผลของการคำนวณ ภาษีทั้งหมดของสินค้าแต่ละชนิดที่ถูกซื้อด้วยจำนวนมาก ๆ การคิดก็จะให้ผลที่แตกต่างกันไป

4.3 การวิเคราะห์และออกแบบออปเจค(Object Analysis and Design)

การออกแบบในระดับนี้ จะเป็นการกำหนด ตัวแปรท้องถิ่นบางตัว(Local variable) หรือ สถานะของออปเจค และเป็นการแสดงว่าข้อมูลนั้น มีเพียงพอในการสร้างระบบงานทั้งหมด เราจะแสดงถึงรายละเอียดต่างๆ ของออปเจค และแสดงความสัมพันธ์ของออปเจคเหล่านี้

ในการออกแบบระดับนี้ จะมีการอ้างถึง CORBA เพียงเล็กน้อย ถึงแม้ว่าขั้นตอนนี้จะเป็นการแสดงถึงตัวแปรสถานะ(State Variable) ซึ่งสิ่งนี้คือ IDL attributes และแสดงให้ชัดเจนถึงสถานะการทำงาน (methods) ที่มาใช้งานตัวแปรเหล่านั้น การวิเคราะห์ออกแบบนี้จะเป็นผลทำให้เหมือนกับระบบถูกกระจายออกไป ซึ่งสิ่งนี้ต้องมาพิจารณาว่าออปเจคใดควรหรือไม่ที่จะกระจายออกไป การเลือกที่จะกระจายออปเจคหรือไม่ต้องคำนึงถึง ความหนาแน่นในการใช้เครือข่ายต้องให้มีความหนาแน่นน้อย แอพพลิเคชั่นทั้งหมดที่เป็นแบบกระจายจะต้องคำนึงถึงความเหมาะสมว่าลักษณะของตัวออปเจค ถ้าเป็นออปเจคที่มีการเปลี่ยนแปลงบ่อยๆ ก็ไม่ควรที่จะกระจายออกไป สิ่งนี้เป็นประสิทธิภาพในการทำงานของแอพพลิเคชั่น

4.3.1 รายละเอียดของออปเจค (Object Definition)

ในการวิเคราะห์ปัญหาจะแบ่งปัญหาทั้งหมดออกเป็นออปเจคให้สอดคล้องกับการทำงานในชีวิตจริงให้มากที่สุด เพื่อที่จะทำให้เกิดเป็นระบบ POS ที่ทำงานได้จริงๆ จากการออกแบบ ได้ออปเจคดังนี้

จากรูป 4.2 แสดงถึงความสัมพันธ์ของออปเจคที่ได้จากการวิเคราะห์ และ การเชื่อมโยงออปเจคเข้าด้วยกัน แบ่งออปเจคออกเป็น 3 กลุ่มคือ

กลุ่มแรก อยู่ที่เครื่องคอมพิวเตอร์สำนักงานคลังสินค้า มีออปเจค Depot ทำหน้าที่จัดการสินค้าคงคลัง และการค้นหาข้อมูลในฐานข้อมูล

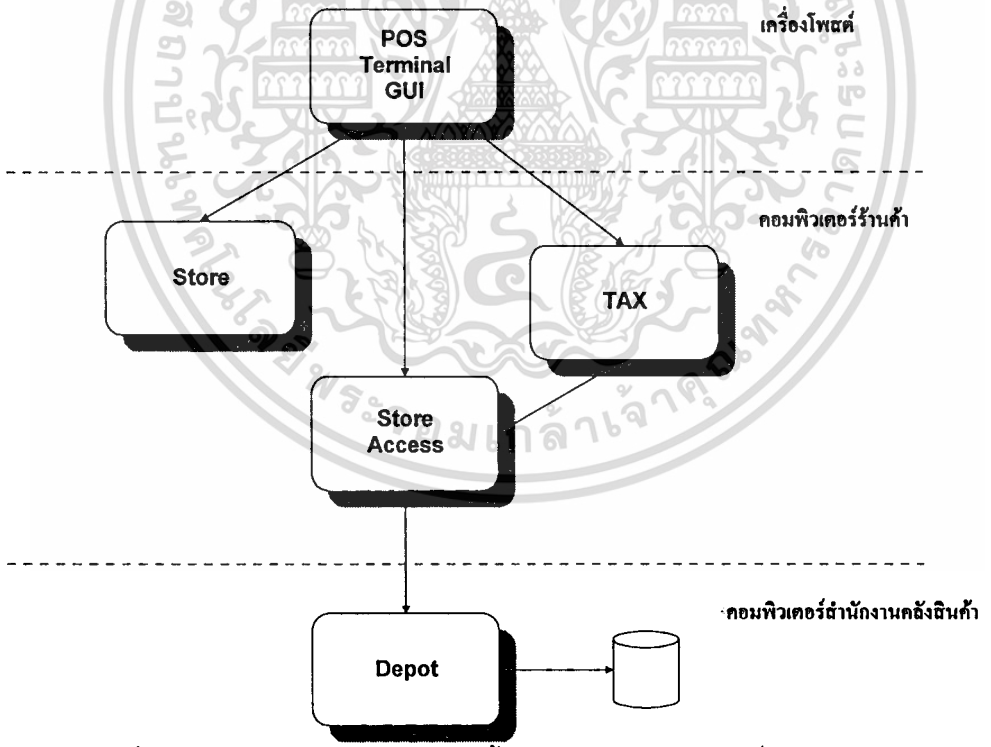
กลุ่มที่สอง อยู่ที่เครื่องเครื่องคอมพิวเตอร์ร้านค้า ประกอบด้วย 3 ออปเจคคือ Store, Tax, StoreAccess ที่มีการเรียกใช้ซึ่งกันและกัน จึงไม่สามารถแยกทั้ง 3 ออปเจคออกจากกันได้เนื่องจากต้องการลดความหนาแน่นในการใช้เครือข่าย

ออปเจค Store ทำหน้าที่ดูแลการทำงานที่เกิดขึ้นทั้งหมดบนเครื่องคอมพิวเตอร์ร้านค้า รวมถึงการเข้าสู่ระบบของเครื่องโพสต์ และบันทึกผลรวมของการขายสินค้าของร้านค้า

ออปเจค Tax ทำหน้าที่คำนวณภาษี กำหนดอัตราภาษี และกำหนดชนิดของสินค้าที่สามารถคำนวณภาษีได้

ออปเจค StoreAccess ทำหน้าที่ติดต่อกับออปเจค Depot ในการสอบถามข้อมูลสินค้าให้กับเครื่องโพสต์

กลุ่มที่สาม อยู่ที่เครื่อง โพสต์ ทำหน้าที่เสมือนเป็นอินพุทและเอาท์พุทของระบบ รับรายการซื้อจากลูกค้า และ แสดงข้อมูลการขายให้กับลูกค้า ประกอบด้วยออปเจค POSTerminal GUI



รูปที่ 4.2 ความสัมพันธ์ระหว่างออปเจคทั้งหมดของระบบขายปลีกที่เป็นสาขาย่อย

ออปเจคทั้งห้าออปเจคมีสถานะและหน้าที่การทำงานที่แตกต่างกัน ตามจุดประสงค์ของการออกแบบมีรายละเอียดดังนี้

4.3.1.1 ออปเจค โพสต์ เทอร์มินอล จียูไอ (POS Terminal GUI Object)

คำอธิบาย (Description)

ออปเจคโพสต์เทอร์มินอล เป็นออปเจคที่เป็นเสมือนอินพุทและเอาท์พุทของระบบ ทำหน้าที่รับรายการซื้อสินค้าจากลูกค้าเข้าสู่ระบบ, แสดงผลรวมของรายการซื้อที่ลูกค้าต้องชำระ, พิมพ์ใบเสร็จให้กับลูกค้า และออกรายงานการขายสินค้าของเครื่องโพสต์และของร้านค้า ในระบบงานจริงอินพุทและเอาท์พุทเป็นอุปกรณ์ที่เป็นแผงคีย์, เครื่องอ่านแถบรหัส, จอแสดงผล(Monitor) และเครื่องพิมพ์ แต่ในการพัฒนาโปรแกรมตัวอย่างของโปรเจคนี้เพื่อให้ง่ายในการพัฒนาและลดการหาอุปกรณ์อินพุทและเอาท์พุทจึงได้จำลองอุปกรณ์อินพุทและเอาท์พุท โดยการแทนแผงคีย์ด้วยการใช้เมาท์(mouse)คลิก ปุ่ม(Botton Control) บนจอแสดงผล แทนเครื่องอ่านแถบรหัสด้วยการป้อนตัวเลขรหัสสินค้าลงในช่องที่ใช้แก้ไข(Edit Control) และใช้การแสดงผลบนจอภาพแทนการแสดงผลด้วยเครื่องพิมพ์ ฉะนั้นออปเจคนี้จึงถูกออกแบบให้รองรับอินพุทและเอาท์พุทที่เป็นกราฟฟิค (Graphic User Interface:GUI) โดยเฉพาะ

ตัวแปรสถานะการทำงาน (State variable)

storeRef: เก็บตัวชี้ที่ใช้อ้างถึงออปเจค Store

storeAccessRef: เก็บตัวชี้ที่ใช้อ้างถึงออปเจค StoreAccess

taxRef: เก็บตัวชี้ที่ใช้อ้างถึงออปเจค Tax

POSId: หมายเลขเครื่องโพสต์

StoreId: หมายเลขร้านค้าที่เครื่องโพสต์อยู่

ItemBarcode: รหัสสินค้าที่ได้จากการอ่านแถบรหัส

ItemQuantity: จำนวนของการซื้อ ปกติจะมีค่าเป็น 1 เสมอ

IInfo: เป็นตัวแปรโครงสร้างที่คืนค่ามาจากคลังสินค้า เก็บข้อมูลของสินค้ามีรายละเอียดดังนี้

- Item รหัสสินค้า เหมือนกับ itemBarcode
- Itemtype ชนิดภาษีของสินค้า มี 3 ชนิดคืออาหาร(FOOD), เสื้อผ้า(CLOTHES), อื่นๆ(OTHER) ใช้ในการหาภาษีของออปเจค Tax
- Itemcost ราคาต้นทุนต่อหน่วย ใช้ในการคำนวณราคาขาย
- Name ชื่อสินค้า
- Quantity จำนวนสินค้าที่สามารถซื้อได้(ต้องไม่เกินจำนวนที่มีในคลังสินค้า) ถ้าจำนวนที่ซื้อมากกว่าในคลังสินค้าค่า Quantity จะมีค่าเท่ากับค่าในคลังสินค้า แต่ถ้าจำนวนที่ซื้อน้อยกว่าในคลังสินค้าค่า Quantity จะมีค่าเท่ากับจำนวนที่ซื้อ

ItemPrice: ราคาขายที่จะขายลูกค้า เป็นราคาที่บวกกำไร คำนวณจากฟังก์ชั่น "FindPrice" ของออปเจค StoreAccess

ItemTaxPrice: ราคาของสินค้าที่สามารถนำมาคิดภาษีได้ หาจากฟังก์ชั่น "FindTaxablePrice" ของออปเจค Tax โดยพิจารณาจากชนิดของสินค้า ถ้าเป็นเสื้อผ้าและอาหาร สินค้าชนิดนั้นไม่สามารถนำมาคิดภาษีได้ (0%) แต่ถ้าเป็นอย่างอื่นนำมาคิดภาษีได้เต็มราคา(100%) ซึ่ง ItemTaxPrice ของทุกรายการจำนำไปคิดภาษีต่อไป

ItemExtension: ราคาขายต่อหนึ่งรายการ คำนวณจาก ราคาขายต่อหน่วย * จำนวน (ItemPrice * Quantity)

SaleSubTotal: ราคาขายรวมของสินค้าทุกรายการ (บวกกำไรแล้ว) หรือผลรวมของ itemExtension

SaleTaxableSubTotal: ราคารวมของสินค้าที่สามารถนำมาคิดภาษีได้ จะต้องมีค่าน้อยกว่าหรือเท่ากับ SaleSubTotal เสมอ

SaleTax: ภาษีของการขายสินค้าครั้งนี้ ได้จากการคืนค่าฟังก์ชั่น "CalculateTax" ของออปเจค Tax

POSTotal: ผลรวมราคาขายทั้งหมดของเครื่องโพสต์ตั้งแต่เข้าสู่ระบบครั้งล่าสุด

POSTaxTotal: ผลรวมภาษีทั้งหมดของเครื่องโพสต์ตั้งแต่เข้าสู่ระบบครั้งล่าสุด (SaleSubTotal + SaleTax)

NetTotal: ผลรวมสุทธิของลูกค้าแต่ละราย (จำนวนเงินที่ลูกค้าต้องชำระ)

QtyTotal: จำนวนชิ้นของสินค้ารวมที่ลูกค้าแต่ละรายซื้อ

การทำงาน (Operation)

Initialization

คำอธิบาย (Descrription): ใช้กำหนดค่าเริ่มต้นของออปเจค POSTerminal

ตัวแปรผ่านค่า (Parameter): ไม่มี

ขั้นตอนการปฏิบัติ: (Implementation):

1. อ่านค่าหมายเลขเครื่องโพสต์จากแฟ้มที่เก็บค่าคงที่ของระบบ (Config file) มาไว้ในตัวแปร POSId
2. สร้างตัวชี้ออปเจค Store เก็บใน StoreRef และออปเจค Tax เก็บใน TaxRef

ถูกเรียกจาก (Called From): เมื่อออปเจค POSTerminal เริ่มต้นทำงาน

Login

คำอธิบาย: ออปเจค POSTerminal ใช้แจ้งออปเจค Store เมื่อมีพนักงานใหม่กำลังจะเข้าสู่ระบบ เพื่อที่ออปเจค Store จะสร้างออปเจค StoreAccess คืนกลับมาให้ การทำงานนี้จะเรียกใช้ฟังก์ชั่น Login ของออปเจค Store

ตัวแปรผ่านค่า: ไม่มี

ขั้นตอนการปฏิบัติ:

1. กำหนดค่าให้ ItemQuantity เป็น 1 และให้ค่าอื่นๆเป็นศูนย์
2. ลงทะเบียนเครื่องโพสต์ใหม่เข้าสู่ระบบ โดยการเรียกฟังก์ชั่น "Login" ของออปเจค Store ใช้ StoreRef ในการอ้างถึงฟังก์ชั่น ผ่านค่าหมายเลขเครื่องโพสต์ (POSId) ให้กับฟังก์ชั่น ฟังก์ชั่นนี้จะสร้างออปเจค StoreAccess ขึ้นใหม่ และคืนค่าตัวชี้กลับมาให้ ออปเจค POSTerminal จะใช้ออปเจค StoreAccess ในการติดต่อกับสำนักงานคลังสินค้า

ถูกเรียกจาก: คำสั่งเมนู POS/Connect to store computer

PrintPOSSalesSummary

คำอธิบาย: แสดงผลรวมการขายสินค้า และ ผลรวมภาษีของเครื่องโพสต์เครื่องนั้นตั้งแต่เริ่มต้นเข้าสู่ระบบ ทุกครั้งที่เข้าสู่ระบบครั้งใหม่ ผลรวมทั้งสองรายการจะมีค่าเริ่มต้นเป็นศูนย์เสมอ

ตัวแปรผ่านค่า: ไม่มี

ขั้นตอนการปฏิบัติ:

1. ฟังก์ชั่นนี้จะไม่ถูกปฏิบัติถ้า ItemBarcode หรือ SaleSubtotal ไม่เท่ากับศูนย์แสดงว่าเครื่องโพสต์อยู่ในระหว่างการขายสินค้า
2. ดึงค่า POSTatol และ POSTaxTotal ของออปเจค POSTerminal แสดงค่าทั้งสองในไดอะล็อก

ถูกเรียกจาก: คำสั่งเมนู POS/Display pos summary

PrintStoreSaleSummary

คำอธิบาย: แสดงผลรวมการขายสินค้า และ ผลรวมภาษีทั้งหมดของร้านค้า โดยแสดงรายละเอียดการขายของเครื่องโพสต์แต่ละเครื่องที่เข้าใช้งานระบบ

ตัวแปรผ่านค่า: ไม่มี

ขั้นตอนการปฏิบัติ:

1. ฟังก์ชั่นนี้จะไม่ถูกปฏิบัติ ถ้า ItemBarcode และ SaleSubTotal ไม่เท่ากับศูนย์
2. ออปเจค POSTerminal ดึงค่าผลรวมการขาย และ ผลรวมภาษีของร้านจากออปเจค Store โดยใช้ StoreRef อ้างถึงฟังก์ชั่น "Totals" ของออปเจค Store
3. แสดงค่าผลรวมทั้งสองที่ได้จากออปเจค Store ในไดอะล็อก
4. สอบถามค่าผลรวมการขาย และ ผลรวมภาษีของแต่ละเครื่องโพสต์ โดยใช้ StoreRef ในการอ้างถึงฟังก์ชั่น "GetPOSTotals" ของออปเจค

5. แสดงผลรวมการขาย และผลรวมภาษีของแต่ละเครื่องโพสต์ Store ฟังก์ชั่นจะส่งลิสต์ข้อมูลของเครื่องโพสต์กลับมาให้ ในไดอะล็อก โดยระบุหมายเลขเครื่องโพสต์ (POSId) ให้ชัดเจน

ถูกเรียกจาก: คำสั่งเมนู POS/Display store summary

ItemQuantity

คำอธิบาย: กำหนดจำนวนของสินค้าที่จะอ่านแถบรหัสครั้งต่อไป , ถูกเรียกใช้เมื่อการขายสินค้าชนิดเดียวกันจำนวนสินค้ามากกว่า 1 ชิ้น เพื่อที่จะลดจำนวนครั้งในการอ่านให้เหลือเพียงครั้งเดียว ฟังก์ชั่นนี้จะกำหนดค่าของตัวแปร ItemQuantity ให้กับจำนวนสินค้าที่จะอ่านแถบรหัสครั้งต่อไป หลังจากที่มีการอ่านแถบรหัส ItemQuantity จะถูกกำหนดให้เป็น 1 เหมือนเดิม

ตัวแปรผ่านค่า:

ตัวแปรเข้า (Input): จำนวนของสินค้า โดยการอ่านค่าจากไดอะล็อก

ขั้นตอนการปฏิบัติ:

1. กำหนดตัวแปร ItemQuantity ให้มีค่าเท่ากับจำนวนสินค้าที่จะอ่านครั้งต่อไป (อ่านค่าจากไดอะล็อก)

ถูกเรียกจาก: การกดปุ่ม Quantity บนไดอะล็อก

SendBarcode

คำอธิบาย: ออปเจค POSTerminal ใช้ส่งแถบรหัสที่อ่านได้จากเครื่องอ่านแถบรหัสไปยังออปเจค StoreAccess (ซึ่งถูกส่งให้กับออปเจค Depot อีกทอดหนึ่ง) เพื่อสอบถามรายละเอียดของสินค้าเกี่ยวกับชื่อสินค้า, ราคาสินค้า และราคาที่สามารถคิดภาษีได้ นำมาแสดงให้ลูกค้าทราบ

ตัวแปรผ่านค่า:

ตัวแปรเข้า:

"ItemBarcode" รหัสสินค้าที่ได้จากการป้อนแถบรหัสช่อง "Barcode" ในไดอะล็อก

"ItemQuantuity" จำนวนสินค้าที่อ่าน

ตัวแปรออก (Output):

"ItemPrice" ราคาขายต่อหน่วย

"ItemIaxPrice" ราคาที่สามารถคิดภาษีได้ต่อหน่วย

"IInfo" ข้อมูลของสินค้า

ขั้นตอนการปฏิบัติ:

1. อ่านค่าแถบรหัสในไดอะล็อก มายังตัวแปร ItemBarcode

2. สอบถามข้อมูลสินค้าโดยเรียกใช้ฟังก์ชั่น "FindPrice" ของออปเจค StoreAccess ผ่านค่าตัวแปร ItemQuantity และ ItemBarcode ให้กับฟังก์ชั่น และตัวแปรออกของฟังก์ชั่นจะคืนค่าราคาขาย (ItemPrice), ราคาที่สามารถคิดภาษี (ItemTaxPrice) และข้อมูลของสินค้า (IInfo)
3. ถ้าการทำงานในข้อ 2 เกิดข้อยกเว้น (Exception) เนื่องจากหาแถบรหัสในฐานข้อมูลไม่พบ (BarcodeNotFound) ฟังก์ชั่นจะไม่มีการผ่านตัวแปรออกมาให้ และจะกำหนดค่า ItemQuantity เป็น 1 และจบการทำงานของฟังก์ชั่น แต่ถ้าไม่เกิดข้อยกเว้นจะทำงานต่อไป
4. คำนวณหาราคาขายของสินค้า (ItemExtension) จากราคาขายต่อหน่วยคูณกับจำนวนสินค้าที่ซื้อ เพื่อที่จะใช้ในการหาผลรวมครั้งสุดท้าย (SaleSubTotal) ของการขายสินค้า และปรับปรุงผลรวมของสินค้า (SaleSubTotal)
5. แสดงรายละเอียดสินค้าบนจอแสดงผลให้ลูกค้าทราบ ประกอบด้วย ชื่อสินค้า, รหัสสินค้า, จำนวน, ราคาต่อหน่วย และราคารวม
6. คำนวณหาราคาสินค้าที่สามารถคิดภาษีได้ต่อสินค้าหนึ่งรายการ จากราคาสินค้าที่สามารถคิดภาษีได้ต่อหน่วย (ItemTaxPrice) คูณกับจำนวนสินค้า (ItemQuantity) และปรับปรุงผลรวมของสินค้าที่คิดภาษีได้ (SaleTaxableSubTotal)
7. กำหนดให้ ItemQuantity เป็น 1

ถูกเรียกจาก: การกดปุ่ม "ScanBarcode" ในไดอะล็อก

EndSale

คำอธิบาย: ออปเจค POSTerminal ใช้บอกถึงการสิ้นสุดการขายของลูกค้าหนึ่งราย โดยคำนวณหาภาษีสินค้าจากราคาสินค้าที่คิดภาษีได้ทั้งหมด, พิมพ์ใบเสร็จของการขายให้กับลูกค้า และรายงานการขายไปยัง Store ว่าลูกค้ามีการซื้อเท่าไร ภาษีที่คิดเท่าไร เพื่อเก็บเป็นข้อมูลของร้านค้า

ตัวแปรผ่านค่า: ไม่มี

ขั้นตอนการปฏิบัติ:

1. แสดงผลรวมราคาของสินค้า และ ผลรวมของสินค้าที่สามารถคิดภาษีได้ ให้ลูกค้าทราบ
2. คำนวณหาภาษีสินค้าจากการเรียกใช้ฟังก์ชั่น CalculateTax ของออปเจค Tax ผ่านผลรวมสินค้าที่คิดภาษีได้ให้กับฟังก์ชั่น ฟังก์ชั่นจะคืนค่าของภาษี (SaleTax) กลับมาให้ อัตราภาษีที่ใช้ในการคำนวณถูกกำหนดจากตัวแปร TaxRate ของออปเจค Tax ค่านี้สามารถถูกแก้ไขได้จากผู้จัดการของร้านค้า
3. คำนวณหาราคาขายสุทธิซึ่งเป็นราคาขายที่รวมภาษีแล้วจากผลรวมราคาของสินค้า (SaleSubTotal) บวกกับภาษีที่ได้จากการคืนค่ากลับของฟังก์ชั่น CalculateTax
4. แสดงภาษีและราคาขายสุทธิ บนจอภาพให้ลูกค้าทราบ

5. ปรับปรุงยอดขายของร้านค้าโดยเรียกฟังก์ชั่น "UpdateStoreTotals" ของออปเจค Store ผ่านค่าของผลรวมราคา และภาษีไปให้ ร้านค้าจะนำไปปรับปรุงยอดขายรวม เพื่อใช้ในการถามถึงยอดต่อไป
6. กำหนดให้ ItemBarcode , SaleSubTotal และ SaleTaxableSubTotal เป็นศูนย์
7. กำหนดให้ ItemQuantity เป็น 1

ถูกเรียกจาก: การกดปุ่ม "End of Sale" ในไดอะล็อก

4.3.1.2 ออปเจค สโตน (Store Object)

คำอธิบาย

ออปเจค Store ทำหน้าที่รับการเข้าสู่ระบบของเครื่องโพสต์ทุกเครื่องในร้านค้า คอยดูแลออปเจค POS Terminal เก็บรายการของการขายสินค้า และรายงานการขายที่เกิดขึ้นในร้านค้า เมื่อเครื่องโพสต์ถามถึง

ตัวแปรสถานะการทำงาน

Total : เป็นตัวแปรโครงสร้าง เก็บผลรวมการขายสินค้าของร้านค้าทั้งหมดตั้งแต่เป็นเครื่องคอมพิวเตอร์ร้านค้าเริ่มทำงาน

StoreTotal ผลรวมของการขายทั้งหมดของร้านค้า

StoreTaxTotal ผลรวมของภาษีทั้งหมด

POSlist: เป็นลิสต์ของตัวแปรโครงสร้างที่เก็บรายละเอียดเกี่ยวกับออปเจค POSTerminal ที่การเข้าสู่ระบบตั้งแต่เครื่องคอมพิวเตอร์ร้านค้าเริ่มทำงาน เช่น หมายเลขเครื่องโพสต์, ผลรวมการขาย และ ผลรวมภาษีของแต่ละเครื่องโพสต์

การทำงาน

Initialization

คำอธิบาย: ใช้กำหนดค่าเริ่มต้นออปเจค Store เมื่อเครื่องคอมพิวเตอร์ร้านค้าเริ่มทำงาน

ตัวแปรผ่านค่า: ไม่มี

ขั้นตอนการปฏิบัติ:

1. กำหนดให้ผลรวมการขายสินค้า และผลรวมภาษี มีค่าเป็นศูนย์
2. กำหนดให้ลิสต์ของโพสต์เป็นลิสต์ว่าง (ไม่มีเครื่องโพสต์เครื่องใดเข้าสู่ระบบ)

ถูกเรียกจาก: ถูกเรียกเมื่อเริ่มต้นใช้ออปเจค Store

Login

คำอธิบาย: แจ้งว่ามีเครื่องโพสต์ใหม่กำลังจะเข้าสู่ระบบให้ร้านค้าทราบ และเริ่มเก็บรายการขายสินค้าของเครื่องโพสต์ และทำการสร้างออปเจค StoreAccess ขึ้นเพื่อให้ออปเจค POS ใช้ในการติดต่อกับออปเจค Depot

ตัวแปรผ่านค่า:

ตัวแปรเข้า: "Id" หมายเลขของเครื่องโพสต์

ตัวแปรคืนกลับ: ตัวชี้ที่ใช้อ้างถึงออปเจค StoreAccess

ขั้นตอนการปฏิบัติ:

1. ถ้าไม่มีโครงสร้างของ POSTerminal อยู่ในลิสต์ (POS List)
   - สร้างออปเจค StoreAccess
   - เพิ่มโครงสร้างข้อมูลของ POSInfo เข้าในลิสต์ของโพสต์ โดยการกำหนดหมายเลขเครื่องโพสต์ กำหนดผลรวมของการขาย และผลรวมภาษีเป็นศูนย์ และกำหนดตัวชี้ที่ใช้อ้างถึงออปเจค StoreAccess ลงในโครงสร้างใหม่นี้ด้วย
2. ถ้ามีโครงสร้างของ POSTerminal อยู่ในลิสต์อยู่แล้ว (คือมีหมายเลขเครื่องโพสต์ ในลิสต์โพสต์ตรงกับหมายเลขเครื่องโพสต์ที่ผ่านค่ามาให้) จะทำการกำหนดผลรวมให้เป็นศูนย์เท่านั้น
3. คืนค่าตัวชี้ออปเจค StoreAccess ให้กับออปเจค POSTerminal

ถูกเรียกจาก: ฟังก์ชั่น "Login" ของออปเจค POSTerminal

GetPOSTotals

คำอธิบาย: อ่านค่าสถานะของออปเจค POSTerminal ที่ที่มีการเข้าใช้งานระบบ จากลิสต์ของโพสต์ในร้านค้าที่มีการเข้าสู่ระบบ

ตัวแปรผ่านค่า:

ตัวแปรออก: POSData สิสต์สถานะของออปเจค POSTerminal

ขั้นตอนการปฏิบัติ:

1. กำหนดว่า POSData ด้วยค่าปัจจุบันของ POSlist

ถูกเรียกจาก: ฟังก์ชั่น "PrintStoreSalesSummary" ของออปเจค POSTerminal

UpdateStoreTotals

คำอธิบาย: ปรับปรุงค่าผลรวมการขาย และผลรวมภาษีของเครื่องโพสต์ และของร้านค้า โดยเพิ่มรายการขายและรายการภาษีใน SaleTotal และ SaleTaxTotal

ตัวแปรผ่านค่า:

ตัวแปรเข้า: หมายเลขเครื่องโพสต์ (Id), ราคาขาย (Price) และภาษี (Taxes)

ขั้นตอนการปฏิบัติ:

1. ค้นหาโครงสร้างของโพสต์ในโพสต์ลิสต์ ที่มีหมายเลขเครื่องตรงกับตัวแปร Id
   - ถ้าไม่พบ แสดงว่าลิสต์มีข้อมูลไม่ถูกต้อง เนื่องจากเครื่องโพสต์ที่เข้าสู่ระบบจะต้องมีโครงสร้างอยู่ในโพสต์ลิสด้วย ฉะนั้นแสดงว่าเกิดผิดพลาดขึ้น ให้จบการทำงานของฟังก์ชั่น
   - ถ้าพบ ให้เพิ่ม "Price" และ "Tax" ในฟิลด์ของโครงสร้างที่ค้นพบ
2. เพิ่ม "Price" ใน StoreTotal และเพิ่ม "Taxes" ลงใน StoreTaxTotal

ถูกเรียกจาก: ออปเจค POSTerminal ฟังก์ชั่น "EndOfSale" ของออปเจค POSTerminal

4.3.1.3 ออปเจค สโตนแอ็คเซส (StoreAccess)

คำอธิบาย

ออปเจค StoreAccess เป็นออปเจคที่เป็นสื่อกลางการเชื่อมต่อระหว่างเครื่องโพสต์ กับสำนักงานคลังสินค้าทำหน้าที่เชื่อมออปเจค POSTerminal กับออปเจค Depot ทำให้ออปเจค POSTerminal สามารถเข้าถึงฐานข้อมูลได้ จากฟังก์ชั่น Login ของออปเจค Store จะสร้างออปเจค StoreAccess คืนให้กับออปเจค POSTerminal แต่ละตัวที่จะเข้าสู่ระบบ

ตัวแปรสถานะการทำงาน

depotRef: เก็บตัวชี้ที่ใช้อ้างถึงออปเจค Depot

taxRef: เก็บตัวชี้ที่ใช้อ้างถึงออปเจค Tax

StoreMarkup: เปอร์เซ็นต์ ที่เครื่องโพสต์เครื่องนั้นจะใช้ในการคิดกำไรที่ต้องการ สามารถแก้ไขอัตรากำไรได้จากผู้จัดการร้านค้า

Store_id: หมายเลขร้านค้าที่จะใช้ในการติดต่อกับออปเจค Depot

การทำงาน

Initialization

คำบรรยาย:

ใช้กำหนดค่าเริ่มต้นของออปเจค StoreAccess

ขั้นตอนการปฏิบัติ:

1. สร้างตัวชี้ออปเจค Depot เก็บใน depotRef, ออปเจค Tax เก็บใน taxRef และกำหนดค่า storeMarkup

ถูกเรียกจาก: ฟังก์ชั่น "Login" ของออปเจค Store

FindPrice

คำอธิบาย: บอกออปเจค Depot ให้ค้นหาข้อมูลของสินค้าในฐานข้อมูล นำข้อมูลที่ได้มาคำนวณราคาขายจากค่า StoreMarkup และหาราคาที่สามารถคิดภาษีได้จากชนิดของภาษี โดยเรียกฟังก์ชั่น "FindTaxTablePrice" ของออปเจค Tax

ตัวแปรผ่านค่า:

ตัวแปรเข้า: "Item" รหัสของสินค้า

"Quautity" จำนวนของสินค้า

ตัวแปรออก: "ItemPrice" ราคาขายต่อหน่วย

"ItemTaxPrice" ราคาต่อหน่วยที่สามารถคิดภาษีได้

"IInfo" ข้อมูลของสินค้า

กรณียกเว้น (Exception): จะเกิดขึ้นเมื่อหารหัสสินค้าในฐานข้อมูลไม่พบ(BarcodeNotFound)

ขั้นตอนการปฏิบัติ:

1. ค้นหาข้อมูลของสินค้า เรียกฟังก์ชั่น "FindItemInfo" ของออปเจค depot โดยใช้ depotRef ผ่านค่ารหัสสินค้า และจำนวนให้กับฟังก์ชั่น ฟังก์ชั่นคืนค่าข้อมูลของสินค้ากลับมาใหม่
2. ถ้าเกิดข้อยกเว้นกรณีหารหัสไม่พบ ให้จบการทำงานของฟังก์ชั่น แต่ถ้าค้นหาข้อมูลพบให้ทำต่อ
3. คำนวณหาราคาขาย "ItemPrice" จากราคาต้นทุน คูณ กับค่าเปอร์เซ็นต์ของกำไร (cost * Storemarkup)
4. หาราคาที่สามารถคิดภาษีได้ จากเรียกฟังก์ชั่น "FindTaxablePrice" ของออปเจค Tax โดยผ่านค่าชนิดของสินค้าไปให้ ฟังก์ชั่นจะคืนค่าราคาที่คิดภาษีได้กลับมายังตัวแปร ItemTaxPrice

ถูกเรียกจาก: ฟังก์ชั่น "SendBarcode" มองออปเจค POSTerminal

4.3.1.4 ออปเจคแทค (Tax)

คำอธิบาย

ให้บริการเกี่ยวกับภาษีของร้านค้า เช่นหาราคาที่สามารถคิดภาษีได้ของสินค้ารายการนั้น และคำนวณหาภาษีที่ลูกค้าจะต้องจ่าย ในตัวอย่างนี้สินค้าที่ไม่สามารถนำมาคิดภาษีได้คือ เสื้อผ้า และ อาหาร นอกเหนือจากนั้นจะนำมาคิดภาษี 100 % การแยกออปเจค Tax ออกจากออปเจค Store ให้เป็นอิสระเพื่อที่จะให้อำนาจการตัดสินใจในการคิดภาษีอยู่ในระดับของร้านค้า และสามารถแก้ไขอัตราภาษีได้โดยผู้จัดการร้านค้า

ตัวแปรสถานะการทำงาน

Rate: อัตราภาษี

การทำงาน

Initialization

คำอธิบาย: ใช้กำหนดค่าเริ่มต้นเมื่อระบบของร้านค้าเริ่มทำงาน

ตัวแปรผ่านค่า: ไม่มี

ขั้นตอนการปฏิบัติ:

1. กำหนดอัตราภาษีโดยอ่านอัตราภาษีจากแฟ้มข้อมูลของระบบ

ถูกเรียกจาก: เมื่อระบบเริ่มทำงาน หรือออปเจค Tax ถูกสร้างขึ้น

CalculateTax

คำอธิบาย: คำนวณภาษีของสินค้าที่ลูกค้าจะต้องจ่าย

ตัวแปรผ่านค่า:

ตัวแปรเข้า: "TaxableAmount" ราคารวมที่ต้องการที่จะคิดภาษี

ตัวแปรกลับ: ราคาภาษีของสินค้าที่ลูกค้าจะต้องจ่าย

ขั้นตอนการปฏิบัติ:

1. คูณ "TaxableAmount" กับอัตราภาษีของร้านค้า แล้วคืนผลลัพธ์ที่ได้กลับคืนออกไป

ถูกเรียกจาก: ฟังก์ชั่น "EndofSale" ของออปเจค POSTerminal

FindTaxablePrice

คำอธิบาย: หาราคาสินค้าที่สามารถคิดภาษีได้จากชนิดของภาษีของสินค้ารายการนั้นโดยมีรายละเอียดดังนี้ถ้าสินค้ามีชนิดเป็นอาหารและเสื้อผ้า จะไม่นำมาคิดภาษี(คิดภาษี 0%) และถ้าชนิดอื่นๆ จะนำมาคิดภาษีเต็ม(คิดภาษี 100%)

ตัวแปรผ่านค่า:

ตัวแปรเข้า: "ItemPrice" ราคาสินค้าต่อหน่วย

"ItemType" ชนิดสินค้า (FOOD, CLOTHES, OTHER)

ค่าคืนกลับ: "taxablePrice" ราคาที่สามารถคิดภาษีได้

ขั้นตอนการปฏิบัติ:

1. ถ้า "ItemType" เป็น FOOD หรือ CLOTHES จะคืนค่า taxablePrice เป็นศูนย์
2. ถ้า "ItemType" เป็น OTHER จะคืนค่า taxablePrice เป็น "ItemPrice"

ถูกเรียกจาก: ฟังก์ชั่น "FindPrice" ของออปเจค StoreAccess

4.3.1.5 ออปเจคดีพอท (Depot)

คำอธิบาย

ควบคุมสินค้าคงคลังของระบบ และค้นหาข้อมูลเกี่ยวกับสินค้า สำหรับการขายสินค้าของทุกๆ ร้าน เมื่อมีการสอบถามข้อมูลสินค้าเข้ามาจะทำการค้นหาข้อมูลในฐานข้อมูลสินค้าคงคลัง และส่งข้อมูลกลับคืนไปให้สำหรับพิมพ์ออกใบเสร็จให้กับลูกค้า

ตัวแปรสถานะการทำงาน

ออปเจค Depot จะมีฐานข้อมูลสำหรับเก็บรายการของสินค้าทั้งหมดที่มีในระบบฐานข้อมูลประกอบด้วยข้อมูลหลายชนิด ซึ่งมีโครงสร้างดังนี้

barcodekey: รหัสสินค้า

itemName: ชื่อสินค้า

inventoryCount: จำนวนสินค้าที่เหลือใน Stock

itemCost: ต้นทุนของสินค้า

taxType: ชนิดของภาษีสินค้า

การทำงาน

Initiatization

คำอธิบาย: เริ่มต้นหลังจากที่ระบบเริ่มทำงานครั้งแรกที่สำนักงานคลังสินค้า สร้างออปเจคเพื่อคลอบฐานข้อมูลของระบบ ไว้สำหรับการเข้าถึงครั้งต่อไป

ขั้นตอนการปฏิบัติ:

1. สร้างออปเจคของฐานข้อมูล คลอบฐานข้อมูลไว้

ถูกเรียกจาก: เมื่อระบบเริ่มทำงานหรือออปเจค Depot ถูกสร้างขึ้น

FindItemInfo

คำอธิบาย: ค้นข้อมูลของสินค้าในฐานข้อมูลคืนให้กับร้านค้า แต่ถ้าค้นหาไม่พบก็จะไปทำข้อยกเว้นกรณีหาข้อมูลไม่พบ หรือ BarcodeNotFound

ตัวแปรผ่านค่า:

ตัวแปรเข้า: "StoreId" หมายเลขร้านค้าใช้ในการอ้างถึงข้อมูลเฉพาะร้าน

"Item" รหัสสินค้า

"Quantity" จำนวนสินค้าที่ซื้อ

ตัวแปรออก: "Iinfo" ข้อมูลของสินค้าที่สอบถามมา

ข้อยกเว้น: เมื่อค้นหาข้อมูลที่ตรงกับ Item ในฐานข้อมูลไม่พบ (BarcodeNotFound)

ขั้นตอนการปฏิบัติ:

1. ค้นหาข้อมูลในฐานข้อมูลที่มีรหัสสินค้าตรงกับ "Item" ถ้าไม่พบให้ทำข้อยกเว้น แต่ถ้าพบให้ทำต่อไป
2. ลดจำนวนสินค้าคงเหลือของข้อมูลที่ค้นหาพบลงเท่ากับจำนวนที่ซื้อ("Quantity")
3. กำหนดค่าจากฐานข้อมูลลงในตัวแปรออก Iinfo

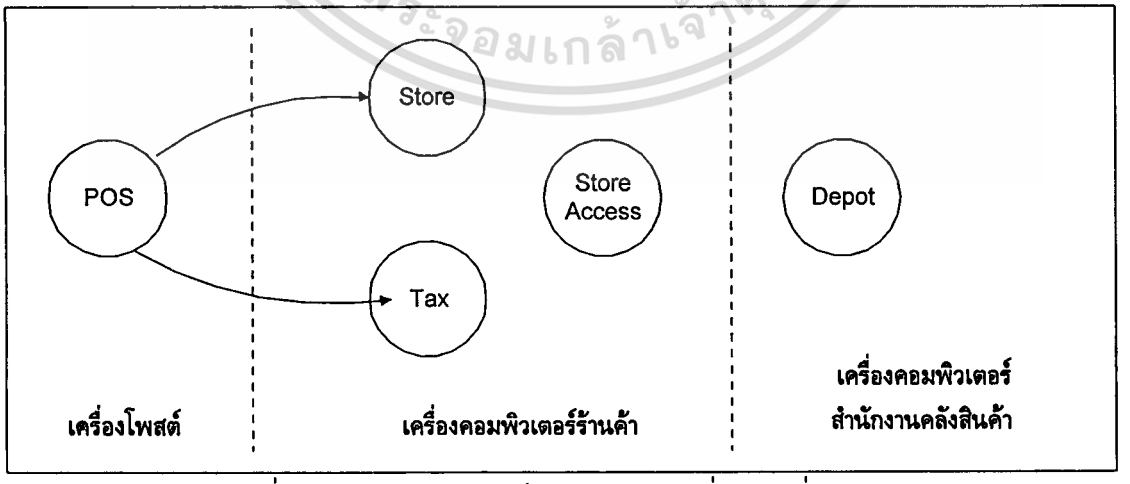
ถูกเรียกจาก: ฟังก์ชั่น "FindPrice" ของออปเจค StoreAccess

4.3.2 การทำงาน และ ความสัมพันธ์ระหว่างออปเจค

ออปเจคทั้งหมดในระบบขายปลีกที่เป็นสาขาย่อยจะมีหน้าที่และการทำงานแตกต่างกันตวัตถุประสงค์ของการออกแบบที่วางไว้ แต่ละออปเจคมีการติดต่อสื่อสารและการเรียกใช้บริการซึ่งกันและกัน การสร้างออปเจคและความสัมพันธ์ของออปเจคอาจเกิดขึ้นตอนเริ่มต้นการทำงานของระบบ หรือ เกิดขณะโปรแกรมกำลังทำงานก็ได้ ซึ่งการทำงานของระบบอาจมีผลทำให้เกิดความสัมพันธ์ระหว่างออปเจคเกิดขึ้น แต่ละการทำงานจะมีความสัมพันธ์ที่แตกต่างกันซึ่งพอจะแยกการทำงานของระบบที่ทำให้เกิดความสัมพันธ์ออกเป็น 3 การทำงานหลักๆ

เมื่อระบบเริ่มทำงานออปเจคต่างๆ ภายในระบบจะถูกสร้างขึ้น โดยมีลำดับการสร้างคือ ออปเจค Depot, Store, Tax และ Pos ส่วนออปเจค StoreAccess จะถูกสร้างขึ้นขณะโปรแกรมกำลังรัน หรือมีการเรียกฟังก์ชั่น "Login" ของออปเจค POS

เมื่อออปเจค POS ถูกสร้างขึ้น จะถือตัวชี้ออปเจค Store ใช้สำหรับติดต่อกับคอมพิวเตอร้านค้าและใช้เก็บสถานะการทำงานทั้งหมดที่เกิดขึ้นกับออปเจค POS และถือออปเจค Tax สำหรับใช้ในการคำนวณภาษีของสินค้า ซึ่งทั้งสองออปเจคจะถูกถือด้วยตัวแปรสถานะการทำงานของออปเจค POS

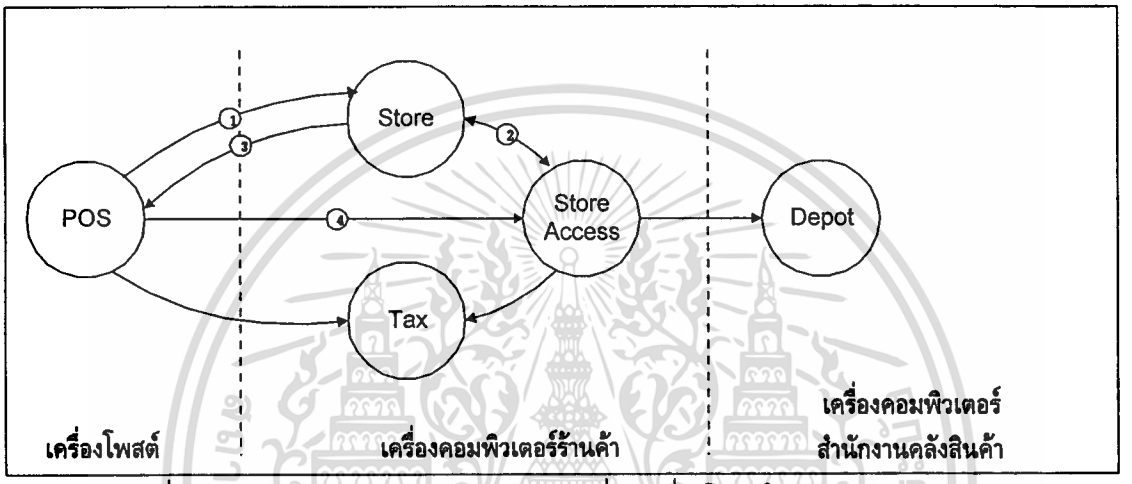


รูปที่ 4.3 แสดงความสัมพันธ์ระหว่างออปเจคเมื่อระบบเริ่มทำงาน

การทำงานหลักของระบบ ที่ทำให้เกิดความสัมพันธ์เกิดขึ้นพอจะแบ่งออกเป็น

4.3.2.1 การทำงานและความสัมพันธ์ระหว่างออปเจคเมื่อเครื่องโพสต์ใหม่เข้าสู่ระบบ(Login)

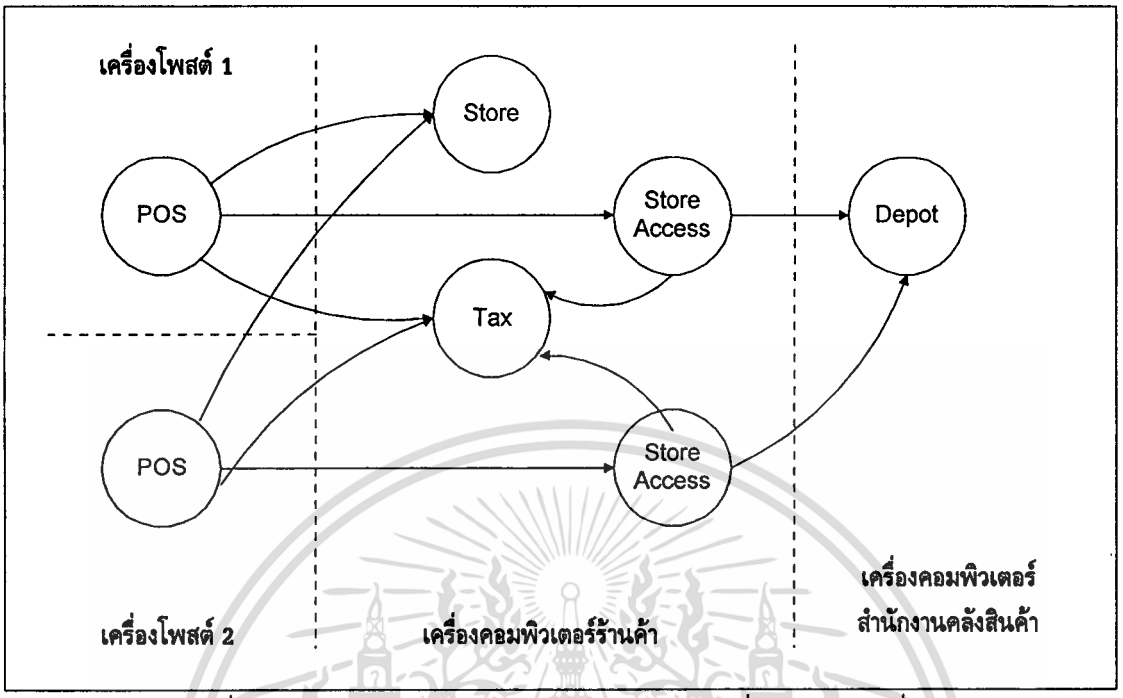
การทำงานนี้จะเกิดขึ้นที่ออปเจค POS เมื่อออปเจค POS ถูกสร้างขึ้นออปเจคนี้จะถือออปเจค Store และ Tax ทั้งสองออปเจคยังไม่สามารถทำให้ออปเจค POS ติดต่อกับออปเจค Depot ได้ หมายความว่าเครื่องโพสต์ไม่สามารถติดต่อกับสำนักงานคลังสินค้าได้ และเมื่อพนักงานเก็บเงินเรียกเข้าสู่ระบบโดยการเรียกฟังก์ชั่น "Login" ของออปเจค POS จะทำให้ออปเจค StoreAccess ถูกสร้างขึ้นและเชื่อมออปเจค POS เข้ากับออปเจค Depot เข้าด้วยกัน



รูปที่ 4.4 แสดงความสัมพันธ์ระหว่างออปเจคเมื่อมีเครื่องโพสต์ใหม่กำลังจะเข้าสู่ระบบ

1. พนักงานเก็บเงินเรียกเข้าสู่ระบบโดยฟังก์ชั่น "Login" ของออปเจค POS ฟังก์ชั่นนี้จะเรียกไปยังฟังก์ชั่น "Login" ของออปเจค Store เพื่อที่จะแจ้งว่ามีเครื่องโพสต์ใหม่กำลังจะเข้าสู่ระบบ และเพื่อสร้างโครงสร้างสำหรับเก็บข้อมูลของเครื่องโพสต์เครื่องนั้นไว้
2. ออปเจค Store ทำการสร้างออปเจค StoreAccess ขึ้นจากคลาส StoreAccess อินพลีเมนท์เทชั่น โดยสร้างให้ออปเจค StoreAccess ถือออปเจคค Tax และ Depot ไว้ จากนั้นคืนค่าตัวชี้ออปเจคกลับไปให้
3. ฟังก์ชั่น Login ของออปเจค Store คืนค่าตัวชี้ออปเจค StoreAccess กลับไปให้กับออปเจค POS
4. ออปเจค POS ถือออปเจค StoreAccess ทำให้สามารถติดต่อกับออปเจค Depot ได้ซึ่งก็หมายความว่าเครื่องโพสต์สามารถติดต่อกับสำนักงานคลังสินค้าได้แล้ว

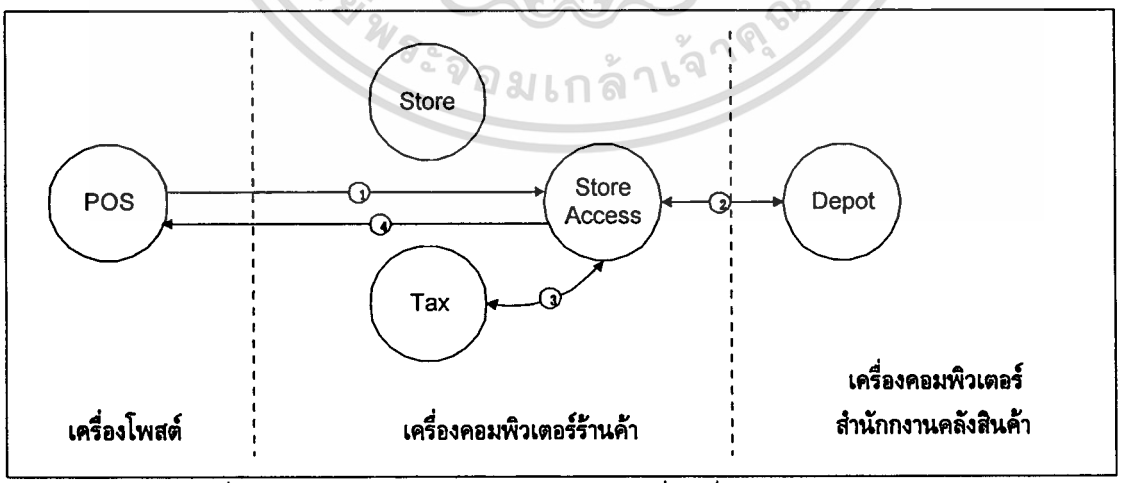
ในกรณีที่มีเครื่องโพสต์หลายเครื่องจะต้องมีออปเจค POS สำหรับเครื่องโพสต์แต่ละเครื่อง ออปเจค POS เหล่านั้นจะมีการใช้ออปเจค Store และออปเจค Tax ร่วมกัน และเมื่อมีการเรียกเข้าสู่ระบบออปเจค StoreAccess ก็จะถูกสร้างขึ้นสำหรับออปเจค POS แต่ละออปเจคด้วย ฉะนั้นออปเจค StoreAccess จะถูกสร้างขึ้นตามจำนวนของออปเจค POS ถ้ามีการเรียกเข้าสู่ระบบของออปเจคหนึ่งมากกว่าหนึ่งครั้ง การเรียกตั้งแต่ครั้งที่สองเป็นต้นไปออปเจค StoreAccess จะไม่ถูกสร้างขึ้นเนื่องจากมีออปเจค StoreAccess อยู่ก่อนแล้วในระบบ



รูปที่ 4.5 แสดงความสัมพันธ์ระหว่างออปเจคกรณีมีเครื่องโพสต์ 2 เครื่อง

4.3.2.2 การทำงานและความสัมพันธ์ระหว่างออปเจคเมื่ออ่านแถบรหัส(Scan barcode)

การทำงานนี้จะทำเมื่อเครื่องโพสต์อ่านแถบรหัสจากเครื่องอ่านแถบรหัส ส่งไปให้กับเครื่องคอมพิวเตอร์ร้านค้า เพื่อที่จะทราบข้อมูลของสินค้าและราคาสินค้า มาแสดงให้ลูกค้าทราบ การทำงานนี้จะทำได้หลังจากที่เครื่องโพสต์เข้าสู่ระบบแล้วเท่านั้น



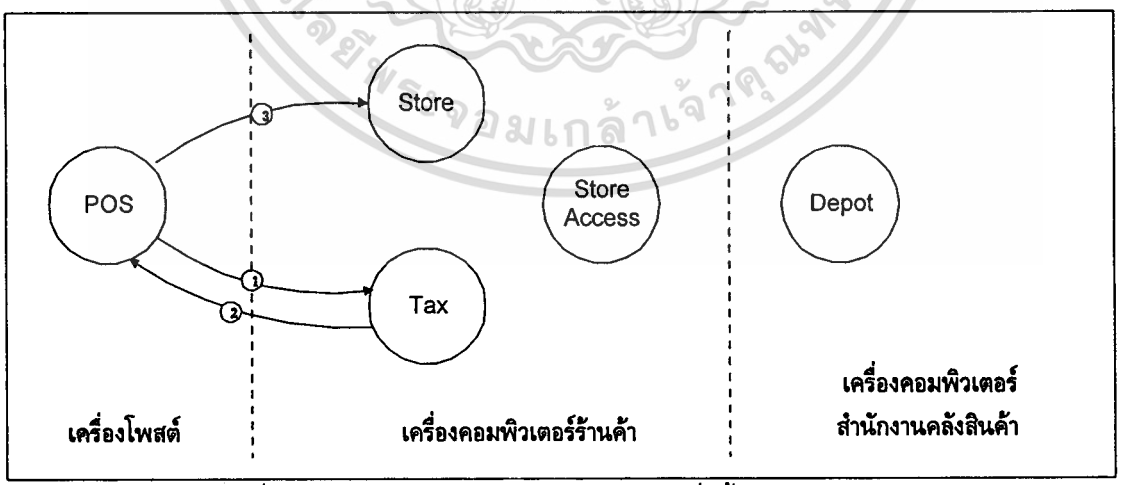
รูปที่ 4.6 แสดงความสัมพันธ์ระหว่างออปเจคเมื่อเครื่องโพสต์อ่านแถบรหัส

1. เมื่อมีการอ่านแถบรหัสฟังก์ชั่น "SendBarcode" ของออปเจค POS จะเริ่มทำงานโดยการอ่านรหัสและจำนวนของสินค้าส่งให้กับออปเจค StoreAccess ฟังก์ชั่น "FindPrice" เพื่อสอบถามข้อมูลของสินค้า
2. ฟังก์ชั่น "FindPrice" ของออปเจค StoreAccess จะเรียกฟังก์ชั่น "FindItemInfo" ของออปเจค Depot โดยส่งรหัสสินค้าและจำนวนที่ได้จากเครื่องโพสต์ไปให้ ฟังก์ชั่นจะค้นหาข้อมูลในฐานข้อมูลคลังสินค้าและคืนค่า ข้อมูลของสินค้าที่สอบถามกลับไปให้เช่น ต้นทุน, ชื่อสินค้า, ชนิดภาษีสินค้า
3. ฟังก์ชั่น "FindPrice" ของออปเจค StoreAccess จะนำค่าที่คืนกลับมาให้ มาคำนวณราคาขายที่จะขายให้กับลูกค้า และหาว่าสินค้าชนิดนั้นคิดภาษีได้หรือไม่ โดยเรียกใช้ฟังก์ชั่น "FindTaxablePrice" ของออปเจค Tax
4. คืนค่าข้อมูลสินค้า, ราคาขาย และ ราคาที่สามารถคิดภาษีได้กลับให้กับออปเจค POS เพื่อที่จะนำมาแสดงให้ลูกค้าทราบ

สังเกตุว่าการทำงานนี้ ออปเจคที่อยู่บนเครื่องคอมพิวเตอร์ทั้งสามเครื่อง (เครื่องโพสต์, เครื่องคอมพิวเตอร์ร้านค้า, เครื่องคอมพิวเตอร์สำนักงานคลังสินค้า) จะเชื่อมต่อถึงกันหมดฉะนั้นระบบเครื่อขายของระบบจะต้องไม่ขัดข้องถึงจะทำให้การทำงานเป็นผลสำเร็จ

4.3.2.3 การทำงานและความสัมพันธ์ระหว่างออปเจคเมื่อสิ้นสุดรายการขาย(End of sale)

เมื่อสิ้นสุดการขาย ราคารวมของสินค้าที่คิดภาษีได้จะถูกนำมาคิดภาษี อัตราภาษีจะถูกกำหนดโดยผู้จัดการร้านค้าและสามารถเปลี่ยนแปลงภาษีได้ ยอดขายและภาษีที่คำนวณได้ จะถูกนำไปปรับปรุงยอดขายของร้านค้า และบันทึกข้อมูลไว้ที่ร้านค้า



รูปที่ 4.7 แสดงความสัมพันธ์ระหว่างออปเจคเมื่อสิ้นสุดรายการขาย

1. เมื่อสิ้นสุดรายการขายฟังก์ชั่น "EndofSale" ของออปเจค POS จะถูกเรียกใช้งาน ทำการคำนวณภาษีโดยเรียกฟังก์ชั่น "CalculateTax" ของออปเจค Tax ส่งราคารวมของสินค้าที่สามารถคิดภาษีได้ไปให้กับฟังก์ชั่น ฟังก์ชั่นจะคืนค่าภาษีที่คำนวณได้กลับมาให้

2. คืนค่าภาษีที่คำนวณได้กลับไปให้ออปเจค POS เพื่อนำมาคิดราคาขายสุทธิที่ลูกค้าจะต้องจ่าย
3. นำราคาขายและภาษีส่งไปให้กับฟังก์ชั่น "UpdateStoreTotals" ของออปเจค Store เพื่อที่จะปรับปรุงยอดขายของร้าค้า และเก็บบันทึกเป็นข้อมูลสำหรับเครื่องโพสต์ต่อไป

4.4 IDL ของระบบขายปลีกที่เป็นสาขาย่อย

ออปเจคที่ได้จากขั้นตอนการวิเคราะห์และออกแบบ จะนำมาเขียน IDL ซึ่งเป็นขั้นตอนต่อไปของการพัฒนา ขั้นตอนการวิเคราะห์และออกแบบจะระบุสถานะ และชื่อการทำงานของออปเจค การเขียน IDL ต้องใช้ชื่อนี้ในการอ้างถึงฟังก์ชั่น

ออปเจคสามารถแปลงเป็น IDL ได้ โดยมีหลักในการแปลงดังนี้คือ

1. แต่ละออปเจคจะถูกกำหนดเป็น IDL อินเตอร์เฟส (IDL interface)
2. ตัวแปรสถานะจะไม่นำมาเขียนใน IDL จะถูกเขียนในขั้นตอนการอิมพลีเมนต์เทชั่น มีเฉพาะการทำงาน และชนิดของข้อมูลที่ผู้ใช้กำหนดเองเท่านั้นที่นำมาเขียนลงใน IDL

จากการวิเคราะห์และออกแบบจะแบ่งออปเจคออกเป็น 3 กลุ่มคือกลุ่ม POS, กลุ่ม Store และ กลุ่ม Depot นำออปเจคแต่ละกลุ่มมาเขียน IDL โดยแยกเขียนแต่ละกลุ่ม กลุ่มละ 1 แฟ้ม ได้ IDL ดังนี้

4.4.1 IDL ของ ออปเจค POS ในแฟ้ม "POS.idl"

ออปเจคในกลุ่มของ POS ประกอบด้วยออปเจค POSTerminal ออปเจคเดียวฉะนั้นการแปลงให้เป็น IDL จะได้อินเตอร์เฟสเดียว แต่มีชนิดข้อมูลที่กำหนดเอง 2 ชนิดคือ POSId และ Barcode จะต้องกำหนดลงใน IDL ด้วย

POS.idl

```
#ifndef POS_IDL
#define POS_IDL
  typedef long  POSId;
  typedef string Barcode;
   interface POSTerminal
  {
    void Login();
    void PrintPOSSalesSummary();
    void PrintStoreSalesSummary();
    void SendBarcode(in Barcode Item);
    void ItemQuantity (in long Quantity);
    void EndOfSale();
  };

#endif
```

รูปที่ 4.8 IDL ที่สมบูรณ์ของออปเจค POS

การทำงาน(Operation) ของออปเจคจะถูกแปลงเป็นฟังก์ชั่นของอินเตอร์เฟส ประกอบด้วย 5 ฟังก์ชั่น

4.4.2 IDL ของออปเจค Store, StoreAccess, Tax ในแฟ้ม "Store.idl"

ออปเจคกลุ่มนี้ประกอบด้วย 3 ออปเจค ฉะนั้นการแปลงเป็น IDL จึงได้ 3 Interface โดยกำหนดชื่ออินเตอร์เฟสเหมือนกับชื่อออปเจค

ออปเจค Store

ออปเจค Store จะมีการเก็บสถานะของร้านค้า ว่ามี POSTerminal ใดเข้าสู่ระบบ และมียอดขายเท่าไรซึ่งใช้ตัวแปรลำดับ(sequence)ของโครงสร้าง ในการเก็บข้อมูลของแต่ละโพสต์ ความยาวของตัวแปรลำดับสามารถเปลี่ยนแปลงได้ขณะกำลังรันโปรแกรม ใช้ IDL ดังนี้

```
Struct POSInfo {
        POS::POSId      Id;
        StoreAccess     StoreAccessReference;
        float           TotalSales;
        float           TotalTaxes;
};
typedef sequence <POSInfo> POSList;
```

ร้านค้าจะใช้ตัวแปรโครงสร้างในการเก็บผลรวมการขายสินค้า และผลรวมภาษี ที่เกิดขึ้นระหว่างที่ร้านค้าเปิดทำงาน การกำหนดจะใช้ IDL attribute ของโครงสร้าง ออปเจคอื่นจะอ่านตัวแปรนี้ได้อย่างเดียว ไม่สามารถแก้ไขได้ การแก้ไขตัวแปรนี้จะต้องทำผ่านฟังก์ชั่นของอินเตอร์เฟสนี้เท่านั้น

```
struct StoreTotal {
        float StoreTotal;
        float StoreTaxTotal;
};
readonly attribute StoreTotals Totals;
```

ออปเจค POSTerminal จะเรียกฟังก์ชั่น Login ในการเริ่มต้นการทำงาน ฟังก์ชั่นจะสร้างออปเจค StoreAccess และคืนค่ากลับมาให้ ต้องมีการระบุการคืนค่าออปเจค StoreAccess กลับของฟังก์ชั่น Login ใน IDL ด้วย

```
StoreAccess Login (in POS::POSId Id);
```

ออปเจค StoreAccess

ออปเจค StoreAccess ที่ได้จากการวิเคราะห์และออกแบบ เป็นตัวกลางระหว่างเครื่องโพสต์ และเครื่องคอมพิวเตอร์ที่สำนักงานคลังสินค้าใช้ในการเข้าถึงข้อมูลในฐานข้อมูล ซึ่งจะถูกสร้างขึ้นให้กับออปเจค POS แต่ละออปเจค ในตอนที่เข้าสู่ระบบ

ออปเจค StoreAccess มีการทำงานเดียวคือ "FindPrice" ใช้เรียกไปยังออปเจค Depot ในการถามถึงข้อมูลของสินค้า โดยระบุรหัสสินค้าไปให้ และถ้าการค้นหาข้อมูลไม่พบรหัสที่ตรงกันในฐานข้อมูล ก็จะเกิดกรณียกเว้น(Exception) เกิดขึ้นจะต้องเขียนยกเว้นลงใน IDL ดังนี้

```
exception BarcodeNoFound { POS::Barcode item; };
```

ฟังก์ชั่น FindPrice มีตัวแปรออกที่มีชนิดเป็น ItemInfo เป็นข้อมูลชนิดโครงสร้างที่เก็บข้อมูลของสินค้า ซึ่งถูกอ้างถึงตัวแปรชนิดนี้ในหลายๆ ออปเจคจึงต้องเขียนโครงสร้างข้อมูลนี้ลง IDL ด้วย

```
enum ItemTypes { FOOD, CLOTHES, OTHER };
Struct ItemInfo {
        POS::Barcode    Item;
        ItemTypes       Itemtype;
        float           Itemcost;
        string          Name;
        long            Quantity;
};
```

ส่วนฟังก์ชั่น FindPrice เขียน IDL ได้ดังนี้

```
Void FindPrice (
        in POS::BarcodeItem
        in long                 Quantity
        out float               ItemPrice
        out float               ItemTaxPrice
        out ItemInfo            IInfo )
raises(BarcodeNotFound);
```

ออปเจค Tax

ออปเจค Tax จะทำเกี่ยวกับภาษีให้กับร้านค้า จากการทำการวิเคราะห์และออกแบบได้การทำงาน 3 การทำงาน คือ

Initialize

CalculateTax

FindTaxablePrice

แต่การทำงาน Initialize จะไม่เขียนในส่วนของ IDL ส่วนทั้ง 2 การทำงานที่เหลือเขียนเป็น IDL ได้ดังนี้

```
float CalculateTax (in float TaxableAmount);
float FindTaxablePrice(
                in float        ItemPrice;
                in ItemType     ItemType);
```

ทั้ง 2 การทำงานจะคืนค่าทศนิยม (float) กลับมาให้ ซึ่งเป็นภาษี และ ราคาที่สามารถคิดภาษีได้ IDL ที่สมบูรณ์ของทั้ง 3 ออปเจคในกลุ่มนี้เขียนได้ดังนี้

Store.idl

```
#ifndef STORE_IDL
#define STORE_IDL

// include interface definition for Point Of Sale objects
#include "POS.idl"

  typedef string ItemTypes ;

  typedef long AStoreId;

  struct ItemInfo
  {
    Barcode     Item;
    ItemTypes       Itemtype;
    float           Itemcost;
    string          Name;
    long            Quantity;
  };

  // The barcodeNotFound exception indicates that the
  // input barcode does not match to any known item.
  exception BarcodeNotFound { Barcode item; };

  interface StoreAccess; // /forward reference
```

รูปที่ 4.9 IDL ที่สมบูรณ์ของออปเจค Store, StoreAccess, Tax

Store.idl (ต่อ)

```
struct POSInfo
{
    POSId   Id;
    StoreAccess  StoreAccessReference;
    float       TotalSales;
    float       TotalTaxes;
};

typedef sequence <POSInfo> POSList;

interface Tax
{
  float CalculateTax(in float         TaxableAmount);
  float FindTaxablePrice(in float       ItemPrice,
                         in ItemTypes   ItemType);
};

interface Store
{
    struct StoreTotals
    {
      float    StoreTotal;
      float    StoreTaxTotal;
    };
    readonly attribute AStoreId StoreId;

    // The struct StoreTotals and this readonly attribute are used
    // in place of a pair of float attributes to avoid data inconsistencies
    // that would result from the following sequence of operations:
    //
    //   POS 1 invokes a method to read StoreTotal
    //   POS 2 invokes the method UpdateStoreTotals
    //   POS 1 invoikes a method to read the StoreTaxTotal
    readonly attribute StoreTotals Totals;

    StoreAccess Login(in POSId Id);
    void      GetPOSTotals(out POSList POSData);
    void      UpdateStoreTotals(
                in POSId Id,
                in float     Price,
                in float     Taxes);
};

interface StoreAccess
{
  // ItemTaxPrice is 0 or return value of FindTaxablePrice()
  void      FindPrice(
              in  Barcode Item,
              in  long        Quantity,
              out float       ItemPrice,
              out float       ItemTaxPrice,
              inout ItemInfo     IInfo)
     raises (BarcodeNotFound);

};

#endif
```

รูปที่ 4.10 IDL ที่สมบูรณ์ของออปเจค Store, StoreAccess, Tax (ต่อ)

4.4.3 IDL ของออปเจค Depot ในแฟ้ม "Central.idl"

ในแฟ้ม Central.idl ประกอบด้วยอินเตอร์เฟสเดียวคืออินเตอร์เฟส Depot มีการทำงานเดียวคือ "FindItemInfo" ทำหน้าที่ค้นหาข้อมูลสินค้าในฐานข้อมูล และควบคุมสินค้าคงคลัง โดยจะคืนค่าข้อมูลสินค้าในตัวแปร ItemInfo ซึ่งตัวแปรชนิดนี้ถูกกำหนดแล้วในแฟ้ม Store.idl การทำงานของฟังก์ชั่นถ้าค้นหาข้อมูลไม่พบจะเกิดกรณียกเว้นเกิดขึ้นด้วย ฉะนั้นจะต้องมีการเขียน IDL ให้เกิดกรณียกเว้นด้วย

```
interface FindItemInfo(
   in AstoreId    StoreId,
   in Barcode     Item,
   in long        Quantity,
   inout ItemInfo Iinfo)
   raises(BarcodeNotFound);
```

Central.idl

```
#ifndef CENTRAL_IDL
#define CENTRAL_IDL

// Include interface definition for store and POS objects.
#include "POS.idl"
#include "Store.idl"

  interface Depot {
    void FindItemInfo(
                in  AStoreId StoreId,
                in  Barcode    Item,
                in  long         Quantity,
                inout ItemInfo IInfo)
      raises (BarcodeNotFound);
  };

#endif
```

รูปที่ 4.11 IDL ที่สมบรูณ์ของออปเจค Depot

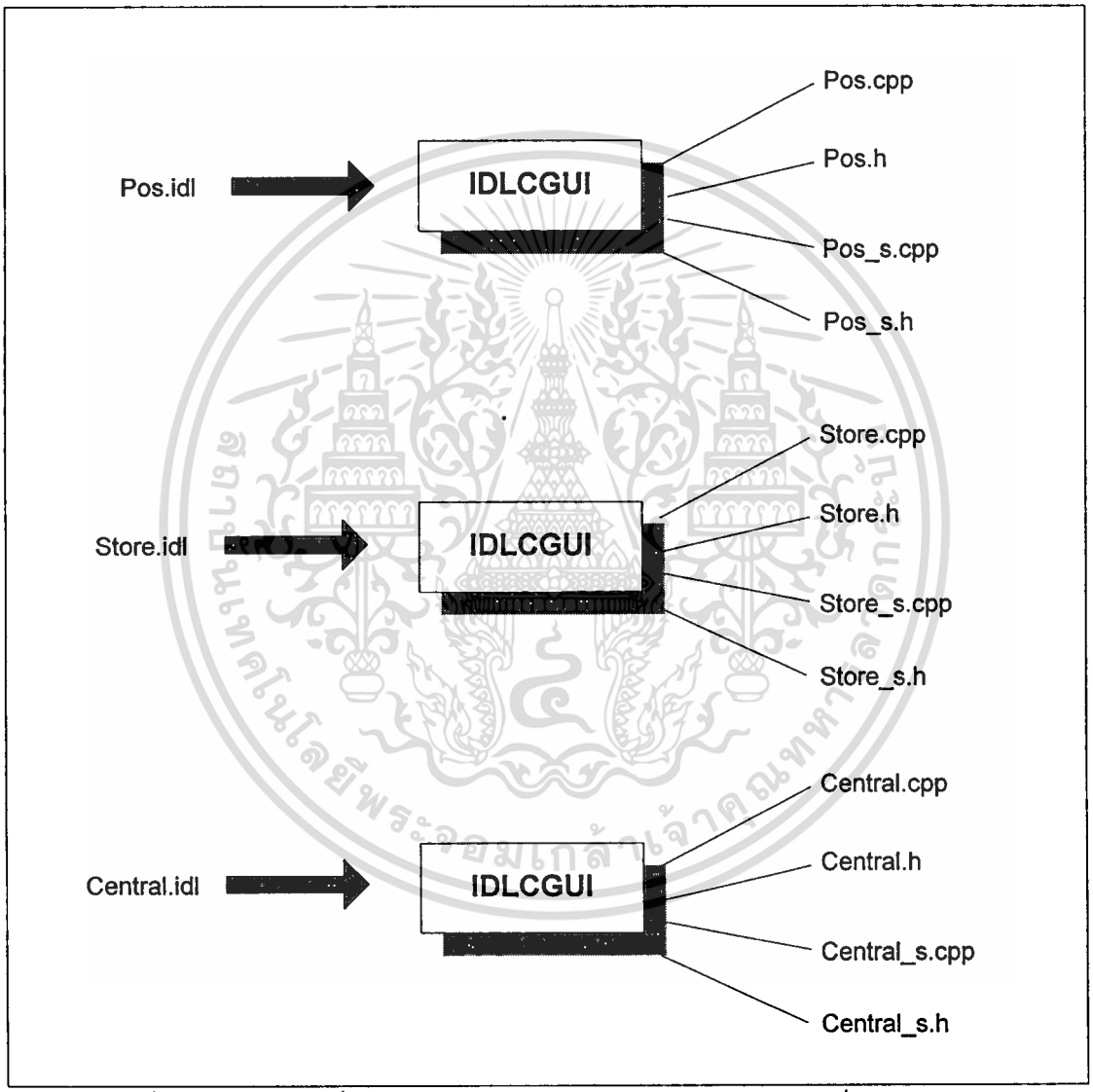
4.4.4 การแปลคำสั่ง IDL เป็นคำสั่งภาษา C++ (Compiling the IDL to C++)

การแปลออปเจคทั้งสามกลุ่มจะได้แฟ้มภาษา IDL 3 แฟ้มคือ

1. Pos.idl
2. Store.idl
3. Central.idl

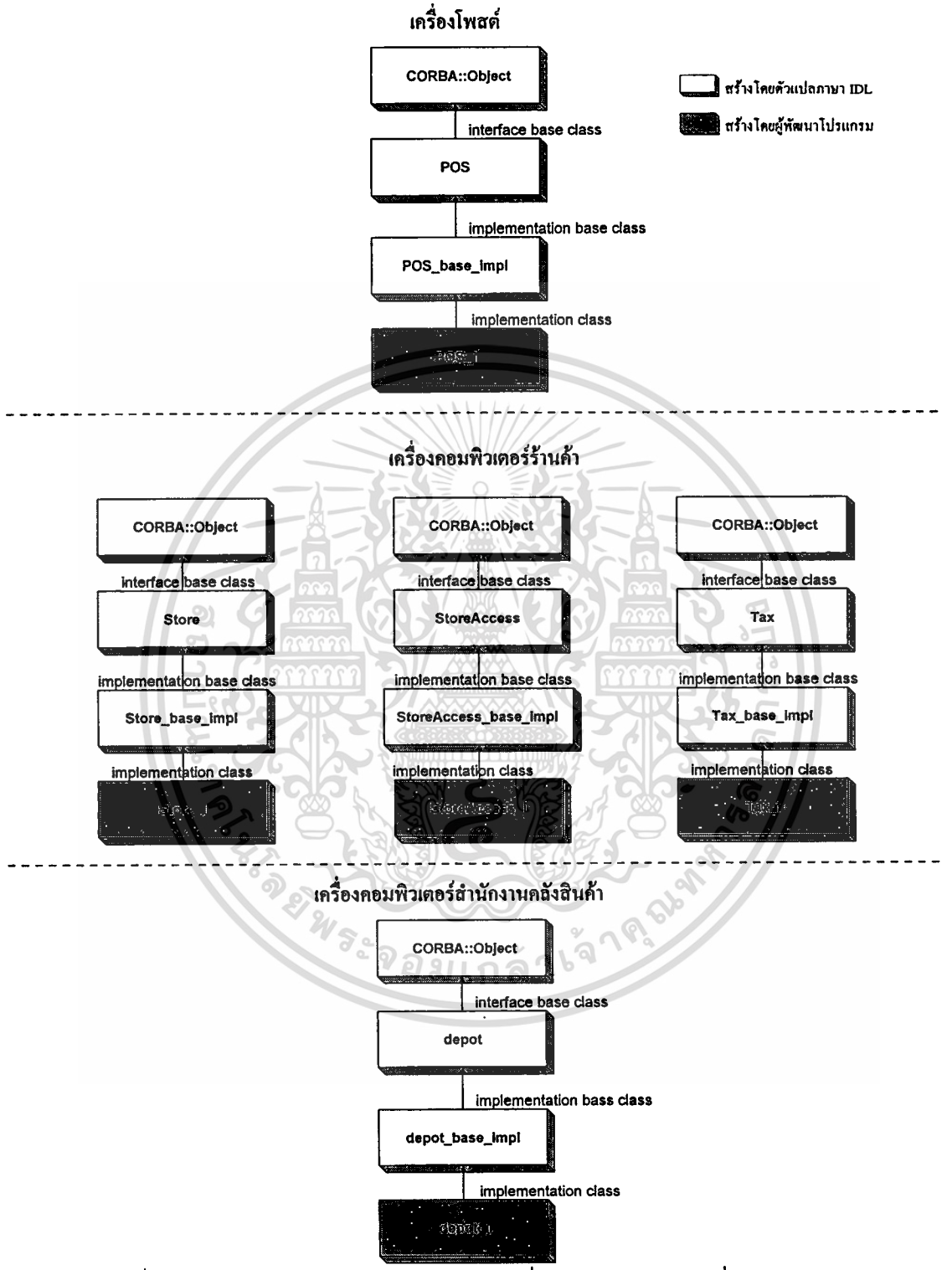
ขั้นตอนต่อไปของการพัฒนาคือการแปลภาษา IDL ในแฟ้ม .idl ให้เป็นภาษา C++ เพื่อนำแฟ้มที่แปลได้มาสร้างออปเจค อิมพลีเมนท์เทชั่น แฟ้ม .idl แต่ละแฟ้มนำมาแปลจะได้ แฟ้มใหม่สี่แฟ้ม โดยแต่ละแฟ้มจะบรรจุคลาสพื้นฐานของ ออปเจคที่ออกแบบไว้ การแปลภาษาใช้โปรแกรมแปลภาษา IDL ของ PowerBroker ที่ชื่อ IDLCGUI

แฟ้มที่ถูกสร้างจากตัวแปลภาษา IDL ของระบบนี้มีทั้งหมด 12 แฟ้มจากแฟ้ม .idl 3 แฟ้ม



รูปที่ 4.12 แฟ้มข้อมูลที่ถูกสร้างจากตัวแปลภาษา IDL ของระบบขายปลีกที่เป็นสาขาย่อย

แต่ละแฟ้มจะบรรจุคลาสพื้นฐานของออปเจคอิมพลีเมนท์เทชั่น การสร้างคลาสอิมพลีเมนท์เทชั่นจะต้องสืบทอดคลาสมาจากคลาสพื้นฐานที่ตัวแปลภาษา IDL สร้างให้ลักษณะการสืบทอดแต่ละคลาสของออปเจค อิมพลีเมนท์เทชั่น จะสืบทอดมาจากคลาสพื้นฐานที่แตกต่างกัน



รูปที่ 4.13 แสดงการสืบทอดคลาสอิมพลีเมนท์เทชั่น ของระบบขายปลีกที่เป็นสาขาย่อย

4.5 การสร้างคลาสอิมพลีเมนท์เทชั่นของระบบขายปลีกที่เป็นสาขาย่อย (Implementation class)

คลาสอิมพลีเมนท์เทชั่น สามารถเขียนได้ 2 วิธีคือทางตรงและทางอ้อม ระบบในตัวอย่างนี้จะใช้การเขียนคลาสอิมพลีเมนท์เทชั่นแบบทางตรง

การสร้างคลาสอิมพลีเมนท์เทชั่นเป็นการเขียนคำสั่งจริงของฟังก์ชั่น โดยกำหนดให้คลาสสืบทอดมาจากคลาสพื้นฐานที่ตัวแปลภาษา IDL สร้างให้ และเขียนคำสั่งของแต่ละฟังก์ชั่น ลงในคลาสนั้น ที่ส่วนต้นแฟ้ม อิมพลีเมนท์เทชั่นจะต้องรวมแฟ้มเฮดเดอร์ของคลาสพื้นฐานเข้าไปด้วย

4.5.1 คลาสอิมพลีเมนท์เทชั่นของออปเจค Depot

ออปเจค Depot เป็นออปเจคที่อยู่ที่เครื่องคอมพิวเตอร์ของสำนักงานคลังสินค้า เขียนอยู่ในแฟ้ม "Depot_i.cpp" ที่มีแฟ้มเฮดเดอร์เป็น "Depot_i.h" โดยมีชื่อคลาสอิมพลีเมนท์เทชั่นเป็น "Depot_i" คลาสนี้จะต้องสืบทอดมาจากคลาส Depot_base_imple ในแฟ้ม Central_s.h ที่ตัวแปลภาษา IDL สร้างให้ส่วนแรกของแฟ้ม Depot_i.h จะต้องรวมแฟ้ม Central_s.h เข้าไว้ด้วย

Depot_i.h

```
#ifndef DEPOT_I_H
#define DEPOT_I_H

#include <Central_s.h>

class Depot_i : virtual public Depot_base_impl {
public:
        CItemStock  *m_pItemStockTable;
public:
  Depot_i();
  virtual void FindItemInfo (CORBA::Long        StoreId,
                const char        *Item,
                CORBA::Long       Quantity,
                ItemInfo                &OutItemInfo);
};
#endif
```

รูปที่ 4.14 การรวมแฟ้มส่วนหัวของคลาสอิมพลีเมนท์เทชั่นพื้นฐานในแฟ้มอิมพลีเมนท์เทชั่น

ในส่วนการประกาศตัวแปรของคลาส มีการประกาศตัวแปรออปเจค "CItemStock" ซึ่งเป็นออปเจคที่ครอบฐานข้อมูลอยู่ ตัวฐานข้อมูลใช้โครงสร้างแฟ้มข้อมูลของไมโครซอฟแอคเซส (Microsoft Access) และการเข้าถึงฐานข้อมูลจะสร้างคลาสฐานข้อมูลแบบ DAO ครอบแฟ้มฐานข้อมูลเอาไว้ ทำให้การเข้าถึงฐานข้อมูลของออปเจค Depot_i จะต้องกระทำกับออปเจค CItemStock

ฟังก์ชั่นสมาชิกของออปเจค Depot_i จะต้องประกาศเป็นฟังก์ชั่นเสมือน

แฟ้มdepot_i.cpp ในส่วนแรกจะต้องรวมแฟ้มเฮดเดอร์ "depot_i.h" และแฟ้มเฮดเดอร์ของฐานข้อมูล "ItemStk.h" ไว้ด้วยฟังก์ชั่น Initialization ของออปเจค depot จะถูกเขียนขึ้นในแฟ้มนี้โดยเขียนลงในส่วนของตัวสร้าง (constructor) ฟังก์ชั่น "FindItemInfo" ค้นหาข้อมูลในฐานข้อมูลโดยใช้คำสั่ง FindFirst ถ้าค้นหาไม่พบ จะแจ้งให้ผู้ที่สอบถามทราบโดยการส่งข้อยกเว้น BarcodeNotFound โดยใช้คำสั่ง throw()

Depot_i.cpp

```
#include "StdAfx.h"
#include "ItemStk.h"
#include "Depot_i.h"

const int NITEMTYPES=3;
const char *itemtypestext[NITEMTYPES] = {"FOOD","CLOTHES","OTHER"};
Depot_i::Depot_i()              //Implementation of Initialization
{
        m_pItemStockTable = new CItemStock;
        m_pItemStockTable -> Open();
        m_pItemStockTable -> MoveFirst();
}

void Depot_i::FindItemInfo(CORBA::Long      StoreId,
                const char      *Item,
                CORBA::Long     Quantity,
                ItemInfo                &IInfo)
{
        CString SQLWhere = "BarcodeItem = '";
        SQLWhere = SQLWhere + Item + "'";
        if (m_pItemStockTable->FindFirst((LPCTSTR) SQLWhere)){
                IInfo.Item = CORBA::string_dup((LPCTSTR)m_pItemStockTable->m_BarcodeItem);
                IInfo.Itemtype = CORBA::string_dup((LPCTSTR)m_pItemStockTable->m_TaxType);
                IInfo.Itemcost = m_pItemStockTable->m_cost;
                IInfo.Name = CORBA::string_dup((LPCTSTR)m_pItemStockTable->m_ItemName);
                if (m_pItemStockTable->m_count >= Quantity){
                        IInfo.Quantity = Quantity;
                        m_pItemStockTable->m_count -= Quantity;
                }
                else{
                        IInfo.Quantity = m_pItemStockTable->m_count;
                        m_pItemStockTable->m_count = 0;
                }
        }
        else{
                BarcodeNotFound * bcnf = new BarcodeNotFound;
                bcnf->item = Item;
                throw(*bcnf);
        }
}
```

รูปที่ 4.15 คำสั่งของฟังก์ชั่นทั้งหมดที่ระบุใน IDL ของออปเจค Depot

4.5.2 คลาสอิมพลีเมนท์เทชั่นของออปเจค Store, StoreAccess และ Tax

คลาสอิมพลีเมนท์เทชั่นของออปเจค Store, StoreAccess และ Tax คือ Store_i, StoreAccess_i, Tax_i ทั้งสามคลาสเขียนรวมอยู่ในแฟ้ม "Store_i.cpp" มีแฟ้มเฮดเดอร์เป็น "Store_i.h" และมีการสืบทอดมาจากคลาสพื้นฐานที่แตกต่างกันคือ Store_Base_impl, StoreAccess_Base_impl และ Tax_Base_impl ตามลำดับเนื่องจากคลาสพื้นฐานอยู่ในแฟ้มเดียวกัน ฉะนั้นตอนต้นของแฟ้ม "Store_i.h" จึงต้องรวมแฟ้ม "Store_s.h" ของคลาสพื้นฐานเข้าไว้ด้วย และนอกจากนี้ออปเจค StoreAccess มีการเรียกใช้ฟังก์ชั่นของออปเจค Depot จึงเปรียบได้ว่า ออปเจค StoreAccess เป็นออปเจคผู้รับบริการ ฉะนั้นจะต้องมีการรวมแฟ้มเฮดเดอร์หรือ แฟ้ม stup ของออปเจค Depot "central.h" ลงในแฟ้ม store_i.h ด้วย

Store.h

```
#ifndef STORE_I_H
#define STORE_I_H
#include <Store_s.h>
#include <Central.h>

class Tax_i :virtual public Tax_base_impl{
private:
        CORBA::Float        m_regionRate;
public:
  Tax_i();
        virtual     CORBA::Float CalculateTax(CORBA::Float TaxableAmount);
        virtual CORBA::Float FindTaxablePrice(CORBA::Float ItemPrice,
                                        const char* ItemType);
};

class Store_i :virtual public Store_base_impl{
private:
        CORBA::ULong          LocatePOSEntry(CORBA::Long);
        AStoreId                    m_storeID;
        CORBA::Float                m_storeTotal;
        CORBA::Float                m_storeTaxTotal;
        CORBA::Float                m_storeMarkup;
        POSList                     m_POSTerminals;
        CosNaming::NamingContext_var   m_pNC;
public:
  Store_i(CosNaming::NamingContext_ptr pNC);
  virtual AStoreId                  StoreId();
  virtual Store::StoreTotals     Totals();
  virtual StoreAccess_ptr           Login(CORBA::Long          Id);
  virtual void  GetPOSTotals(POSList_out  POSData);
  virtual void  UpdateStoreTotals(
                                    CORBA::Long       Id,
            CORBA::Float       Price,
            CORBA::Float       Taxes);
};

class StoreAccess_i :virtual public StoreAccess_base_impl{
private:
  Depot_var                         m_depot;
  Tax_var                           m_tax;
  Store_var                         m_store;
public:
  StoreAccess_i(CosNaming::NamingContext_ptr pNC,Store_ptr store);
  virtual void FindPrice(
            const char         *Item,
            CORBA::Long              Quantity,
            CORBA::Float&            ItemPrice,
            CORBA::Float&            ItemTaxPrice,
            ItemInfo                   &IInfo);
};
#endif
```

รูปที่ 4.16 แฟ้มอิมพลีเมนท์เทชั่นเฮดเดอร์ของออปเจค Store, StoreAccess, Tax

คลาส Store_i

คลาส store_i เป็นคลาสอิมพลีเมนท์ของคลาส Store ตัวแปรสถานะจะถูกเขียนเป็น ดาต้าเมมเบอร์ (Data Member) ของคลาส

```
AstoreId            m_storeID;
CORBA::Float        m_storeTotal;
CORBA::Float        m_storeTaxTotal;
CORBA::Float        m_storeMarkup;
POSList             m_POSTerminal;
CosNaming::NamingContext  m_pNC;
```

ฟังก์ชั่นตัวสร้าง(constructor) ของคลาส Store_i จะเป็นการแทนฟังก์ชั่น Initialzation ที่ได้จากการวิเคราะห์และออกแบบ และเนื่องจากออปเจค Store มีการเรียกใช้ออปเจคโดยใช้บริการ Naming ของ CORBA ฉะนั้นจะต้องมีการสร้างออปเจค NameService ขึ้นเพื่อใช้ค้นหาออปเจคอื่นด้วยคำสั่ง

```
m_pNC = CosNaming::NamingContext::_duplicate (pNC)
```

คำสั่งสำหรับทุกฟังก์ชั่นของคลาส store_i แสดงดังนี้

Implementation of Store_i

```
Store_i::Store_i(CosNaming::NamingContext_ptr pNC): m_POSTerminals(10 )Start off with space for 25
POSs
{
        m_storeID = 1;
        m_pNC = CosNaming::NamingContext::_duplicate(pNC);
        m_storeTaxTotal = 0;
        m_storeTotal = 0;
}

CORBA::Long Store_i::StoreId()
{
  return m_storeID;
}

Store::StoreTotals Store_i::Totals()
{
  Store::StoreTotals ST;
  ST.StoreTotal = m_storeTotal;
  ST.StoreTaxTotal = m_storeTaxTotal;
  return ST;
}

StoreAccess_ptr Store_i::Login(CORBA::Long Id)
{
  CORBA::ULong loc = LocatePOSEntry(Id);
  m_POSTerminals[loc].Id        = Id;
  m_POSTerminals[loc].TotalSales = 0;
  m_POSTerminals[loc].TotalTaxes = 0;
  // check to see of a StoreAccess object exists for this m_POSTerminal
  // allocate new one if needed.
  if (CORBA::is_nil((StoreAccess_ptr &)m_POSTerminals[loc].StoreAccessReference)) {
          //if have StoreAccessRef :  No allocate
     // create a local instance of the SToreAccess Object
    m_POSTerminals[loc].StoreAccessReference = new StoreAccess_i(m_pNC,_this());
    if (CORBA::is_nil((StoreAccess_ptr &)m_POSTerminals[loc].StoreAccessReference))

      cerr << "Store_i::Login: Unable to create StoreAccess object for POS Login" << endl;
  }
  return StoreAccess::_duplicate(m_POSTerminals[loc].StoreAccessReference);
}
```

รูปที่ 4.17 คำสั่งของฟังก์ชั่นทั้งหมดที่ระบุใน IDL ของออปเจค Store

Implementation of Store_i (ต่อ)

```
void Store_i::GetPOSTotals(POSList_out POSData)
{
        POSData = new POSList(m_POSTerminals);

        // This command "POSList(m_POSTerminals)" is mean allocate
        // new POSList pointer give with POSData and  assign with value
        // of m_POSTerminals
}

void Store_i::UpdateStoreTotals(
                CORBA::Long  Id,
                CORBA::Float Price,
                CORBA::Float Taxes)
{
        m_storeTaxTotal += Taxes;
        m_storeTotal += Price;

}

CORBA::ULong Store_i::LocatePOSEntry(CORBA::Long Id)
{
  CORBA::ULong loc      = EMPTY;
  CORBA::ULong availloc  = EMPTY;
  CORBA::ULong len      = m_POSTerminals.length();
  int keepgoing         = 1;
  CORBA::ULong i        = 0;

  // locate POSId or first available slot
  while (loc == EMPTY && i < len)
  {
    if (m_POSTerminals[i].Id == Id)
      loc = i;
    else if (availloc == EMPTY && m_POSTerminals[i].Id == EMPTY)
      availloc = i;
    else
      i++;
  }

  // if we did not find POSId then use the available slot or append new
  // slot at end of m_POSTerminals
  if (loc == EMPTY)
  {
    if (availloc != EMPTY)
      loc = availloc;
    else
                    {
      m_POSTerminals.length(i+1);
      loc = i;
                    }
  }
  return loc;

}
```

รูปที่ 4.18 คำสั่งของฟังก์ชั่นทั้งหมดที่ระบุใน IDL ของออปเจค Store (ต่อ)

คลาส StoreAccess_i

คลาส StoreAccess_i เป็นคลาสอิมพลีเมนท์ของคลาส StoreAccess ตัวแปรสถานะถูกเขียนเป็นดาต้าเมมเบอร์ของคลาส

Depot_var m_depot;

Tax_var m_tax;

Store_var m_store;

คลาส StoreAccess_i จะมีฟังก์ชั่นสมาชิกเพียงฟังก์ชั่นเดียว ได้จากการวิเคราะห์และออกแบบ คือ ฟังก์ชั่น "FindPrice" การเรียกใช้ฟังก์ชั่นของออปเจคอื่นจะต้องมีการสร้างตัวชี้ ชี้ไปยังออปเจคนั้นก่อน สร้างตัวชี้ในออปเจค NameService โดยการสร้างออปเจค Naming ด้วยคำสั่ง

m_pNC = CosNaming::NamingContext::_duplicate(pNC)

Implementation of StoreAccess_i

```
StoreAccess_i::StoreAccess_i(CosNaming::NamingContext_ptr pNC,Store_ptr store)
{
        CORBA::Object_ptr           StoreAccessObj;
        CosNaming::NameComponent ncStoreAccess;
        CosNaming::Name             StoreAccessName(1);
try{
        char refstr[255];
        AStoreId id = store->StoreId();
        sprintf(refstr,"Tax_%ld",id);
        ncStoreAccess.id    = (const char*) "Tax";//refstr;
        StoreAccessName[0]          = ncStoreAccess;
        StoreAccessObj = pNC->resolve(StoreAccessName);
        m_tax = Tax::_narrow(StoreAccessObj);
        ncStoreAccess.id    = (const char*) "Depot";
        StoreAccessName[0]          = ncStoreAccess;
        StoreAccessObj = pNC->resolve(StoreAccessName);
        m_depot = Depot::_narrow(StoreAccessObj);
        m_store = Store::_duplicate(store);
}
catch(...){
        cerr << "Trouble finding tax, store, or depot " << endl;
}
}
void StoreAccess_i::FindPrice(
                const char          *Item,
                CORBA::Long         Quantity,
                CORBA::Float&       ItemPrice,
                CORBA::Float&       ItemTaxPrice,
                ItemInfo            &IInfo)
{
        m_depot->FindItemInfo(m_store->StoreId(),Item,
                                        Quantity, IInfo);
        ItemPrice = IInfo.Itemcost * stormarkup;
        ItemTaxPrice = m_tax->FindTaxablePrice(         ItemPrice
                                        ,IInfo.Itemtype);
}
```

รูปที่ 4.19 คำสั่งของฟังก์ชั่นทั้งหมดที่ระบุใน IDL ของออปเจค StoreAccess

คลาส Tax_i

Tax_i เป็นคลาสอิมพลีเมนท์เท่ชั่นของออปเจค Tax มีดาต้าเมมเบอร์ m_regionRate เก็บอัตราภาษีที่ใช้ในการคำนวณ

ฟังก์ชั่น "CalculateTax" จะคืนค่าภาษีจากผลคูณของราคาขาย กับ อัตราภาษีของสินค้าที่ผู้จัดการร้านค้าเป็นผู้กำหนด (TaxableAmount * m_regionRate) และฟังก์ชั่น "FindTaxablePrice" จะหาว่าสินค้าชนิดนั้นคิดภาษีได้หรือไม่ โดยพิจารณาจากชนิดของภาษี ถ้าเป็นเสื้อผ้าหรืออาหาร จะคืนค่า 0 กลับไปให้ (สินค้ายกเว้นภาษี) แต่ถ้าเป็นสินค้าชนิดอื่นจะคืนค่าราคาสินค้ากลับไปให้ (สินค้าคิดภาษี)

คำสั่งสำหรับทุกฟังก์ชั่นของคลาส Tax_i แสดงดังนี้

Implementation of Tax_i

```
Tax_i::Tax_i()
{
        m_regionRate = 0.5;    // Initial Tax Rate
}

CORBA::Float Tax_i::CalculateTax(CORBA::Float TaxableAmount)
{
  return TaxableAmount*m_regionRate;
}

CORBA::Float Tax_i::FindTaxablePrice(CORBA::Float ItemPrice,
                                     const char* ItemType)
{
  CORBA::Float taxprice;
  if (!strcmp(ItemType,"OTHER"))  {
        taxprice = ItemPrice;
  }
  else  {
        taxprice = 0.0;
  }
  return taxprice;
}
```

รูปที่ 4.20 คำสั่งของฟังก์ชั่นทั้งหมดที่ระบุใน IDL ของออปเจค Tax

เมื่อได้คลาสเฮดเดอร์ที่ตัวแปลภาษา IDL สร้างให้ และคลาสอิมพลีเมนท์เทชั่น แล้วทำให้ออปเจคสามารถกรจายอยู่บนเน็ทเวิร์คได้ แต่การที่ออปเจคทั้ง 5 ออปเจคจะถูกกระจาย และถูกเรียกกลับมาใช้ได้นั้นจะต้องมีการนำออปเจคมาลงทะเบียน และ ประกาศให้โปรแกรมผู้รับบริการทราบก่อน การลงทะเบียนและการประกาศจะกระทำทางด้านโปรแกรมผู้ให้บริการ และ การเรียกออปเจคมาใช้จะกระทำทางด้านโปรแกรมผู้รับบริการ

4.5.3 คลาสอิมพลีเมนท์เทชั่มของออปเจค POS

ออปเจค POS เป็นออปเจคที่เป็นอินพุทและเอาท์พุทของระบบ ซึ่งเป็นอินพุทและเอาท์พุทที่เป็นกราฟฟิก ฉะนั้นการเขียนอิมพลีเมนท์เทชั่นของออปเจคนี้จะต้องสร้างให้รองรับกับการทำงานที่เป็นกราฟฟิก

ด้วย จึงได้สร้างออปเจค POS ไว้ภายใต้คลาสของไดอะล็อกอยู่ในแฟ้ม "POS_MainDlg.cpp" โดยมีแฟ้มเฮดเดอร์เป็น "POS_MainDlg.cpp" คำสั่งของออปเจค POS จะเขียนในคลาส CPOS_MainDlg

4.6 โปรแกรมหลักของระบบขายปลีกที่เป็นสาขาย่อย(Main Program)

โปรแกรมหลักเป็นโปรแกรมที่ทำหน้าที่สร้างออปเจคขึ้นจากคลาสอิมพลีเมนท์เทชั่น และนำออปเจคมาลงทะเบียน ให้เป็นออปเจคแบบกระจายเพื่อรอรับการเรียกใช้จากโปรแกรมผู้รับบริการ ในขั้นตอนการลงทะเบียนออปเจคมีการนำบริการ Naming ของ CORBA มาใช้ในการประกาศออปเจค การนำบริการ Naming มาใช้ต้องรันโปรแกรม pbnamed.exe ซึ่งเป็นโปรแกรมผู้ให้บริการเกี่ยวกับการประกาศออปเจคที่มีการลงทะเบียนไว้แล้วให้โปรแกรมผู้รับบริการทราบ โปรแกรมนี้มีออปเจค "NameService" ทำหน้าที่เก็บข้อมูลของออปเจคอื่นที่ลงทะเบียนแล้วในออปเจค NameService ถ้าโปรแกรมผู้ให้บริการต้องการยกเลิกออปเจค จะต้องจบข้อมูลของออปเจคนั้นออกจากออปเจค NameService ด้วยเพื่อไม่ให้เกิดข้อผิดพลาดเนื่องจากไม่สามารถอ้างออปเจคได้ (ออปเจคถูกยกเลิกแต่ตัวชี้ออปเจคใน NameService ยังไม่ถูกยกเลิก) ทั้งโปรแกรมผู้ให้บริการและโปรแกรมผู้รับบริการ มีการติดต่อกับออปเจค NameService จะต้องมีการสร้างตัวชี้ชี้ไปยังออปเจค NameService ที่ช่องทางการติดต่อหมายเลข 6004 (เป็นหมายเลขช่องทางการติดต่อที่ทางผู้ผลิต ผลิตภัณฑ์ PowerBroker CORBAplus เป็นผู้กำหนด)

ส่วนเริ่มต้นของโปรแกรมหลักที่มีการใช้บริการ Naming ต้องมีการรวมแฟ้มเฮดเดอร์ และประกาศคำสั่งต่อไปนี้

```
#include <pbroker/corba/orb.h>
#include <pbroker/corba/xpsboa.h>
#include <pbroker/pberr.h>
#include <pbroker/winuc/winsuc.h>
#include <pbroker/corba/naming.h>
int arge_central = 4;
char*argv_central[4] = {"central","_pbinit","NameService","iiop://161.246.2.31:6004/
                NameServiceRoot"};
CORBA::ORB_ptr orb = CORBA::ORB_init(arge_central, argv_central);
CORBA::BOA_ptr aboa = orb->BOA_init(arge_central, argv_central);
XpsBOA*          pboa = XpsBOA::_narrow(aboa);
CORBA::Object_ptr Obj = orb->resolve_initial_references("NameService");
COSNaming::NamingContext_ptr pNC=CosNaming::NamingContext::_narrow(obj);
```

4.6.1 โปรแกรมหลักของเครื่องคอมพิวเตอร์สำนักงานคลังสินค้า

เครื่องคอมพิวเตอร์สำนักงานคลังสินค้าทำหน้าที่ให้บริการในการค้นหาข้อมูลสินค้าเพียงอย่างเดียว ไม่มีการเรียกใช้บริการจากที่ใด ฉะนั้นโปรแกรมหลักจึงมีเฉพาะโปรแกรมผู้ให้บริการเท่านั้น ออปเจคของสำนักงานคลังสินค้ามีเพียงออปเจค Depot เพียงออปเจคเดียว การลงทะเบียนกระทำโดยการสร้างสภาพแวดล้อมของ ORB, สภาพแวดล้อม BOA และสร้างตัวชี้ออปเจค NameService แล้วสร้างออปเจค Depot จากคลาส Depot_i มาลงทะเบียนบน BOA

```
Depot_ptr DP = new Depot_i();          //สร้างออปเจค Depot
pboa->obj_is_ready(Dp,nil);            //ลงทะเบียนบน BOA
```

ประกาศออปเจค Depot ที่ลงทะเบียนแล้วโดยใช้ชื่อเรียกว่า "Depot" ลงในออปเจค NameService ซึ่งจะเก็บชื่อและตัวชี้ที่ใช้อ้างถึงออปเจคไว้

```
CosNaming::NameComponent   ncCentral;
CosNaming::Name            CentralName(1);
ncCentral.id          =    (Const char *) "Depot";
CentralName[0]        =    ncCentral;
pNC->rebind                (CentralName, Dp);//ประการออปเจค Depot
                                             //บนออปเจค NameService
```

4.6.2 โปรแกรมของเครื่องคอมพิวเตอร์ร้านค้า

เครื่องคอมพิวเตอร์ร้านค้าทำหน้าที่ให้บริการหาข้อมูลสินค้า, คำนวณภาษี และเก็บสถานะการทำงานแก่เครื่องโพสต์ทุกเครื่องที่เข้าสู่ระบบ และยังเรียกใช้บริการของเครื่องคอมพิวเตอร์สำนักงานคลังสินค้าในการสอบถามข้อมูลของสินค้า โปรแกรมหลักของเครื่องคอมพิวเตอร์ร้านค้าจึงเป็นทั้งโปรแกรมผู้ให้บริการ และโปรแกรมผู้รับบริการในโปรแกรมเดียวกัน มีการลงทะเบียนออปเจค Store, Tax, StoreAccess และมีการเรียกใช้ออปเจค Depot เริ่มต้นการลงทะเบียนโดยการสร้างสภาพแวดล้อมของ ORB, สภาพแวดล้อม BOA และสร้างตัวชี้ออปเจค NameService จากนั้นสร้างออปเจค Store, Tax จากคลาส Store_i, Tax_i ตามลำดับ ลงทะเบียนออปเจคทั้งสองลงบน BOA ส่วนออปเจค StoreAccess จะยังไม่นำมาลงทะเบียนเพราะออปเจคนี้จะถูกสร้างเมื่อมีเครื่องโพสต์ใหม่เข้าสู่ระบบเท่านั้น

```
Store_ptr  Sp = new Store_i (pNC);     //สร้างออปเจค Store
pboa->object _is_ready(Sp,nil);        //ลงทะเบียนออปเจค Store บน BOA
Tax_ptr    Tp = new Tax_i();           //สร้างออปเจค Tax
pboa->object_is_ready(Tp,nil);         //ลงทะเบียนออปเจค Tax บน BOA
```

ประกาศออปเจค store ใช้ชื่อว่า "Store" ลงในออปเจค NameService

```
CosNaming::NameComponent   ncStore;
CosNaming::Name            storeName(1);
ncStore.id          =      (const char *) "Store";
StoreName[0]        =      ncStore;
pNC->rebind(storeName, Sp); //ประกาศออปเจค Store บน NameService
```

ประกาศออปเจค Tax ใช้ชื่อว่า "Tax" ลงในออปเจค NameService

```
ncStore.id          = (const char *) "Tax";
StoreName[0]        = ncStore;
pNC->rebind (StoreName, Tp);  //ประกาศออปเจค Tax บน NameService
```

การเรียกใช้ออปเจค Depot จะต้องมีการสร้างตัวชี้ของออปเจค Depot ก่อนโดยผู้ที่จะสร้างจะต้องรู้ชื่อที่โปรแกรมผู้ให้บริการใช้เรียกแทนออปเจคนั้น ในที่นี้ชื่อที่ใช้เรียกแทนออปเจค Depot ชื่อ "Depot" ทำการค้นหาชื่อนี้ในออปเจค NameService และสร้างตัวชี้ขึ้น

```
CORBA::Object_ptr   depotObj;
ncStore.id     =    (const char *) "Depot";
StoreName[0]   =    ncStore;
depotObj       =    pNC->resolve(storeName);//ค้นหาออปเจค Depot
Depot_ptr DP_ptr = Depot::_narrow(depotObj);    //สร้างตัวชี้ออปเจค Depot
```

4.6.3 โปรแกรมหลักของเครื่องโพสต์

เครื่องโพสต์ติดต่อกับเครื่องคอมพิวเตอร์ร้านค้าโดยการเรียกใช้บริการของออปเจค Store, Tax, StoreAccess ไม่มีการให้บริการกับออปเจคใดๆ โปรแกรมหลักของเครื่องโพสต์จึงเป็นโปรแกรมผู้รับบริการเพียงอย่างเดียว เมื่อเริ่มต้นโปรแกรมหลักของเครื่องโพสต์จะมีการสร้างตัวชี้ออปเจคไปยังออปเจค Store และ Tax และเมื่อพนักงานเก็บเงินเข้าสู่ระบบด้วยการเรียกใช้ฟังก์ชั่น Login ของออปเจค Store ฟังก์ชั่นนี้จะสร้างออปเจค StoreAccess ขึ้นและคืนค่าตัวชี้กลับไปให้ออปเจคโพสต์ เอาไว้สำหรับติดต่อกับออปเจค Depot ในการสอบถามข้อมูล การคืนค่าตัวชี้ออปเจค StoreAccess ของฟังก์ชั่น Login จะต้องประกาศการคืนค่ากลับที่เป็นออปเจค StoreAccess ไว้ในแฟ้ม IDL ด้วยฟังก์ชั่นถึงจะคืนค่าตัวชี้ออปเจคกลับมาได้

การสร้างตัวชี้ออปเจค Store และ Tax จะต้องค้นหาชื่อที่ใช้แทนออปเจค ในออปเจค NameService ให้พบก่อนถึงจะสามารถสร้างตัวชี้ออปเจคได้

สร้างตัวชี้ออปเจค Store ซึ่งใช้เรียกแทนออปเจคชื่อ "Store"

```
CORBA::Object_ptr               POSObj;
CosNaming::NameComponent        ncPOS;
CosNaming::Name                 POSName(1);
ncPOS.id        =       (const char *) "Store";
POSName[0]      =       ncPOS
POSObj          =       pNC->resolve(POSName); //ค้นหาออปเจค Store
Store_ptr  SP_ptr  =  Store::_narrow(POSObj);    //สร้างตัวชี้ออปเจค Store
```

สร้างตัวชี้ออปเจค Tax ชื่อใช้เรียกแทนออปเจคชื่อ "Tax"

```
ncPOS.id        =       (const char *) "Tax";
POSName[0]      =       ncPOS
POSObj  =  pNC->resolve(POSName);          //ค้นหาออปเจค Tax
Tax_ptr TP_ptr = Tax::_narrow(POSObj);//สร้างตัวชี้ออปเจค Tax
```

4.7 การแปลภาษา และ การรันโปรแกรม (Compiling and Running the Application)

แฟ้มของโปรแกรมที่ได้จากการพัฒนาระบบขายปลีกที่เป็นสาขาย่อยแบ่งออกเป็น 3 โปรแกรมคือ

1. แฟ้มของโปรแกรมสำนักงานคลังสินค้า

โปรแกรมหลักคือ Central_Main.cpp

แฟ้มโปรเจคคือ Central_Main.mdp

และแฟ้มที่เกี่ยวข้องคือ

แฟ้มเฮดเดอร์	แฟ้มอิมพลีเมนท์เทชั่น
- Central_Main.h	- Central_Main.cpp
- Central_MainDlg.h	- Central_MainDlg.cpp
- depot_i.h	- depot_i.cpp
- itemstk.h	- itemstk.cpp
- central.h	- central.cpp
- central_s.h	- central_s.cpp
- stdafx.h	- stdafx.cpp
- resource.h	

2. แฟ้มของโปรแกรมร้านค้า

โปรแกรมหลักคือ Store_Main.cpp

แฟ้มโปรเจคคือ Store_Main.mdp

และแฟ้มที่เกี่ยวข้องคือ

แฟ้มเฮดเดอร์	แฟ้มอิมพลีเมนท์เทชั่น
- Store_Main.h	- Store_Main.cpp
- Store_MainDlg.h	- Store_MainDlg.cpp
- Store_i.h	- Store_i.cpp
- Store.h	- Store.cpp
- Store_s.h	- Store_s.cpp
- subdlg.h	- subdlg.cpp
- stdafx.h	- stdafx.cpp
- resource.h	

3. แฟ้มของโปรแกรมเครื่องโพสต์

โปรแกรมหลักคือ POS_Main.cpp

แฟ้มโปรเจคคือ POS_Main.mdp

และแฟ้มที่เกี่ยวข้องคือ

แฟ้มเฮดเดอร์	แฟ้มอิมพลีเมนท์เทชั่น
- POS_Main.h	- POS_Main.cpp
- POS_MainDlg.h	- POS_MainDlg.cpp
- POS_i.h	- POS_i.cpp
- POS.h	- POS.cpp
- POS_S.h	- POS_S.cpp
- subdialog.h	- subdialog.cpp
- rw_ini.h	- rw_ini.cpp
- stdafx.h	- stdafx.cpp
- resource.h	

แฟ้มของทั้งสามโปรแกรมถูกแปลให้เป็นแฟ้มเอ็กซีคิวชั่น (Execution file) หรือแฟ้มที่มีนามสกุลเป็น EXE โดยใช้โปรแกรมแปลภาษา Visual C++ 4.0 โปรแกรมแปลภาษาจะมีการลิงค์แฟ้ม naming.lib, corba.lib, pbroker.lib, winsvc.lib รวมเข้าไว้ด้วย แฟ้มเอ็กซีคิวชั่นที่ได้จากการแปลภาษาและการลิงค์คือ

1. โปรแกรมสำนักงานคลังสินค้า แฟ้มชื่อ Central.exe
2. โปรแกรมร้านค้า แฟ้มชื่อ Store.exe
3. โปรแกรมเครื่องโพสต์ แฟ้มชื่อ Pos.exe

การรันโปรแกรม

โปรแกรมระบบขายปลีกที่เป็นสาขาย่อย เป็นโปรแกรมที่มีการสร้างและเรียใช้ออปเจคโดยใช้บริการ Naming ซึ่งเป็นบริการของ CORBA ฉะนั้นทำให้การรันโปรแกรมระบบขายปลีกที่เป็นสาขาย่อยจะต้องรันบริการ Naming ก่อนถึงจะทำให้โปรแกรมทำงานได้ ผลิตภัณฑ์ PowerBroker CORBAplus ได้สร้างบริการ Naming ไว้ในแฟ้ม "pbnamed.exe" ในไดเรคทรอรี(Directory) "/pbcp/bin" การรันทำได้โดยการเรียกคำสั่ง

```
/pbcp/bin/pbnamed.exe
```

โปรแกรม pbnamed.exe จะสร้างออปเจค NameSerivce ที่ใช้ชื่อเรียกแทนออปเจคว่า NameServiceRoot และมีช่องทางการติดต่อหมายเลข 6004 ฉะนั้นโปรแกรมที่จะมาประกาศและเรียกใช้ออปเจคจะต้องใช้ชื่อและช่องทางติดต่อนี้ในการติดต่อกับออปเจค NameService และนอกจากนั้น การรันโปรแกรมยังต้องการ run time libarary อีก 4 แฟ้มคือ

```
/pbcp/bin/corba.dll
/pbcp/bin/pbroker.dll
/pbcp/bin/winsvc.dll
/pbcp/bin/naming.dll
```

จะต้องมีการสร้างเส้นทางผ่านไดเร็คทรอรี(path directory) ในแฟ้น "Autoexec.bat" ด้วยคำสั่ง

```
PATH %PATH%;c:\pbcp/bin
```

การรันโปรแกรมระบบขายปลีกที่เป็นสาขาย่อยสามารถรันได้ 2 กรณีคือ

1. รันทั้งสามโปรแกรมบนเครื่องคอมพิวเตอร์เครื่องเดียว
2. รันทั้งสามโปรแกรมบนแยกเครื่องคอมพิวเตอร์

การรันโปรแกรมทั้ง 2 กรณีจะขั้นตอนและหลักการเหมือนกัน โดยมีลำดับขั้นตอนดังนี้

1. รันโปรแกรม pbnamed.exe

โปรแกรมนี้ต้องรันบนเครื่องที่ติดตั้งระบบปฏิบัติการวินโดวส์ เอ็นที เท่านั้น โดยจะมีออปเจค NameService คอยให้บริการ Naming ให้กับออปเจคอื่นในระบบ โปรแกรมนี้ถูกรันบนเครื่องที่มีหมายเลข IP เป็น 161.246.2.31 ซึ่งหมายเลขนี้จะนำไปใช้ในการรันโปรแกรมของระบบขายปลีกที่เป็นสาขาย่อย โปรแกรมของระบบขายปลีกที่เป็นสาขาย่อยต้องมีลำดับการรันโปรแกรมดังนี้

2. รันโปรแกรม Central.exe

เป็นโปรแกรมที่ต้องรันเป็นอันดับสองเนื่องจากมีออปเจค Depot ที่จะต้องถูกเรียกใช้โดยออปเจค StoreAccess ของโปรแกรม Store.exe ฉะนั้นจึงต้องมีการลงทะเบียนออปเจค Depot ก่อนที่ออปเจค StoreAccess จะเรียกใช้

เรียกใช้โปรแกรมโดยใช้คำสั่ง

Central -pbinit NameService iiop://xxx.xxx.xxx.xxx:6004/NameServiceRoot

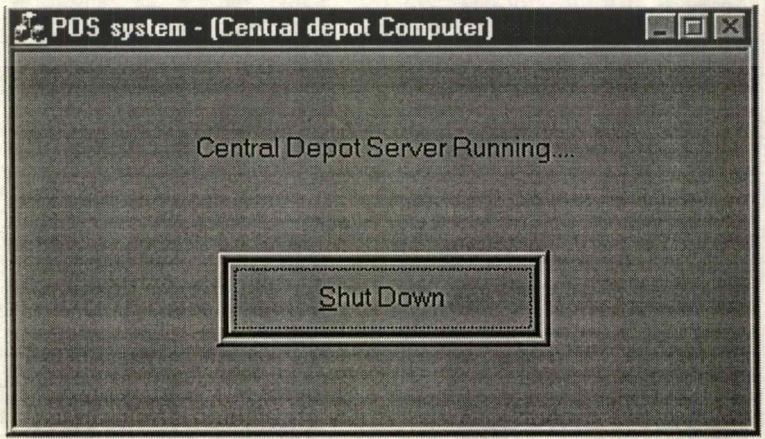
NameService หมายถึงชื่อจริงของออปเจคในบริการ Naming.

iiop://xxx.xxx.xxx.xxx หมายเลข IP ของเครื่องที่รันโปรแกรม pbnamed.exe ซึ่งก็คือ 161.246.2.31

6004 เป็นหมายเลขช่องทางที่ใช้ติดต่อกับออปเจค NameService ซึ่งมาตราฐานของผลิตภัณฑ์นี้ใช้ช่องทางหมายเลข 6004

NameServiceRoot เป็นชื่อที่โปรแกรม pbnamed.exe ใช้เรียกแทนออปเจค NameSerivce

โปรแกรมนี้ทำหน้าที่ให้บริการแก่ร้านค้าเพียงอย่างเดียวไม่มีการให้บริการแก่ผู้ใช้ จึงไม่ได้ออกแบบส่วนติดต่อกับผู้ใช้ มีแต่เพียงปุ่มที่ใช้ปิดโปรแกรม(Shut Down) เท่านั้น



รูปที่ 4.21 แสดงส่วนติดต่อผู้ใช้ ของโปรแกรม Central.exe

3. รันโปรแกรม Store.exe

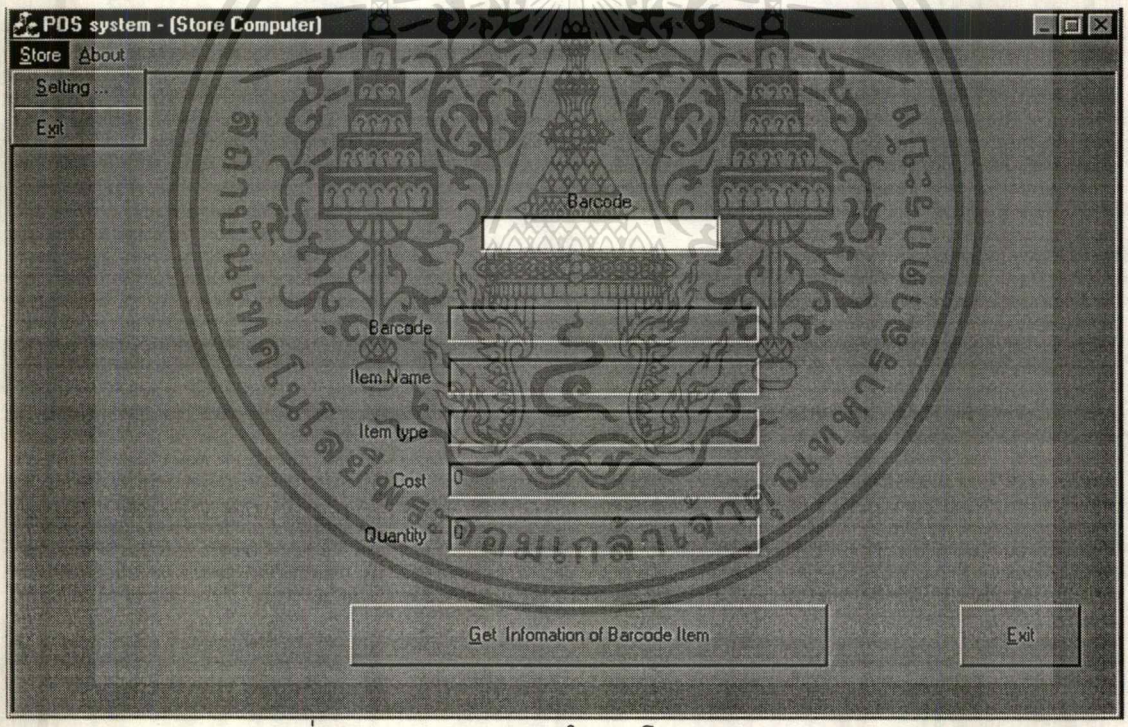
เป็นโปรแกรมที่รันเป็นอันดับสาม มีออปเจค Store และ Tax ที่ถูกเรียกใช้โดยออปเจค POS ของโปรแกรม POS ต้องมีการลงทะเบียนออปเจค Store และ Tax ก่อนที่ออปเจค POS จะเรียกใช้

เรียกใช้โปรแกรมโดยใช้คำสั่ง

Store -pbinit NameService iiop://xxx.xxx.xxx.xxx:6004/NameServiceRoot

การรันโปรแกรม Store.exe จะมีการใช้พารามิเตอร์เหมือนกับการรันโปรแกรม Central.exe แสดงให้เห็นว่าผู้ที่จะเรียกใช้ออปเจครู้เพียงที่อยู่และหมายเลขช่องทางติดต่อของออปเจค NameService ก็สามารถใช้ออปเจคได้แล้ว

โปรแกรมมีหน้าที่ดูแลเครื่องโพสต์ทุกเครื่องในร้าน คอยให้บริการ และ เก็บข้อมูลของเครื่องโพสต์ กำหนดอัตราภาษีของสินค้าทั้งหมด และออกรายงานการขายสินค้า โดยมีส่วนติดต่อผู้ใช้ดังนี้



รูปที่ 4.22 แสดงส่วนติดต่อผู้ใช้ ของโปรแกรม Store.exe

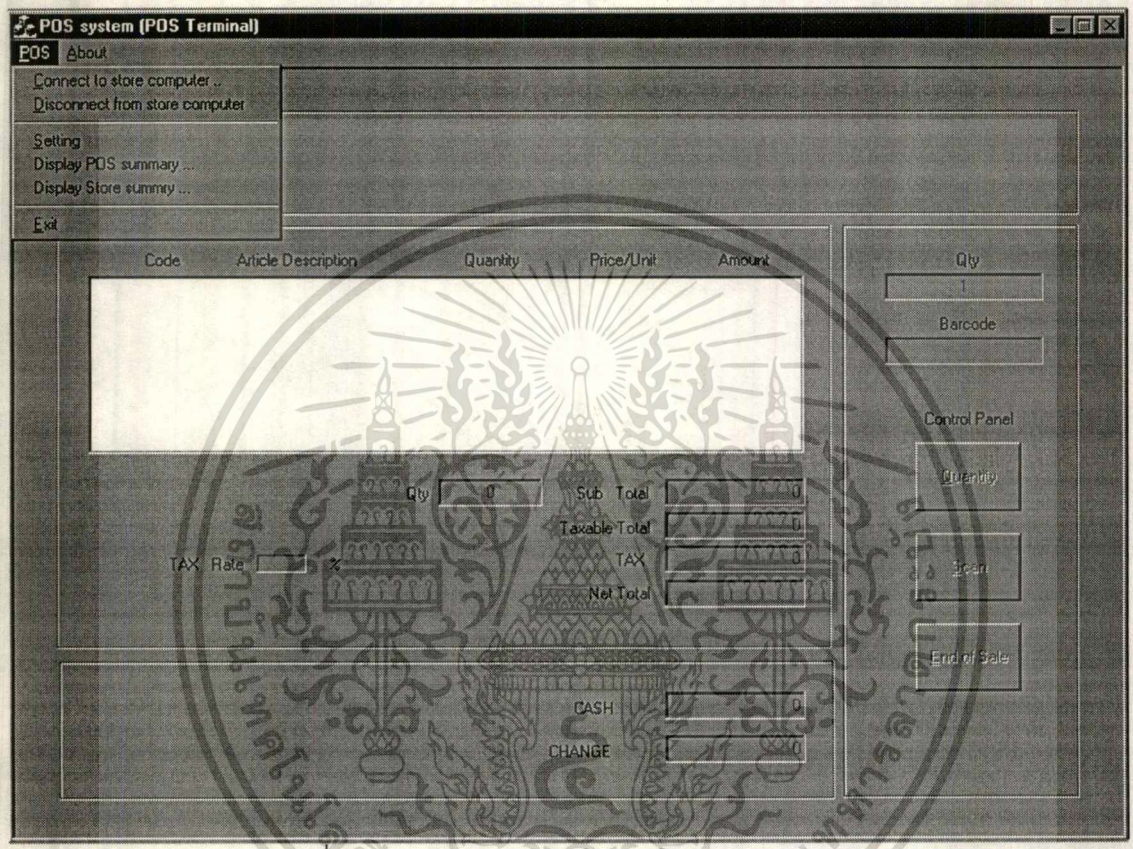
4. รันโปรแกรม Pos.exe

เป็นโปรแกรมที่รันเป็นอันดับสุดท้าย เนื่องจากต้องรอให้ออปเจค Store และ Tax ถูกลงทะเบียนไว้ก่อน ถึงจะเริ่มรันได้เพราะออปเจค POS มีการเรียกใช้ออปเจค Store และ Tax

การเรียกใช้โปรแกรมจะคล้ายกับการรันโปรแกรม Central.exe และ Store.exe โดยใช้คำสั่ง

Pos -pbinit NameService iiop://xxx.xxx.xxx.xxx:6004/NameServiceRoot

โปรแกรมนี้ทำหน้าที่รับรายการซื้อจากลูกค้า และแสดงยอดซื้อที่ลูกค้าได้ซื้อไป ซึ่งเป็นเสมือนอินพุทและเอาท์พุทของระบบ ผู้จัดการร้านค้าสามารถใช้ถามถึงยอดการขายของเครื่องโพสต์ หรือ ยอดการขายของร้านค้าได้



รูปที่ 4.23 แสดงส่วนติดต่อผู้ใช้ ของโปรแกรม Pos.exe

4.8 ปัญหา และ แนวทางการพัฒนาต่อไป

ปัญหา: การเข้าสู่ระบบของพนักงานเก็บเงิน ยังไม่มีระบบรักษาความปลอดภัย มีแต่เพียงการแจ้งไปยังเครื่องคอมพิวเตอร์ร้านค้าเท่านั้น

แนวทางการพัฒนา: มีการใส่รหัสพนักงาน และรหัสผ่าน และเพิ่มการตรวจสอบรหัสผ่านเข้าไปในฟังก์ชั่น Login ของออปเจค Store

ปัญหา: ระบบขายปลีกที่เป็นสขาย่อยที่สร้างนี้เป็นระบบที่มีร้านค้าเพียงร้านเดียว ถ้านำไปใช้กับระบบที่มีร้านค้ามากกว่า 1 ร้านจะทำให้เกิดปัญหา เนื่องจากแต่ละร้านค้าจะสร้างออปเจคชื่อ Store เหมือนกันทำให้เมื่อเครื่องโพสต์จะเข้าสู่ระบบ จะไม่ทราบว่าจะเรียกไปยังออปเจค Store ใด เนื่องจากมีออปเจค Store มากกว่า หนึ่งออปเจค

แนวทางการพัฒนา: เปลี่ยนแปลงวิธีการประกาศออปเจคใหม่ โดยให้ชื่อที่ใช้แทนออปเจคเปลี่ยนแปลงได้ตามหมายเลขร้านค้า เช่นร้านค้าที่ 1 มีชื่อที่ใช้เรียกแทนออปเจค Store_1 ส่วนร้านค้าที่ 2 มีชื่อที่ใช้เรียกแทนออปเจค Store ว่า Store_2 เป็นต้น และเมื่อออปเจค POS ของร้านที่ 1 จะสร้างตัวชี้ก็ให้ใช้ชื่อ Store_1 ในการค้นหาออปเจค

ปัญหา: ไม่มีระบบป้องกัน เมื่อมีการตั้งหมายเลขเครื่องโพสต์ตรงกันมากกว่า 1 เครื่อง เมื่อเครื่องโพสต์ที่มีหมายเลขเครื่องหมายเลขเดียวกันเข้าสู่ระบบ เครื่องโพสต์เหล่านั้นจะมีโครงสร้างข้อมูลที่ใช้เก็บสถานะในออปเจค Store ตัวเดียวกันด้วย ทำให้เมื่อมีการปรับปรุงยอดขายของเครื่องโพสต์จะปรับปรุงที่โครงสร้างนี้ ซึ่งจะไม่รู้เลยว่ายอดขายที่แท้จริงของเครื่องโพสต์แต่ละเครื่องเป็นเท่าไร

แนวทางการพัฒนา: เพิ่มตัวแปรสถานะในโครงสร้างข้อมูลนี้ โดยเมื่อเครื่องโพสต์เครื่องแรกเข้าสู่ระบบก็จะมีการเซตสถานะนี้ว่ามีเครื่องโพสต์ถืออยู่ และเมื่อเครื่องโพสต์อื่นที่มีหมายเลขเดียวกันเข้าสู่ระบบก็จะมาเช็คที่สถานะนี้ก่อนว่ามีเครื่องโพสต์อื่นถืออยู่หรือไม่ ถ้าไม่มีถึงจะเข้าใช้งานระบบได้

ปัญหา: ไม่มีระบบป้องกันการลักลอบใช้ออปเจค ทำให้ถ้ามีบุคคลอื่นที่รู้ชื่อออปเจค และ หมายเลข IP ของเครื่องที่รันบริการ Naming จะสามารถเรียกใช้ออปเจคได้โดยไม่ได้รับอนุญาติ ซึ่งอาจทำให้เกิดความเสียหาได้เช่น ถ้าลักลอบใช้ออปเจค Store อาจจะเข้าไปเปลี่ยนเปอร์เซนต์ของกำไรที่จะคิด ซึ่งค่านี้จะต้องกำหนดโดยผู้จัดการร้านค้าเท่านั้น และถ้าเป็น ออปเจค Tax อาจเข้าไปปรับอัตราภาษีได้

แนวทางการพัฒนา: เพิ่มระบบรักษาความปลอดภัยของการเรียกใช้ออปเจค โดยนำบริการรักษาความปลอดภัย (Security Service) ของ CORBA มาใช้ บริการนี้จะทำให้การเรียกใช้ออปเจคจะต้องรู้รหัสผ่านของแต่ละออปเจคถึงจะนำมาใช้ได้

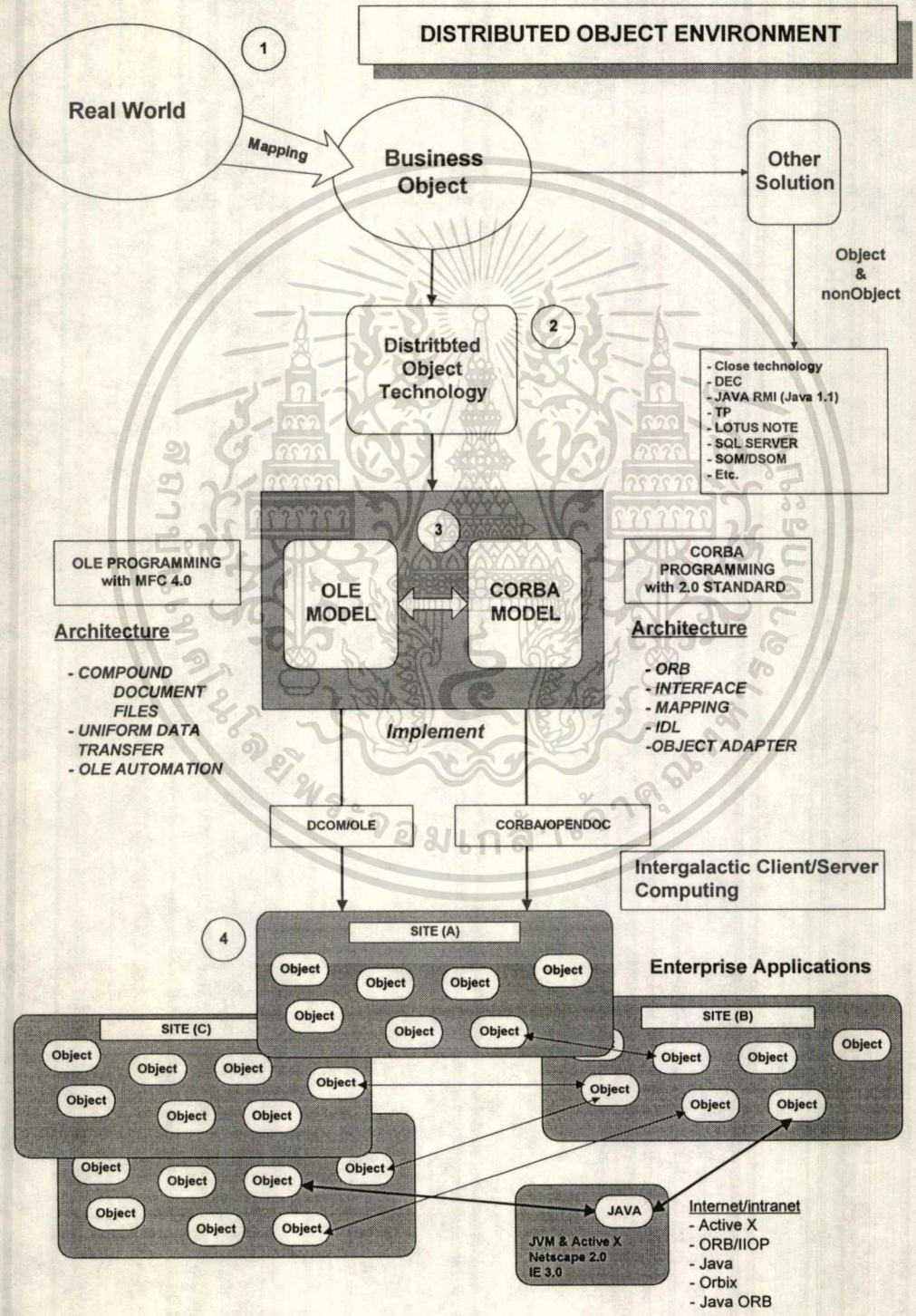
ปัญหา: การเข้าสู่ระบบไม่มีการบันทึกรหัสประจำตัวของพนักงานเก็บเงิน ทำให้ถ้ามีกรณีของการคิดรายการสินค้าของวันที่ผ่านมาผิดพลาด จะไม่สามารถหาพนักงานเก็บเงินที่รับผิดชอบส่วนนั้นได้

แนวทางการพัฒนา: มีการใส่รหัสประจำตัวพนักงานเก็บเงิน โดยการอ่านรหัสประจำตัวจากเครื่องอ่านแถบรหัส เพื่อป้องกันการนำรหัสผู้อื่นมาใช้ และบันทึกเวลา, รหัส และหมายเลขเครื่องโพสต์ ไว้ที่ออปเจค Store

บทที่ 5

บทวิจารณ์และสรุป

จะการศึกษาเทคโนโลยีการโปรแกรมเชิงวัตถุแบบกระจาย ทั้งทางด้านการออกแบบแอพพริเคชั่น และ โปรแกรมมิ่งเราสามารถที่จะวาดแผนผังของเทคโนโลยีทางด้านนี้ได้ดังรูปที่ 5.1



รูปที่ 5.1 แผนภาพของเทคโนโลยีทางด้านออปเจคแบบกระจาย

1. การสร้างแอพพริเคชั่น จะต้องถูกสร้างขึ้นเพื่อที่จะช่วยให้การทำงานทำงานให้มีประสิทธิภาพมากยิ่งขึ้น โดยแอพพริเคชั่นทุกชนิดจะถูกออกแบบได้หลายวิธีด้วยกัน เช่น การใช้ ดาต้าโฟล์ไดอแกรม (Data Flow Diagram) , อี-อาร์ไดอแกรม(E-R Diagram) แต่ถ้าเราจะใช้เทคโนโลยีทางด้านออปเจค เราควรที่จะทำการออกแบบด้วยวิธีการวิเคราะห์แบบออปเจค (Object Analysis and Design) ซึ่งจะมีการออกแบบเป็นเสมือนว่าเป็นโครงสร้างของแอพพริเคชั่นจริงๆ ทำให้เราเห็นโครงสร้างทุกๆส่วนประกอบไปด้วย ออปเจคหลายตัวทำงานร่วมกัน และการโปรแกรมมิ่งก็คือการสร้างออปเจคแต่ละตัวทำงานตาโครงสร้างของออปเจคที่ได้ออกแบบไว้ ตัวอย่าง การวิเคราะห์ออกแบบได้แสดงไว้ในส่วนของแอพพริเคชั่นตัวอย่างในบทที่ 4
2. หลังจากการออกแบบแอพพริเคชั่นขั้นตอนต่อไปคือ การโปรแกรมมิ่งปัจจุบันมีหลายวิธีการที่เราสามารถทำได้ทั้งที่เป็นเทคโนโลยีออปเจคแบบกระจาย หรือแบบที่เป็นออปเจคธรรมดา และอาจสร้างจากซอฟแวร์ทูลต่างๆ ในหัวข้อที่เรากำลังสนใจจะเป็นในส่วนของเทคโนโลยีออปเจคแบบกระจาย การสร้างแอพพริเคชั่นจะสามารถมองได้ว่ามีออปเจคต่างๆ ทำงานร่วมกันเพื่อเกิดเป็นแอพพริเคชั่นขึ้นมา โดยที่ออปเจคไม่จำเป็นที่จะต้องอยู่บนเครื่องเดียวกันก็ได้ ออปเจคแต่ละตัวจะไปทำงานอยู่ในที่สภาพแวดล้อมที่เหมาะสมในเครื่องต่างๆหรือเครื่องเดียวกัน และทำงานร่วมกันโดยใช้โปรโตคอลแบบเดียวกันเพื่อที่จะทำงานร่วมกันได้
3. สำหรับโมเดลที่เป็นมาตรฐานจะมีอยู่สองชนิดใหญ่ๆคือ CORBA และ DCOM (เป็นมาตรฐานของบริษัทไมโครซอฟ) ในที่นี้เราได้เลือกทางด้าน CORBA ทั้งสองเทคโนโลยีสามารถที่จะทำงานร่วมกันได้แต่ต้องใช้เครื่องมือช่วยในการทำแมพปิ่ง
4. แอพพริเคชั่นจะถูกสร้างขึ้นมาโดยมีออปเจคต่างๆทำงานร่วมกัน และสามารถที่จะทำงานร่วมกับอินเตอเนต/อินทราเนตแอพพริเคชั่น โดยทำการกระจายออปเจคเล็กๆที่ทำหน้าที่เป็นไคลเอ็นต์แล้วทำงานร่วมกับออปเจคของเซริฟเวอร์ในลักษณะ การเรียกใช้งานแบบรีโมทออปเจค

ข้อดีที่สำคัญของการใช้เทคโนโลยีออปเจคเบบกระจายได้ดังนี้

1. เป็นวิธีที่เหมาะสมในการพัฒนาแอพริเคชั่นขนาดใหญ่ เพราะโครงสร้างของแอพพริเคชั่นเป็นการยึดเกาะกันแบบหลวม การเปลี่ยนแปลงแก้ไขบางส่วนของโปรแกรมจะทีผลกระทบกับส่วนอื่นน้อยมาก ทำให้ลดค่าใช้จ่ายต้นทุนในการพัฒนาแอพพริเคชั่น
2. สามารถทำการรวมระบบได้ง่ายขึ้น เพราะการทำงานของออปเจคสามารถทำงานต่างสภาวะแวดล้อมได้ และยังทำงานร่วมกับแอพพริเคชั่นของเก่าได้

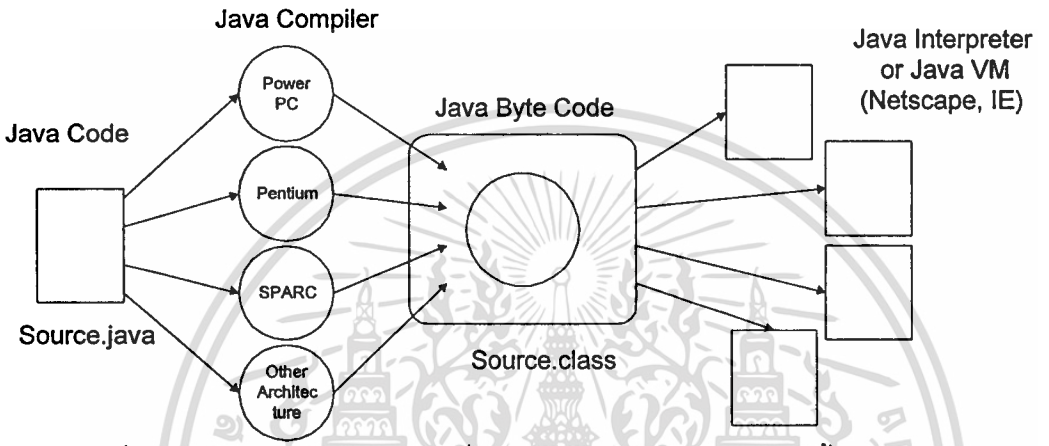
ข้อเสียของการใช้เทคโนโลยีออปเจคเบบกระจายได้ดังนี้

1. ORB ของแต่และบริษัทจะมีการเพิ่มมาตรฐานของตัวเองเข้าไปทำให้เกิดความสับสนว่าอะไรที่เป็นมาตรฐานจริงๆ และ โคดของโปรแกรมจะสูญเสียความเป็นอิสระไป
2. ต้องใช้ระยะเวลาในการศึกษาการใช้งานมาก จึงจะสามารถทำการโปรแกรมได้
3. ราคาของซอฟแวร์ CORBA นั้นสูงมากจึงไม่เหมาะสมกับการพัฒนาแอพพริเคชั่นที่มีขนาดเล็ก ซึ่งเราสามารถใช้วิธีการโปรแรมทั่วไปก็สามารถสร้างแอพพริเคชั่นได้

ภาคผนวก

จาวา (JAVA)

ภาษาจาวาเป็นภาษาที่ออกแบบมาให้มีความสามารถในการทำงานข้ามสภาวะแวดล้อมการทำงานของ ระบบปฏิบัติการที่แตกต่างกัน(Platform) และจัดอยู่ในภาษาประเภท OOP(object oriented Programming)



โค้ดคำสั่งของภาษาจาวาจะเก็บอยู่ในไฟล์ที่มีนามสกุลเป็น .java ต่อจากนั้นจะต้องถูกคอมไฟล์ให้เป็นไบท์โค้ด(Byte Code) ที่เป็นโค้ดกลางที่ทุกๆ จาวาวิชวลแมทชีนเข้าใจ (Java Vitual Machine หรือ Java-VM) จะเห็นว่าพื้นฐานการทำงานของภาษาจาวาจะมีความสามารถในการทำออปเจคแบบกระจายได้เป็นอย่างดีแต่โครงสร้างโดยลำพังของจาวาอย่างเดียวยังทำออปเจคแบบกระจายไม่สมบูรณ์ บริษัท SUN ได้ ออกแบบ จาวาให้มีความสามารถทางด้านการทำ รีโมทอินโวเคชั่นขึ้นมาและเรียกว่า JAVA-RMI (RMI = Remote Method Invocation)

JAVA-RMI จะทำให้ผู้พัฒนาสร้างระบบออปเจคแบบกระจายระหว่าง Java to Java ได้ในลักษณะของการเรียกเมธอดจาก รีโมท์จาวาออปเจคจากวิชวลแมทชีนเครื่องอื่นได้ จาวาโปรแกรมสามารถเรียกรีโมท์ออปเจคได้จะต้องรับ ออปเจคเรฟเฟอร์เรนท์และมองหารีโมทออปเจคในเวลาหลังจากที่มีการทำบูทสแตป โดยใช้การบริการชื่อ (Naming Service) ของ RMI ดังนั้น ไคลเอ็นต์สามารถเรียกรีโมท์ออปเจค ให้กับรีโมทออปเจคตัวอื่นได้เช่นกัน สำหรับรุ่นของ จาวาที่สนับสนุน RMI คือ จากรุ่น 1.02 เป็นต้นไป

การทำงานออปเจคแบบกระจายโดยใช้วิธีนี้ ถูกออกแบบมาใช้กับภาษาจาวาเพียงอย่างเดียว แต่เราสามารถที่จะทำงานร่วมกับ COM ได้โดยการใช้ชุด SDK ของ Microsoft

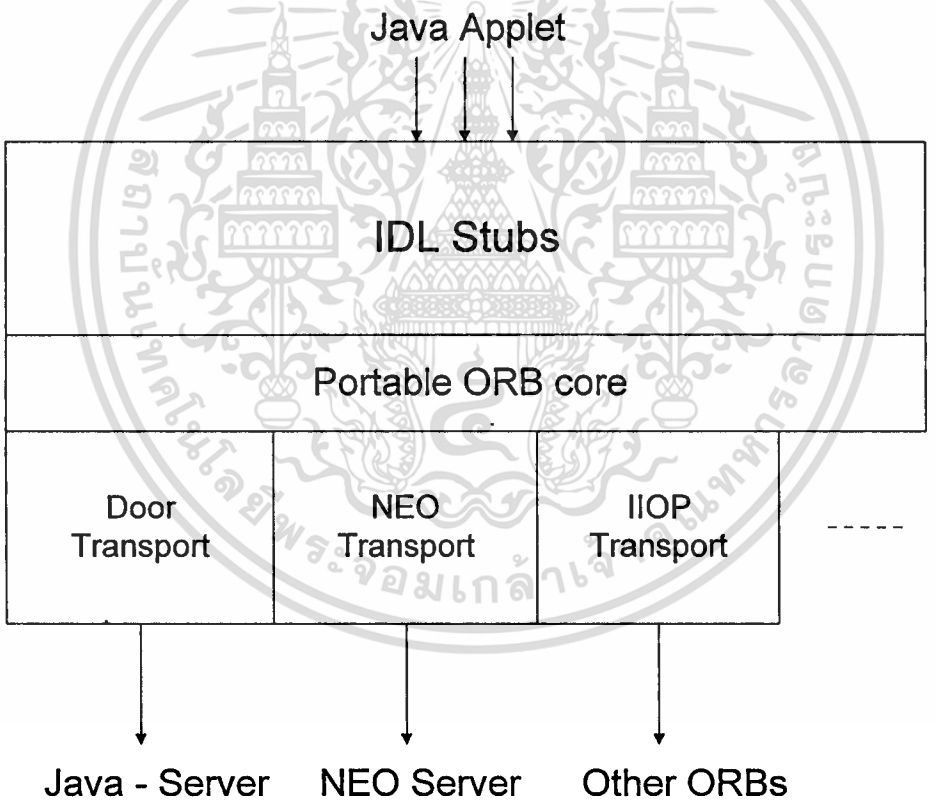
JAVA-IDL

JAVA-IDL เป็นวิธีการที่เราสามารถเชื่อมต่อกับออปเจคของจาวาให้เป็นลักษณะ Transparently ได้โดยทำการเชื่อมต่อกับ จาวาไคลเอ็นต์ โดยใช้มาตราฐานของ IDL ในภาษาจาวารุ่น JAVA-IDL ภาษา IDL จะถูกคอมไพล์ด้วย idlgen เพื่อสร้างเป็นไคลเอ็นต์และเซิร์ฟเวอร์สตีป การเขียน IDL สำหรับ JAVA นั้นสามารถหาข้อมูลเพิ่มเติมได้จาก (http://splash.javasoft.com/JavaIDL/pages/IDLtoJava.html)

การเขียนโดยใช้ JAVA-IDL จะเป็นผลทำให้เราสามารถทำให้ จาวาไคลเอ็นต์เรียก IDL ออปเจคที่อยู่บนรีโมทเซิร์ฟเวอร์ได้ในแบบ transparently หรืออาจมองได้ว่าเป็นลักษณะที่ยอมให้จาวาเซิร์ฟเวอร์กำหนดออปเจคที่สามารถเรียกได้แบบ transparently จาก IDL ไคลเอ็นต์

การเชื่อมต่อกับ ORB

ระบบของ JAVA - IDL มีพื้นฐานอยู่บน Portable JAVA ORB core ซึ่งมีโครงสร้างที่ สนับสนุนในการทำงานร่วมกับ ORB โปรโตคอลตัวอื่น



การใช้ JAVA - IDL จะทำให้เราเขียน Applet ที่สามารถไปเรียกออปเจคบน ORB ตัวอื่นได้โดย Applet จะทำเสมือนเป็น GUI ออปเจค

DCOM (Distributed Component Object Model)

DCOM เป็นเทคโนโลยีที่พัฒนามาจากบริษัท Microsoft เพียงบริษัทเดียว DCOM จะใช้เทคโนโลยีของ COM (Commond Object Model) เป็นคอมโพเนนท์หลักเพียงแต่ว่าเป็น COM ที่มีการเชื่อมต่อกันได้โดยมีพื้นฐานอยู่บน Open Software Foundation's DCE-RPC ที่ระดับล่างของ COM จะเป็นแกนหลักที่ใช้ใน Microsoft ActiveX เทคโนโลยีของ ActiveX สามารถที่จะใช้ร่วมกับ อินเตอร์เนตเทคโนโลยีที่มีอยู่แล้วเช่น JAVA, HTTP โปรโตคอล และ TCP/IP โปรโตคอล ซึ่งสามารถทำให้เราสามารถรวมเดสทอป แอพพริเคชั่น ให้สามารถทำงานข้ามเครือข่ายอินเตอร์เนต ผ่านทาง WWW (World Wide Web) ได้

แต่ในช่วงเวลานี้ จะอยู่ในช่วงพัฒนาโดยอยู่ในช่วงการทดสอบของรุ่นเบต้า (Beta testing)

เนื่องจาก DCOM โดยพื้นฐานของมันจะใช้ COM เป็นหลัก ดังนั้นการทำการเปรียบเทียบข้อดีข้อเสียสามารถที่จะกระทำระหว่าง คอมโพแนนท์ของ COM และ CORBA

ตารางการเปรียบเทียบระหว่าง COM และ OLE ในลักษณะความเป็นออปเจค

เรื่อง	**COM/OLE**	**CORBA/OpenDoc**
ลักษณะของคอมโพเนนท์มีลักษณะเหมือนอะไร **หมายเหตุ** white box สามารถที่จะทำการสืบทอดได้	Black box	Black box หรือ White box
คอมโพเนนท์สามารถทำการ multiple อินเตอร์เฟส	ใช่	ใช่
คาสมี IDL ที่ไม่ซ้ำกัน	ใช่	ใช่
รันไทม์ออปเจคมี IDL ที่ไม่ซ้ำกัน	ไม่	ใช่

การเปรียบเทียบลักษณะของออปเจคบัส

เรื่อง	COM/OLE		CORBA/OpenDoc	
	Local	Remote	Local	Remote
การเรียกเมธอดแบบสแตติก	ใช่	ไม่	ใช่	ใช่
การเรียกแบบไดนามิก	ใช่	ไม่	ใช่	ใช่
มีการใช้ IDL **หมายเหตุ** OLE มี ODL เพื่อใช้อธิบายเกี่ยวกับเมต้าดาต้า และ Microsoft ใช้ IDL/MIDL อธิบายการเชื่อมต่อผ่าน RPC	ใช่	ไม่	ใช่	ใช่
Implementation Repository	ใช่	ไม่	ใช่	ใช่

การเปรียบเทียบในรูปบริการต่างๆ ของระบบ (The System Service)

เรื่อง	COM/OLE		CORBA/OpenDoc	
	Local	Remote	Local	Remote
Events	ใช่	ไม่	ใช่	ใช่
Life Cycle (**Note:** OLE provides Life Cycle via IClassFactory and IUnknown)	ใช่	ไม่	ใช่	ใช่
Naming (**Note:** OLE provides naming via monikers)	ใช่	ไม่	ใช่	ใช่
Persistence	ใช่	ไม่	ใช่	ใช่
ODBMS integration	ไม่	ไม่	ใช่	ใช่
RDBMS integration (**Note:**Microsoft is working on an OLE DB specification)	ไม่	ไม่	ใช่	ใช่
Externalization	ใช่	ไม่	ใช่	ใช่
Transactions (**Note:** Microsoft is working on an OLE Tramsactions specification)	ไม่	ไม่	ใช่	ใช่
Concurrency Control	ไม่	ไม่	ใช่	ใช่
Relationships	ไม่	ไม่	ใช่	ใช่
Query (**Note:**Microsoft is working on an OLE DB specification)	ไม่	ไม่	ใช่	ใช่
Licensing	ใช่	ไม่	ใช่	ใช่
Properties	ใช่	ไม่	ใช่	ใช่
Versioning (**Note:** OpenDoc provides version control today; CORBA Change Management is expected in 1996)	ไม่	ไม่	ใช่	ใช่
Security (**Note:** Individual ORBs provide security; the CORBA Security Service is expected late 1995)	ไม่	ไม่	ใช่	ใช่
Garbage collection (**Note:** OpenDoc augments CORBA by providing reference counting; OLE provides reference counting via IUnknown)	ใช่	ไม่	ใช่	ไม่

การเปรียบเทียบในเชิงดอคคิวเมนท์ เฟรมเวิร์ค (document framework)

เรื่อง	COM/OLE	CORBA/OpenDoc
Compound document storage	ใช่ (DocFiles)	ใช่ (Bento)
Document draffs	ไม่	ใช่ (Bento versions)
Automation and scripting	ใช่	ใช่
Semantic message resolution	ไม่	ใช่
Recordable scripts	ไม่	ใช่
Irregularly-shaped parts	ไม่	ใช่
Multiframed parts	ไม่	ใช่
In-place editiong and inside-out levels of nesting	2	ไม่จำกัด
Bitmap caching for inactive parts	ใช่	ไม่
Drag-and-Drop	ใช่	ใช่
Clipboard	ใช่	ใช่
Linking (**Note:** OpenDoc provides persistent Link Ids that can be distributed if a networked ORB is persent)	ใช่	ใช่

จากการเปรียบเทียบเราจะพบว่า CORBA ออปเจคจะเป็นคอมโพเนนท์ที่สามารถทำออปเจคแบบกระจายได้ดีกว่า COM ซึ่งเมื่อมาตรฐานของ DCOM สร้างขึ้นมาสำเร็จ อาจมีความสามารถเท่ากันหรือดีกว่าได้ (CORBA นั้นพัฒนามาก่อน DCOM เป็นเวลา 2 ปี)

ข้อได้เปรียบที่สำคัญของ COM คือ ระบบปฏิบัติการที่ใช้ส่วนใหญ่บนเครื่องเดสทอป จะเป็นวินโดว์ ซึ่งใช้ OLE/COM เป็นมาตรฐานอยู่แล้ว ดังนั้นจึงมีฐานการสนับสนุนอยู่มากกว่าทางCORBA ซึ่งผู้ใช้ส่วนใหญ่จะเป็นนักพัฒนาโปรแกรมในองค์กรขนาดใหญ่

กิตติกรรมประกาศ

ขอขอบคุณ บริษัท Expersoft Corporation แห่งประเทศอเมริกา ที่ได้เอื้อเฟื้อโปรแกรมชุดทดลองในการศึกษาระบบการโปรแกรมออปเจคแบบกระจาย Powerbroker CORBAplus for VC++ 4.0 และขอขอบคุณ สถานีวิทยุกระจายเสียงแห่งประเทศไทย ที่ได้เอื้อเฟื้อสถานที่ และ ระบบคอมพิวเตอร์ซึ่งประกอบไปด้วยเครือข่ายระบบแลน และ เซิร์ฟเวอร์ เพื่อใช้ในการทดลองการโปรแกรม

หนังสืออ้างอิง

1. Robert Orfali, Dan Harkey, AND Jeri Edwards, "Intergalactic Client/Server Computing", Byte, Vol.20, No. 4, 1995, pp. 108-160
2. Robert Orfali, Dan Harkey, AND Jeri Edwards, "เตรียมต้อนรับระบบ Client/Server ระดับจักรวาล", Byte Thailand, ฉบับที่ 13, 2538, หน้า 79-87
3. เกษมสันต์ พานิชการ, "C++ และหลักการของ OOP ฉบับเริ่มต้น", บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน), 312 หน้า, 2537.
4. David J.Kruglinski และ โชคชัย เตชพรรุ่ง, "การเขียนโปรแกรม Visual C++ เวอร์ 1.5 บนวินโดวส์", บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน), 297 หน้า, 2538
5. Kaare Christian และ อนิรุต ลิ่วหาทอง, "การเขียนโปรแกรมบนวินโดวส์ด้วย Microsoft C++", บริษัท ซีเอ็ดยูเคชั่น จำกัด(มหาชน), 649 หน้า, 2539
6. Richard C. Leinecker&Jamie Nye, "Visual C++ Power Toolkit", Ventana Press, 785 p., 1995
7. Chane Cullens, Mark Davidson, Chris Corry, Paul Robichaux, Steve Potts, and Kate Gregory, "Special Edition Using Visual C++ 4", Que, 664 p., 1996
8. David Curtis, "PowerBroker CORBAplus for C++ User Manual", Expersoft Corporation, 257p.,1995.
9. David Curtis, "PowerBroker Reference Manual", Expersoft Corporation, 1995
10. David J. Kruglinski, "Inside Visual C++ ", Microsoft Press, 896 p., 1996.
11. Jon Siegel, "CORBA Fundamentals and Programming", John Wiley&Sons, 693 p., 1996.
12. Thomas J. Mowbray. PhD,and Ron Zahavi, "The Essential CORBA : Systems Integration Using Distributed Object", John Wiley&Sons, 316 p.,1995.
13. Robert Orfali, Dan Harkey, and Jeri Edwards, "The Essential Distributed Objects Survival Guide", John Wiley&Sons, 604 p., 1996