การเขียนโปรแกรมใช้งานร่วมภายใต้ระบบเครือข่าย

Distributed Object Client/Server using CORBA

โดย

นาย ภักดี	มูซอ	รหัสนักศึกษา	38013284
นาย วารินทร์	รัตนรจนานนท์	รหัสนักศึกษา	38013288
นาย สุวิทย์	ไพบูลย์	รหัสนักศึกษา	38013299

อาจารย์ที่ปรึกษา

อาจารย์ บรรจง ปิยธำรง

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การเขียนโปรแกรมใช้งานร่วมภายใต้ระบบเครือข่าย

(Distributed Object Client/Server using CORBA)

ผู้จัดทำ

1. นาย ภักดี มูซอ รหัสนักศึกษา 38013284
2. นาย วารินทร์ รัตนรจนานนท์ รหัสนักศึกษา 38013288
3. นาย สุวิทย์ ไพบูลย์ รหัสนักศึกษา 38013299

.....อาจารย์ที่ปรึกษา

(อาจารย์ บรรจง ปิยธำรง)

การเขียนโปรแกรมใช้งานร่วมภายใต้ระบบเครือข่าย

นาย ภักดี มูซอ
นาย วารินทร์ รัตนรจนานนท์
นาย สุวิทย์ ไพบูลย์
อาจารย์ที่ปรึกษา อาจารย์ บรรจง ปิยะธำรง

บทคัดย่อ

CORBA เป็นมาตรฐานหนึ่งของการสร้างโปรแกรมประยุกต์บนระบบเครือข่าย ซึ่งประกอบไปด้วยองค์ประกอบหลายส่วนด้วยกันเช่น ORB ที่ช่วยในการติดต่อสื่อสารระหว่างไคลเอนท์กับเซริฟเวอร์ หรือ IDL ซึ่งเป็นภาษาที่ช่วยให้การเขียนโปรแกรมภายใต้ระบบเครือข่ายง่ายขึ้น จุดสำคัญอีกประการของมาตรฐาน CORBA ก็คือการมุ่งเน้นไปในเรื่องของ Object Oriented ในแบบของ Common Object ซึ่งเป็นการมองทุกอย่างในระบบเป็นออบเจ็กต์รวมทั้งเซริฟเวอร์ด้วย ซึ่งออบเจ็กต์แต่ละตัวจะมีความสามารถในการให้บริการได้เหมือนกัน

สำหรับลักษณะการทำงานตามมาตรฐาน CORBA สามารถแสดงได้โดยการจำลองระบบการถอนยอดบัญชีของลูกค้าธนาคาร ซึ่งจะแสดงให้เห็นได้ถึงลักษณะการทำงานของโปรแกรมประยุกต์ภายใต้มาตรฐานนี้เริ่มตั้งแต่การสร้างออบเจ็กต์เซริฟเวอร์ไปจนกระทั่งเมื่อออบเจ็กต์เซริฟเวอร์ให้บริการเรียบร้อยมีการส่งค่ากลับมาให้ไคลเอนท์อย่างไรด้วย

CORBA เป็นมาตรฐานที่น่าจะเป็นที่ยอมรับกันโดยทั่วไป เนื่องจากเกิดขึ้นโดยความร่วมมือของบริษัทต่างๆ มากมายกว่า 700 บริษัท แต่เราก็คงต้องยอมรับเช่นกันว่า CORBA ยังคงเป็นมาตรฐานที่มีทั้งข้อดี-ข้อเสีย และจุดที่ยังคงต้องได้รับการพัฒนาต่อไปในอนาคต

Distributed Object Client/Server using CORBA

Mr. Pakdee	Muso	ID	38013284
Mr. Warin	Rattanarojjananon	ID	38013288
Mr. Suwit	Phaiboon	ID	38013299
Advisor	Mr. Bunjong	Piyathumrong	

Abstract

CORBA is a standardization that used for implement an application on the Client/Server system. It consists of many subjects such as ORB that link the communication between client and server or an IDL. An IDL is a language that used to write the application easily. The topic of CORBA standardization is to focus in Object Oriented base for Common Object, so every one on the system is the object include server. Each object has an ability to service look like ones.

CORBA standardization can be present by simulation a check balance system. So we can present since initialize the object server, the client invoke it and the final is that the object server return result to the client.

CORBA standardization will be accepted, because it was born by the consortium of more than 700 companies. But we must accept that CORBA is a standardization that has the good-the bad and want to be developed in the future.

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญภาพ

หน้า

สารบัญภาพ (ต่อ)

วัตถุประสงค์

1. เพื่อศึกษาหารายละเอียด,เนื้อหาที่เกี่ยวกับ Distributed Object Client/Server
2. เพื่อทำการศึกษาลักษณะของสถาปัตยกรรมที่มีชื่อว่า CORBA หรือ Common Object Request Broker Architecture
3. เพื่อทำการศึกษา Tools ต่างๆ ที่สร้างขึ้นโดยอ้างอิงกับมาตรฐาน CORBA ในการนำมาประยุกต์ใช้ภายใต้ระบบ Distributed Object Client/Server
4. เพื่อแสดงให้เห็นถึงลักษณะการทำงานของโปรแกรมประยุกต์ที่สร้างขึ้นโดยอาศัยมาตรฐาน CORBA
5. เพื่อวิเคราะห์หาข้อดีและข้อเสียของโปรแกรมประยุกต์ที่ใช้มาตรฐาน CORBA

แนวทางการทำงาน

1. ศึกษาถึงที่มาและจุดประสงค์ในการกำหนดมาตรฐาน CORBA
2. ศึกษาลักษณะการทำงานของโปรแกรม Client/Server ที่มีมาแต่เดิม
3. ค้นคว้าเพื่อเปรียบเทียบระหว่าง CORBA กับมาตรฐานอื่นๆ ที่มีความใกล้เคียงกัน
4. ทำการศึกษาเครื่องมือต่างๆ ที่สร้างขึ้นเพื่อรองรับการทำงานในมาตรฐาน CORBA
5. ทำการเลือกเครื่องมือที่มีความชัดเจนและมีปัญหาในการใช้งานน้อยที่สุด
6. ทำการวางขอบเขตของโปรแกรมประยุกต์ที่ต้องการจะสร้าง
7. ทำการแก้ไขไฟล์บางส่วนที่สามารถนำมาใช้กับโปรแกรมประยุกต์ได้ และทำการสร้างไฟล์บางส่วนขึ้นมาใหม่เพื่อให้โปรแกรมประยุกต์มีความสมบูรณ์มากยิ่งขึ้น
8. ทดสอบระบบงานที่สร้างขึ้นและแก้ไขพัฒนาให้เกิดความสมบูรณ์และมีความสวยงาม

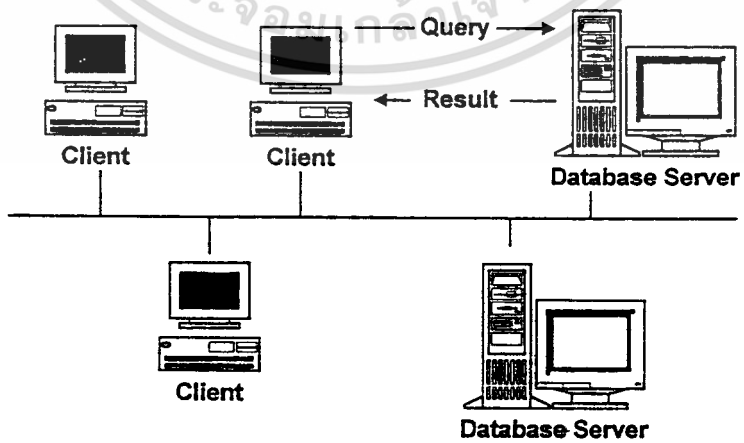
บทที่ 1 นำเรื่อง (Introduction)

Distributed Object Computing เป็นเทคโนโลยีที่ช่วยทำให้โปรแกรมประยุกต์ต่างๆ ที่อยู่บนเครือข่ายสามารถทำงานร่วมกันได้สะดวกมากขึ้น งานบริการและโปรแกรมประยุกต์เหล่านี้นั้นถูกมองเป็นออบเจ็กต์ที่สามารถใช้วิธีการติดต่อพื้นฐานสื่อสารกันได้

การทำงานร่วมกันหรือการติดต่อกันระหว่างออบเจ็กต์ที่อยู่ในโปรแกรมเดียวกันนั้นไม่ได้ยุ่งยากซับซ้อนมากนักเพราะมักสร้างขึ้นด้วยภาษาที่เหมือนกัน ทำงานบนเครื่องเดียวกันและบนระบบปฏิบัติการ (Operating System : OS) ชนิดเดียวกัน แต่ในระบบ Distributed Object นั้น ออบเจ็กต์กระจัดกระจายอยู่ทั่วไปในระบบเครือข่ายที่หลากหลายรูปแบบและโปรแกรมประยุกต์ต่างๆ ก็สร้างขึ้นด้วยภาษาที่แตกต่างกัน สิ่งที่ยากลำบากคือจะทำอย่างไรที่จะทำให้โปรแกรมประยุกต์ทั้งหลายนั้นสามารถติดต่อสื่อสารกันได้ถึงแม้ว่ามันจะถูกสร้างขึ้นด้วยภาษาที่แตกต่างกัน ทำงานบนเครื่องที่แตกต่างกันและบนระบบปฏิบัติการต่างกัน

หลักการของ Client/Server

Client/Server เป็นโครงสร้างของระบบคอมพิวเตอร์รูปแบบหนึ่งที่แบ่งแยกการประมวลผลข้อมูลออกเป็น 2 ฝั่งคือฝั่งไคลเอนท์ (Client) หรือผู้ขอใช้บริการและเซริฟเวอร์ (Server) หรือผู้ให้บริการ โดยฝั่งไคลเอนท์จะมีระบบฟรอนท์เอ็นด์ (Front End System) หรือส่วนที่มีโปรแกรมประยุกต์ด้านฐานข้อมูล (Database Application) ทำงานอยู่ และฝั่งฐานข้อมูลเซริฟเวอร์ (Database Server) จะมีระบบแบคเอ็นด์ (Back End System) หรือส่วนที่ทำหน้าที่เป็น DBMS ตัวจริง ๆ ทำงานอยู่และอินพุท เอาท์พุทของผู้ใช้และระบบแบคเอ็นด์จะจัดการการประมวลผลข้อมูลและทำการติดต่อกับดิสก์ (Disk Access) เช่น เมื่อผู้ใช้ระบบฟรอนท์เอ็นด์สร้างคิวรี่ (Query) เพื่อสอบถามข้อมูลจากฐานข้อมูลเซริฟเวอร์โดยผ่านระบบเครือข่าย ส่วนเซริฟเวอร์ก็จะทำการค้นหาข้อมูลที่ผู้ใช้ต้องการแล้วส่งข้อมูลกลับไปให้ดังรูป



รูป 1-1 Client/Server

วัตถุประสงค์หลักของระบบ Client/Server คือการอนุญาตให้โปรแกรมประยุกต์ของผู้ขอใช้บริการเข้ามาเรียกใช้ข้อมูลที่ถูกจัดการโดยผู้ให้บริการได้ โดยผู้ให้บริการสามารถทำงานอยู่ในเครื่องที่ตั้งอยู่ในที่ห่างไกลกับเครื่องที่ผู้ขอใช้บริการกำลังทำงานอยู่

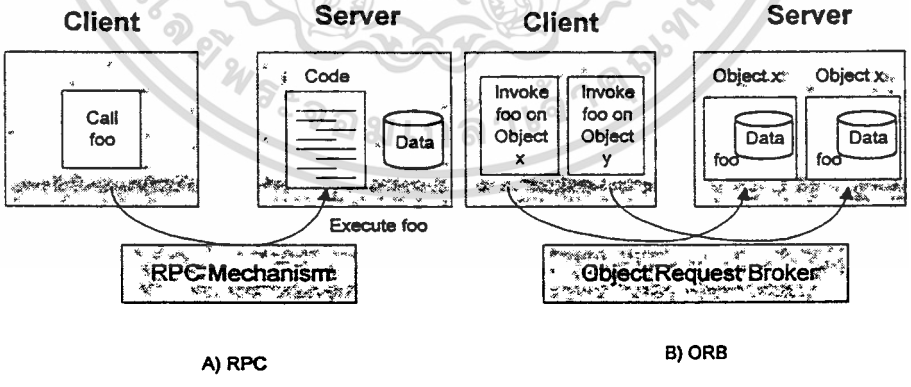
ระบบเชิงกระจายวัตถุและไคลเอนท์เซริฟเวอร์

(Distributed Object System &Client/Server)

ระบบเชิงกระจายวัตถุเป็นระบบที่มีโปรแกรมประยุกต์ต่างๆ ที่ถูกมองอยู่ในรูปของออบเจ็กต์ทำงานร่วมกันผ่านระบบเครือข่าย โดยมีความสัมพันธ์กันแบบ Client/Server การนำ CORBA มาพัฒนาใช้กับระบบเช่นนี้เป็นข้อได้เปรียบสำหรับผู้พัฒนามาก เพราะช่วยลดความซับซ้อนในการสร้างโปรแกรมประยุกต์เหล่านั้นได้

ORB เป็น middleware ที่ก่อให้เกิดความสัมพันธ์แบบ Client/Server ระหว่าง Object โดยไคลเอนท์สามารถร้องขอการทำงานจากเซริฟเวอร์ (อาจจะอยู่บนเครื่องเดียวกันหรือบนเครือข่าย) ผ่าน ORB ที่ทำงานอยู่เบื้องหลัง โดยที่ ORB บนเครื่องเซริฟเวอร์ จะทำการค้นหาออบเจ็กต์ที่ไคลเอนท์ต้องการจาก ORB ที่กระจายอยู่ในเครื่องอื่นๆ และส่ง operation และ parameter ไปให้มันทำงาน จากนั้นก็จะส่งผลลัพธ์ที่ได้กลับไปให้ไคลเอนท์

จากรูปจะเห็นการเปรียบเทียบระหว่างวิธีการทำงานของ RPC หรือ Remote Procedure Call นั้นจะเรียกใช้ฟังก์ชันใช้งานเฉพาะที่ต้องการเท่านั้น แต่วิธีการของ ORB จะเรียกใช้หรือร้องขอทุกอย่างเป็นออบเจ็กต์



รูป 1-2 แสดงการเปรียบเทียบการเรียกใช้ Client/Server ระหว่าง RPC กับ ORB

ประวัติความเป็นมาของ CORBA

CORBA ย่อมาจากคำว่า Common Object Request Broker Architecture เกิดขึ้นในปี ค.ศ.1990 โดยกลุ่มบริษัทที่เรียกตัวเองว่า OMG หรือ Object Management Group อันประกอบไปด้วยบริษัทต่าง ๆ มากมายทั้งเล็กและใหญ่กว่า 700 บริษัทได้แก่ Microsoft, DEC, IBM, Visigenic etc. CORBA เป็นมาตรฐานในการติดต่อสื่อสารของโปรแกรมใช้งานในเครือข่ายที่เรียกว่า Client/Server ซึ่งอยู่ในรูปแบบของการใช้งานออบเจ็กต์ร่วมกัน (Common Object) การที่บริษัทต่างๆ มารวมตัวกันนั้นก็เนื่องมาจากความต้องการที่จะให้มีมาตรฐานเดียวกันในการติดต่อสื่อสารระหว่างโปรแกรมใช้งานในระบบเครือข่าย Client/Server โดยที่ไม่คำนึงถึงว่าจะมีความแตกต่างกันมากน้อยเพียงใดระหว่างเครื่องไคลเอนท์กับเซริฟเวอร์ไม่ว่าจะเป็นโปรแกรมประยุกต์ใช้งาน (Application), ระบบปฏิบัติการ(Operating System : OS) หรือแพลตฟอร์ม (Platform) ของเครื่องคอมพิวเตอร์นั้นๆ

ในที่สุดทางกลุ่มก็จึงได้ประกาศมาตรฐานใหม่ของการติดต่อสื่อสารระหว่างโปรแกรมในเครือข่าย Client/Server นี้ออกมาชื่อว่า CORBA ซึ่งกำหนดรูปแบบหรือสถาปัตยกรรมของการติดต่อสื่อสารระหว่างไคลเอนท์กับเซริฟเวอร์ที่มีชื่อเรียกว่า OMA หรือ Object Management Architecture ขึ้นมา อันเป็นการแสดงให้เห็นถึงโครงสร้างในการติดต่อเรียกใช้บริการและการให้บริการระหว่างไคลเอนท์กับเซริฟเวอร์ในรูปแบบของออบเจ็กต์ ในช่วงแรกมีชื่อรุ่นว่า CORBA 1.0 ซึ่งนอกจากจะประกาศใช้มาตรฐาน CORBA 1.0 แล้วสมาชิกของ OMG ก็ได้สร้างเครื่องมือต่างๆ ที่อ้างอิงกับมาตรฐาน CORBA 1.0 หลายตัวเช่น Visibroker ของ Visigenic หรือ Orbix Web ของ Iona โดยในช่วงแรกเครื่องมือส่วนใหญ่ที่ออกมายังมุ่งเน้นไปในเรื่องของการเขียนโปรแกรม Client/Server แบบ LAN (Local Area Network) มากกว่าแบบ Internet แต่ต่อมา OMG ก็ได้ทำการศึกษาและพัฒนา CORBA 1.0 จนทำให้ในปี ค.ศ.1994 ทาง OMG ก็ได้ออกมาตรฐาน CORBA ตัวใหม่ออกมา ซึ่งมีการเพิ่มเติมคุณสมบัติต่าง ๆ เข้าไปอีกมากมาย และหนึ่งในคุณสมบัติที่เพิ่มเข้าไปก็คือ Protocol ที่ใช้บนเครือข่าย Internet ที่มีชื่อว่า IIOP หรือ Internet Inter-ORB Protocol ซึ่งมาตรฐาน CORBA ที่ออกมาในภายหลังมีชื่อรุ่นว่า CORBA 2.0 โดยปัจจุบันนี้เครื่องมือที่ถูกพัฒนาขึ้นมาส่วนใหญ่ก็มักจะอ้างอิงกับมาตรฐาน CORBA 2.0 นี้เอง เนื่องจากมีคุณสมบัติและความสามารถค่อนข้างที่จะสมบูรณ์และครอบคลุมการทำงานได้ดีมากพอสมควร

การเปรียบเทียบระหว่าง CORBA กับ Active X

มาตรฐาน CORBA นั้นเป็นมาตรฐานซึ่งกำเนิดจากกลุ่ม OMG(Object Management Group) โดยที่ CORBA นี้มิได้เป็นมาตรฐานของ Distributed Object เพียงมาตรฐานเดียว แต่กลับมีคู่แข่งอีกซึ่งพยายามที่จะประกาศตัวให้เป็นที่ยอมรับโดยทั่วกันว่าเป็นมาตรฐานของ Distributed Object ด้วยเช่นเดียวกันเช่น Active X เป็นต้น Active X เป็นชื่อที่เรียกกันโดยทั่วไปของเทคโนโลยีจำพวก OLE (Object Link and Embedding), COM (Component Object Model) หรือ DCOM (Distributed Component Object Model) ซึ่งเป็นของบริษัทไมโครซอฟท์ แต่ในขณะที่ Active X กำลังจะก้าวขึ้นมาแข่งขันกับ

CORBA นั้นก็ดูเหมือนว่าจะพบกับจุดบกพร่องมากมายซึ่งทำให้ CORBA ยังคงได้เปรียบอยู่ สำหรับข้อเปรียบเทียบบางอย่างระหว่าง CORBA กับ Active X สามารถที่จะแสดงได้ดังนี้

1. สถาปัตยกรรม,นิยามและรายละเอียด (Architecture, Definition and Specification)

เนื่องจากไมโครซอฟท์ได้กล่าวว่า Active X นั้นเป็นชื่อทั่วไปของเทคโนโลยีจำพวก OLE หรือ DCOM ดังนั้นจึงเป็นการยากที่จะให้คำนิยามเกี่ยวกับ Active X นอกจากนี้ในรายงานของไมโครซอฟท์ยังได้กล่าวไว้ว่า OLE นั้นได้ถูกนิยามให้เป็นขอบข่ายของการติดต่อในแบบ COM ซึ่งนั่นก็หมายถึงว่า Active X จะถูกแสดงในลักษณะของภาษาที่เป็นพื้นฐานในการติดต่อกันบนส่วนประกอบต่างๆ ที่เป็นเทคโนโลยีแบบ Active X แต่อย่างไรก็ตามได้เกิดความขัดแย้งกันขึ้น เนื่องจาก Active X นั้นบังคับให้มีการเขียนโปรแกรมโดยใช้การติดต่อแบบ 32 บิต แต่ได้เกิดการต่อต้านขึ้นโดยองค์กรที่ชื่อ European Computer Manufacturers Association (ECMA) จะเห็นได้ว่า Active X นั้นขาดภาพรวมของสถาปัตยกรรมซึ่งแตกต่างกับ CORBA โดยสิ้นเชิง เพราะ OMG ได้มีการสร้างสถาปัตยกรรมที่เป็นที่ยอมรับกันโดยทั่วไป รวมทั้งมีการกำหนดการให้บริการไว้ภายในโดยเรียบร้อยแล้ว จึงทำให้ CORBA มีความชัดเจนในเรื่องของสถาปัตยกรรมและมีนิยามที่ชัดเจนมากกว่า Active X

2. การรองรับการทำงานข้ามระบบปฏิบัติการ (Cross-platform Support)

ในปัจจุบัน Active X หรือ DCOM สามารถทำงานได้ดีบนระบบปฏิบัติการที่เป็นของค่ายไมโครซอฟท์เท่านั้น โดยเฉพาะอย่างยิ่งบน Windows '95 หรือ Windows NT ซึ่งต่างกับ CORBA ที่มีความสามารถในการทำงานครอบคลุมทั้งในระดับของระบบปฏิบัติการรุ่นเก่าเช่น MS-DOS,Windows 3.x จนกระทั่งระบบปฏิบัติการรุ่นใหม่อย่างเช่น UNIX, OS/2 หรืออื่นๆ สำหรับในจุดนี้ทางไมโครซอฟท์เองก็ได้ยอมรับว่า Active X สามารถทำงานได้ดีและให้ความมั่นใจได้เฉพาะบน Windows เท่านั้น ดังนั้นจึงเห็นได้ว่า CORBA นั้นเป็นสถาปัตยกรรมที่สามารถทำงานข้ามระบบปฏิบัติการกันได้ โดยที่ไม่เกิดปัญหาแต่อย่างใด

3. การรองรับการทำงานข้ามโปรแกรมภาษา (Cross-language Support)

เป็นที่ยอมรับกันโดยทั่วไปว่าโปรแกรมที่ใช้กันในธุรกิจปัจจุบันนี้ ส่วนหนึ่งยังเป็นโปรแกรมที่ถูกสร้างขึ้นโดยใช้ภาษา COBOL โดยที่อีกส่วนหนึ่งกลับถูกสร้างขึ้นด้วยภาษา C และ C++ ด้วยความหลากหลายของภาษาที่ใช้ดังกล่าวทำให้การเชื่อมต่อของโปรแกรมเหล่านั้นจำเป็นจะต้องมีตัวกลางที่สมบูรณ์พอ แต่สำหรับ COM และ DCOM แล้วเบื้องหลังยังคงเป็น C++ ซึ่งทำให้เกิดปัญหาได้เช่น DCOM จะให้โปรแกรมต่างๆ ติดต่อกันในลักษณะของพอยน์เตอร์ ในขณะที่ COBOL และ Java นั้นไม่รองรับในเรื่องของพอยน์เตอร์ ตรงกันข้าม, ภาษาของ CORBA กลับเป็นภาษาที่มีความเป็นกลางสามารถรองรับการทำงานกับภาษาต่างๆ ได้มากมายอันเนื่องมาจากการร่วมมือกันของสมาชิกในกลุ่มของ OMG ในการกำหนดรูปแบบของการแปลงภาษา (Language Mapping) นอกจากนี้สมาชิกบางรายยังได้รวมวิธีการแปลงภาษาสำหรับภาษาอื่นๆ อีกมากมายเช่น Visual Basic หรือ FORTRAN

4. การเจริญเติบโตระหว่าง Active X กับ CORBA (Maturity and deployment)

สำหรับ Active X นั้นพึ่งจะมีการกำหนดรูปแบบขึ้นเมื่อไม่นานมานี้ รวมทั้งเอกสารต่างๆ ที่เกี่ยวกับ Active X ก็ยังมีลักษณะที่ไม่สมบูรณ์ชัดเจนนัก นอกจากนี้การโฆษณารวมถึงการประกาศ Active X ก็ยังกระทำกันในช่วงระยะเวลาอันสั้น ซึ่งในจุดนี้จะสังเกตุเห็นได้ว่า CORBA นั้นมีความแตกต่างอย่างสิ้นเชิง เนื่องจาก CORBA มีการประกาศตัวในครั้งแรกก่อนปี ค.ศ.1991 และเริ่มรวมกลุ่มกันกำหนดมาตรฐานประมาณเดือนมีนาคมปี ค.ศ.1991 ต่อมาในเดือนธันวาคมปี ค.ศ.1994 ซึ่งเป็นระยะเวลาถึง 3 ปีจึงได้มีการประกาศ CORBA 2.0 ออกมาพร้อมกับ ORB นอกจากนั้น CORBA ยังได้มีการทดสอบและแก้ไขข้อบกพร่องทั้งจากผู้ผลิตและลูกค้ารอบโลกโดยตลอดระยะเวลานานถึง 7 ปี จึงแสดงให้เห็นว่า CORBA ได้มีการพัฒนามาอย่างเป็นลำดับขั้นตอนและเป็นที่ยอมรับกันโดยทั่วไป

5. การรองรับการทำงานบน World Wide Web (Support on World Wide Web)

สำหรับ Active X แล้วตัวโปรแกรมที่เป็น Web browser จะต้องสามารถรองรับการทำงานแบบ Active X ได้เท่านั้น ซึ่งในปัจจุบันก็ยังจำกัดอยู่เฉพาะโปรแกรม Web browser ที่ทำงานบน Windows '95 หรือ Windows NT 4.0 เท่านั้นเช่น Internet Explorer แต่ในขณะที่ CORBA มีลักษณะการทำงานที่เรียกว่า ORBlet ซึ่งมีลักษณะการทำงานที่คล้ายคลึงกับ Java applet จึงสามารถทำงานได้บนหลากหลายแพลตฟอร์ม (Platform) มากกว่า

6. การรักษาความปลอดภัยในระบบ (Security)

CORBA มีการให้บริการซึ่งเกี่ยวข้องกับด้านของความปลอดภัยซึ่งมุ่งเน้นไปทางด้านของระบบธนาคารหรือการเงิน ดังนั้นจึงมีความมั่นใจได้ค่อนข้างมากว่าระบบความปลอดภัยของ CORBA จะมีประสิทธิภาพ ซึ่งในส่วนนี้สำหรับ Active X แล้วยังคงมีช่องว่างอยู่ค่อนข้างมาก เนื่องจาก Active X มีวิธีการทำงานใน 2 ลักษณะคือใช้การก๊อปปี้ตัว Active X Control ไปยังเครื่องคอมพิวเตอร์กับวิธีที่ 2 คือการใช้ RPC (Remote Procedure Call) ซึ่งยังมีช่องทางให้บุคคลอื่นเจาะระบบเข้าไปได้

7. ขอบเขตความสามารถ (Scalability)

ในเรื่องนี้จะเห็นได้ว่า CORBA เองนั้นได้ถูกออกแบบมาเพื่อให้ใช้งานในระดับของโปรแกรมเครือข่ายขนาดใหญ่อยู่แล้ว ในขณะที่ DCOM/Active X ไม่เคยมีแนวคิดในเรื่องนี้เลย

จากข้อเปรียบเทียบดังกล่าวจะเห็นได้ชัดเจนว่า Active X ในปัจจุบันยังคงมีข้อบกพร่องที่จะต้องได้รับการปรับปรุงแก้ไขจากไมโครซอฟท์อีกมาก ในขณะที่ CORBA นั้นมีความสมบูรณ์มากตั้งแต่ในเรื่องของสถาปัตยกรรม, นิยาม รวมจนถึงเรื่องของการประยุกต์ใช้งานแล้ว ซึ่งในส่วนของ CORBA เองคงยังเหลือเฉพาะการพัฒนาประสิทธิภาพ ให้กว้างขวางออกไปอีกเท่านั้น

แปลจากเอกสาร Active X - CORBA .html จาก Web site ของ http://www.omg.org/

ศึกชิงนักพัฒนาโปรแกรมประยุกต์

ข่าวหนาหูเกี่ยวกับเทคโนโลยี Com3 ที่ไมโครซอฟท์กำลังจะออกช่วยเน้นให้เห็นช่องว่างในรูปแบบปัจจุบันของโปรแกรมประยุกต์ที่เป็นแบบออบเจ็กต์ของบริษัทให้เด่นชัดขึ้น

กล่าวโดยทั่วไปข่าวนี้เป็นข่าวดีสำหรับผู้มีหน้าที่รวมแพลตฟอร์มต่างๆ เข้าด้วยกัน นักพัฒนาโปรแกรมประยุกต์และผู้บริหารระบบเครือข่ายคอมพิวเตอร์ไม่จำเป็นต้องเลือกระหว่างความสะดวกที่ได้จาก DCOM (Distributed Component Object Model) ของไมโครซอฟท์กับสมรรถนะที่ได้จาก CORBA ของผู้ค้าหลายรายอีกต่อไปเพราะทางเลือกที่ให้ทำงานร่วมกันได้จะช่วยให้โปรแกรมประยุกต์ได้ประโยชน์จากวิธีการทั้งสองแบบในการตอบโต้ออบเจ็กต์บนเครือข่าย แต่ข่าวร้ายก็คือต้องมีความเข้าใจทั้ง CORBA และ DCOM เป็นอย่างดี ปัจจุบันบริษัทต่างๆ ต้องการทั้งสองวิธีเพื่อบรรลุความต้องการของผู้ใช้ในระดับโปรแกรมประยุกต์บนพีซีแบบเดิมๆ ทั้งในแง่สมรรถนะและการโต้ตอบได้ ขณะเดียวกันก็ให้นักพัฒนามีความยืดหยุ่นในซอฟต์แวร์ที่ใช้ส่วนประกอบและมีความรวดเร็วซึ่งจำเป็นในสภาพแวดล้อมของโปรแกรมประยุกต์แบบกระจายวัตถุ (Distributed Object)

DCOM หรือ COM ยอมให้โปรแกรมประยุกต์บนเดสก์ท็อปทำงานกับส่วนประกอบระยะไกลบนการเชื่อมต่อเครือข่ายชนิดต่างๆ โดยใช้ COM API ซึ่งนักพัฒนาได้ใช้ในโปรแกรมประยุกต์ของ Windows ที่แบ่งเป็นส่วนประกอบอยู่แล้ว COM API เป็นเทคโนโลยีที่พัฒนามาจาก OLE ของไมโครซอฟท์และสามารถใช้ได้กับเทคโนโลยี Dynamic Data Exchange ซึ่งทำให้นักพัฒนามีมาตรฐานแบบเปิดสำหรับการโต้ตอบของโปรแกรมประยุกต์ที่โปรแกรมได้ภายใต้ Windows อย่างไรก็ตาม ชื่อเสียงของ COM ก็มีทั้งในแง่บวกและแง่ลบ เราสามารถทำงานพื้นๆ ในสภาพแวดล้อมท้องถิ่นที่มี COM ได้โดยง่ายและย้ายไปสู่การโต้ตอบแบบกระจายบนเครือข่ายด้วย DCOM ได้โดยง่ายเช่นเดียวกัน แต่ก็ไม่ได้แปลว่า DCOM เป็นวิธีที่เหมาะสมเสมอไปเมื่อเทียบกับ CORBA ที่ซับซ้อนกว่า มีผู้เปรียบไว้ว่าการปีนต้นไม้ย่อมง่ายกว่าการสร้างและปล่อยจรวด แต่เราก็ไม่อาจไปถึงดวงจันทร์ได้โดยการมองหาต้นไม้สูงๆ เพื่อที่จะปีน

จุดอ่อนของ DCOM เกิดจากความพยายามที่จะทำให้เครือข่ายปรากฏ (ต่อโปรแกรมประยุกต์ใด) เหมือนเป็นสภาพแวดล้อมท้องถิ่น DCOM สร้างขึ้นสำหรับพีซีรูปแบบเดิมที่มีผู้ใช้คนเดียวเป็นผู้ควบคุมจากศูนย์กลางและหน่วยความจำท้องถิ่นเป็นทรัพยากรส่วนกลางที่ใช้ร่วมกันได้ สมมุติฐานนี้ทำให้ต้องการการสนับสนุนจากซอฟต์แวร์หลายชั้น DCOM สร้างและทำลายออบเจ็กต์แบบยืดหยุ่น ออบเจ็กต์ทุกตัวเป็นชุดของตัวที่ชี้ไปยังตัวติดต่อซึ่งออบเจ็กต์หนึ่งเสนอแก่ออบเจ็กต์อื่น ถ้าไม่มีสิ่งใดต้องการใช้ออบเจ็กต์นั้นออบเจ็กต์ก็จะหายไป และตราบใดที่ยังมีสิ่งที่ต้องการใช้งานออบเจ็กต์นั้นออบเจ็กต์นั้นก็จะต้องคงอยู่ CORBA ใช้วิธีการที่แตกต่างออกไปออบเจ็กต์จะยังมีชื่ออยู่แม้ว่าออบเจ็กต์นั้นจะไม่อยู่ในหน่วยความจำแล้วก็ตาม หน้าที่หลักของ ORB คือให้ออบเจ็กต์ใช้บริการของออบเจ็กต์อื่นได้ แต่ไม่ให้รู้ตำแหน่งและสถานะของออบเจ็กต์ที่เรียกใช้ ถ้ามีออบเจ็กต์ของ CORBA ตัวหนึ่งมองหาบริการจากออบเจ็กต์อื่น ORB ก็ต้องพิจารณาว่าออบเจ็กต์อื่นอยู่ที่ไหนและสั่งงานออบเจ็กต์นั้นถ้าจำเป็นเพื่อให้มองเห็นออบเจ็กต์ที่เรียกไป

กรณีเซริฟเวอร์ของ DCOM มีไคลเอนท์ที่กำลังทำงานอยู่เซริฟเวอร์ต้องเก็บไคลเอนท์ไว้ในหน่วยความจำตลอดเวลา แม้ว่าตอนนี้ไม่จำเป็นต้องสนใจแล้วส่วนกรณี CORBA เซริฟเวอร์สามารถลบตัวเองออกจากหน่วยความจำเพื่อให้ใช้งานอื่น เนื่องจากไคลเอนท์ทุกตัวจะสามารถเข้าถึงเซริฟเวอร์ที่ต้องการได้ด้วย ORB ข้อบกพร่องด้านสถาปัตยกรรมบางอย่างใน DCOM อาจได้รับการแก้ไขโดยแผนของไมโครซอฟท์สำหรับ COM รุ่นที่สามเชื่อว่าแผนเหล่านี้จะรวมอยู่ในสภาพแวดล้อมแบบรันไทม์ซึ่งเพิ่มความเชื่อถือได้ด้วยวิธีจัดการกับกรณีผิดพลาดแบบ Java นับเป็นสิ่งดีที่ไมโครซอฟท์ชิงลงมือเสียก่อนเพราะถ้าไมโครซอฟท์ไม่ทำก็จะมีบริษัทอื่นมาอุดช่องว่างนี้แทน

	CORBA	DCOM
พื้นฐาน	ออบเจ็กต์คงอยู่บนเครือข่ายแบบผสม	ออบเจ็กต์ไม่อยู่บนเดสก์ท็อป Windows ตลอด
การใช้ทรัพยากร	ออบเจ็กต์ไคลเอนท์, เซริฟเวอร์ ถอนจากหน่วยความจำแล้วโหลดอีกครั้งเมื่อต้องการใช้	ออบเจ็กต์ยังอยู่ในหน่วยความจำตราบเท่าที่ยังมีการใช้
โมเดลส่วนประกอบซอฟต์แวร์	JavaBeans: - ปลอดภัย - คำสั่งไบต์: ช้าแต่ไม่ขึ้นกับแพลตฟอร์ม	ส่วนประกอบ ActiveX: - ต้องการคำสั่งที่เชื่อถือได้ - คำสั่งพื้นฐาน: เร็วแต่ใช้เฉพาะแพลตฟอร์ม
การเข้ากันได้	ผู้ค้าหลายรายต้องมาตกลงกัน	กำหนดโดยไมโครซอฟท์

รูป 1-3 ข้อแตกต่างระหว่าง CORBA กับ DCOM

บทที่ 2 ทฤษฎีของ CORBA (CORBA Theory)

มาตรฐาน CORBA เริ่มต้นตั้งแต่ปี ค.ศ. 1990 ในรุ่น CORBA 1.0 และในปี ค.ศ. 1994 ก็พัฒนามาเป็นรุ่น CORBA 2.0 โดยการร่วมมือกันของสมาชิกกลุ่ม OMG ด้วยความต้องการที่จะให้มีมาตรฐานที่เป็นส่วนกลางสำหรับโปรแกรมภาษาทุกภาษา ในการที่จะสร้างโปรแกรมประยุกต์ให้สามารถทำงานร่วมกันได้ สำหรับในปัจจุบันมีเครื่องมือมากมายที่อ้างอิงอยู่กับมาตรฐานของ CORBA 2.0 เช่น Visibroker ของบริษัท Visigenic, Orbix Web ของ Iona หรือ PowerBroker ของ Expert Soft เป็นต้น โดยถ้าเปรียบเทียบกับเครื่องมืออื่นๆ ที่ไม่ได้อ้างอิงกับมาตรฐาน CORBA 2.0 แล้วจะเห็นได้ว่า CORBA 2.0 เป็นมาตรฐานที่เอื้ออำนวยและรองรับการทำงานในลักษณะต่างๆ ดังนี้

* **Static & Dynamic Invocation :** สนับสนุนการทำงานทั้งในลักษณะของ Static เพื่อที่จะกำหนดถึงวิธีการเรียกใช้ method ในช่วงทำการคอมไพล์หรือเป็นลักษณะของ Dynamic ก็ได้ซึ่งจะทำให้ ORB สามารถจะรับรู้ถึงวิธีกรเรียกใช้ method ในขณะทำงานได้ ซึ่งโดยทั่วไปแล้วมาตรฐานต่างๆ ที่ไม่ใช่มาตรฐานของ CORBA 2.0 นี้ก็มักจะสนับสนุนเพียงรูปแบบของ Static Invocation เท่านั้น
* **High-level Language Binding :** เป็นการสนับสนุนผู้ใช้ในการใช้ภาษาระดับสูงเช่น C++, Java ในการเขียนโปรแกรมที่เป็นโปรแกรมประยุกต์ได้ โดยไม่จำกัดว่าออบเจ็กต์ที่เรียกใช้จะถูกสร้างขึ้นโดยโปรแกรมภาษาอะไรหรือว่าเซริฟเวอร์จะใช้ระบบปฏิบัติการใด ๆ ก็ตาม
* **Self Describing System :** CORBA จะทำการสร้างข้อมูลขึ้นมาเกี่ยวกับเซริฟเวอร์ที่ติดต่อด้วย สำหรับ CORBA ORB ทุกตัวจะต้องรองรับการทำงานกับ Interface Repository ซึ่งภายในจะบรรจุข้อมูลซึ่งอธิบายถึง method ต่างๆ ที่เซริฟเวอร์ตัวหนึ่งๆ มี รวมทั้งส่วนที่เป็นพารามิเตอร์ของมันด้วย ไคลเอนท์จะใช้ข้อมูลเหล่านี้เพื่อที่จะหาวิธีในการเรียกใช้บริการต่างๆ ข้อมูลเหล่านี้จะถูกสร้างโดยอัตโนมัติทั้งจากการคอมไพล์ไฟล์ .idl หรือจากคอมไพเลอร์ที่สร้าง IDL จากภาษาต่างๆ ก็ได้
* **Local/Remote transparrency :** เป็นคุณสมบัติที่ทำให้สามารถใช้งานได้ทั้งแบบที่เป็นเครือข่าย Client/Server บนเครื่องเดียวกันหรือจะต่อเชื่อมไปยัง ORB ของเครือข่าย Client/Server อื่น ๆ ก็ได้ โดยผ่าน IIOP ของ CORBA 2.0 โดยที่ไม่จำเป็นต้องรู้รายละเอียดของเซริฟเวอร์เลย
* **Built in Security and Transaction :** มีการทำ Security สำหรับการติดต่อกันระหว่าง ORB กับ ORB ไว้แล้ว
* **Polymorphic messaging :** ORB จะแตกต่างกับตัวกลางประเภทอื่นๆ ในจุดที่ ORB จะไม่ได้เรียกใช้ฟังก์ชันที่มีอยู่ในระบบโดยตรง แต่เสมือนว่า ORB ได้ทำการเรียกใช้ฟังก์ชันที่อยู่บนตัวออบเจ็กต์ ซึ่งหมายถึงฟังก์ชันตัวเดียวกันอาจมีผลตอบสนองที่แตกต่างกันได้ถ้าออบเจ็กต์นั้นมีความแตกต่างกันอยู่
* **Coexistence with existing systems :** เนื่องจาก CORBA จะมีการแยกออบเจ็กต์กับส่วนของการเรียกใช้งานไว้ต่างหากซึ่งทำให้สามารถสร้างโปรแกรมประยุกต์ใหม่ได้โดยเรียกใช้บริการออบเจ็กต์เก่าได้ทันที

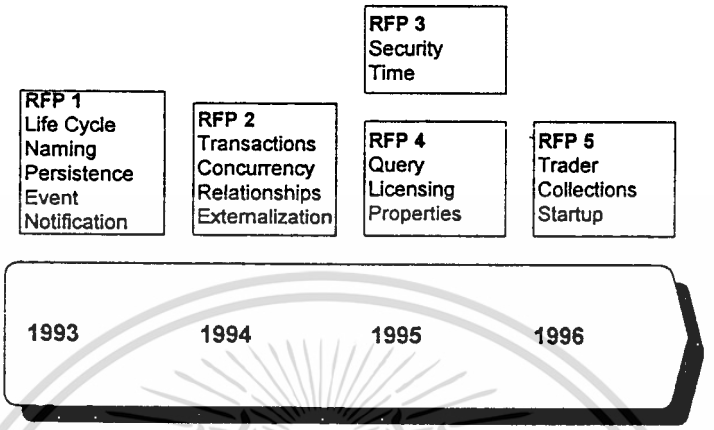
จากที่กล่าวมาจะเห็นว่าข้อดีของ CORBA 2.0 นั้นทำให้ CORBA 2.0 มีความโดดเด่นเหนือมาตรฐาน Client/Server อื่น ๆ มาก ซึ่งในทางทฤษฎีแล้วอาจกล่าวได้ว่า CORBA เป็นตัวกลางการติดต่อที่ดีที่สุดเท่าที่เคยมีมาเลยทีเดียว แต่สำหรับในทางปฏิบัติแล้วถึงจะไม่ดีที่สุดแต่ก็นับได้ว่าเป็นตัวกลางการติดต่อที่ดีมากพอสมควร

COS (Common Object Service)

นอกจากข้อดีดังกล่าวมาแล้ว CORBA 2.0 ยังมีการให้บริการเพื่อช่วยอำนวยความสะดวกให้กับผู้ใช้อีกด้วย ซึ่งเป็นหน้าที่การให้บริการโดยระบบที่เรียกว่า Object Service หรือ Common Object Service หรืออาจเรียกว่า COS ก็ได้ โดยที่ ORB จะคอยให้บริการในส่วนของ COS ให้กับผู้ใช้ที่เรียกใช้ออบเจ็กต์ ซึ่ง COS จะมีทั้งหมด 16 บริการคือ

1. **Life Cycle Service** ให้บริการเกี่ยวกับการสร้าง ทำสำเนา ย้ายหรือลบออบเจ็กต์ต่างๆ
2. **Persistence Service** ดูแลควบคุมเกี่ยวกับการติดต่อกับออบเจ็กต์ที่อยู่ในหลายเซริฟเวอร์
3. **Naming Service** อนุญาตให้ส่วนประกอบหนึ่งอ้างอิงถึงอีกส่วนประกอบหนึ่งได้โดยใช้ชื่อของส่วนประกอบนั้น และรวมถึงการผนวกไดเร็กทอรีต่างๆ ที่อยู่ในเครือข่ายด้วย
4. **Event Service** เก็บรวบรวมข้อมูลต่างๆ ซึ่งเกี่ยวข้องกับเหตุการณ์ต่างๆ ที่เกิดขึ้น โดยในส่วนนี้ส่วนประกอบต่างๆ สามารถที่จะนำข้อมูลในส่วนนี้ไปใช้ได้
5. **Concurrency control Service** ควบคุมให้เกิดการทำงานที่ถูกต้องของแต่ละ Transaction
6. **Transaction Service** ดูแลไม่ให้เกิดปัญหาต่าง ๆ อันเนื่องมาจากการทำงานแบบ Multi Transaction
7. **Relationship Service** ดูแลในเรื่องของการเชื่อมต่อกันระหว่างส่วนประกอบต่าง ๆ ที่เกี่ยวข้อง
8. **Externalization Service** ควบคุมการรับของมูลเข้าและส่งข้อมูลออก
9. **Query Service** ดูแลการใช้งานในลักษณะของ Query กับ Objects ต่าง ๆ ซึ่งจะใช้ลักษณะของ SQL3 หรือ OQL(Object Query Language)
10. **License Service** ดูแลควบคุมจำนวนการใช้งานของ User ในแต่ละส่วนย่อย
11. **Properties Service** ดูแลการระบุสถานะของส่วนประกอบนั้น ๆ ให้ส่วนประกอบอื่น ๆ ได้รู้เช่น วันที่ของไฟล์ที่เป็นส่วนประกอบ
12. **Time Service** ควบคุมในเรื่องของการทำงานโดยอ้างอิงกับเวลา
13. **Security Service** ควบคุมในเรื่องของการตรวจสอบสิทธิในการเรียกใช้บริการ
14. **Trader Service** อนุญาตให้ผู้อื่นสามารถเข้ามาใช้และสั่งงานออบเจ็กต์ได้
15. **Collection Service** ดูแลการสร้างและจัดการส่วนประกอบร่วมทั่ว ๆ ไป
16. **Startup Service** อนุญาตให้ทำการร้องขอโดยอัตโนมัติเมื่อ ORB ถูกเรียกใช้

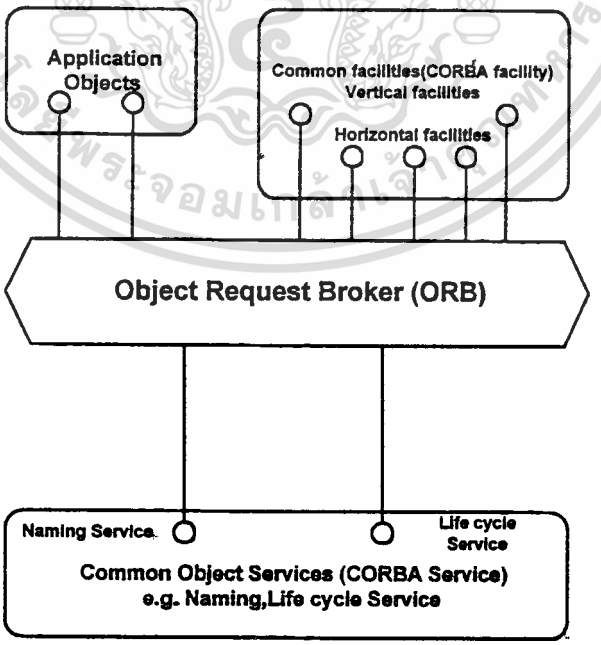
สำหรับทั้ง 16 บริการของ COS ที่กล่าวมาถือได้ว่ามีส่วนช่วยให้ระบบของ Distributed Object มีเสถียรภาพดียิ่งขึ้น ซึ่ง COS นี้มิได้มีการกำหนดในคราวเดียว แต่ได้มีการกำหนดเพิ่มเติมมาเรื่อย ๆ จากรูปแสดงให้เห็นถึงการกำหนด Common Object Service ของ OMG ตั้งแต่ปี ค.ศ.1993 จนถึงปัจจุบัน



รูป 2-1 พัฒนาการของ Common Object Service

OMA (Object Management Architecture)

ดังที่ได้กล่าวมาแล้วว่า CORBA คือมาตรฐานในการติดต่อสื่อสารกันในเครือข่าย Client/Server ซึ่งมีลักษณะของโครงสร้างที่เรียกว่า OMA(Object Management Architecture) ซึ่ง OMA นี้ถือได้ว่าเป็นสถาปัตยกรรมที่แสดงให้เห็นถึงวิธีการติดต่อสื่อสารในรูปแบบของมาตรฐาน CORBA ด้วยดังรูป



รูป 2-2 Object Management Architecture (OMA)

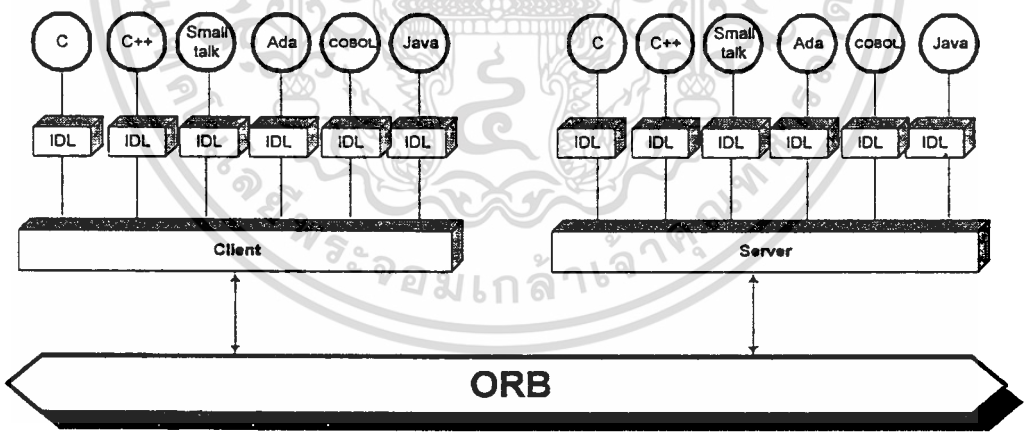
จากรูปจะเห็นได้ว่าองค์ประกอบของ OMA จะสามารถแบ่งออกได้เป็น 4 ส่วนดังนี้

1. Application Objects หรือโปรแกรมที่ผู้ใช้งานกำลังเรียกใช้อยู่นั่นเอง
2. Common facilities เป็นองค์ประกอบที่เอื้ออำนวยความสะดวกต่างๆ ในการเรียกใช้งานออบเจ็กต์
3. Object Request Broker เป็นตัวกลางที่ช่วยในการเรียกใช้ออบเจ็กต์และดูแลให้การติดต่อสื่อสารระหว่างไคลเอนท์กับเซริฟเวอร์เป็นไปด้วยความมีเสถียรภาพ
4. Common Object Service เป็นส่วนของบริการต่าง ๆ ที่ ORB จะสามารถเรียกใช้เพื่ออำนวยความสะดวกในการเรียกใช้ออบเจ็กต์

องค์ประกอบทั้ง 4 ที่กล่าวมานั้นล้วนมีความสัมพันธ์กัน โดยผู้ใช้งานนั้นจะใช้โปรแกรมประยุกต์ของตนเรียกใช้ออบเจ็กต์ที่อยู่ในส่วนของเซริฟเวอร์โดยผ่านตัวกลางการจัดการที่ชื่อ ORB และในส่วนของ ORB เองนอกจากจะดูแลและจัดการในเรื่องของการเรียกใช้บริการของออบเจ็กต์แล้วก็ยังสามารถที่จะให้บริการในการควบคุมดูแลออบเจ็กต์ให้มีเสถียรภาพด้วย

IDL (Interface Definition Language)

IDL เป็นภาษาที่เกิดจากกลุ่มของ OMG โดยบางครั้งอาจเรียกว่า OMG IDL มีจุดประสงค์ที่จะสร้างโปรแกรมประยุกต์ที่สามารถเชื่อมต่อกันได้ระหว่างโปรแกรมภาษาที่แตกต่างกัน โดยตัว IDL เองจะมีลักษณะที่เป็นมาตรฐานซึ่งสามารถที่จะเชื่อมต่อโปรแกรมประยุกต์ที่สร้างขึ้นจากภาษาต่างๆ เข้าด้วยกันได้มีลักษณะดังรูป



รูป 2-3 ลักษณะการทำงานของ OMG IDL

IDL เป็นภาษาที่อยู่ในระดับของการต่อเชื่อมการติดต่อหรือที่เรียกว่า Middleware เพื่อที่จะกำหนดขอบเขตของการติดต่อรวมทั้งรูปแบบของ method ที่ใช้ในการทำงานของโปรแกรมทางไคลเอนท์ โดยที่ตัวภาษา IDL นี้จะมีรูปแบบการเขียนที่เป็นแบบฉบับของตัวเองแต่จะมีรูปแบบของภาษาที่มีความคล้ายคลึงกับภาษา C++ ด้วยจึงทำให้ดูเหมือนไม่มีอะไรใหม่ ปกติแล้วในการสร้างโปรแกรมประยุกต์ตามมาตรฐาน CORBA นั้นจำเป็นจะต้องมีการสร้างไฟล์ภาษา IDL ก่อนเสมอ เนื่องจาก

องค์ประกอบของ CORBA นั้นจะประกอบด้วยไฟล์ที่เกิดจากไฟล์ภาษา IDL ทั้งสิ้น โดยการแปลงนั้นเรียกว่าเป็นการทำ IDL Mapping ซึ่งขึ้นอยู่กับเครื่องมือที่เลือกใช้ว่าจะทำการแปลงออกมาเป็นภาษาอะไร เช่น Visibroker for Java ของ Visigenic ก็จะมีตัวคอมไพเลอร์(Compiler) ซึ่งจะถูกเรียกว่า Pre-Compiler ทำการแปลงจากไฟล์ภาษา IDL ไปเป็นไฟล์ภาษา Java เป็นต้น และไฟล์ภาษา IDL ปกติแล้วก็มักจะมีนามสกุลของไฟล์เป็น .idl เช่น Bank.idl เป็นต้น องค์ประกอบของภาษา IDL ที่สำคัญจะมีอยู่หลายส่วนด้วยกัน ดังนี้

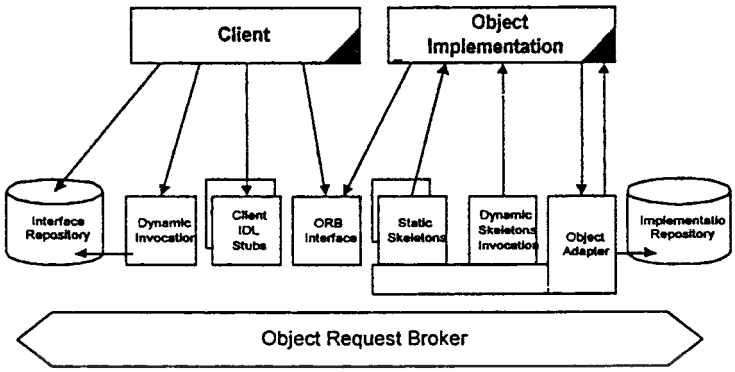
```
module <identifier>  ——————————  Define a naming context

{
 <type declarations>;
 <constant declarations>;
 <exception declarations>;

   interface <identifier>  [:<inheritance>]——————————  Define a CORBA class
    {
        <type declarations>;
        <constant declarations>;
        <attribute declarations>;
        <exception declarations>;

        [<op_type>] <identifier> (<parameters>)——————————  Define a method
        [raises exception] [context];
                .
                .
        [<op_type>] <identifier> (<parameters>)  ——————————  Define a method
        [raises exception] [context];
                .
                .
    }
   interface <identifier> [:<inheritance>]  ——————————  Define a CORBA Class
                .
                .
}
```

1. Module เป็นส่วนที่ทำการกำหนดพื้นที่ในการรวบรวมไฟล์ต่างๆ ที่เกิดจากการคอมไฟล์ไฟล์ .idl หรือกล่าวได้ว่า Module ก็คือส่วนที่กำหนดให้มีการสร้างไดเร็คทอรีที่ใช้ในการเก็บรักษาไฟล์ที่ได้จากการคอมไพล์ไฟล์ .idl ให้เป็นหมวดหมู่นั่นเอง
2. Interface เป็นตัวกำหนดตัวไฟล์ที่ต้องการติดต่อด้วย
3. Operation เป็นเสมือน method ที่โปรแกรมทางไคลเอนท์สามารถเรียกใช้ได้นั่นเอง นอกจากนั้นตัว method ยังสามารถที่จะมีอาร์กิวเมนต์ได้ด้วย ซึ่งอาร์กิวเมนต์ของ method ในไฟล์ .idl สามารถทำการกำหนดสถานะได้คือ
   - in เป็นการกำหนดให้มีการส่งอาร์กิวเมนต์จากไคลเอนท์ไปยังเซริฟเวอร์
   - out เป็นการกำหนดให้มีการส่งอาร์กิวเมนต์จากเซริฟเวอร์ไปยังไคลเอนท์
   - inout เป็นการกำหนดให้มีการส่งอาร์กิวเมนต์ระหว่างเซริฟเวอร์กับไคลเอนท์ได้
4. Data type ครอบคลุมถึงการกำหนดค่าที่รับมาได้จากตัวอาร์กิวเมนต์ กำหนดแอทริบิวท์การส่งค่ากลับและการดูแลในเรื่องของ Exception ในส่วนของ Data type นั้นมีลักษณะพื้นฐานที่เกือบเหมือน C++ คือ short, long etc. ส่วน ORB นั้นสามารถที่จะรองรับการทำงานกับ Data type ได้ทั้งหมด 2 แบบด้วยกันคือรูปแบบพื้นฐาน (basic) และแบบ Constructed

ภาษา IDL เป็นภาษาที่ทำให้การเขียนโปรแกรมมีความกระทัดรัดและสั้นลงมาก โดย IDL จะช่วยให้สามารถกำหนดขอบเขตของงานและ method ต่างๆ ที่ไคลเอนท์ต้องการเรียกใช้ได้ IDL เป็นภาษาที่เป็นตัวกลางในการสร้างไฟล์และกำหนด method ที่ต้องใช้ แต่นั่นก็หมายถึง IDL ไม่สามารถช่วยในการกำหนดรายละเอียดของงานได้

ORB (Object Request Broker)

หน้าที่หลักของ ORB ก็คือเป็นตัวกลางในการจัดการทั้งในเรื่องของการเรียกใช้บริการจากผู้ใช้ไปยังเซริฟเวอร์และนำผลที่ได้มาให้กับไคลเอนท์ นอกจากนั้นแล้วตัว ORB ก็ยังคอยจัดการในการเรียกใช้บริการต่าง ๆ ด้วย หรืออาจกล่าวได้ว่าทุกสิ่งทุกอย่างที่เกิดขึ้นบนเครือข่ายนั้นจะต้องผ่าน ORB เสมอ และเนื่องจาก ORB มีบทบาทสำคัญในการจัดการการติดต่อสื่อสารจึงทำให้ ORB มีองค์ประกอบที่มากมายดังแสดงได้ในรูปต่อไปนี้



รูป 2-4 โครงสร้างการทำงานของ CORBA 2.0 ORB

* **Client IDL Stubs** สนับสนุนการติดต่อแบบ Static เพื่อกำหนดรูปแบบที่ไคลเอนท์จะเรียกใช้งานบริการจากออบเจ็กต์ผ่าน ORB การบริการนี้จะถูกกำหนดและอ้างถึงโดย IDL ซึ่งทั้งไคลเอนท์และเซริฟเวอร์ ในส่วนของ Stub นี้จะถูกสร้างจาก IDL Compiler
* **Dynamic Invocation APIs** เป็นวิธีการที่จะเรียกใช้งานบริการขณะที่กำลังสั่งให้ตัวโปรแกรมทำงานอยู่ โดยที่ CORBA กำหนดมาตรฐาน APIs สำหรับการกำหนดเพื่อระบุถึงงานบริการ การสร้างพารามิเตอร์ การสั่งงานระยะไกล (Remote call) และการรับผลที่ต้องการ
* **Interface Repository APIs** เป็นส่วนที่เก็บรูปแบบของ Class ทั้งหมดที่ได้เก็บข้อมูลไว้ให้ ORB รู้จักวิธีที่ Class เหล่านั้นสนับสนุนและพารามิเตอร์ที่การทำงานนั้นต้องการ ทั้งหมดนี้ CORBA จะเรียกว่า Method Signatures โดยเก็บรวบรวมไว้ใน Interface Repository ซึ่งเป็นฐานข้อมูลให้กับโปรแกรมค้นหาในขณะที่กำลังทำงาน ORB Interface ประกอบด้วย API บางประเภทที่โปรแกรมประยุกต์ต้องการงานบริการแบบภายใน (Local) การเรียกใช้นี้ไม่ได้เป็นส่วนหนึ่งของกลไกการเรียกใช้ของระบบ Client/Server
* **Server IDL Stubs (Skeletons)** ทำการจัดหาการติดต่อแบบ Static สำหรับงานบริการแต่ละอย่าง ในส่วนนี้คล้ายกับ IDL Stubs ทางฝ่ายของไคลเอนท์มันจะถูกสร้างโดย IDL Compiler ฝั่งเซริฟเวอร์ไม่สามารถแยกความแตกต่างระหว่างการร้องขอที่เป็นแบบ Static หรือ Dynamic ได้ มันได้รับการร้องขอการบริการทั้ง 2 แบบผ่านส่วนเซริฟเวอร์IDL Stubs เพียงทางเดียวเท่านั้น
* **Object Adapter** เป็นตัวติดต่อสื่อสารระหว่าง ORB กับออบเจ็กต์ของเซริฟเวอร์ช่วยในการกำหนด ID ของออบเจ็กต์ที่ต้องการ CORBA เรียกใช้ออบเจ็กต์โดยใช้ออบเจ็กต์อ้างอิง(Object Reference) Object Adapter จะทำการลงบันทึก Class ทั้งหมดที่มันสนับสนุนและออบเจ็กต์ที่กำลังทำงานอยู่ทั้งหมดไว้ในที่เก็บข้อมูลที่เรียกว่า Implementation Repository CORBA จะกำหนด Adapter พื้นฐานที่ ORB ทุกตัวต้องสนับสนุนเป็นมาตรฐานเรียกว่า Basic Object Adapter ซึ่งเซริฟเวอร์อาจจะสนับสนุนมากกว่าหนึ่ง Adapter ก็ได้

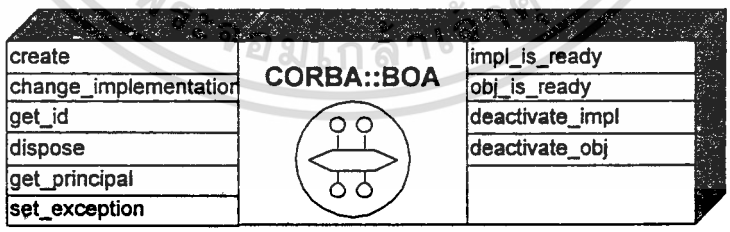
* **Implementation Repository** เป็นแหล่งข้อมูลเกี่ยวกับ Class และออบเจ็กต์ที่เซริฟเวอร์สนับสนุนและกำลังทำงานอยู่ในขณะนั้น เป็นที่เก็บข้อมูลเพิ่มเติมที่เกี่ยวกับการประยุกต์ใช้ เช่น Trace information, Audit trails, Security etc.
* **Dynamic Skeleton Invocation** ทำหน้าที่ในการตรวจสอบตัวพารามิเตอร์ของการเรียกใช้บริการและทำการระบุให้เห็นถึงออบเจ็กต์ที่ต้องการเรียกใช้บริการ ซึ่งมักจะถูกใช้ในกรณีของการติดต่อสื่อสารระหว่าง ORB กับ ORB
* **ORB Interface** ประกอบด้วย API ขนาดเล็กซึ่งให้บริการในลักษณะแบบภายใน (Local) การให้บริการนั้นจะเป็นบริการในเรื่องที่ไม่ยุ่งยากและไม่ใหญ่โตมากนัก เช่น แปลงออบเจ็กต์อ้างอิงให้เป็นข้อความ สิ่งสำคัญก็คือ ORB Interface นี้จะมีอยู่ทั้งทางฝั่งของไคลเอนท์และเซริฟเวอร์

IIOP (Internet Inter-ORB Protocol)

IIOP เป็น Protocol ที่ทำงานเกี่ยวข้องกับคำสั่งของ CORBA โดยจุดประสงค์เพื่อที่จะให้เครือข่ายของ CORBA สามารถขยายกว้างขวางออกไปได้ ตัว IIOP นั้นจะทำงานเพื่อดูแลการติดต่อระหว่าง ORB กับ ORB ซึ่งอาจจะอยู่ห่างกันออกไป โดยที่ IIOP นั้นจะทำงานอยู่บน TCP/IP อีกทีหนึ่ง IIOP เป็น Protocol ที่พัฒนามาจาก GIOP หรือ General Inter-ORB Protocol ซึ่งเป็น Protocol ที่ใช้ในการจัดการทั่วไปของ CORBA นั่นเอง

BOA (Basic Object Adapter)

BOA ถือได้ว่าเป็นออบเจ็กต์ตัวหนึ่งที่ถูกสร้างขึ้นโดย ORB เพื่อที่จะรองรับการทำงานของเซริฟเวอร์ให้สามารถให้บริการตัวออบเจ็กต์ต่างๆ ได้ โดยในส่วนของ BOA จะมี method ให้เรียกใช้งานได้มากมายซึ่งรองรับทั้งการสร้างออบเจ็กต์และการทำลายออบเจ็กต์นั้นดังรูป



รูป 2-5 method ต่างๆ ที่ออบเจ็กต์ BOA สามารถเรียกใช้ได้

— **create ใช้เพื่อสร้างข้อมูลของออบเจ็กต์ใหม่ รวมถึงจดเตรียมการอ้างอิงให้กับ BOA เพื่อให้พร้อมที่ BOA จะเรียกใช้**

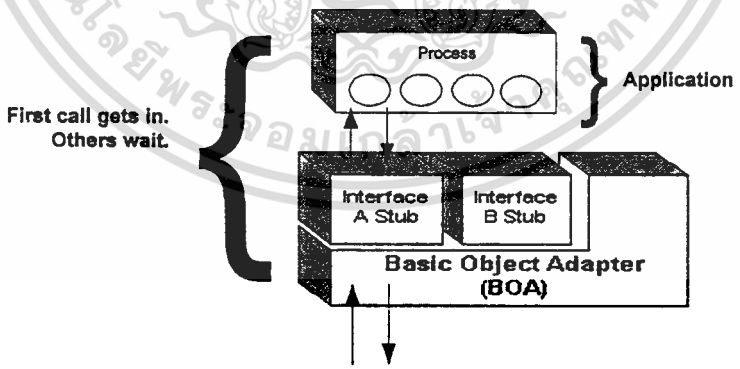
— **change_implementation เป็นคำสั่งเพื่อเปลี่ยนแปลงข้อมูลที่เกี่ยวข้องกับออบเจ็กต์ต่างๆ ให้สอดคล้องกับออบเจ็กต์ใหม่ในขณะนั้น**

- get_id ใช้เมื่อต้องการข้อมูลที่อ้างอิงถึงออบเจ็กต์นั้นๆ ซึ่งอาจจะเป็นหมายเลขของออบเจ็กต์ (Object ID) หรือข้อมูลอื่นๆ ที่เกี่ยวข้อง
- dispose ใช้เมื่อต้องการทำลายการอ้างอิงถึงออบเจ็กต์นั้นๆ
- get_principal คำสั่งนี้เป็นคำสั่งที่กันไว้ให้ส่วนของ Security Service เรียกใช้
- set_exception ใช้เพื่อแจ้งเตือน ORB ในกรณีที่เกิดข้อผิดพลาดหรือความผิดปกติอย่างใดอย่างหนึ่งขึ้น
- impl_is_ready, obj_is_ready ใช้เพื่อสั่งการให้การทำงานและตัวออบเจ็กต์ที่อยู่ภายใต้การลงทะเบียนข้อมูลของ BOA พร้อมทำงาน
- deactivate_impl, deactivate_obj ตรงกันข้ามกับข้างต้น

แนวทางการให้บริการของ BOA (BOA Activation Policy)

ปกติแล้ว BOA จะมีวิธีการทำงานที่ไม่เหมือนกันขึ้นกับว่าลักษณะงานที่ใช้นั้นเป็นเช่นไร อย่างไรก็ดีอาจสามารถที่จะทำการแบ่งแนวทางในการให้บริการหรือแนวทางการทำงานของ BOA ออกได้เป็นแนวทางใหญ่ๆ ทั้งหมด 4 แนวทางด้วยกันดังนี้คือ

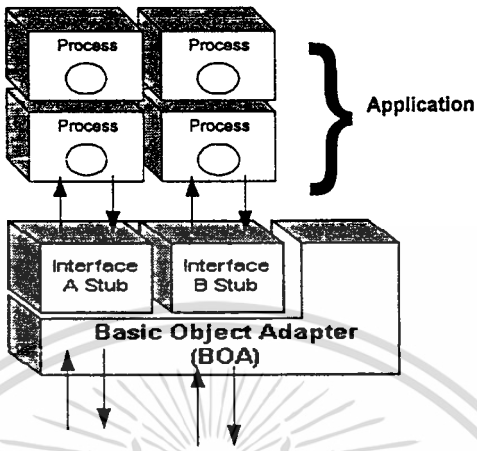
1. **BOA Shared Server :** ในหลักการดังกล่าวจะมีออบเจ็กต์มากมายหลายตัว ซึ่งอาจจะอยู่ในโปรแกรมหรือโปรเซสเดียวกัน BOA จะสั่งการให้เซริฟเวอร์ทำงานแล้วจากนั้นเซริฟเวอร์ก็จะสั่งการต่อไปยังออบเจ็กต์ที่เซริฟเวอร์นั้นดูแลอยู่ หลังจากที่ตัวเซริฟเวอร์ทำการตรวจสอบตัวเองแล้วก็จะทำการแจ้งกลับไปยัง BOA ให้รู้ว่าเซริฟเวอร์พร้อมแล้วที่จะให้บริการ โดยใช้คำสั่ง impl_is_ready ซึ่งเมื่อ BOA รับรู้แล้วก็จะมีการบันทึกรูปแบบวิธีการการทำงานของโปรเซสนั้นๆ ไว้และจะไม่มีการสั่งการให้เซริฟเวอร์อื่นๆ ทำงานขึ้นมาซ้ำซ้อนอีก



รูป 2-6 BOA Shared Server Activation Policy

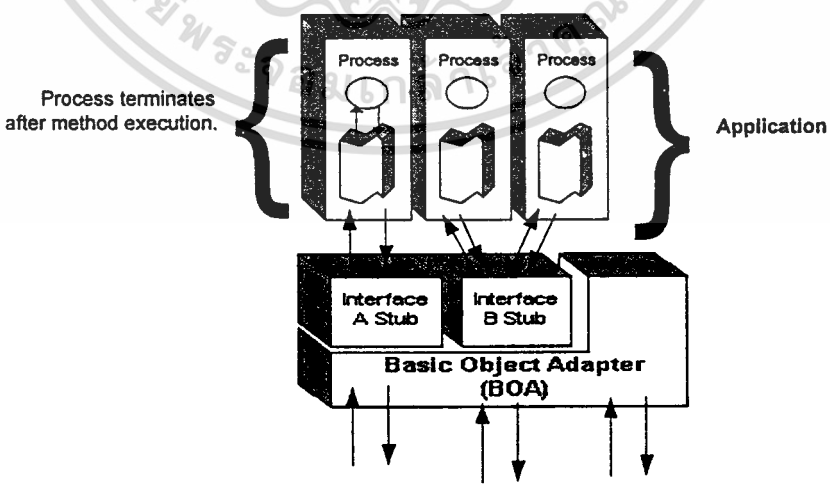
2. **BOA Unshared Server :** แต่ละออบเจ็กต์จะอยู่ในแต่ละเซริฟเวอร์โปรเซส โดยเซริฟเวอร์เริ่มทำงานและมีการติดต่อกับออบเจ็กต์ เมื่อออบเจ็กต์ทำการกำหนดรูปแบบของตัวเองเรียบร้อยแล้วจะบอกให้ BOA รู้โดยใช้คำสั่ง obj_is_ready และในหลักการนี้เซริฟเวอร์ตัวใหม่จะสามารถเริ่มการทำงานได้

ถ้าหากออบเจ็กต์ที่ต้องการไม่ได้ทำงานอยู่ ถึงแม้จะมีรูปแบบและวิธีการทำงานที่เหมือนกับของเดิมก็ตาม หลักการนี้มักจะถูกใช้เมื่อมีออบเจ็กต์ที่ใช้ในงานใดโดยเฉพาะ เช่นออบเจ็กต์ในรูปของพรินท์เตอร์หรือหุ่นยนต์ในโรงงาน ซึ่งแต่ละโปรเซสก็จะดูแลออบเจ็กต์ของตนเองเท่านั้น



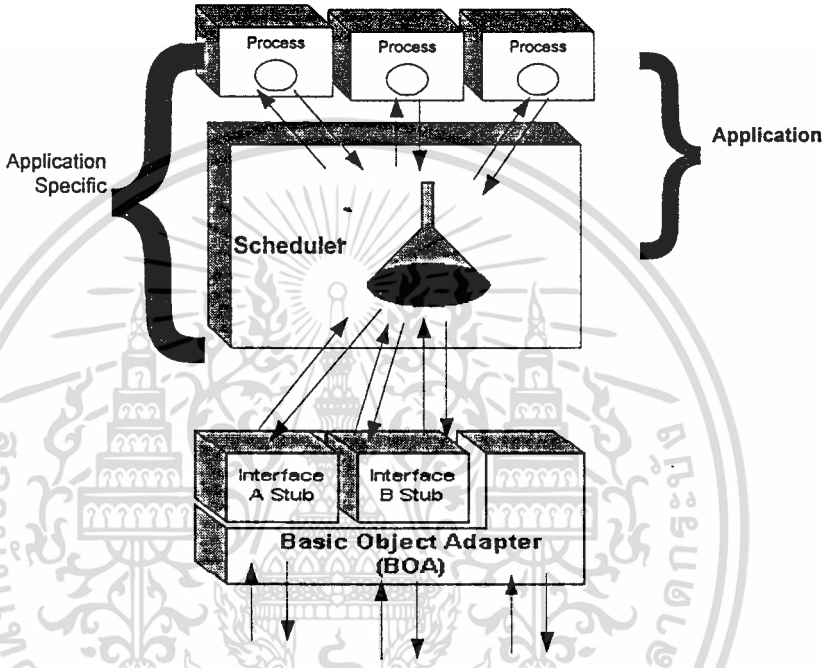
รูป 2-7 BOA Unshared Server Activation Policy

3. **BOA Server-per-method** : เซริฟเวอร์ในหลักการนี้จะทำงานในช่วงเวลาสั้นๆ จำพวกโปรแกรมเอนกประสงค์หรือพวกสคริปท์ (Script) โดยหลักการนี้เซริฟเวอร์ไม่จำเป็นต้องแจ้งข้อมูลกลับไปยัง BOA BOA จะทำการเรียกใช้หรือสั่งการให้เซริฟเวอร์ทำงานในทุกๆ ครั้งที่มีการร้องขอ ในรูปจะเห็นว่ามีเซริฟเวอร์โปรเซสหลายตัวสำหรับออบเจ็กต์ตัวเดียวกัน หรือ method ตัวเดียวกันในออบเจ็กต์ตัวเดียวกันอาจมีการทำงานติดต่อกันหลายครั้งก็ได้



รูป 2-8 The BOA Server-per-method Activation Policy.

4. **BOA Persistent Server :** ในกรณีนี้เซริฟเวอร์จะถูกสั่งให้ทำงานโดยอยู่นอกเหนือภารทำงานของ BOA โดยหลังจากเซริฟเวอร์ทำงานจนพร้อมแล้วจะทำการแจ้งกลับไปยัง BOA โดยใช้คำสั่ง impl_is_ready จากนั้น BOA ก็จะทำงานเหมือนกับเป็น Shared Server จะเห็นว่า BOA Persistent Server เป็นกรณีพิเศษของ Shared Server แต่แตกต่างกันตรงที่การทำงานของเซริฟเวอร์จะไม่ได้ถูกดูแลโดย ORB แต่เซริฟเวอร์จะถูกสั่งให้ทำงานโดยผู้ใช้งานซึ่งจะต้องพิมพ์คำสั่งเข้าไปโดยเฉพาะในกรณีนี้อาจจะมีการสร้างสคริปท์เริ่มการทำงานของเซริฟเวอร์ก็ได้



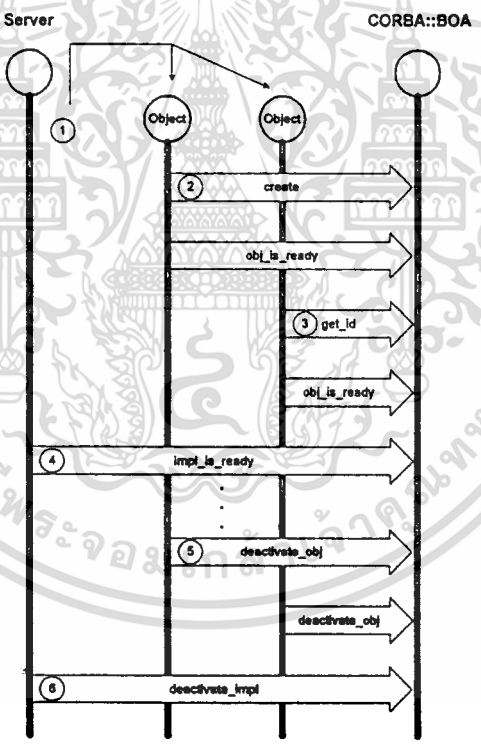
รูป 2-9 BOA Persistent Server Activation Policy

เค้าร่างการทำงานของออบเจ็กต์ในหลักการแบบ Shared Server

สำหรับในที่นี้จะยกตัวอย่างการทำงานของหลักการให้บริการของ BOA ในแบบที่เรียกว่า Shared Server เพื่อเป็นการแสดงให้เห็นถึงลักษณะการทำงานคร่าวๆ ดังนี้

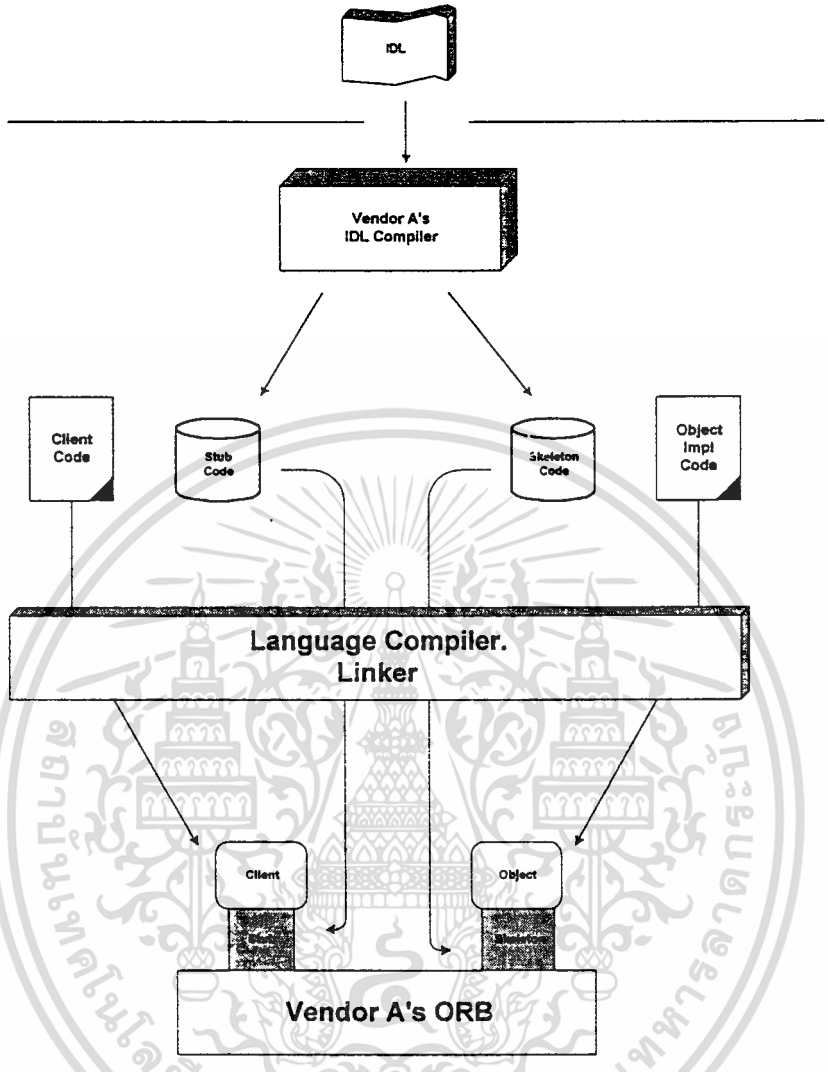
1. เซริฟเวอร์ทำการสร้างออบเจ็กต์ 2 ตัวโดยการเรียกใช้ CORBA factory หรือ Java constructor
2. BOA ทำการลงทะเบียนออบเจ็กต์ใหม่ที่เกิดขึ้น ออบเจ็กต์ใหม่ที่เกิดขึ้นจะต้องเรียกใช้คำสั่ง BOA::create ซึ่งจะต้องส่งค่าต่างๆ ไปให้เช่น ชื่อที่ใช้ในการติดต่อ ข้อมูลจำเพาะของออบเจ็กต์ (persistent ID) ซึ่งคำสั่ง create จะส่งค่าอ้างอิงถึงตัวออบเจ็กต์นั้นๆ กลับมา ซึ่งสามารถที่จะส่งค่าอ้างอิงเหล่านี้ให้รู้ได้โดยใช้คำสั่ง object_is_ready เพื่อบอกให้ ORB รู้ว่าออบเจ็กต์พร้อมแล้วสำหรับการใช้งาน

3. ออบเจ็กต์ต่างๆ ที่มีอยู่แล้วทำการลงทะเบียนกับ BOA เสียก่อน หลังจากที่ BOA ทำการลงทะเบียนออบเจ็กต์ใหม่แล้วก็จะมาทำการลงทะเบียนกับออบเจ็กต์เดิมที่มีอยู่ในระบบด้วยคำสั่ง obj_is_ready ด้วยเช่นเดียวกัน
4. เซริฟเวอร์แสดงสถานะความพร้อมในการให้บริการ โดยเซริฟเวอร์จะทำการสั่งให้ออบเจ็กต์ต่างๆ เตรียมพร้อมและทำการแจ้งให้ ORBรู้ด้วยคำสั่ง impl_is_ready สำหรับขั้นตอนนี้จะต้องกระทำเสมอสำหรับเซริฟเวอร์นี้
5. สั่งงานให้ออบเจ็กต์หยุดทำงานหรืออยู่ในสภาวะหยุดนิ่ง (Object deactivate) สำหรับการสั่งให้ออบเจ็กต์หยุดนิ่งสามารถกระทำได้ในหลายลักษณะเช่น กระทำเมื่อไม่มีไคลเอนท์ใดเรียกใช้หรืออ้างอิงถึงออบเจ็กต์ตัวนี้อีก สำหรับขั้นตอนนี้ตัวออบเจ็กต์จะเป็นตัวสั่งการเองโดยใช้คำสั่ง deactivate_obj
6. เซริฟเวอร์หยุดการทำงาน ในขั้นตอนหรือกรณีสุดท้ายเซริฟเวอร์จะเรียกใช้คำสั่ง deactivate_impl เพื่อบอกให้ ORB รู้ว่าเซริฟเวอร์ตัวนี้หยุดให้บริการแล้ว



รูป 2-10 โครงร่างการทำงานของ BOA Shared Server

วิธีการสร้างโปรแกรมประยุกต์



รูป 2-11 กระบวนการในการสร้างโปรแกรมประยุกต์

จากรูปจะเห็นว่าในการสร้างโปรแกรมประยุกต์นั้นจะต้องประกอบไปด้วยส่วนต่างๆ ดังต่อไปนี้

1. IDL Code
2. IDL Compiler หรือบางครั้งอาจเรียกว่า IDL Pre-compiler
3. Client Code และ Object Implementation Code หรือโปรแกรมที่เป็นของเซริฟเวอร์นั่นเอง
4. Compiler ของภาษาที่ใช้ในการสร้างโปรแกรมประยุกต์
5. ORB ซึ่งจะต้องเป็นของบริษัทผู้ผลิตเดียวกันกับตัว IDL Compiler

ในรูปได้แสดงให้เห็นถึงลักษณะของการทำงานเพื่อสร้างโปรแกรมประยุกต์ขึ้นมา โดยอาศัยเครื่องมือที่เกี่ยวข้องกับมาตรฐาน CORBA ซึ่งจะต้องเป็นของผู้ผลิตรายเดียวกัน สำหรับขั้นตอนในการทำงานสามารถที่จะลำดับได้ดังนี้

1. ทำการสร้างไฟล์ภาษา IDL ขึ้นมาเสียก่อน โดยในไฟล์นั้นจะต้องทำการระบุถึงโครงร่างของการติดต่อและเรียกใช้ method ต่างๆ รวมทั้ง method ที่ไคลเอนท์จำเป็นต้องใช้ด้วย
2. นำไฟล์ที่สร้างขึ้นในภาษา IDL นั้นมาทำการคอมไพล์โดยใช้ตัว IDL Compiler ของบริษัทผู้ผลิตรายหนึ่งซึ่งสมมุติให้เป็นผู้ผลิต A
3. หลังจากที่ทำการคอมไพล์ไฟล์ภาษา IDL แล้วจะได้ไฟล์ขึ้นมาจำนวนหนึ่งซึ่งอาจจะมีจำนวนไม่เท่ากันขึ้นกับตัวคอมไพเลอร์ที่เลือกใช้ นอกจากนั้นชื่อของไฟล์ที่ได้ก็อาจจะไม่เหมือนกันทั้งหมดด้วย แต่สำหรับไฟล์ที่ได้ทั้งหมดนั้นสามารถที่จะแบ่งออกได้เป็น 2 จำพวกด้วยกันคือ Stub กับ Skeleton ในส่วนของไฟล์ที่เป็น Stub จะมีหน้าที่ในการแสดงวิธีการทำงานของ method ต่างๆ ที่ไคลเอนท์จะเรียกใช้ ส่วน Skeleton จะเป็นไฟล์ที่มีหน้าที่ในการแสดงรูปแบบการให้บริการของออบเจ็กต์รวมทั้งรูปแบบการทำงานหรือเรียกใช้ออบเจ็กต์นั้น
4. ทำการสร้าง Client Code และ Object Impl Code ซึ่งโปรแกรมทั้งสองตัวก็จะต้องมีการเรียกใช้ไฟล์ที่เป็น Stub และ Skeleton ตามลำดับด้วย โดยในส่วนของ Client Code ก็จะเป็นโปรแกรมที่ให้ทางไคลเอนท์เป็นผู้เรียกใช้ ส่วน Object Impl Code ก็จะเป็นโปรแกรมที่ทำหน้าที่เป็นตัวเซริฟเวอร์นั่นเอง
5. นำไฟล์ที่เป็น Client Code มาทำการคอมไพล์ด้วยคอมไพเลอร์ที่สอดคล้องกับภาษาที่ใช้เป็น Client Code ซึ่งหลังจากนั้นจะได้ไฟล์ที่เป็นโปรแกรมของไคลเอนท์ส่วนหนึ่งและอีกส่วนหนึ่งจะได้เป็น Stub ไฟล์ซึ่งพร้อมที่จะทำงานร่วมกับโปรแกรมของไคลเอนท์ได้
6. นำไฟล์ที่เป็น Object Impl Code มาทำเช่นเดียวกันกับ Client Code ซึ่งจะได้ไฟล์สองส่วนเช่นกันคือส่วนที่เป็นเซริฟเวอร์ไฟล์และส่วนที่เป็น Skeleton ไฟล์
7. หลังจากนั้นก็สามารถที่จะทำการรันโปรแกรมต่างๆ เพื่อทำงานตามที่ต้องการได้

ความรู้พื้นฐานเกี่ยวกับ Java

เนื่องจาก Java เป็นเรื่องที่ค่อนข้างใหม่ ยังไม่เป็นที่คุ้นเคยของผู้ใช้ (User) การทำความเข้าใจกับ Java จึงเป็นสิ่งสำคัญ Java เป็นได้หลายอย่างในมุมมองที่ต่างออกไป ในขณะที่ Scott Mcnealy จาก Sun บอกว่ามันคือ การปฏิวัติอุตสาหกรรมไอทีครั้งใหม่บิลล์เกตส์แห่งไมโครซอฟท์ กลับมองเห็นมันเป็นเพียงภาษาโปรแกรมมิ่งภาษาใหม่ที่อาจเข้ามาแทนที่ C/C++ ในอนาคตเท่านั้น

Java ถูกแนะนำสู่วงการอุตสาหกรรมไอทีในฐานะของเทคโนโลยีใหม่ ที่จะเข้ามาแทนที่ช่องว่างและการแก้ปัญหาในการพัฒนาโปรแกรมประยุกต์สำหรับระบบปฏิบัติการต่างๆ ที่มีอยู่มากมายในตลาด ซึ่งประกอบด้วยส่วนที่เป็นภาษาสำหรับเขียนโปรแกรม ซึ่งอาจกล่าวได้ว่าเป็นภาษาที่พัฒนาต่อจาก C/C++ ซึ่งเป็นภาษาโปรแกรมมิ่งหลักในอุตสาหกรรมไอทีในขณะนี้ โดย Java ได้ทำการปรับปรุงจุดอ่อนในภาษา C/C++ ทำให้ Java เป็นภาษาที่เรียนรู้ได้ง่ายและมีความน่าเชื่อถือมากขึ้น กับส่วนประกอบที่สำคัญที่เป็นประเด็นที่ถกเถียงกันมากอีกส่วนหนึ่งคือ Java Virtual Machine หรือ Java VM ซึ่งเป็นสิ่งที่หลายฝ่ายในอุตสาหกรรมไอทีให้ความสนใจมากเป็นพิเศษ แต่เป็นสิ่งที่บิลล์เกตส์พยายามมองข้ามไป ความตั้งใจเบื้องต้นของ Sun ในการพัฒนา Java ขึ้นมาก็เพื่อเป็นภาษาพื้นฐานในการพัฒนาโปรแกรมประยุกต์สำหรับอุปกรณ์สื่อสารขนาดเล็กและเครื่องใช้ไฮเทคส่วนบุคคลทั้งหลาย เช่น โทรศัพท์

มือถือ PDA และอื่นๆ โดยเพียงแค่บรรจุ Java VM เข้าไปในอุปกรณ์เหล่านี้ก็จะทำให้อุปกรณ์เหล่านี้สามารถใช้ซอฟต์แวร์ และโปรแกรมประยุกต์ที่พัฒนาขึ้นด้วย Java ได้ทันที อย่างไรก็ตามเมื่อ Java เริ่มเป็นที่รู้จักกันในตลาดความสามารถที่ซ่อนเร้นอยู่ก็เผยตัวออกมา ในวันนี้ Java ในสายตาของนักพัฒนาซอฟต์แวร์จำนวนไม่น้อย ไม่ได้จำกัดอยู่ในฐานะของภาษาสำหรับงานควบคุมเท่านั้น หากแต่มีศักยภาพพอที่จะพัฒนาโปรแกรมประยุกต์ขนาดใหญ่ได้ด้วย โดยมีแรงผลักดันที่สำคัญอยู่ที่โครงสร้างของภาษาที่คล้ายคลึงกับ C และความสามารถของ Java VM

Java VM แท้ที่จริงก็คือเครื่องมือที่แปลชุดคำสั่งที่เป็นภาษา Java ที่ผ่านการคอมไพล์แล้ว (byte code) ไปเป็นคำสั่งที่แพลตฟอร์มนั้นๆ เข้าใจ ดังนั้นด้วย Java VM นี้ ผู้พัฒนาโปรแกรมประยุกต์ไม่จำเป็นต้องแปลงโปรแกรมประยุกต์หรือซอฟท์แวร์ของตนให้สามารถทำงานบนแพลตฟอร์มใหม่ๆ ที่จะเกิดขึ้นในอนาคต ซึ่งมักจะเป็นค่าใช้จ่ายก้อนโตเสมอ ส่วนทางด้านผู้ออกแบบและผลิตแพลตฟอร์มใหม่ๆ ก็ไม่จำเป็นต้องคำนึงการให้การสนับสนุนเป็นพิเศษจากผู้ผลิตซอฟท์แวร์ทั้งหลาย สิ่งที่ต้องทำมีเพียงการพัฒนา Java VM สำหรับแพลตฟอร์มของตนเท่านั้น แพลตฟอร์มของตนก็สามารถใช้งานซอฟต์แวร์ Java ที่มีอยู่ทั้งหมด ความสามารถนี้เป็นความฝันของอุตสาหกรรมนี้มาเป็นเวลานาน ในนามของ Portability ของซอฟต์แวร์ การกล้านำเสนอ Portability แบบ 100% ของ Java สู่อุตสาหกรรมนี้จึงเป็นเรื่องที่น่าตื่นเต้นเป็นอย่างยิ่ง ไม่ใช่เรื่องน่าสงสัยว่าทำไม Java จึงเป็นเทคโนโลยีที่ถูกจับตามองมากที่สุดในระยะที่ผ่านมา

โปรแกรมประยุกต์ต่างๆ ของ Java สามารถรันบนแพลตฟอร์มใดก็ได้ อย่างไรก็ตามแพลตฟอร์มนั้นจะต้องมีตัว Java VM อยู่เพื่อแปลชุดคำสั่งในโปรแกรมประยุกต์ให้อยู่ในรูปของคำสั่งที่แพลตฟอร์มนั้นๆ รู้จัก การใช้เทคโนโลยี Virtual Machine ทำให้ประสิทธิภาพของ Java VM บนแต่ละแพลตฟอร์มมีผลต่อประสิทธิภาพของ โปรแกรมประยุกต์โดยตรง Java ยังสามารถนำไปใช้ในการพัฒนาโปรแกรมประยุกต์ต่างๆ ที่มีขนาดใหญ่อย่างเช่น Word Processor โปรแกรมประยุกต์ทางด้าน Graphics หรือแม้กระทั่งโปรแกรมประยุกต์ที่เป็น Client/Server ได้เช่นเดียวกับการที่ใช้ภาษา C/C++ และ Java ยังช่วยลดความผิดพลาด (Bugs) ที่อาจเกิดเนื่องจาก จุดอ่อนในภาษา C/C++ ลงได้อีกด้วย

ในแง่ของคุณสมบัติทางด้านภาษา Java เป็นภาษาออบเจ็กต์ที่เป็นเทคโนโลยีของยุคทศวรรษที่ 90 นี้ ในขณะที่ภาษา C เป็นภาษาของทศวรรษที่ผ่านมาและ C++ เป็นความพยายามในการต่ออายุให้แก่ภาษา C โดยปรับปรุงโครงสร้างให้เป็นออบเจ็กต์ซึ่งทำให้ C ++ ขาดคุณสมบัติบางประการที่ควรจะมีในภาษาแบบออบเจ็กต์เช่น การเคลียร์ทรัพยากรที่ไม่ได้ใช้คืนให้แก่ระบบโดยอัตโนมัติ

บทที่ 3 การประยุกต์ใช้งาน (Implementation)

ในบทนี้จะเป็นการแสดงขั้นตอนของการทำงาน ซึ่งเริ่มตั้งแต่การออกแบบ การสร้างองค์ประกอบต่าง ๆ ของงานไปจนถึงการทดสอบงานที่ทำการสร้างขึ้นมา แต่เนื่องจากงานที่สร้างขึ้นมานั้นอ้างอิงอยู่กับมาตรฐาน CORBA ดั้งนั้นในขั้นแรกจึงจำเป็นต้องทำการเลือกเครื่องมือที่ช่วยในการทำงานของระบบเสียก่อน สำหรับลำดับการทำงานสามารถแสดงได้ดังนี้

การเลือกเครื่องมือใช้งาน

ในการเลือกเครื่องมือใช้งานที่อ้างอิงกับมาตรฐาน CORBA นั้นมีเครื่องมือหลายตัวจากหลายบริษัทเช่น Orbix Web ของ Iona, PowerBroker ของ ExperSoft หรือ Visibroker ของ Visigenic นอกจากนั้นยังมีทั้งแบบที่ใช้กับภาษา C++ และภาษา Java ด้วย สำหรับเครื่องมือที่เลือกใช้เป็นเครื่องมือของบริษัท Visigenic ที่มีชื่อว่า Visibroker for java V3.0 ซึ่งถือได้ว่าเป็นเครื่องมือที่ได้มีการแก้ไขให้ทำงานในลักษณะของมาตรฐาน CORBA 2.0 ได้ดีมากและมีความชัดเจนสมบูรณ์ในการทำงานมากด้วย ซึ่งในตัวเครื่องมือ Visibroker นั้นนอกจากจะมีตัวโปรแกรมที่เป็น ORB แล้วยังมีโปรแกรมที่เป็นคอมไพเลอร์ที่สำคัญสำหรับมาตรฐาน CORBA อีกอย่างหนึ่งนั่นคือโปรแกรม idl2java ซึ่งเป็นตัวโปรแกรมที่ใช้ในการแปลงไฟล์ที่มีนามสกุล idl เป็นไฟล์ต่างๆที่อยู่ในภาษา java ด้วย สำหรับในเครื่องมือที่ชื่อ Visibroker นั้นโปรแกรมที่เป็นตัว ORB จะมีชื่อว่า Smart Agent นอกจากนั้นยังมีโปรแกรมที่ใช้รันไฟล์ที่มีนามสกุล .class อีกด้วยซึ่งมีชื่อว่า vbj แต่สำหรับ vbj นั้นอาจไม่ใช้ก็ได้เนื่องจากทำหน้าที่เหมือนกับโปรแกรม java ของ JDK นั่นเอง นอกจากเครื่องมือดังกล่าวแล้วยังต้องมีเครื่องมือเสริมการทำงานอีกซึ่งเป็นเครื่องมือที่ใช้ในการคอมไพล์จากไฟล์นามสกุล .java เป็น .class โดยได้เลือก JDK 1.1.4 เนื่องจากทำงานได้ดีและไม่ซับซ้อนมากนัก

ส่วนสำหรับระบบปฏิบัติการนั้นมาตรฐาน CORBA สามารถใช้งานบนระบบปฏิบัติการได้มากมายทั้งในลักษณะของ UNIX และลักษณะของ Windows ทั้งนี้ขึ้นกับว่าเครื่องมือนั้นระบุไว้สำหรับระบบปฏิบัติการอะไร ซึ่งระบบปฏิบัติการที่เลือกคือ Windows NT 4.0 เนื่องจาก Visibroker for java ที่มีความสมบูรณ์มากที่สุดจะเป็นรุ่นที่ใช้สำหรับ Windows ซึ่งทำงานได้ทั้งบน Windows NT และ Windows '95 แต่สำหรับบน Windows '95 แล้วการทำงานจะมีปัญหาในเรื่องของความเร็วซึ่งน่าจะมีผลจากการที่สภาพแวดล้อมส่วนใหญ่ของ Windows '95 ไม่ได้รองรับในลักษณะของการ Service โดยตรงหรือกล่าวได้ว่า Windows '95 ไม่ได้มีสภาพแวดล้อมในลักษณะของเซริฟเวอร์โดยตรงจึงทำให้ผลการตอบสนองนั้นช้ากว่า Windows NT มาก

สภาพแวดล้อมของระบบ (Environment)

Server

เครื่องคอมพิวเตอร์ Pentium P-166 MMX Ram 32 MB

ระบบปฏิบัติการ Windows NTเซริฟเวอร์4.0

เครื่องมือที่มีตัวโปรแกรม ORB เป็นของ Visigenic ซึ่งมีชื่อว่า Visibroker for Java V3.0

เครื่องมือที่มีโปรแกรมที่ใช้ในการคอมไพล์เป็นของ Sun Microsystem Co., Ltd. ชื่อ JDK 1.1.4

Client

เครื่องคอมพิวเตอร์ Pentium P-166 MMX Ram 32 MB

ระบบปฏิบัติการ Windows NTเซริฟเวอร์4.0

เครื่องมือที่มีตัวโปรแกรม ORB เป็นของ Visigenic ซึ่งมีชื่อว่า Visibroker for Java V3.0

เครื่องมือที่มีโปรแกรมที่ใช้ในการคอมไพล์เป็นของ Sun Microsystem Co., Ltd. ชื่อ JDK 1.1.4

จุดประสงค์และสมมุติฐานของโปรแกรมประยุกต์

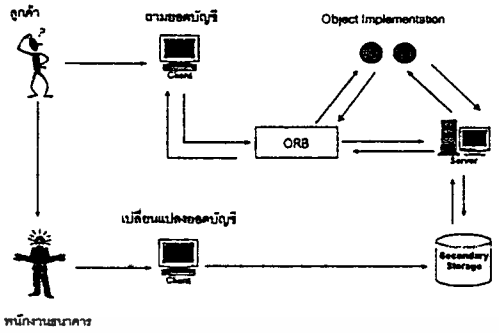
สำหรับการสร้างโปรแกรมประยุกต์ซึ่งอ้างอิงกับมาตรฐาน CORBA นั้นจะมีลักษณะเป็นการจำลองการทำกิจกรรมของธนาคาร ซึ่งกิจกรรมดังกล่าวนั้นเป็นเพียงการจำลองรูปแบบกิจกรรมเท่านั้น ดังนั้นโครงสร้างของข้อมูลที่ใช้ในโปรแกรมประยุกต์จึงเป็นเพียงข้อมูลธรรมดาเท่านั้นมิได้มีความสลับซับซ้อนอะไร จุดประสงค์สำคัญของโปรแกรมประยุกต์นี้เพื่อที่จะทำการแสดงให้เห็นถึงลักษณะการทำงานตามมาตรฐาน CORBA โดยในโปรแกรมประยุกต์นี้จะมีการสร้างโปรแกรมที่กระทำตามกิจกรรมข้างต้น และเนื่องจากกิจกรรมบางอย่างสร้างขึ้นเพื่อให้บุคคลบางกลุ่มเป็นผู้เรียกใช้งานเท่านั้น ฉนั้นจึงกำหนดเป็นสมมุติฐานในเบื้องต้นว่ากิจกรรมใดๆก็ตามที่มีความซับซ้อนก็จะมีไว้สำหรับบุคคลที่มีความรู้ในการใช้งานแล้ว ซึ่งจะแสดงต่อไปในส่วนของการปฏิบัติ สำหรับในเบื้องต้นนี้จะสมมุติให้มีบัญชีทั้งหมด 3 บัญชีและมียอดเงินดังนี้ดังนี้

Warin 300.00

Pakdee 218.00

Suwit 74.00

การออกแบบ



รูป 3-1 รูปแบบการทำงานของโปรแกรมประยุกต์

จากรูปจะเห็นว่ามีการจำลองลักษณะงานแบ่งได้เป็น 2 ลักษณะงานคือ กิจกรรมการเปลี่ยนแปลงยอดบัญชีและกิจกรรมการถามยอดบัญชี โดยที่กิจกรรมการถามยอดบัญชีนั้นลูกค้าสามารถจะกระทำการได้ด้วยตนเองโดยผ่านโปรแกรมไคลเอนท์และในส่วนของกิจกรรมการถามยอดบัญชีนี้เองที่เป็นส่วนที่สร้างขึ้นโดยอิงกับมาตรฐาน CORBA ดังนั้นจึงเห็นว่าเมื่อโปรแกรมไคลเอนท์มีการทำงานเพื่อสอบถามยอดบัญชีของลูกค้าแล้ว ตัว ORB (Object Request Broker) จะถูกเรียกใช้เพื่อทำการค้นหาและติดต่อกับเซริฟเวอร์เพื่อที่จะทำการตรวจสอบชื่อลูกค้าและค้นหายอดบัญชีส่งกลับมายังไคลเอนท์ในส่วนของการส่งค่ากลับมายังไคลเอนท์นั้นก็ยังคงต้องใช้บริการผ่านทางตัว ORB ด้วยเช่นกัน ส่วนกิจกรรมอีกอย่างหนึ่งคือการเปลี่ยนแปลงยอดบัญชีนั้นลูกค้าไม่สามารถจะทำการได้ด้วยตนเอง แต่จะต้องกระทำการโดยผ่านทางพนักงานธนาคาร ซึ่งในการเปลี่ยนแปลงยอดบัญชีนั้นพนักงานธนาคารจะกระทำการโดยตรงไปยังส่วนของ Secondary Storage ในส่วนของการเปลี่ยนแปลงยอดบัญชีนี้มิได้มีการอ้างอิงกับมาตรฐานของ CORBA เนื่องจากเป็นการซ้ำซ้อน แต่ที่ต้องมีเนื่องจากต้องการสร้างข้อมูลที่เป็นยอดบัญชีซึ่งมีลักษณะที่ถาวรเก็บอยู่ในส่วนของ Secondary Storage นั่นเอง

กิจกรรมการถามยอดบัญชี

ลักษณะการทำงานเริ่มต้นจากการที่ลูกค้าธนาคารมีการเรียกใช้โปรแกรมไคลเอนท์โดยจะต้องมีการระบุชื่อของเจ้าของบัญชีเข้าไปในโปรแกรมเป็นอาร์กิวเมนต์ด้วย หลังจากนั้นโปรแกรมไคลเอนท์จะทำการเรียกใช้ตัว ORB โดยส่งค่าของอาร์กิวเมนต์มาให้ ORB ซึ่งตัว ORB จะทำการติดต่อไปยังฝั่งของเซริฟเวอร์และส่งอาร์กิวเมนต์ต่อไปให้ตัวเซริฟเวอร์ด้วย ส่วนเซริฟเวอร์นั้นเมื่อได้รับอาร์กิวเมนต์ที่เป็นชื่อเจ้าของบัญชีแล้วจะทำการตรวจสอบตารางข้อมูลที่เป็นยอดบัญชีซึ่งมีการสร้างไว้แล้วเป็นลักษณะของตารางข้อมูลชั่วคราว (Dynamic Table) ว่ามีชื่อลูกค้าดังกล่าวในตารางหรือไม่ ถ้าหากมีก็จะนำเอาตัวเลขยอดบัญชีของชื่อนั้นส่งกลับมายัง ORB แต่ถ้าหากไม่พบก็จะทำการตรวจสอบกับข้อมูลที่อยู่ใน Secondary Storage อีกทีหนึ่งว่ามีข้อมูลที่ตรงกับชื่อที่เป็นอาร์กิวเมนต์ที่ได้รับมาหรือไม่ ถ้าพบก็จะนำข้อมูลใน

Secondary Storage นั้นออกมาสร้างเป็นข้อมูลในตารางก่อนที่จะนำเอาข้อมูลนี้ส่งกลับไปยัง ORB แต่ถ้าหากยังไม่พบอีกก็จะแจ้งกลับไปยัง ORB ซึ่งทางฝั่งไคลเอนท์ก็จะรับรู้ได้ทันทีว่าไม่มียอดบัญชีของผู้ฝากรายนี้อยู่เลย

กิจกรรมการเปลี่ยนแปลงยอดบัญชี

ในส่วนของกิจกรรมการแปลี่ยนแปลงยอดบัญชีนี้จะแตกต่างกับกิจกรรมการถามยอดบัญชีอย่างสิ้นเชิง สำหรับกิจกรรมการเปลี่ยนแปลงยอดบัญชีจะเริ่มจากการที่พนักงานธนาคารทำการเรียกใช้โปรแกรม Create ซึ่งจะมีการส่งค่าอาร์กิวเมนต์เข้าไปทั้งหมด 2 ตัวด้วยกันคือชื่อบัญชีและยอดบัญชี ซึ่งตัวโปรแกรม Create เองก็จะมีการตรวจสอบอาร์กิวเมนต์เพื่อให้มั่นใจว่าถูกต้อง เริ่มตั้งแต่จะต้องมีอาร์กิวเมนต์ทั้งหมดสองตัวด้วยกัน และอาร์กิวเมนต์ตัวที่ 2 ต้องมีทศนิยมไม่เกิน 2 ตำแหน่งเท่านั้น ถ้าหากจำนวนและลักษณะของอาร์กิวเมนต์ทั้งหมดอยู่ในลักษณะที่ถูกต้องแล้วก็จะมีการสร้างไฟล์ทั้งหมด 2 ตัวด้วยกันคือไฟล์ที่มีนามสกุลเป็น .name และ .balance ซึ่งไฟล์ที่มีนามสกุล .name จะเก็บข้อมูลชื่อเจ้าของบัญชี ส่วนไฟล์ที่มีนามสกุล .balance นั้นจะเก็บข้อมูลของตัวเลขยอดบัญชีไว้

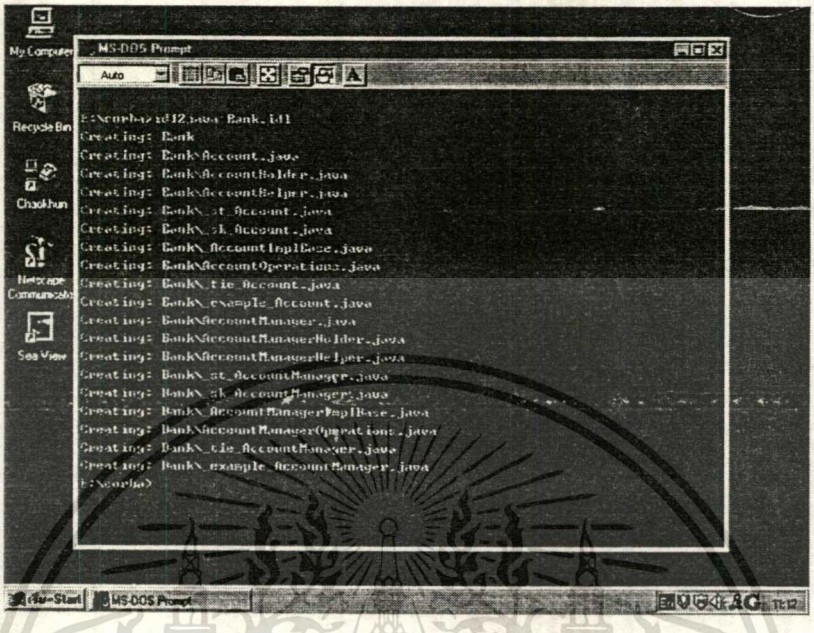
ไฟล์ที่เกี่ยวข้อง

- **Bank.idl** : ทำหน้าที่เป็นไฟล์กำหนดตัว class ที่ทั้งฝั่งของไคลเอนท์และเซริฟเวอร์จะต้องติดต่อด้วย ซึ่งในส่วนของ Bank.idl นั้นยังระบุถึงลักษณะของตัวแปรและชื่อของ method บางส่วนที่อยู่ใน class แต่ละตัวด้วย ลักษณะของไฟล์ Bank.idl จะมีลักษณะดังนี้

```
module Bank {
 interface Account {
  float balance();
 };
 interface AccountManager {
  Account open(in string name);
 };
};
```

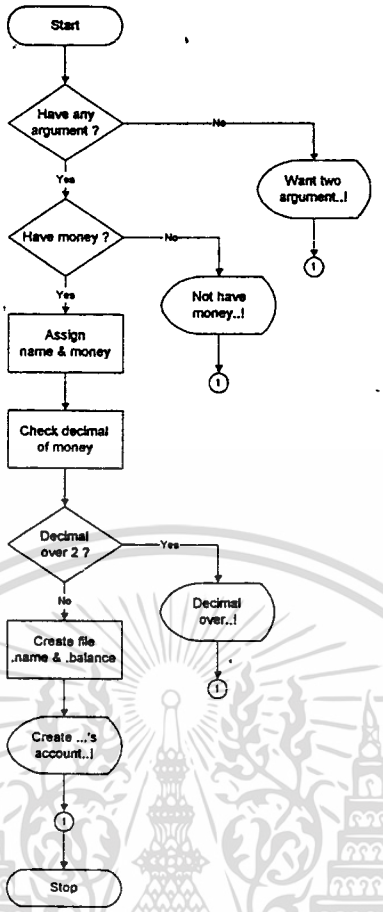
ในไฟล์ Bank.idl จะมีการกำหนดให้สร้างไดเร็กทอรี่ Bank สำหรับเก็บไฟล์ที่จะสร้างขึ้นทั้งหมด โดยใช้ถ้อยแถลง module Bank ส่วน interface Account เป็นการระบุให้สร้างไฟล์ Account.java เพื่อเป็นไฟล์ที่จะเรียกใช้โดยภายในจะมี method balance ซึ่งมี การส่งค่ากลับ(return type) เป็นค่าจำนวนจริง (Float) ส่วน interface AccountManager เป็นการระบุให้สร้างไฟล์ AccountManager.java ซึ่งภายในจะมีการเรียกใช้ method open จากไฟล์ Account โดยส่งอาร์กิวเมนต์เข้าไปหนึ่งตัวคือ name มีลักษณะเป็นข้อความ (String) หลังจากนั้นก็นำไฟล์ Bank.idl ไปผ่านการคอมไพล์โดยใช้โปรแกรมที่ทำหน้าที่ในการ

คอมไพล์ของ Visibroker คือ idl2java โดยเรียกโปรแกรมแล้วตามด้วยชื่อของไฟล์ที่เป็น .idl ซึ่งจะได้ผลลัพธ์ดังรูป



รูป 3-2 ผลลัพธ์การคอมไพล์ไฟล์ Bank.idl

– **Create.java** : ทำหน้าที่ในการสร้างยอดบัญชี เนื่องจากในการถามยอดนั้นจำเป็นจะต้องมียอดบัญชีที่เป็นข้อมูลถาวรอยู่ในส่วนของ Secondary Storage ดังนั้นจึงมีการสร้างไฟล์ Create.java ขึ้นมาเพื่อใช้งานตามวัตถุประสงค์ดังกล่าว ลักษณะการทำงานของไฟล์สามารถแสดงได้ด้วย Flow Chart ดังนี้



รูป 3-3 Flow chart แสดงการทำงานของโปรแกรม Create

สำหรับไฟล์ Create.java จะเป็นไฟล์ที่ทำการสร้างข้อมูลโดยตรงไปยังส่วนของ Secondary Storage เลย ดังนั้นจึงไม่มีการเรียกใช้ ORB ไฟล์ Create.java นี้จะประกอบไปด้วยส่วนที่เป็นโปรแกรมหลักและส่วนที่เป็น method convert ซึ่งใช้สำหรับการตรวจจำนวนทศนิยมของอาร์กิวเมนต์ตัวที่ 2 ลักษณะของไฟล์ Create.java จะเป็นดังนี้

```
// Create.java for teller.
import java.lang.*;
import java.io.*;
import java.awt.event.*;
import java.security.*;

class Create {
 public static void main(String[] arg) {
  String name = arg.length > 0 ? arg[0] : null;
```

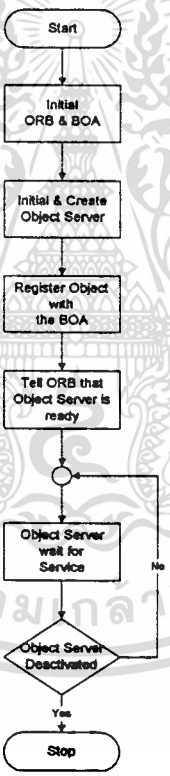
```
if (name==null) {
  System.out.println("Want two argument..!");
} else {
   String money = arg.length > 1 ? arg[1] : null;
   if (money==null) {
    System.out.println("Not have money..!");
   } else {
      char[] ch = money.toCharArray();
      int dot_position = convert(ch);
      if (dot_position>2) {
      System.out.println("Decimal Over..!");
      } else {
         Float t0 = Float.valueOf(money);
         float balance = t0.floatValue();
         String fname = name+".name";
         String fbal = name+".balance";
         //Save the account to File.
         try {
          FileOutputStream fnameout = new FileOutputStream(fname);
          FileOutputStream fbalout = new FileOutputStream(fbal);
          DataOutputStream nameout = new DataOutputStream(fnameout);
          DataOutputStream balout = new DataOutputStream(fbalout);
          nameout.writeBytes(name);
          balout.writeFloat(balance);
          System.out.println("Created "+name+"'s account..!");
          fnameout.close();
          fbalout.close();
         } catch (IOException e){}
       }
    }
  }
}

private static int convert(char[] ch_in) {
```

```
        int len_t =  ch_in.length;
        int len = len_t;
        int temp1 = 0;
        int pos_dot = 0;
        if (len<=14) {
         do {
          char temp0 = ch_in[len-1];
          --len;
          if (temp0!='.') {
           temp1++;
          } else {
            pos_dot = temp1;
           }
         } while(len!=0);
        }
        return pos_dot;
       }

      }
```

การทำงานของ Create เริ่มจากการที่ผู้เรียกใช้โปรแกรมมีการส่งค่าอาร์กิวเมนต์เข้ามาทั้งหมด 2 ตัว ตัวหนึ่งเป็นชื่อเจ้าของบัญชีส่วนอีกตัวหนึ่งเป็นยอดเงินของบัญชีนั้น ซึ่งอยู่ในลักษณะอาเรย์ของข้อความ(array of String) จากนั้นก็จะทำการกำหนดค่าของอาร์กิวเมนต์ตัวแรกให้ตัวแปรที่ชื่อ name ซึ่งในจุดนี้จะมีการตรวจสอบด้วยว่ามีอาร์กิวเมนต์หรือไม่เพราะถ้าไม่มีอาร์กิวเมนต์เลยตัวแปร name ก็จะมีค่าเป็น null ซึ่งแสดงว่าไม่สามารถจะทำการสร้างยอดบัญชีได้ โปรแกรม Create ก็จะพิมพ์คำว่า " Want two argument..! " แล้วจบโปรแกรม แต่ถ้าหากตัวแปร name ไม่ใช่ค่า null ก็จะทำการกำหนดค่าอาร์กิวเมนต์ตัวที่ 2 ให้กับตัวแปร money เช่นเดียวกันถ้าเกิดในการเรียกใช้โปรแกรม Create ในครั้งนี้มีอาร์กิวเมนต์เพียงตัวเดียวตัวแปร money ก็จะมีค่าเป็น null และถ้า money เป็น null แล้วโปรแกรม Create ก็ไม่สามารถจะกระทำการได้ ซึ่งก็จะพิมพ์คำว่า " Not have money..! " แล้วจบโปรแกรมเช่นเดียวกัน แต่ถ้าเกิดตัวแปร money ไม่ใช่ค่า null ก็แสดงว่ามีอาร์กิวเมนต์ครบทั้ง 2 ส่วน ซึ่งโปรแกรม Create ก็จะสามารถกระทำการต่อไปได้ หลังจากตรวจสอบจำนวนอาร์กิวเมนต์แล้วก็จะเป็นการตรวจสอบจำนวนทศนิยมของอาร์กิวเมนต์ตัวที่ 2 บ้าง โดยการนำเอาอาร์กิวเมนต์ตัวที่ 2 มาแปลงให้เป็นอาเรย์ของตัวอักษร (array of character) ก่อนที่จะทำการส่งผ่านไปยัง method convert สำหรับใน method convert นี้ก็จะทำการตรวจสอบหาตำแหน่งของตัวอักษร '.' เริ่มจากด้านท้ายเป็นต้นมา ซึ่งในระหว่างที่หาอยู่นั้นก็จะมีการนับจำนวนของตัว

อักษรหรือตัวเลขที่อยู่หลังจุดทศนิยมด้วย ซึ่งในส่วนสุดท้ายของ method convert นี้จะมีการส่งค่าของจำนวนทศนิยมกลับเป็นลักษณะของจำนวนเต็ม การส่งค่าของจำนวนจุดทศนิยมกลับนี้จะถูกเก็บไว้ที่ตัวแปร dot_position ของโปรแกรมหลัก (main program) แล้วโปรแกรมหลักก็จะตรวจสอบต่อว่าจำนวนทศนิยมนั้นเกิน 2 หลักหรือไม่ถ้าเกินก็จะพิมพ์คำว่า " Decimal Over..! " แล้วจบโปรแกรมอีกเช่นเดิม แต่ถ้าหากไม่เกินโปรแกรมหลักก็จะทำการบันทึกข้อมูลลงใน Secondary Storage โดยเก็บข้อมูลชื่อบัญชีไว้ในไฟล์นามสกุล .name และตัวเลขยอดบัญชีเก็บในไฟล์นามสกุล .balance สำหรับชื่อของไฟล์ทั้งสองจะใช้อาร์กิวเมนต์ที่เป็นชื่อเจ้าของบัญชีที่รับเข้ามานั่นเอง หลังจากสร้างเสร็จก็จะแสดงข้อความ "Create ... 's account..!" ก่อนที่จะจบการทำงานของโปรแกรม

— **Server.java** : ทำหน้าที่ในการกำหนดและเตรียมพร้อมออบเจ็กต์เซริฟเวอร์ในการให้บริการ ลักษณะการทำงานของไฟล์สามารถแสดงได้ดังนี้



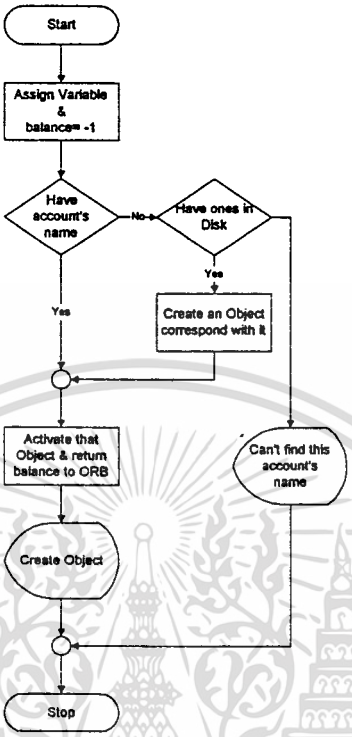
รูป 3-4 Flow chart แสดงการทำงานของโปรแกรมเซริฟเวอร์

นอกจากไฟล์ตัวนี้จะทำการกำหนดค่าเริ่มต้นให้ออบเจ็กต์เซริฟเวอร์แล้วยังมีการเรียกใช้ตัว BOA ด้วยเพื่อที่จะทำการลงทะเบียนออบเจ็กต์เซริฟเวอร์ตัวนี้ไว้และบอกให้ ORB รู้ว่าออบเจ็กต์เซริฟเวอร์ตัวนี้พร้อมแล้วสำหรับการให้บริการ รูปแบบการเขียนของไฟล์ Server.java จะเป็นดังนี้

```
public class Server {
 public static void main(String[] args) {
  // Initialize the ORB.
  org.omg.CORBA.ORB orb = org.omg.CORBA.ORB,init(args,null);
  // Initialize the BOA.
  org.omg.CORBA.BOA boa = orb.BOA_init();
  // Create the account manager object.
  Bank.AccountManager manager =
   new AccountManagerImpl("BankChecking");
  // Export the newly create object.
  boa.cbj_is_ready(manager);
  System.out.println(manager + " is ready.");
  // Wait for incoming requests
  boa.impl_is_ready();
 }
}
```

จากรูปแบบของไฟล์จะสังเกตุเห็นว่าในไฟล์ Server.java นั้นจะเริ่มต้นด้วยการสร้างออบเจ็กต์ที่เป็นตัว ORB และ BOA(Basic Object Adapter) เสียก่อน โดยมีการกำหนดค่าอาร์กิวเมนต์เบื้องต้นให้กับออบเจ็กต์ที่เป็นตัว ORB ด้วย จากนั้นในส่วนถัดมาจะเป็นการสร้างออบเจกต์เซริฟเวอร์ที่ใช้ชื่อตัวแปรว่า manager สร้างจากไฟล์ AccountManagerImpl และมีชื่อในการเรียกใช้ออบเจ็กต์เซริฟเวอร์ดังกล่าวคือ BankChecking จากนั้นได้เรียกการให้บริการของตัว BOA โดยใช้ถ้อยแถลง boa.obj_is_ready(manager); ซึ่งเป็นลักษณะของการลงทะเบียนออบเจ็กต์ตัวนั้นนั่นเอง และในส่วนสุดท้ายจะเป็นการประกาศให้ ORB รู้ว่าออบเจ็กต์เซริฟเวอร์พร้อมที่จะให้บริการกับการติดต่อจาก ORB แล้วโดยใช้ถ้อยแถลงว่า boa.impl_is_ready();

— **AccountManagerImpl.java** : เป็นไฟล์ที่จะเป็นตัวออบเจ็กต์เซริฟเวอร์นั่นเอง มีหน้าที่ในการติดต่อและสร้างตัวออบเจกต์ตามที่ทางฝั่งของไคลเอนท์เรียกใช้ ลักษณะการทำงานเป็นดังนี้



รูป 3-5 Flow chart แสดงการทำงานของออบเจ็กต์เซริฟเวอร์

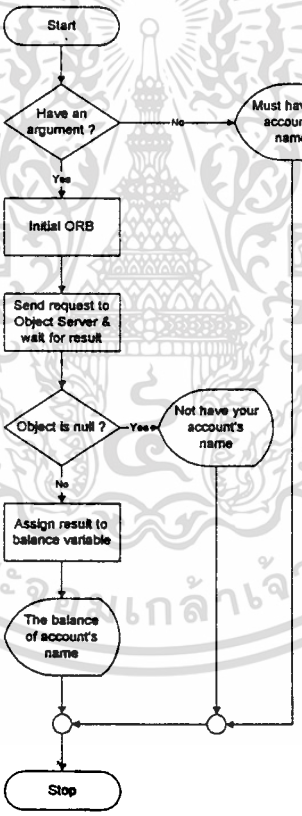
ฉนั้นจึงสามารถกล่าวได้ว่าไฟล์ AccountManagerImpl.java นี้เป็นไฟล์ที่ทำหน้าที่สำคัญมากในกิจกรรมการสอบถามยอดบัญชี ในไฟล์ AccountManagerImpl.java จะมีลักษณะดังนี้

```
// AccountManagerImpl.java
import java.util.*;
import java.io.*;
import java.io.IOException;

public class AccountManagerImpl extends Bank._AccountManagerImplBase {
 public AccountManagerImpl(String name) {
  super(name);
 }
 public synchronized Bank.Account open(String name) {
  String fname = name+".name";
```

```
String fbal = name+".balance";
float balance = -1;
// Lookup the account in the account dictionary.
Bank.Account account = (Bank.Account) _accounts.get(name);
// If there was no account in the dictionary, create one.
if(account == null) {
 try {
  FileInputStream fnamein = new FileInputStream(fname);
  DataInputStream namein = new DataInputStream(fnamein);
  FileInputStream fbalin = new FileInputStream(fbal);
  DataInputStream balin = new DataInputStream(fbalin);
  balance = balin.readFloat();
  fbalin.close();
  fnamein.close();
 } catch (IOException e){}
 if (balance<0) {
  System.out.println("Can't find this account's name");
 } else {
  // Create the account implementation, given the balance.
  account = new AccountImpl(balance);
  // Make the object available to the ORB.
  _boa().obj_is_ready(account);
  // Print out the new account.
  System.out.println("Created " + name + "'s account: " + account);
  // Save the account in the account dictionary.
  _accounts.put(name, account);
  }
 }
 // Return the account.
 return account;
}
private Dictionary _accounts = new Hashtable();
private Random _random = new Random();
}
```

โปรแกรมจะมีการทำงานใน 2 ส่วนด้วยกันโดยในส่วนแรกจะทำการตรวจสอบข้อมูลของบัญชีที่อยู่ในตัวออบเจ็กต์ต่างๆ ก่อน แต่ถ้าหากไม่พบก็จะเข้าไปทำงานในส่วนที่สองซึ่งจะตรวจสอบข้อมูลที่อยู่ใน Secondary Storage ต่อไป ในส่วนที่สองนี้เป็นการตรวจสอบว่าใน Secondary Storage มีข้อมูลในส่วนที่ตรงกับที่ต้องการหรือไม่ ในส่วนของ Secondary Storage นี้ถือว่าเป็นจุดสุดท้ายของข้อมูลถ้าหากพบก็ถือว่ามีข้อมูลนี้อยู่จริง แต่ถ้าไม่พบก็แสดงว่าในขณะนั้นข้อมูลบัญชีของบุคคลดังกล่าวไม่มีแล้ว สำหรับข้อมูลที่พบในส่วนของ Secondary Storage นั้นจะถูกนำไปสร้างเป็นออบเจ็กต์ใหม่ทันที นั่นหมายความว่าหลังจากนี้ไปถ้าหากมีการถามยอดบัญชีของบุคคลดังกล่าวตัวออบเจ็กต์นี้ก็จะให้บริการได้โดยทันที

- **Client.java** : ทำหน้าที่เป็นโปรแกรมทางฝั่งของไคลเอนท์ที่ใช้ในการรับความต้องการในการถามยอดบัญชีของลูกค้าแล้วส่งอาร์กิวเมนต์ต่อไปยัง ORB รูปแบบการทำงานเป็นดังนี้



รูป 3-6 Flowchart แสดงการทำงานของโปรแกรมไคลเอนท์

ในส่วนของไคลเอนท์นี้จะมีการตรวจสอบอาร์กิวเมนต์ด้วย โดยถ้าเกิดไม่มีอาร์กิวเมนต์ก็จะไม่มีการติดต่อแต่ประการใด แต่ถ้ามีอาร์กิวเมนต์แล้วก็จะทำการเรียกใช้บริการจาก ORB ทันที สำหรับค่าที่ไคลเอนท์ต้องการนั้นจะเป็นตัวเลขจำนวนจริงของตัวแปร balance แต่ในการตรวจสอบจะดูว่าออบเจ็กต์

account เป็น null หรือเปล่า ถ้าเป็น null แสดงว่าไม่พบข้อมูลของบัญชีดังกล่าว แต่ถ้าหากไม่เป็น null แสดงว่ามีข้อมูลของลูกค้านั้นอยู่จริง ซึ่งก็จะทำการแสดงผลลัพธ์ออกมา

```
// Client.java
public class Client {
 public static void main(String[] args) {
  // use args[0] as the account name.
  String name = args.length > 0 ? args[0] : null;
  if (name==null) {
   System.out.println("Must have an account's name");
  } else {
    // Initialize the ORB.
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    // Locate an account manager.
    Bank.AccountManager manager =
          Bank.AccountManagerHelper.bind(orb, "BankChecking");
    // Request the account manager to open a named account.
    Bank.Account account = manager.open(name);
    if (account==null) {
     System.out.println("Not have your account's name..!");
    } else {
       float balance = account.balance();
       System.out.println
        ("The balance in " + name + "'s account is $" + balance);
      }
    }
  }

}
```

— **AccountImpl.java : เป็นไฟล์ที่เขียนขึ้นเพื่อแสดงตัว method ที่จะทำการส่งค่าตัวเลขซึ่งเป็นยอดบัญชีที่ต้องการสร้างไปยังออบเจ็กต์ใหม่ที่กำลังจะถูกสร้างขึ้นจากออบเจ็กต์เซริฟเวอร์ ซึ่งจะเห็นได้จากว่าไฟล์ดังกล่าวนั้นจะมีการส่งค่าของยอดบัญชีกลับมาหลังจากเรียกใช้แล้วนั่นเอง ดังจะเห็นได้จากตัวโปรแกรมดังนี้**

```
// AccountImpl.java
public class AccountImpl extends Bank._AccountImplBase {
  public AccountImpl(float balance) {
    _balance = balance;
  }
  public float balance() {
    return _balance;
  }
  private float _balance;
}
```

การทดสอบการทำงานของระบบ

หลังจากทำการสร้างไฟล์ดังกล่าวทั้งหมดแล้วก็จะทำการคอมไพล์เพื่อที่จะทำการรันทดสอบในโอกาสต่อไปสำหรับไฟล์ทั้งหมดนั้นไม่จำเป็นจะต้องทำการคอมไพล์ทั้งหมด แต่ไฟล์ที่จำเป็นต้องคอมไพล์จะมีอยู่ 3 ตัวด้วยกันคือไฟล์ Create.java, Client.java และไฟล์ Server.java ส่วนที่เหลือนั้นจะถูกคอมไพล์เองเมื่อคอมไพล์ไฟล์ดังกล่าวแล้ว สำหรับการคอมไพล์ไฟล์ทั้ง 3 จะทำเช่นเดียวกันดังนี้

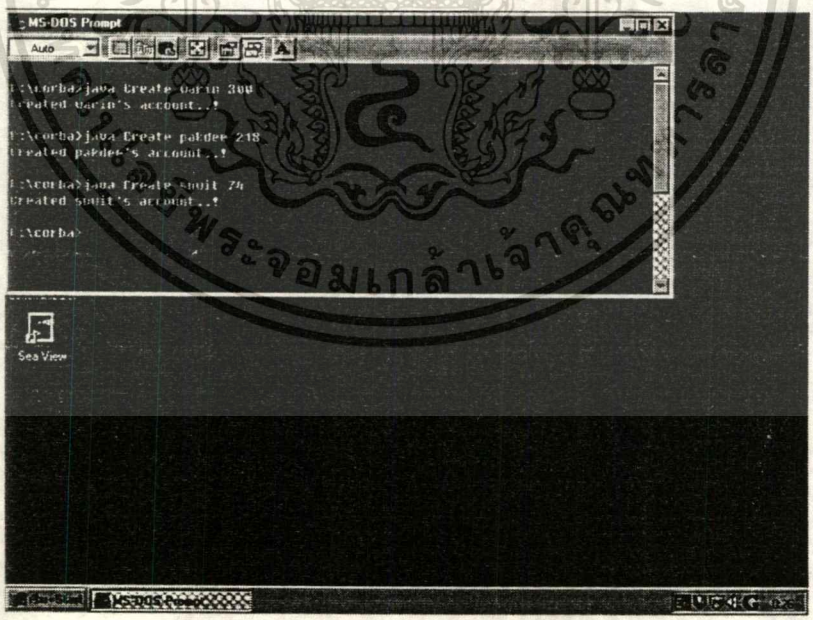
```
prompt > javac Create.java
prompt > javac Client.java
prompt > javac Server.java
```

หลังจากคอมไพล์เรียบร้อยแล้วจะได้ไฟล์ที่มีนามสกุล .class ที่สามารถจะรันได้โดยใช้โปรแกรม java ของ JDK หรือ ใช้ vbj ที่มากับเครื่องมือของ Visibroker ก็ได้

การทำงานของระบบ

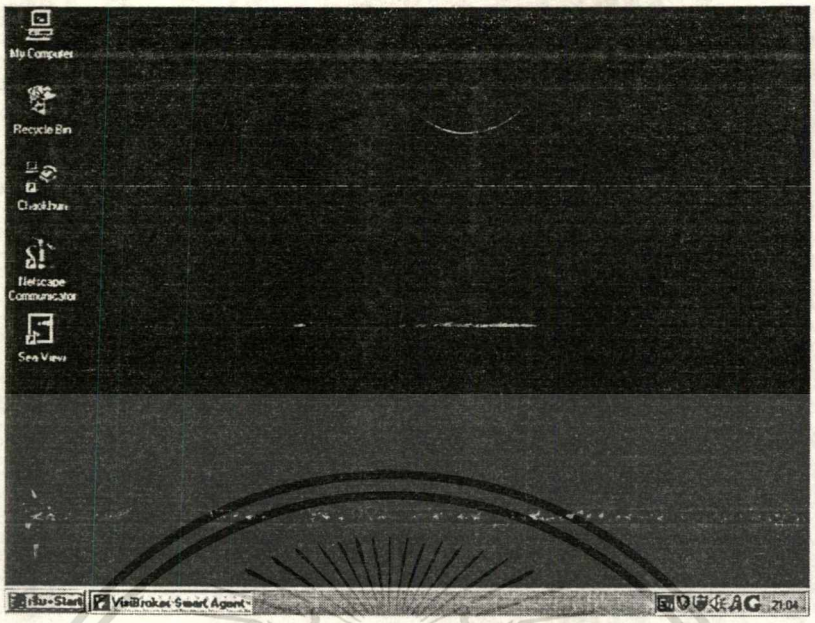
หลังจากที่ได้ทำการออกแบบและทำการสร้างองค์ประกอบที่เกี่ยวข้องสำหรับระบบงานที่ต้องการเรียบร้อยแล้ว ในขั้นตอนต่อไปก็จะเป็นการทดสอบและพัฒนาระบบงานให้มีความสมบูรณ์และใช้งานง่ายขึ้น โดยการตรวจสอบการทำงานของโปรแกรมประยุกต์ในช่วงแรกจะทำการตรวจสอบที่จุดของการทำงานจริงก่อน กล่าวคือจะทำการรันโปรแกรมโดยใช้คำสั่งที่ Command prompt เป็นอันดับแรก ซึ่งพอจะลำดับขั้นตอนได้ดังนี้

1. ทำการสร้างข้อมูลที่เป็นยอดบัญชีต่างๆ ตามที่ได้กำหนดไว้ก่อน ดังรูป



รูป 3-7 การสร้างยอดบัญชีของบุคคลต่างๆ ตามที่กำหนด

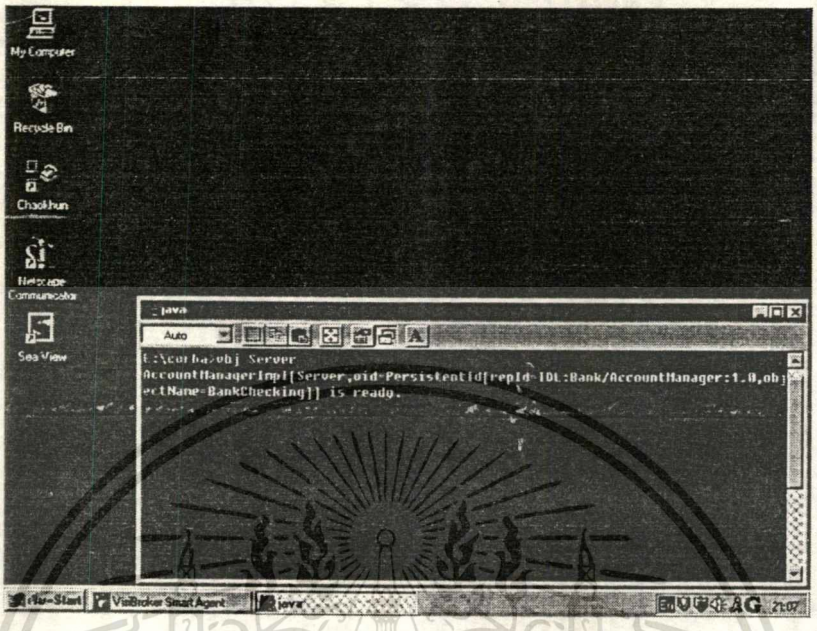
2. หลังจากที่ทำการสร้างยอดของบัญชีต่างๆ เรียบร้อยแล้ว เมื่อต้องการทดสอบระบบงานของโปรแกรมประยุกต์ให้รันโปรแกรม Smart Agent ไว้เป็น Background Process ก่อนดังรูป



รูปที่ 3-8 การรันโปรแกรม Smart Agent เป็น Background Process

Smart Agent คือโปรแกรมตัวหนึ่งซึ่งแท้ที่จริงแล้วก็คือตัว ORB นั่นเอง ในส่วนของ Visibroker นั้นได้นำมาตั้งชื่อเป็น Visibroker Smart Agent ซึ่งทำหน้าที่ในการติดต่อและลงทะเบียนออบเจ็กต์เซริฟเวอร์รวมถึงออบเจ็กต์ต่างๆ ที่กำลังให้บริการอยู่ในขณะนั้น เพื่อที่จะเรียกใช้บริการจากออบเจ็กต์ที่ไคลเอนท์ต้องการ ในส่วนของการลงทะเบียนออบเจ็กต์นั้นกระทำเพื่อที่จะเก็บข้อมูลของออบเจ็กต์เซริฟเวอร์ต่างๆ ไว้และยังรวมไปถึงการเก็บข้อมูลของออบเจ็กต์ที่เกี่ยวข้องกับออบเจ็กต์เซริฟเวอร์นั้นๆ ด้วย สำหรับข้อมูลดังกล่าวนั้นจะเป็นประโยชน์ในการที่ ORB ตัวอื่นๆ ที่ทำงานขึ้นมาในขณะนั้นจะได้สามารถรับรู้สภาพแวดล้อมในขณะนั้นว่ามีออบเจ็กต์เซริฟเวอร์อะไรที่ให้บริการอยู่บ้าง สำหรับในส่วนของ Smart Agent นั้นจะเป็นลักษณะของ Background Process ซึ่งในการทำงานทุกๆ ครั้งของโปรแกรมประยุกต์ตามมาตรฐาน CORBA จำเป็นจะต้องมีการรันโปรแกรมSmart Agent ตัวนี้เป็น Background Process ก่อนเสมอ

3. รันโปรแกรมเซริฟเวอร์โดยพิมพ์คำสั่ง vbj Server หรือ java Server ก็ได้ ซึ่งหลังจากรันเสร็จแล้วจะแสดงสถานะว่าพร้อมต่อการทำงานให้บริการดังรูป



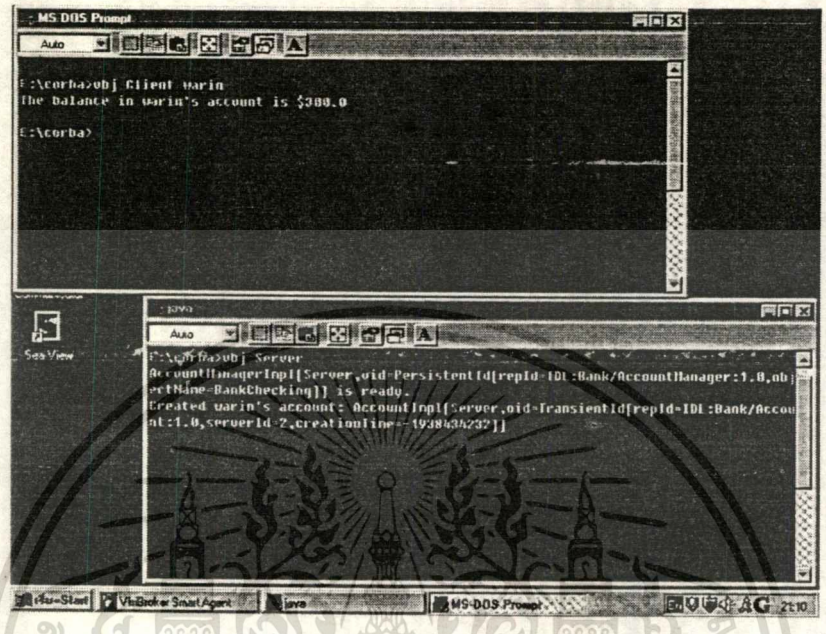
รูปที่ 3-9 การรันโปรแกรม Server เพื่อเตรียมพร้อมที่จะให้บริการ

สำหรับโปรแกรมที่เป็นตัวเซริฟเวอร์บางครั้งอาจจะเรียกว่า Object Implement Code ซึ่งหมายถึงโปรแกรมที่ทำการเตรียมตัวออบเจ็กต์เซริฟเวอร์ให้พร้อมที่จะให้บริการ โดยในส่วนตัวเซริฟเวอร์นั้นเมื่อรันเรียบร้อยแล้วบางครั้งอาจจะเรียกว่าออบเจ็กต์เซริฟเวอร์ก็ได้ จากรูปจะสังเกตได้ว่าหลังจากที่รันโปรแกรมเซริฟเวอร์แล้วจะแจ้งข้อความที่บอกให้ทราบว่าออบเจ็กต์ที่จะเป็นตัวเซริฟเวอร์นั้นพร้อมแล้ว ซึ่งในส่วนของคำสั่งที่ใช้ก็จะเป็นคำสั่งที่เป็นของ BOA ดังนี้

- orb.obj_is_ready(manager) : บอกให้รู้ว่ามีการสร้างออบเจ็กต์เซริฟเวอร์ที่ใช้ชื่อของตัวแปรออบเจ็กต์ว่า manager ให้เรียบร้อยแล้ว
- System.out.println(manager+ " is ready.") : สั่งให้พิมพ์ออกมาที่หน้าจอว่าออบเจ็กต์ดังกล่าวพร้อมแล้ว
- orb.impl_is_ready() : บอกให้รู้ว่าออบเจ็กต์เซริฟเวอร์ตัวดังกล่าวพร้อมที่จะรับการร้องขอจากไคลเอนท์แล้ว

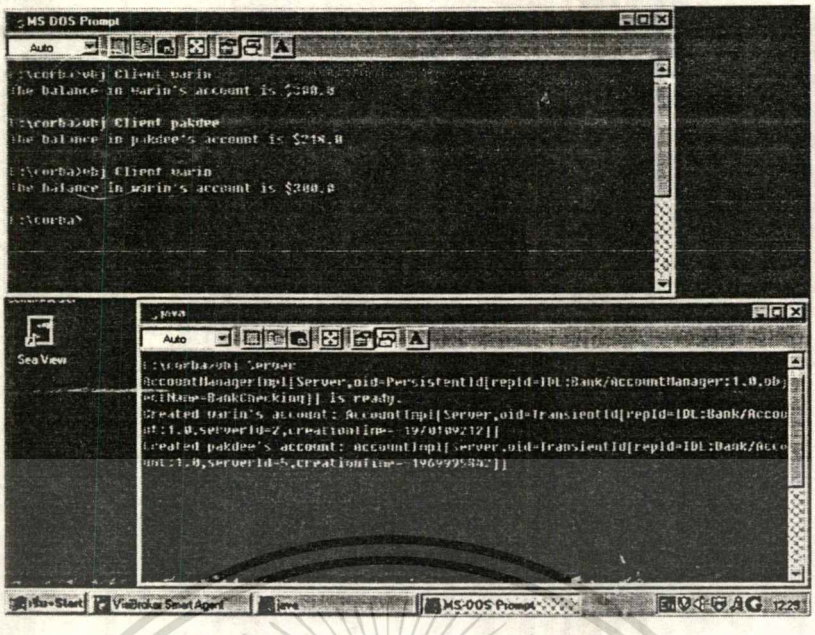
จากคำสั่งที่ให้พิมพ์ผลลัพธ์ออกมาทางหน้าจอกับผลลัพธ์จริงๆ ที่ออกมาที่หน้าจอนั้นจะเห็นว่าตัวออบเจ็กต์ที่มีชื่อตัวแปรคือ manager นั้นจะมีรายละเอียดอยู่ภายในเริ่มตั้งแต่บอกว่าออบเจ็กต์ตัวนี้เป็นออบเจ็กต์เซริฟเวอร์ นอกจากนั้นยังมีข้อมูลซึ่งเป็นส่วนที่เรียกว่า PID (Persistent ID) ด้วย ซึ่งในส่วนของ PID นี้เองที่ ORB ใช้ประโยชน์ในการเรียกใช้บริการจากออบเจ็กต์เซริฟเวอร์ เช่น ชื่อในการเรียกใช้ออบเจ็กต์ที่เรียกว่า Implement name ซึ่งในที่นี้ออบเจ็กต์เซริฟเวอร์ดังกล่าวจะมีชื่อในการเรียกใช้ออบเจ็กต์ว่า Bankchecking

4. เมื่อออบเจ็กต์เซริฟเวอร์พร้อมที่จะให้บริการแล้วก็สามารถที่จะทำการสอบถามยอดของบัญชีนั้นๆ ได้เลย โดยใช้โปรแกรมของไคลเอนท์เช่นถ้าต้องการสอบถามยอดของบัญชีที่ชื่อ warin ก็ให้พิมพ์คำสั่ง vbj Client warin ซึ่งก็จะได้ผลลัพธ์ออกมาดังรูป



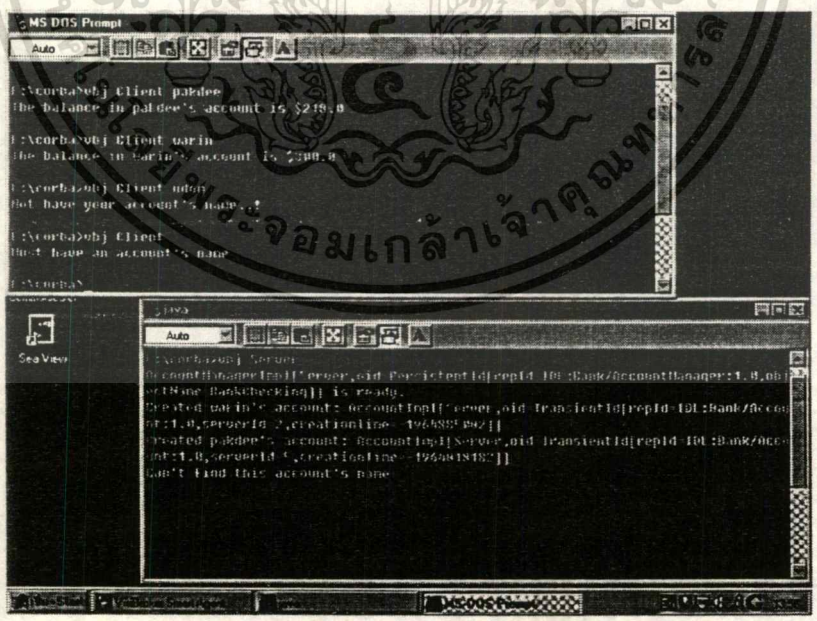
รูป 3-10 การทำงานของโปรแกรมประยุกต์ทั้งทางไคลเอนท์และเซริฟเวอร์

จากรูปจะสังเกตเห็นได้ว่าทางออบเจ็กต์เซริฟเวอร์ได้มีการสร้างออบเจ็กต์ใหม่ที่สอดคล้องกับการเรียกใช้บริการของทางไคลเอนท์ขึ้นมา และออบเจ็กต์ที่ถูกสร้างขึ้นมาใหม่นี้ก็จะมีข้อมูลจำเพาะที่แสดงถึงความเป็นออบเจ็กต์ที่ไม่เหมือนออบเจ็กต์ตัวอื่นๆ ในระบบอีกด้วย ดังจะเห็นได้จากการที่ออบเจ็กต์เซริฟเวอร์จะมีการแสดงผลลัพธ์โดยการพิมพ์ข้อมูลของออบเจ็กต์ที่สร้างขึ้นใหม่ขึ้นมา และในข้อมูลนั้นจะมีส่วนที่เรียกว่า Transient ID ซึ่งมีลักษณะคล้ายกับ Persistent ID จะแตกต่างกันก็ตรงที่ Transient ID จะเป็นข้อมูลจำเพาะของออบเจ็กต์ที่แสดงในขณะนี้เท่านั้น ในกรณีที่ไคลเอนท์มีการติดต่อกับออบเจ็กต์เซริฟเวอร์แต่ละครั้ง ตัวออบเจ็กต์เซริฟเวอร์ก็จะเรียกใช้ออบเจ็กต์ต่างๆ ที่ลงทะเบียนไว้แล้ว ซึ่งถ้าออบเจ็กต์ที่ไคลเอนท์เรียกใช้ได้ลงทะเบียนไว้แล้วก็จะทำงานขึ้นมาทันทีแต่ถ้าในกรณีที่ออบเจ็กต์ที่ไคลเอนท์ต้องการเรียกใช้ยังไม่ได้ถูกลงทะเบียน ตัวออบเจ็กต์เซริฟเวอร์ก็จะทำการสร้างออบเจ็กต์ที่ต้องการนั้นขึ้นมาดังรูปข้างล่าง



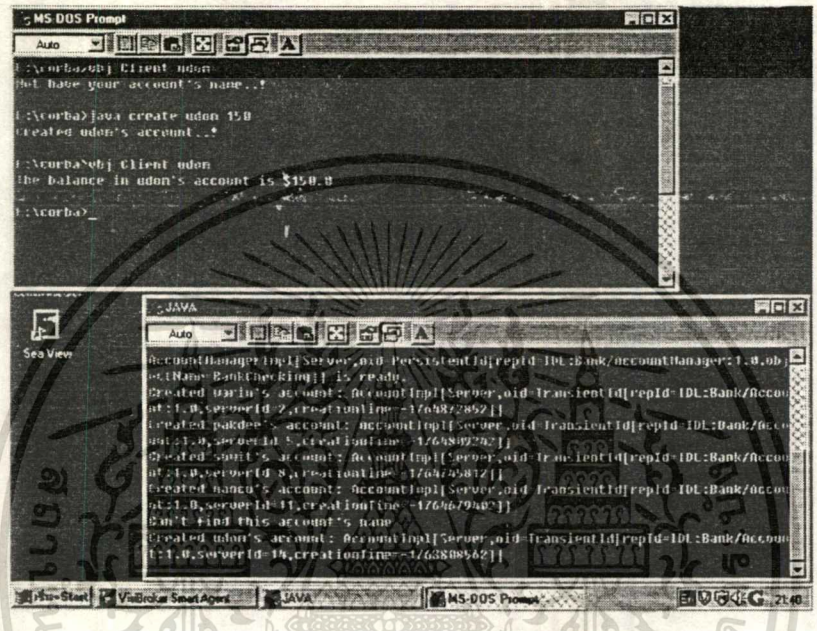
รูป 3-11 การติดต่อกับออบเจ็กต์ที่ลงทะเบียนแล้วกับออบเจ็กต์ที่ยังไม่ได้ลงทะเบียน

5. ในกรณีที่ไคลเอนท์มีการกำหนดแอทริบิวท์หรือชื่อบัญชีที่ไม่มีตรงกับความเป็นจริง กล่าวคือชื่อบัญชีดังกล่าวไม่ตรงกับข้อมูลทั้งในส่วนที่อยู่ในออบเจ็กต์และในส่วนของ Secondary Storage ตัวออบเจ็กต์เซริฟเวอร์ก็จะทำการตรวจสอบให้แต่เมื่อไม่พบข้อมูลที่ตรงกับที่ไคลเอนท์ต้องการก็จะทำการแจ้งให้ไคลเอนท์ทราบว่าข้อมูลของบัญชีนี้ไม่มีอยู่จริงดังรูป



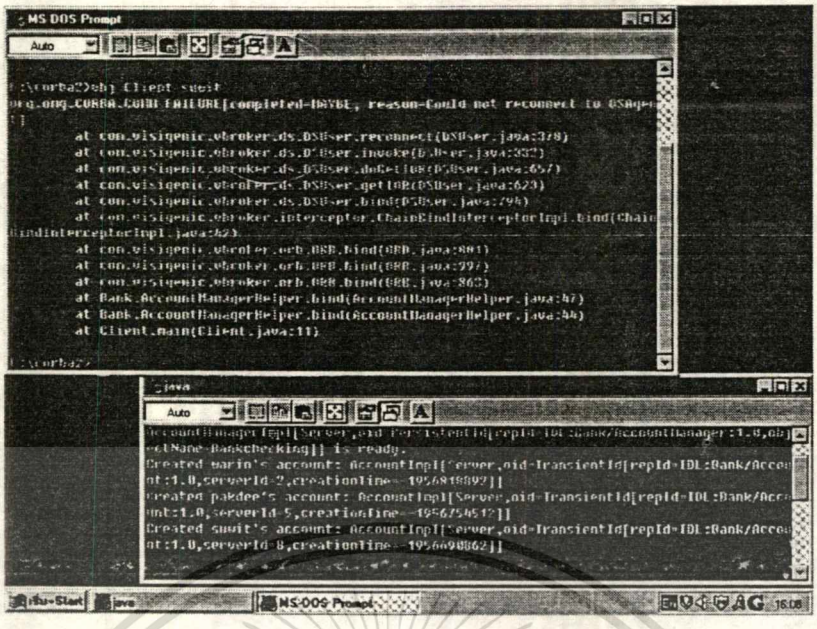
รูป 3-12 ผลลัพธ์เนื่องจากการที่ระบุแอทริบิวท์ไม่ครบหรือระบุผิดพลาด

6. หลังจากที่ได้มีการสอบถามยอดบัญชีของบุคคลที่ไม่มีข้อมูลอยู่จริงแล้ว ถ้ามีการสร้างยอดของบัญชีนั้นๆ เข้ามาในภายหลังแล้วก็สามารถที่จะทำการสอบถามยอดได้อีกโดยผ่านทางไคลเอนท์เช่นเดิม ซึ่งจะสังเกตเห็นว่าในครั้งหลังนี้ออบเจ็กต์เซริฟเวอร์จะทำการสร้างออบเจ็กต์ของข้อมูลดังกล่าวขึ้นมาได้ ทั้งนี้เนื่องจากเมื่อมีการเรียกใช้โปรแกรม Create แล้วก็จะมีข้อมูลของยอดบัญชีดังกล่าวอยู่ใน Secondary Storage ของทางเครื่องเซริฟเวอร์ทันที ดังนั้นเมื่อออบเจ็กต์เซริฟเวอร์ตรวจพบก็สามารถที่จะนำมาสร้างเป็นออบเจ็กต์ได้ดังรูป



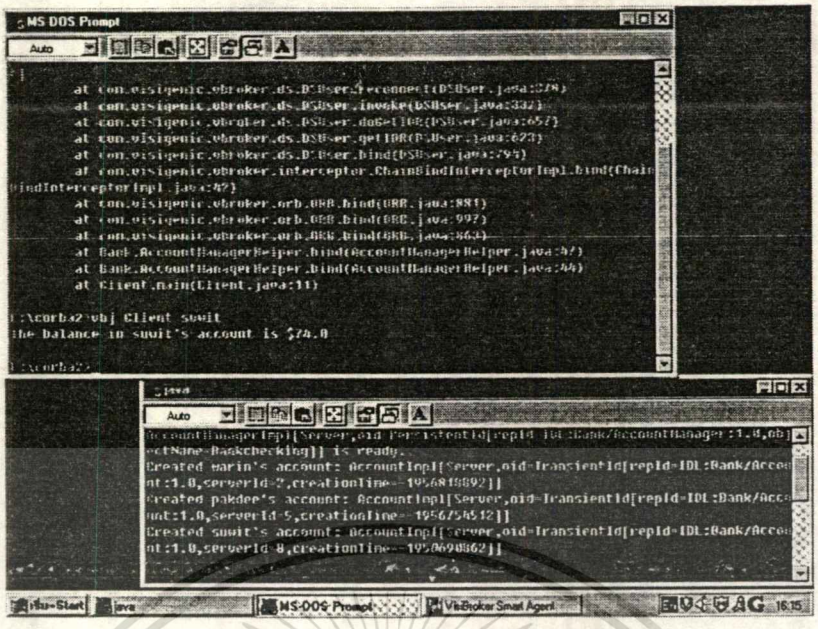
รูป 3-13 การถามยอดบัญชีของบุคคลหนึ่งก่อนและหลังการสร้างยอดบัญชีนั้นๆ

7. ในกรณีที่ ORB ไม่ได้ทำงานอยู่หรือมีการยุติการทำงานไปในระหว่างที่มีการทำงานของโปรแกรมประยุกต์อยู่นั้น ถ้าหากมีการเรียกใช้งานออบเจ็กต์จากไคลเอนท์เข้ามาก็จะมีการแจ้งให้ทางไคลเอนท์ทราบว่า ORB ที่เป็นตัวกลางติดต่อนั้นไม่ได้ทำงานอยู่แล้ว โดยจะเป็นลักษณะของการเกิดความผิดพลาดอันเนื่องมาจากการหาตัว ORB ไม่พบ ส่วนทางออบเจ็กต์เซริฟเวอร์ก็จะยังคงพร้อมให้บริการอยู่เหมือนเดิมหากแต่จะไม่ได้รับรู้ถึงการร้องขอของไคลเอนท์ ดังรูป



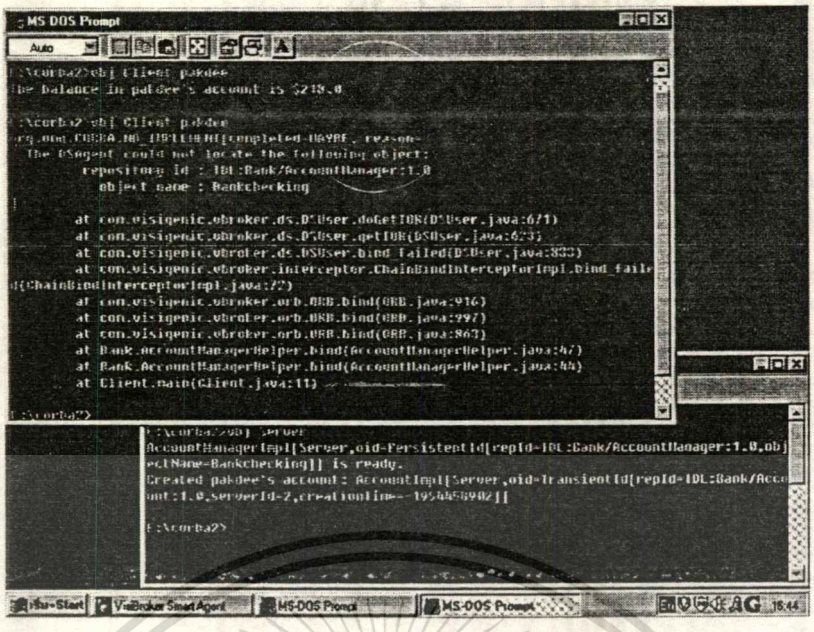
รูป 3-14 การเกิดความผิดพลาด (Error) อันเนื่องมาจาก ORB หยุดทำงาน

สำหรับตัว ORB นั้นตามที่ได้กล่าวมาแล้วว่าจะมีการลงทะเบียนออบเจ็กต์เป็นแบบ Dynamic ดังนั้นถึงแม้ว่า ORB จะถูกปิดไปแล้วแต่ถ้าเกิดมีการสั่งงานให้ ORB ทำงานขึ้นมาอีกครั้งหนึ่งก็สามารถที่จะทำการลงทะเบียนออบเจ็กต์ที่มีอยู่ในระบบได้อีกเช่นเดิม แต่ในจุดนี้จะมีความแตกต่างกันบ้างระหว่างระบบปฏิบัติการ Windows '95 กับ Windows NT ตรงจุดที่ถ้าเป็น Windows '95 แล้วจะสามารถทำการลงทะเบียนออบเจ็กต์และทำงานได้ทันทีแต่ถ้าเป็น Windows NT แล้วจะต้องมีการสั่งการให้ออบเจ็กต์เซริฟเวอร์ทำการสร้างออบเจ็กต์ใหม่ขึ้นมาเสียก่อนจึงจะมีการลงทะเบียนออบเจ็กต์ตัวอื่นๆ ตามมา แล้วหลังจากนั้นจึงจะสามารถติดต่อกับออบเจ็กต์เก่าที่ยังคงสภาพความเป็นออบเจ็กต์อยู่ในระบบได้ อย่างไรก็ดีจะสังเกตเห็นว่าถ้ามีการสั่งงานให้ ORB ทำงานขึ้นมาอีกครั้งหนึ่งแล้วก็สามารถที่จะทำงานกับออบเจ็กต์เดิมได้ต่อไปดังรูป



รูป 3-15 การตอบสนองของออบเจ็กต์หลังจาก ORB หยุดทำงานชั่วขณะ

8. สำหรับออบเจ็กต์ทั้งหมดที่ให้บริการอยู่นั้นเป็นลักษณะของ Dynamic ซึ่งเกิดจากการสร้างและเรียกใช้โดยผ่านตัวออบเจ็กต์เซริฟเวอร์ ดังนั้นเมื่อออบเจ็กต์เซริฟเวอร์ได้ยุติการทำงานลง ออบเจ็กต์ต่างๆ เหล่านั้นก็จะเหมือนกับหยุดการทำงานตามไปด้วย นอกจากนั้นตัว ORB เองก็ได้มีการลงทะเบียนออบเจ็กต์เซริฟเวอร์ไว้ด้วยซึ่งเมื่อออบเจ็กต์เซริฟเวอร์ได้ยุติการทำงานลงแล้วข้อมูลการลงทะเบียนของออบเจ็กต์เซริฟเวอร์นั้นก็จะถูกทำลายไปด้วย ซึ่งเมื่อไคลเอนท์มีการเรียกใช้ออบเจ็กต์เหล่านั้นแล้วการร้องขอของไคลเอนท์ก็จะผ่านไปยัง ORB แต่เมื่อ ORB ตรวจสอบแล้วเห็นว่าออบเจ็กต์เซริฟเวอร์ที่ดูแลออบเจ็กต์เหล่านั้นได้หยุดการทำงานลงแล้วก็จะทำการแจ้งผลกลับมายังไคลเอนท์ดังรูป



รูป 3-16 การทำงานของระบบในกรณีที่ออบเจ็กต์เซริฟเวอร์ยุติการทำงาน

การพัฒนาสู่ Web Browser

หลังจากได้มีการทดสอบเป็นที่เรียบร้อยแล้วก็จะนำเอาโปรแกรมของทางไคลเอนท์ไปสร้างรูปแบบให้เกิดความสวยงามขึ้น โดยในการนี้ได้ทำการสร้างเป็นลักษณะของ Applet เพื่อที่จะได้เอื้ออำนวยต่อการทำงานบน Web Browser ต่อไป โดยในการนี้จะมีไฟล์ที่สร้างเพิ่มขึ้นใหม่เพื่อเอื้ออำนวยต่อการทำงานบน Web Browser ดังนี้

- **ClientApplet.java** : เป็นไฟล์สร้างขึ้นเพื่อที่จะทำให้โปรแกรมของทางไคลเอนท์มีการทำงานในแบบของ Java Applet ซึ่งจะได้นำไปใช้ในการทำงานบน Web Browser ต่อไป โดยในไฟล์ดังกล่าวมีรูปแบบดังต่อไปนี้

```
// ClientApplet.java


import java.awt.*;
import java.lang.*;
import org.omg.CORBA.ORB;


public class ClientApplet extends java.applet.Applet {


  private TextField _nameField, _balanceField;
  private TextArea _statusArea;
```

```
private Button _checkBalance;
private Bank.AccountManager _manager;
private float nonbalance;
String sumtext="";

public void init() {
 // This GUI uses a 2 by 2 grid of widgets.
 setLayout(new FlowLayout(FlowLayout.LEFT));
//  setLayout(new GridLayout(4, 2, 5, 5));
 // Add the four widgets.
 add(new Label("Account Name  "));
 add(_nameField = new TextField(10));
 add(new Label("                                    "));

 add(_checkBalance = new Button("Check Balance "));
 add(_balanceField = new TextField(10));
 add(new Label("                                    "));

 // make the balance text field non-editable.
 _balanceField.setEditable(false);

 add(new Label("Status   "));
 add(_statusArea = new TextArea("",10,60));
 _statusArea.setEditable(false);

 // Initialize the ORB.
 org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
  // Locate an account manager.
  _manager = Bank.AccountManagerHelper.bind(orb, "BankChecking");

}
 public boolean action(Event ev, Object arg) {
  if(ev.target == _checkBalance) {
   // Request the account manager to open a named account.
```

```
            // Get the account name from the name text widget.
            Bank.Account account = _manager.open(_nameField.getText());
            nonbalance=account.balance();
            if (nonbalance!=-1.0)
             {
             // Set the balance text widget to the account's balance.
             _balanceField.setText(Float.toString(account.balance()));
                    sumtext=_nameField.getText()+"'s account have balance is "+Float.toString
(account.balance())+" $ ... \n"+sumtext;
             }
            else
             {
             _balanceField.setText("          ");
             sumtext=_nameField.getText()+" have not in Account.  Try again....\n"+sumtext;
            }
           _statusArea.setText(sumtext);
           return true;
          }
          return false;
        }
      }
```

— **CreateApplet.java : สร้างขึ้นโดยมีจุดประสงค์เดียวกันกับไฟล์ ClientApplet.java เพียงแต่ว่าไฟล์นี้ใช้สำหรับโปรแกรม Create เท่านั้น ซึ่งก็จะถูกใช้เพื่อสร้างรูปแบบของ Web Browser ที่ทำการเปลี่ยนแปลงยอดบัญชีนั่นเอง โดยมีรูปแบบของการเขียนดังนี้**

```
// CreateApplet.java for Admin

import java.lang.*;
import java.io.*;
import java.awt.*;
// import java.security.*;

public class CreateApplet extends java.applet.Applet {
```

```
    private TextField _nameField, _balanceField;
    private TextArea _statusArea;
    private Button _createBalance;
    private Bank.AccountManager _manager;
    private String  name,sumtext="";


  public void init() {
    // This GUI uses a 3 by 3 grid of widgets.
    setLayout(new FlowLayout(FlowLayout.LEFT));
//    setLayout(new GridLayout(3, 2, 5, 5));
    // Add the four widgets.
    add(new Label("Account Name   "));
    add(_nameField = new TextField(10));
    add(new Label("                                  "));

    add(new Label("Create Balance "));
    add(_balanceField = new TextField(10));
    add(new Label("                                  "));

    add(_createBalance = new Button("Create Balance"));
    add(new Label("                                                                  "));

    add(new Label("Status        "));
    add(_statusArea = new TextArea("",10,60));
    _statusArea.setEditable(false);

   // Initialize the ORB.
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
     // Locate an account manager.
      _manager = Bank.AccountManagerHelper.bind(orb, "BankChecking");
  }
```

```
public boolean action(Event ev, Object o) {
 if(ev.target == _createBalance) {
  try {
  // Get the account name from the name text widget.
  name=(_nameField.getText());
  String fname = name+".name";
  String fbal = name+".balance";
   int balance=Integer.parseInt(_balanceField.getText());

   //Save the account to File.

   FileOutputStream fnameout = new FileOutputStream(fname);
   FileOutputStream fbalout = new FileOutputStream(fbal);
   DataOutputStream nameout = new DataOutputStream(fnameout);
   DataOutputStream balout = new DataOutputStream(fbalout);
   nameout.writeBytes(name);
   balout.writeFloat(balance);
   fnameout.close();
   fbalout.close();

                sumtext=_nameField.getText()+"'s account created balance was
"+_balanceField.getText()+".00 $ \n"+sumtext;
   showStatus("Ready..");
  }
  catch (IOException e){
                sumtext=_nameField.getText()+"'s account can't create balance
"+_balanceField.getText()+" \n"+sumtext;
   showStatus("Exception: "+e.toString());
  }
  catch(NumberFormatException e){
                sumtext=_nameField.getText()+"'s account can't create balance
"+_balanceField.getText()+" \n"+sumtext;
   System.out.print("Exception Number Format Error. Try again ...\n");
            showStatus("Exception: "+e.toString());
```

```
            }
            _statusArea.setText(sumtext);
                _nameField.setText("");
                _balanceField.setText("");

                return true;
            }
          return false;
        }
       }
```

— ClientApplet.html : เป็นไฟล์ที่สร้างขึ้นเพื่อเรียกใช้ไฟล์ ClientApplet.java เพื่อที่จะนำไปทำงานบน Web Browser เป็นอันดับสุดท้าย ลักษณะของไฟล์ดังกล่าวก็จะเหมือนกับไฟล์ที่สร้างขึ้นเพื่อทำงานบน Web Browser โดยทั่วไปเนื่องจากเป็นเพียงการกำหนดให้นำ Java Byte Code ไปทำงานบนโปรแกรม Web Browser ต่างๆ เท่านั้น รูปแบบของไฟล์จะเป็นดังนี้

```
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
  <META NAME="GENERATOR" CONTENT="Mozilla/4.04 [en] (WinNT; I) [Netscape]">
</HEAD>
<BODY>

<CENTER>
<H1>
DISTRIBUTED OBJECT CLIENT/SERVER</H1></CENTER>

<CENTER>
<H1>
USING CORBA</H1></CENTER>

<CENTER><IMG SRC="hline.gif" HEIGHT=5 WIDTH=576></CENTER>
```

```
<P> 
<CENTER><APPLET
  code=ClientApplet.class  width=450 height=250>
<PARAM  name=org.omg.CORBA.ORBClass  value=com.visigenic.vbroker.orb.ORB>
</CENTER>
//<PARAM  name=org.omg.CORBA.ORBClass  value=c:\CorbaJavaBook\Classes\ORB>
</CENTER>

<CENTER></APPLET></CENTER>

<CENTER> </CENTER>

<CENTER><IMG SRC="ball_red.gif" HEIGHT=14 WIDTH=14><B><FONT SIZE=+3>This
is examples applet checking bank from Client to Server to use CORBA</FONT></B>
</CENTER>

<CENTER> </CENTER>

<CENTER>
<H2>
You are probably not running a Java enabled browser. Please use a Java
enabled browser (or enable your browser for Java) to view this applet...</H2></CENTER>

<HR>
</BODY>
</HTML>
```

- CreateApplet.html : ใช้สำหรับเรียก Java Byte Code ไปทำงานบนโปรแกรม Web Browser โดยทั่วๆ ไป โดยมีลักษณะการเขียนโปรแกรมดังนี้

```
<HTML>
<HEAD>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
  <META NAME="GENERATOR" CONTENT="Mozilla/4.04 [en] (WinNT; I) [Netscape]">
```

```
<TITLE>CreateApplet</TITLE>
</HEAD>
<BODY>

<CENTER>
<H1>
DISTRIBUTED OBJECT CLIENT/SERVER</H1></CENTER>

<CENTER>
<H1>
USING CORBA</H1></CENTER>

<CENTER><IMG SRC="hline.gif" HEIGHT=5 WIDTH=576></CENTER>

<P> 
<CENTER><APPLET
  code=CreateApplet.class
  width=460 height=290><PARAM name=org.omg.CORBA.ORBClass
value=com.visigenic.vbroker.orb.ORB></CENTER>

<CENTER></APPLET></CENTER>

<CENTER> </CENTER>

<CENTER><IMG SRC="ball_red.gif" HEIGHT=14 WIDTH=14><B><FONT SIZE=+3>This
is examples applet create account bank from Client to Server to use CORBA</FONT></B>
</CENTER>

<CENTER><B><FONT SIZE=+3></FONT></B> </CENTER>

<CENTER>
<H2>
You are probably not running a Java enabled browser. Please use a Java
```

enabled browser (or enable your browser for Java) to view this applet...</H2></CENTER>

<CENTER></CENTER>

<HR>

</BODY>

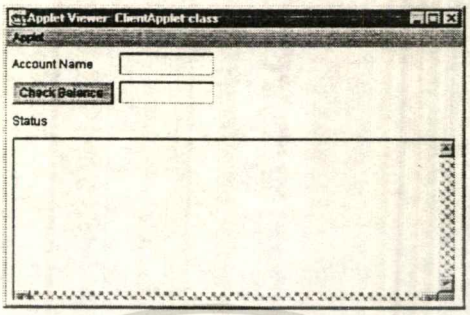
</HTML>

หลังจากที่ทำการสร้างไฟล์ทั้งหมดเหล่านี้ขึ้นมาแล้ว ในขั้นตอนต่อไปก็จะเป็นการทดสอบสิ่งที่ได้ทำการพัฒนาแล้ว ซึ่งในขั้นตอนต่อไปก็จะทดสอบในลักษณะการทำงานในแบบของ Java Applet โดยมีขั้นตอนในการทดสอบดังนี้

1. สำหรับในการทำงานบน Web Browser นั้นจำเป็นจะต้องทำการรันโปรแกรมอีกตัวหนึ่งเพื่อสนับสนุนการทำงานในลักษณะของ Applet นั่นคือโปรแกรม GateKeeper ซึ่งลักษณะการทำงานหลังจากที่ทำการรันโปรแกรม GateKeeper แล้วจะมีลักษณะดังรูป

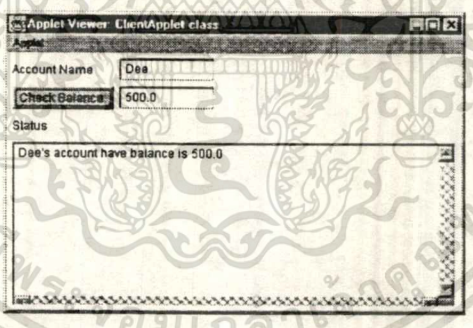
รูป 3-17 ลักษณะการแสดงผลหลังจากทำการรันโปรแกรม GateKeeper

2. เมื่อทำการรันโปรแกรมในการสนับสนุนแล้วก็สามารถที่จะทำการรันโปรแกรมของไคลเอนท์ในรูปแบบของ Applet ได้ และเพื่อความง่ายในการทำความเข้าใจจึงได้รันโดยใช้โปรแกรม AppletViewer ซึ่งจะได้ผลดังรูป



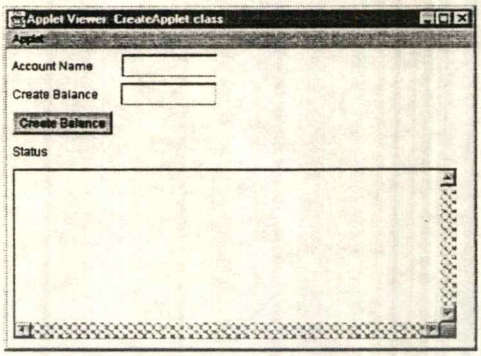
รูป 3-18 ลักษณะการทำงานของ ClientApplet บนโปรแกรม AppletViewer

จากรูปจะมีช่องสำหรับป้อนชื่อที่เป็นเจ้าของบัญชีและปุ่ม Check Balance เพื่อสั่งให้ตรวจสอบยอดเงินในบัญชี หลังจากใส่ชื่อและกดปุ่มนี้แล้วตัวไคลเอนท์ก็จะเรียกใช้บริการจากออบเจ็กต์เซริฟเวอร์ผ่านไปทาง ORB และเมื่อออบเจ็กต์เซริฟเวอร์ทำงานเสร็จก็จะส่งผลกลับมาให้ไคลเอนท์ได้แสดงออกมาดังรูปข้างล่างนั่นเอง



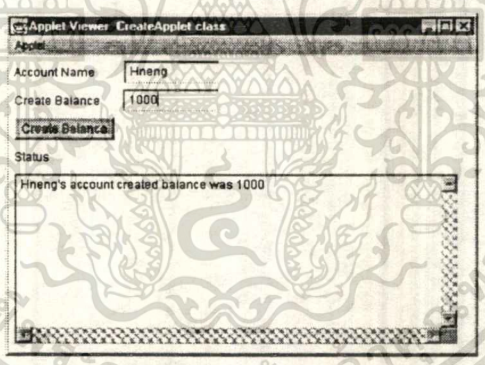
รูป 3-19 การแสดงผลของไคลเอนท์บนหน้าจอ AppletViewer

3. นอกจากนั้นยังได้มีการสร้างโปรแกรม Create ให้เป็นแบบ Applet ด้วยเพื่อให้เกิดความสวยงามในการใช้งานและให้มีลักษณะเหมือนกับโปรแกรมของไคลเอนท์



รูป 3-20 ลักษณะของโปรแกรม Create เมื่อเป็นแบบ Applet

สำหรับตัวโปรแกรม Create นั้นจะมีลักษณะการทำงานที่ไม่ยุ่งยากเท่าไรนัก ซึ่งจะมีกรอบสำหรับป้อนข้อมูลของชื่อบัญชีและยอดเงินของบัญชี เมื่อป้อนเข้าไปครบแล้วก็ให้คลิกไปที่ปุ่ม Create Balance ซึ่งโปรแกรมก็จะทำการสร้างยอดของบัญชีให้และแสดงผลออกมาดังรูป



รูป 3-21 การแสดงผลของโปรแกรม Create ในแบบ Applet

บทที่ 4 สรุปและวิเคราะห์การทำงาน

จากการที่ได้ศึกษาถึงมาตรฐาน CORBA และการสร้างโปรแกรมประยุกต์ที่ทำงานตามมาตรฐาน CORBA ทำให้เห็นว่าความสามารถในการพัฒนาโปรแกรมประยุกต์ในรูปแบบมาตรฐาน CORBA นั้นมีความสะดวกและกะทัดรัดมาก โดยจะเห็นว่าสามารถที่จะสร้างโครงสร้างการติดต่อและการเรียกใช้ออบเจ็กต์ได้ด้วยไฟล์เพียงตัวเดียว นอกจากนั้นการทำงานของโปรแกรมประยุกต์นั้นก็ยังมีประสิทธิภาพที่ดีรองรับการทำงานในขอบเขตที่กว้างออกไปได้ แต่สำหรับการทำงานตามตัวอย่างนี้คงจะจำกัดอยู่เฉพาะคุณสมบัติเพียงส่วนเดียวของมาตรฐาน CORBA เท่านั้น ซึ่งจากการทำงานที่ผ่านมาพอที่จะทำให้เห็นว่าการสร้างโปรแกรมประยุกต์ภายใต้มาตรฐาน CORBA นั้นมีทั้งข้อดีและข้อเสีย ซึ่งพอที่จะสรุปได้ดังนี้

ข้อดีของ CORBA

จากการทดลองจะเห็นว่าข้อดีของการสร้างโปรแกรมประยุกต์ภายใต้มาตรฐาน CORBA นั้นมีอยู่ไม่น้อยซึ่งพอจะแสดงเป็นข้อๆ ได้ดังนี้

- ลดช่องว่างในเรื่องของการติดต่อข้ามระบบเนื่องจากการกำหนดโครงสร้างการทำงานของโปรแกรมโดยอาศัยตัวกลางการติดต่อที่ชื่อ ORB (Object Request Broker) ซึ่งเป็นวิธีที่ทำให้การติดต่อกันระหว่างโปรแกรมต่างๆ ที่อยู่บนระบบเครือข่ายคอมพิวเตอร์ที่มีอยู่ในปัจจุบันนั้นสามารถลดภาระของนักพัฒนาระบบให้ลดลงอยู่เฉพาะในกรอบของการพัฒนาตัวโปรแกรมประยุกต์เท่านั้น
- เนื่องจาก CORBA ใช้แนวความคิดการออกแบบโปรแกรมในลักษณะของ Object Oriented จึงทำให้การพัฒนาโปรแกรมมีขีดความสามารถมากขึ้น ทำการสร้างโปรแกรมประยุกต์ที่มีขนาดใหญ่ได้ และสามารถใช้งานกับภาษาของโปรแกรมในยุคใหม่ๆ ได้ เช่น C++, Java เป็นต้น
- การใช้ภาษา IDL (Interface Definition Language) ในการกำหนดโครงร่างในการทำงานทำให้เกิดความสะดวกและเกิดความชัดเจนในการทำงานหรือกำหนดจุดประสงค์ของการทำงานได้ นอกจากนี้การใช้ภาษา IDL ยังทำให้ผู้ที่จะทำการสร้างโปรแกรมประยุกต์ที่มีลักษณะการทำงานแบบเดียวกันนี้สามารถนำไปใช้งานได้ทันที เนื่องจาก IDL เป็นภาษาที่มีความเป็นกลางไม่ขึ้นกลับภาษาใดภาษาหนึ่ง แต่กลับสามารถที่จะนำไปใช้กับภาษาได้เกือบทุกภาษา
- ลดข้อจำกัดสำหรับนักพัฒนาโปรแกรมที่มีความถนัดในการใช้งานโปรแกรมภาษาใดภาษาหนึ่ง เนื่องจากโครงสร้างของ CORBA สามารถรองรับกับการทำงานได้กับโปรแกรมภาษาเกือบทุกภาษาที่มีความสำคัญทางด้านธุรกิจปัจจุบัน ฉนั้นถึงแม้ว่านักพัฒนาโปรแกรมจะมีความถนัดกับโปรแกรมภาษาเพียงภาษาเดียวก็สามารถที่จะสร้างโปรแกรมประยุกต์ที่ความเป็นสากลสามารถติดต่อกับโปรแกรมประยุกต์ของภาษาอื่นๆ ได้ โดยที่ไม่เกิดความผิดพลาดแต่อย่างใด
- เพิ่มความสะดวกในการเขียนโปรแกรม เนื่องจากการที่โปรแกรมประยุกต์ต่างๆ ภายใต้มาตรฐาน CORBA นั้นจะต้องมีการทำงานผ่านตัวกลางการติดต่อที่ชื่อ ORB อยู่แล้ว โดยที่ ORB จะมีการเก็บข้อมูลของออบเจ็กต์เซริฟเวอร์และออบเจ็กต์ต่างๆ ที่ให้บริการอยู่ดังนั้นในการเขียนโปรแกรมของไคลเอนท์ก็จึงไม่มีความจำเป็นที่จะต้องทราบหรือระบุเป้าหมายปลายทางของออบเจ็กต์ต่างๆ เหล่านั้น

จึงทำให้ลักษณะการเขียนโปรแกรมตามมาตรฐาน CORBA ซึ่งเป็นโปรแกรมที่ทำงานบนเครือข่ายคอมพิวเตอร์มีความใกล้เคียงกับลักษณะการเขียนโปรแกรมของโปรแกรมประยุกต์ต่างๆ ที่ทำงานเป็นแบบเครื่องเดียวหรือ Stand Alone

- งานบริการต่างๆ ที่จำเป็นสำหรับโปรแกรมที่ทำงานในระบบเครือข่ายคอมพิวเตอร์ทั้งในระดับเล็กและใหญ่เช่นการรักษาความปลอดภัยของระบบถูกดูแลโดย ORB จึงทำให้การทำงานหรือสร้างโปรแกรมประยุกต์ทั้งหมดเหล่านั้นลดความยุ่งยากลงไปได้มาก

ข้อเสียของ CORBA

ถึงแม้ว่า CORBA จะมีข้อดีในการทำงานมากมายแต่ก็ไม่ได้หมายความว่าจะไม่มีข้อเสียใดๆ เลย จากการทดลองจะเห็นได้ชัดเจนว่า CORBA ยังคงมีข้อเสียอีกหลายอย่างซึ่งทำให้มาตรฐานตัวนี้ยังคงไม่เป็นที่พึงพอใจของนักพัฒนาโปรแกรมประยุกต์อีกหลายคน สำหรับข้อเสียของ CORBA สามารถจะแจกแจงออกเป็นข้อๆ ได้ดังนี้

- เนื่องจากการใช้แหล่งข้อมูลที่ได้จากการลงทะเบียนของ ORB ในการติดต่อเพื่อเรียกใช้ออบเจ็กต์ต่างๆ แต่ข้อมูลการลงทะเบียนเหล่านั้นกลับไม่ได้มีการระบุถึงตำแหน่งของออบเจ็กต์แต่ละตัว จึงทำให้เสมือนกับว่าในแต่ละครั้งที่มีการเรียกใช้ออบเจ็กต์ ORB จะต้องมีการค้นหาออบเจ็กต์เหล่านั้นใหม่ ซึ่งทำให้เกิดความสูญเสีย (Overhead) สูงมาก ซึ่งในส่วนของความสูญเสียนี้อาจจะมากหรือน้อยขึ้นอยู่กับโครงสร้างของระบบที่โปรแกรมประยุกต์นั้นๆ ทำงานอยู่ด้วย
- ในส่วนของข้อดีจะเห็นว่าผู้กำหนดมาตรฐาน CORBA ได้พยายามที่จะทำให้ CORBA เป็นมาตรฐานที่ลดภาระต่างๆ ของการโปรแกรมลง แต่ด้วยเหตุนี้เองทำให้เพิ่มภาระในส่วนของโปรแกรมที่เข้ามาเสริมการทำงาน ดังนั้นการทำงานของโปรแกรมประยุกต์ตามมาตรฐาน CORBA มักจะต้องมีการเรียกใช้โปรแกรมเสริมต่างๆ เป็นจำนวนมากทำให้เสียทรัพยากร (Resource) ไปเป็นจำนวนมากด้วย โดยจะเห็นว่าในการทำงานของโปรแกรมทางไคลเอนท์ในลักษณะของ Applet ก็จะต้องมีการเรียกใช้โปรแกรมเสริมที่ชื่อ GateKeeper เพื่อเสริมการทำงานเป็นต้น

วิเคราะห์การทำงาน

สำหรับข้อดีและข้อเสียของ CORBA ที่ได้สรุปไปแล้วนั้นสามารถกล่าวได้ว่าเป็นเพียงข้อดีและข้อเสียที่เห็นได้ชัดเจนจากการทำงาน แต่ยังคงมีประเด็นอีกมากที่อาจเป็นได้ทั้งข้อดีและข้อเสียสำหรับมาตรฐาน CORBA ซึ่งพอสรุปได้ดังต่อไปนี้

1. มาตรฐาน CORBA เป็นมาตรฐานที่ผู้กำหนดพยายามที่จะสร้างให้เกิดความครอบคลุมและในขณะเดียวกันก็พยายามที่จะพลักดันให้เป็นมาตรฐานของการสร้างโปรแกรมประยุกต์ในระบบเครือข่ายคอมพิวเตอร์ทั้งระบบเลย จึงทำให้ข้อกำหนดของมาตรฐานค่อนข้างที่จะมาก นอกจากนี้สมาชิกของกลุ่มที่กำหนดมาตรฐานนี้คือ OMG (Object Management Group) ก็ได้มีการนำเอามาตรฐานนี้ไปใช้กับ

เครื่องมือของตนเองเป็นจำนวนมาก แต่การนำเอาไปอ้างอิงกับเครื่องมือของตนนั้นก็ยังคงมีรูปแบบที่แตกต่างกันออกไปอีก ซึ่งทำให้เกิดความรู้สึกว่ายังไม่เกิดความเป็นอันหนึ่งอันเดียวกันอยู่

2. ในขณะทำการทดลองได้พบว่าการตอบสนองของออบเจ็กต์เซริฟเวอร์ที่ทำงานบนระบบปฏิบัติการ Windows '95 กับ Windows NT มีช่วงเวลาที่แตกต่างกันมาก โดยบน Windows '95 ให้ผลตอบสนองกลับมายังไคลเอนท์ใช้เวลาเกือบ 1 นาที ซึ่งทำให้เกิดข้อสงสัยว่ามาตรฐาน CORBA นี้จะไม่ได้คำนึงถึงระบบปฏิบัติการที่มิได้มีความสามารถในการให้บริการหรือเป็นเซริฟเวอร์โดยตรง นอกจากนี้ในบางช่วงของการทำงานยังพบว่าผลตอบสนองจากระบบปฏิบัติการ Windows '95 กับ Windows NT มีความแตกต่างกันด้วยเช่นกัน
3. มาตรฐาน CORBA โดยมากแล้วมักจะมีลักษณะเป็นนามธรรมมากกว่ารูปธรรมกล่าวคือ การทำงานส่วนใหญ่ภายใต้มาตรฐาน CORBA จะไม่สามารถเห็นได้ด้วยตาแต่จะรับรู้ได้เพียงว่าได้เกิดพฤติกรรมตามทฤษฎีดังกล่าวแล้วเท่านั้น เช่นการให้บริการของ ORB ก็จะไม่สามารถเห็นการทำงานของ ORB ได้ แต่จะสามารถแสดงได้ว่า ORB นั้นได้ทำงานจริงตามทฤษฎีแล้วเท่านั้น
4. การสร้างโปรแกรมประยุกต์ตามมาตรฐาน CORBA ให้มีความสมบูรณ์พร้อมที่จะใช้งานบน Web Browser ได้นั้นจะมีลักษณะที่คล้ายคลึงกัน Java Byte Code มาก กล่าวคือจะทำการสร้างตัวโปรแกรมให้เรียบร้อยในลักษณะทั่วไปก่อนจากนั้นก็จะใช้วิธีการนำตัวโปรแกรมที่ใช้งานได้ทำการโหลด (Load) เข้าไปยังตัว Web Browser ให้ไปรันบน Web Browser อีกทีหนึ่ง
5. การที่จะสร้างโปรแกรมประยุกต์ขึ้นมาตามมาตรฐานนี้สักตัวหนึ่งนั้น จำเป็นอย่างยิ่งที่จะต้องรู้ถึงรูปแบบการทำงานของโปรแกรมนั้นอย่างดีตั้งแต่ต้นแล้ว ซึ่งแตกต่างจากการโปรแกรมในรูปแบบเดิมๆ ที่อาจมีการเสริมการทำงานเข้าไปในภายหลังได้ ทั้งนี้เนื่องจากรูปแบบการติดต่อของโปรแกรมนั้นได้ถูกกำหนดโดยตัว IDL ตั้งแต่ต้นแล้ว ดังนั้นถ้ามีการเพิ่มเติมในภายหลังก็อาจจะต้องมีการแก้ไข IDL ใหม่และต้องทำการคอมไพล์ใหม่ด้วย
6. ภาษา IDL เป็นภาษาที่นับได้ว่าดีมากทั้งนี้เนื่องมาจากการที่ภาษา IDL นี้สามารถที่จะทำการคอมไพล์เป็นโปรแกรมในภาษาต่างๆ ได้มากมายขึ้นอยู่กับคอมไพล์เลอร์นั่นเอง ดังนั้นจึงสามารถที่จะใช้ไฟล์ภาษา IDL ตัวเดียวเป็นแหล่งข้อมูล (Source Code) ของตัวคอมไพล์เลอร์ของหลายบริษัทหลายภาษาได้โดยไม่เกิดความเสียหายแต่อย่างใดเลย
7. ในการเปรียบเทียบหลายอย่างที่ผ่านมาทำให้เห็นว่า CORBA มีการทำงานโดยที่ผู้ให้บริการหรือออบเจ็กต์ต่างๆ ที่สร้างขึ้นมานั้นจะมีข้อมูลเป็นของตนเอง ซึ่งทำให้เกิดความสิ้นเปลืองเนื้อที่ในหน่วยความจำเป็นจำนวนมาก
8. ORB ของ CORBA มีข้อดีตรงที่มีลักษณะที่เป็น Dynamic ซึ่งทำให้มีความยืดหยุ่นในการทำงานมาก นอกจากนั้นในการทดลองสำหรับเครื่องคอมพิวเตอร์หลายเครื่องในระบบเครือข่ายเดียวกันยังพบว่าสามารถที่จะทำการรันตัว ORB ตัวที่สองขึ้นมาคร่อม (Overlap) กับ ORB ตัวแรกก็ได้ในกรณีที่เครื่องที่ ORB ตัวแรกทำงานอยู่อาจเกิดปัญหาไม่สามารถให้บริการต่อไปได้ ซึ่งเมื่อทำการรัน ORB ตัวที่สองขึ้นมาคร่อมแล้ว ORB ตัวที่สองก็จะทำการลงทะเบียนออบเจ็กต์ต่างๆ ตามข้อมูลของ ORB ตัวแรก ซึ่งทำให้สามารถให้บริการต่อไปได้ทันที

9. ถ้ามองในแง่ของการโปรแกรมแล้วการลงทะเบียนออบเจ็กต์ของ ORB นับได้ว่ามีประโยชน์และช่วยทำให้เกิดมิติใหม่ของการสร้างโปรแกรมที่ง่ายและสะดวกกว่าการสร้างโปรแกรมในแบบเดิมมาก แต่ถ้าหากมองในแง่ของการตอบสนองผลลัพธ์ของเซริฟเวอร์แล้วนับได้ว่าค่อนข้างที่จะช้า โดยเฉพาะอย่างยิ่งถ้าเป็นลักษณะของ Internet ซึ่งมีการติดต่อกันบนระบบเครือข่ายที่มีปัญหาเรื่องของความเร็วแล้วจะยิ่งทำให้การตอบสนองช้าลงไปอีก
10. เนื่องจาก CORBA มีหลักการทำงานที่เรียกว่า Common Object ดังนั้นจึงทำให้ระบบงานเกิดความเป็นเอกภาพมากขึ้น เกิดความมั่นใจได้ว่าออบเจ็กต์ที่โปรแกรมต่างๆ ในระบบงานนั้นเรียกใช้อยู่นั้นจะทำให้ได้ข้อมูลที่เหมือนกัน

ภาคผนวก
คำย่อและข้อความที่ควรทราบ

คำย่อ

CORBA	Common Object Request Broker Architecture
OMG	Object Management Group
OMA	Object Management Architecture
ORB	Object Request Broker
BOA	Basic Object Adapter
IDL	Interface Definition Language
COS	Common Object Service
IIOP	Internet Inter-ORB Protocol

ข้อความภาษาไทย

โปรแกรมประยุกต์ใช้งาน	Application
ประยุกต์ใช้งาน	Implement
ไคลเอนท์	Client
เซริฟเวอร์	Server
ออบเจ็กต์	Object
มาตรฐาน	Standardization

กิตติกรรมประกาศ

ขอขอบคุณ Electro-Optics Lab (NECTEC) ที่เอื้อเฟื้อสถานที่ และ ระบบเครือข่ายคอมพิวเตอร์ (LAN) สำหรับใช้ติดต่อ Internet (TCP/IP) และ เครื่อง Client/Server สำหรับทดลองตัวโปรแกรม และขอขอบคุณ บริษัท Visigenic Visibroker, Expersoft Corporation และ Iona Orbix Corporation ที่ให้ทดลองใช้โปรแกรม Visigenic Visibroker Developer for Java 3.0 , Powerbroker CorbaPlus for Java และ Orbix Web 3.0 ตามลำดับ

บรรณานุกรม

1. Robert Orfall & DanHarkey,"Client/Server Programming with Java and CORBA", Copyright © 1997 by John Wiley & Sons, Inc.
2. Robert Orfali & DanHarkey,"Client/Server Survival Guide with OS/2", Copyright © 1994 Van Nostrand Reinhold.
3. John Siegel, "CORBA Fundamentals and Programming", Copyright © 1996 by Object Management Group.
4. Visigenic Programming Guide Version 3.0
5. Visigenic Reference Guide Version 3.0
6. Visigenic Gatekeeper Guide Version 3.0
7. Document support http://www.omg.org
8. Technical support, http://www.visigenic.com/, http://www.iona.com and http://www.expersoft.com