การปรับปรุงความเร็วการสื่อสารของคอร์บาด้วยเทคนิคฟาสต์ไอดีแอล

FAST IDL TECHNIQUE FOR CORBA COMMUNICATION

อาวีศรี บุตรโพธิ์

ARVEESRI BUTPO

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรมหาบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัย

สถานบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2546

ISBN 974-324605-3

FAST IDL TECHNIQUE FOR CORBA COMMUNICATION

ARVEESRI BUTPO

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2003

ISBN 974-324605-3

หัวข้อวิทยานิพนธ์	การปรับปรุงความเร็วการสื่อสารของคอร์บาด้วยเทคนิคฟาสต์ไอดีแอล
นักศึกษา	นางสาวอาวีศรี บุตรโพธิ์
รหัสประจำตัว	41067044
ปริญญา	วิทยาศาสตรมหาบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
พ.ศ.	2546
อาจารย์ผู้ควบคุมวิทยานิพนธ์	รศ.บรรจง ปิยธำรง

บทคัดย่อ

การพัฒนาแอพพลิเคชั่นและโปรแกรมประยุกต์ด้วยเทคโนโลยีเชิงวัตถุแบบกระจายและภาษาจาวาผ่านระบบเครือข่ายเป็นวิธีการหนึ่งที่ถูกนำไปใช้แพร่หลายมากขึ้นในปัจจุบัน คอร์บาเป็นหนึ่งในโครงสร้างสถาปัตยกรรมการติดต่อสื่อสารแบบหนึ่ง วัตถุแบบกระจายแบบหนึ่ง การศึกษาวิจัยนี้ได้ทำการศึกษาการทำงานระหว่างออบเจ็กต์ภายใต้สถาปัตยกรรมวิเคราะห์การส่งผ่านออบเจ็กต์และข้อมูลในรูปแบบต่างๆ พบว่าการออกแบบออบเจ็กต์ที่ใช้สถาปัตยกรรมคอร์บามีผลต่อความเร็วในการส่งข้อมูลผ่านระบบเครือข่าย เนื่องจาก IDL หรือภาษาที่ใช้ในการกำหนดโครงสร้างออบเจ็กต์ภายใต้คอร์บาสามารถกำหนดปริมาณข้อมูล ที่จะทำการส่งได้ ดังนั้นการออกแบบ IDL ที่เหมาะสมจะทำให้ความเร็วในการทำงานน้อยลงและส่งข้อมูลได้มากขึ้น

Title	Fast IDL Technique for CORBA Communication
Student	Miss Arveesri Butpo
Student ID.	41067044
Degree	Master of Science
Programme	Information Technology
Year	2003
Thesis Advisor	Assoc. Prof. Banjong Piyatamrong

ABSTRACT

Application and system development using Common Object Request Broker Architecture and Java provides robustness, flexibility, and reusability. IDL design is a key of CORBA performance to be concerned in the high-performance network computing systems. So this paper will aim at studying and evaluating the object communication mechanism under CORBA environment in order to analyze object operations and overheads of communications. Also this paper will propose the Fast IDL technique for designing the proper IDL to improve the performance of CORBA communications.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดีจากความกรุณาของ ผู้ช่วยศาสตราจารย์ บรรจง ปิยธำรง ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ที่คอยให้คำแนะนำ คำปรึกษา และแนะแนวทางการแก้ปัญหาของงานวิจัยรวมทั้งวิชาการต่างๆ ที่เกี่ยวข้อง ซึ่งผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์จากท่าน และขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ คุณเจอรัลด์ เลียม เอ็นไรท์ (Mr.Gerald Liam Enright) และบริษัทเดอะเดฟช็อป จำกัด (TheDevShop Ltd.)ที่สนับสนุนและเอื้อเฟื้ออุปกรณ์และสถานที่ในการทดสอบงานวิจัย

ขอขอบพระคุณ คุณประพันธ์ กรณีกิจ และคุณชินวุธ โทปุรินทร์ ที่คอยให้คำปรึกษาเกี่ยวกับการพัฒนาโปรแกรมที่ใช้ในการทดสอบผลการวิจัย ให้สามารถทำงานได้อย่างถูกต้องและตรงตามวัตถุประสงค์

ขอขอบคุณบิดา มารดา เพื่อนๆ พี่ๆ สาขาวิทยาการสารสนเทศทุกคนในคณะเทคโนโลยีสารสนเทศที่ให้กำลังใจและคำปรึกษาตลอดมา

ขอขอบคุณเจ้าหน้าที่คณะเทคโนโลยีสารสนเทศที่คอยอำนวยความสะดวกเกี่ยวกับอุปกรณ์ ตำราเรียนและงานวิจัยที่เกี่ยวข้อง

ขอขอบคุณบุคคลต่างๆ ที่มิได้เอ่ยนามทุกท่าน ซึ่งคอยให้กำลังใจและแนวคิดเกี่ยวกับวิทยานิพนธ์ฉบับนี้

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

อาวีศรี บุตรโพธิ์

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ หน้า

สารบัญภาพ

ภาพที่ หน้า

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ตลอดเวลา 15 ปีที่ผ่านมามีความเปลี่ยนแปลงเกิดขึ้นมากมายในการออกแบบ พัฒนา และบำรุงรักษาระบบสารสนเทศสำหรับองค์กรขนาดใหญ่ ช่วงเวลาดังกล่าวนี้เริ่มต้นจากระบบการทำงานจากศูนย์กลาง หรือระบบเมนเฟรม (Monolithic mainframe system) แต่ละส่วนของระบบจัดเก็บข้อมูลสำหรับใช้ในการประมวลผลทั้งหมด ตั้งแต่ส่วนนำเสนอข้อมูล เงื่อนไขทางด้านธุรกิจ และการเข้าถึงข้อมูล หากแต่มิได้มีการใช้ข้อมูลร่วมกัน และแต่ละส่วนต้องเก็บข้อมูลที่ต้องการใช้งานไว้อีกชุดหนึ่งด้วย เพราะระบบที่แตกต่างกันมีความจำเป็นต้องใช้งานข้อมูลชุดเดียวกันจากส่วนอื่นๆ เช่นกัน ทำให้องค์การต้องเก็บรักษาข้อมูลที่ซ้ำซ้อนกันบนระบบต่างๆ โปรแกรมประยุกต์หรือแอพพลิเคชั่น (Application) ที่ใช้งานบนระบบข้างต้นจึงค่อนข้างมีประสิทธิภาพต่ำและต้องเสียค่าใช้จ่ายสูง ด้วยเหตุนี้จึงได้มีการนำเทคโนโลยีฐานข้อมูลสัมพันธ์ (Relational database technology) และแบบโครงสร้างไคลเอนท์/เซิร์ฟเวอร์ (Client/Server model) ถือเป็นการบรรจบกันของเทคโนโลยีที่เป็นไปได้ อาทิเช่น ระบบเครือข่าย เครื่องคอมพิวเตอร์ราคาต่ำ รูปแบบการติดต่อกับผู้ใช้ด้วยรูปภาพ (Graphical User Interface) และฐานข้อมูลเชิงสัมพันธ์ (Relational database) การประมวลผลแบบไคลเอนท์/เซิร์ฟเวอร์ (Client/Server computing) ส่อให้เห็นถึงการพัฒนาและบำรุงรักษาแอพพลิเคชั่นที่ซับซ้อนที่ชัดเจนและง่ายขึ้น โดยแยกระบบการทำงานรวมศูนย์กลางเป็นส่วนประกอบย่อยซึ่งพัฒนาและบำรุงได้ง่ายกว่า

แอพพลิเคชั่นที่ใช้งานจะถูกแบ่งออกเป็นส่วนประกอบย่อยฝั่งลูกข่ายหรือไคลเอนท์ (Client component) ซึ่งทำงานส่วนการนำเสนอข้อมูลและเก็บเงื่อนไขธุรกิจบางส่วน และส่วนประกอบทางฝั่งแม่ข่ายหรือเซิร์ฟเวอร์ (Server component) ทำหน้าที่เก็บขั้นตอนการทำงานของเงื่อนไขธุรกิจ ส่วนการเข้าใช้ข้อมูลนั้นสามารถเข้ามาจากไคลเอนท์หรือเซิร์ฟเวอร์ก็ได้ ขึ้นอยู่กับวิธีการที่ใช้ในการพัฒนาแอพพลิเคชั่น การแก้ปัญหาแบบระบบไคลเอนท์/เซิร์ฟเวอร์จำนวนมากถูกสร้างขึ้นเป็นระบบเดี่ยวๆ 2 ระบบที่ในท้ายที่สุดถูกนำมารวมเป็นหนึ่งเดียว ปัจจุบันยังคงมีความยากในการสร้าง บำรุงรักษา และขยายการทำงานของแอพพลิเคชั่นไคลเอนท์/เซิร์ฟเวอร์ที่ใช้งานเฉพาะทาง ทำให้ทีมพัฒนาต้องสร้างฟังก์ชั่นหรือกระบวนการทำงานเช่นเดียวกันซ้ำแล้วซ้ำอีก การนำกฎเกณฑ์หลักการ หรือที่เรียกว่า "โค้ด" (code) มาใช้งานอีกใหม่ค่อนข้างยาก โดยปกติแล้วจะหมายถึงการคัดลอกบางส่วนโค้ดมาใช้งาน ดัดแปลงแก้ไข และกระจายโค้ดที่แก้ไขไปใช้งาน เมื่อผ่านไปสักระยะหนึ่ง ส่วนการทำงานหรือโมดูล (module) ที่คล้ายคลึงกันที่ถูกนำไปใช้งานต้องถูกปรับปรุงแยกจากกัน การเปลี่ยนแปลงที่เกิดขึ้นกับโมดูลหนึ่งจะถูกถ่ายทอดไปยังโมดูลอื่นๆ ที่

คล้ายกันทั้งองค์กร เพราะเหตุว่าการกระทำดังกล่าวพิสูจน์ให้เห็นถึงสิ่งที่นอกเหนือการจัดการและความไม่สอดคล้องกันของการทำงานที่ค่อยๆ เกิดขึ้นกับระบบสารสนเทศขององค์กรขนาดใหญ่

โดยรากฐานของเทคโนโลยีเชิงวัตถุแบบกระจาย (Distributed object technology) ได้เปลี่ยนแปลงการทำงานที่กล่าวมาข้างต้น เนื่องจากเชื่อมการทำงานเข้ากับโครงสร้างพื้นฐานทางด้านติดต่อสื่อสาร วัตถุแบบกระจายจัดแบ่งแอพพลิเคชั่นไคลเอนท์/เซิร์ฟเวอร์ที่ยังคงมีอยู่ทุกวันนี้ ออกเป็นส่วนประกอบย่อยที่มีการจัดการภายในตัวเองได้ (self-managing component) หรือที่เรียกกันว่า "ออบเจ็กต์" (Object) ที่สามารถทำงานข้ามเครือข่ายและระบบปฏิบัติการที่แตกต่างกัน รูปแบบโครงสร้างการประมวลผลเชิงวัตถุแบบกระจายหรือคอมโพเน็นต์เบส (Component based, distributed object computing model) ทำให้หน่วยงานเทคโนโลยีสารสนเทศสร้างโครงสร้างพื้นฐานในการทำงานที่สามารถปรับให้เหมาะกับการเปลี่ยนแปลงอย่างต่อเนื่อง และตอบสนองต่อโอกาสทางการตลาด ในยุคสมัยที่การแข่งขันที่เพิ่มขึ้นทั่วโลกบนหน้าจอเล็กๆ บริษัทที่มีการปรับเปลี่ยนอย่างรวดเร็ว ไม่จำเป็นขานรับสิ่งที่กำลังเข้ามาก เป็นการดีที่จะเตรียมและประเมินโอกาสที่จะเกิดขึ้นและความสำเร็จที่น่าจะเป็นไปได้ แอพพลิเคชั่นเชิงวัตถุแบบกระจาย (Distributed application) สามารถกำหนดเงื่อนไขเพื่อทำให้เกิดโอกาสที่เป็นไปได้ในอันที่จะบรรลุและธำรงไว้ซึ่งความได้เปรียบในการแข่งขัน ด้วยการสร้างโครงสร้างพื้นฐานทางเทคโนโลยีสารสนเทศที่ปรับเปลี่ยนได้ รวมทั้งความต้องการใหม่ๆ จากผู้ใช้งาน การปฏิบัติงานในสภาพแวดล้อมการประมวลผลที่ไม่เหมือนกัน แอพพลิเคชั่นเชิงกระจายธุรกิจต้องทำงานบนความหลากหลายของอุปกรณ์อิเล็กทรอนิกส์คอมพิวเตอร์และโปรแกรมประยุกต์ สิ่งเหล่านี้จำต้องมีการประสานการทำงานเทคโนโลยีทั้งเก่าและใหม่ ให้ใช้งานร่วมกับโครงสร้างที่มีอยู่เดิม โดยเฉพาะอย่างยิ่งเหมาะกับความต้องการการใช้งานแอพพลิเคชั่นในระดับองค์กรขนาดใหญ่ เพื่อให้สามารถรองรับการประมวลผลแบบข่ายงานหรือเว็บ (Web-based computing) ทั้งในแง่การขยายงาน การใช้ประโยชน์อย่างคุ้มค่า ง่ายต่อการจัดการดูแล มีประสิทธิภาพการทำงานสูง และความสมบูรณ์ของข้อมูล

เทคโนโลยีเชิงวัตถุ (Object-Oriented Technology: OO Technology) จึงถูกนำมาใช้ในการพัฒนาระบบงานและแอพพลิเคชั่น (application) บนระบบเครือข่าย โดยเฉพาะแนวความคิดทางเทคโนโลยีวัตถุแบบกระจาย (Distributed Object Technology: DOT) ซึ่งนำแนวความคิดการทำงานเชิงวัตถุ (object-oriented model) และการเขียนโปรแกรมเชิงวัตถุ (object-oriented programming) มาใช้ร่วมกับโครงสร้างพื้นฐานของวัตถุแบบกระจาย (distributed object infrastructure) เพื่อให้เหมาะสมกับการทำงานบนระบบไคลเอนท์และเซิร์ฟเวอร์ที่แบ่งเป็น 3 ส่วน (3-tier client/server system) หรือสถาปัตยกรรมของระบบการทำงานแบบหลายระดับชั้น (multi-tier system architecture) โดยการแบ่งฟังก์ชันการทำงานออกเป็นออบเจ็กต์หรือส่วนประกอบย่อยๆ ที่กระจายอยู่บนแพลตฟอร์ม (platform) ที่แตกต่างกัน และไม่ว่าจะถูกเขียนขึ้นด้วยภาษาใดก็สามารถ

ติดต่อสื่อสารและทำงานร่วมกันได้ผ่านตัวกลางในการสื่อสารระหว่างออบเจ็กต์ที่เรียกว่า Object Request Broker (ORB) หรือโออาร์บี ระหว่างไคลเอนท์และออบเจ็กต์ที่อยู่บนเครื่องเซิร์ฟเวอร์ ORB ที่เป็นนิยมและถูกนำมาใช้อย่างแพร่หลายในวงการธุรกิจและการศึกษาวิจัยคือ Common Object Request Broker Architecture (CORBA) หรือคอร์บา ไม่ว่าจะเป็นในระบบธุรกิจแบบ 3-tier (3-tier business system) ระบบโมบายเอเจ็นท์ (mobile agent system) เป็นต้น คอร์บาจะควบคุมการทำงานโปรแกรมเครือข่ายทั่วไป (common network programming tasks) อาทิเช่น การกำหนดที่ตั้งของออบเจ็กต์ (object location) การกระตุ้นการทำงานของออบเจ็กต์ (object activation) การส่งและจัดรูปแบบพารามิเตอร์ (parameter marshalling/demarshalling) การกำหนดรูปแบบการส่งข้อมูล (framing) และการจัดการความผิดพลาดในระบบ (error handling) นอกจากนี้คอร์บายังจัดเตรียมมาตรฐานสำหรับการกำหนดการให้บริการในระดับสูง ได้แก่ การตั้งชื่อ (naming) การจัดการเหตุการณ์ต่างๆ ในระบบ (event) การทำสิ่งจำลองของออบเจ็กต์ (replication) การบันทึกรายการ (transaction) และความปลอดภัย (security) ประสิทธิภาพการทำงานของ ORB จึงมีผลต่อประสิทธิภาพการทำงานของระบบบนเครือข่ายด้วยเช่นกัน

อย่างไรก็ตามยังมีจุดบกพร่องสำคัญอย่างหนึ่งในการใช้ CORBA ORB เป็นสื่อกลางเชื่อมการติดต่อในระดับสูง (higher-level middleware) คือ ทำงานค่อนข้างช้ากว่าวิธีการส่งข้อมูลผ่านโปรแกรมระบบเครือข่ายโดยทั่วไป (conventional network programming technique) เช่น C/C++ socket เป็นต้น ทำให้ได้ปริมาณงานที่ทำสำเร็จ (throughput) น้อยลงตามลำดับ เพราะเกิดส่วนเกิน (overhead) ในกรณีที่มีการส่งข้อมูลขนาดใหญ่ เนื่องมาจากต้องเปลี่ยนแปลงรูปแบบข้อมูลในระดับการนำเสนอและแสดงผล (presentation layer conversion) และการทำสำเนาข้อมูล (data copying) ดังนั้นงานวิจัยนี้ทำการศึกษากระบวนการทำงานของ ORB ประเมินและปรับปรุงการทำงานของ ORB ให้ได้ปริมาณงานที่ทำสำเร็จมากขึ้น และลดระยะเวลาในการส่งข้อมูล โดยเฉพาะการทดสอบความสามารถในด้านการรับส่งข้อมูลผ่านระบบเครือข่าย โดยวัดความเร็วในการส่งข้อมูลตั้งแต่เริ่มติดต่อกับเครื่องเซิร์ฟเวอร์จนได้ผลลัพธ์ หรือที่เรียกกันว่า Round-Trip Time (RTT) และจำนวนของผลการทำงานที่ทำสำเร็จ (Throughput) ซึ่งเป็นการทดสอบการทำงานโดยรวมของทั้งระบบ เพื่อวิเคราะห์การทำงานของออบเจ็กต์และแก้ไขจุดบกพร่องในการรับส่งข้อมูล นำไปใช้ปรับปรุงการรับส่งข้อมูลให้รวดเร็วและเพิ่มขึ้น

1.2 วัตถุประสงค์

1) เพื่อศึกษากระบวนการการทำงานของสถาปัตยกรรม Common Object Request Broker Architecture (CORBA)

2) เพื่อศึกษาเครื่องมือที่สร้างขึ้นเพื่อรองรับการทำงานในมาตรฐาน CORBA

3) เพื่อประเมินประสิทธิภาพการทำงานภายใต้สถาปัตยกรรม CORBA และปัจจัยที่มีผลต่อการทำงานภายใต้สถาปัตยกรรมคอร์บา

4) เพื่อหาศึกษารูปแบบการเขียนอินเทอร์เฟซของออบเจ็กต์ (CORBA IDL) ให้เหมาะสมกับการการทำงานภายใต้สถาปัตยกรรม CORBA ให้ทำงานได้เร็วขึ้น และสอดคล้องสภาพการใช้งานในปัจจุบัน

1.3 ขอบเขตงานวิจัย

งานวิจัยนี้ได้ทำการศึกษาถึงวิธีการและขั้นตอนการทำงานของ CORBA ORB จากผลิตภัณฑ์ ORB ที่ได้รับการยอมรับ และประเมินการทำงานในด้านต่างๆ ได้แก่ ความสามารถในการตอบสนองต่อการเรียกใช้งานวัตถุของ CORBA ตามลักษณะของข้อมูลแต่ละชนิด และวิธีที่ใช้ในการส่งข้อมูลที่มีขนาดใหญ่ โดยเปรียบเทียบกับระยะเวลาทั้งหมดที่ใช้ในการรับ-ส่งข้อมูล (round trip time) เพื่อวิเคราะห์การทำงานของออบเจ็กต์ และข้อด้อยในการรับ-ส่งข้อมูล พร้อมทั้งนำเสนอเทคนิคที่ใช้ปรับปรุงการออบแบบอินเทอร์เฟซของออบเจ็กต์ด้วย IDL ซึ่งเป็นภาษาที่ใช้ในการกำหนดโครงสร้างของออบเจ็กต์ที่ใช้สำหรับการพัฒนาแอพพลิเคชั่นภายใต้สภาพแวดล้อมของคอร์บา เนื่องจากการออกแบบภาษาที่ใช้กำหนดการติดต่อสื่อสารระหว่างวัตถุ ซึ่งมีผลต่อการกำหนดปริมาณข้อมูลที่จะเกิดขึ้นบนเครือข่าย เพื่อปรับปรุงประสิทธิภาพและลดจำนวนการส่งข้อมูลผ่านเครือข่าย

1.4 ขั้นตอนการดำเนินงานวิจัย

1. ศึกษางานวิจัยและบทความต่างๆ ที่เกี่ยวข้อง
2. เก็บข้อมูลตัวอย่างจากการทดสอบการส่งข้อมูลชนิดต่างๆ ทั้งที่ทำงานบนเครื่องเดียวกัน และภายในระบบเครือข่าย
3. ศึกษารูปแบบการจัดรูปแบบการส่งข้อมูลของ ORB ก่อนเข้าสู่ระดับเครือข่าย (Network layer)
4. เขียนโปรแกรมทดสอบความสามารถการส่งข้อมูลในรูปแบบข้อมูลขนาดต่างๆ ของ CORBA ORB
5. รวบรวมผลการทดลองที่ได้จากโปรแกรม และวิเคราะห์ข้อบกพร่องในการส่งข้อมูลด้วย CORBA ORB
6. ศึกษาเทคนิคในการออกแบบออบเจ็กต์สำหรับสถาปัตยกรรม CORBA
7. นำเสนอรูปแบบการออกแบบออบเจ็กต์สำหรับปรับปรุงความเร็วการส่งข้อมูลภายใต้สถาปัตยกรรม CORBA

8. วัดประสิทธิภาพเทคนิคใหม่ในการออกแบบออบเจ็กต์ที่นำเสนอในข้อ 7
9. สรุปผลและวิธีการที่เป็นไปได้ พร้อมทั้งทำเอกสารนำเสนอเป็นงานวิจัย

1.5 สรุปงานวิจัยที่เกี่ยวข้อง

1.5.1 การประเมินประสิทธิภาพการทำงานของเทคโนโลยีการกระจายวัตถุด้วยภาษาจาวา (Performance Evaluation of Popular Distributed Object Technologies for Java)

ศึกษาโดย Satoshi Hirano Yoshiji Yasu และ Hirotaka Igarashi ในปี ค.ศ. 1999 ได้ทำการประเมินประสิทธิภาพโปรแกรมสำหรับเทคโนโลยีเชิงวัตถุ ได้แก่ Java Socket HORB version 1.3.b2 จาก HIRANO Satoshi of Electrotechnical Laboratory (ETL) RMI ของ Sun's JDK 1.1.4 Voyager 2.0 beta 1 ของ ObjectSpace โปรแกรมที่พัฒนา CORBA IIOP ด้วยจาวา เช่น Visibroker for Java ของบริษัท Visigenic และ OrbixWeb 2.0.1 ของบริษัท IONA เป็นต้น และ Microsoft SDK for Java 2.0 สำหรับ DCOM โดยประเมินความสามารถเกี่ยวกับการสร้างและติดต่อออบเจ็กต์ (object connection and creation) การเรียกใช้วิธีการทำงานจากระยะไกล (remote method call) การส่งชุดอะเรย์ของออบเจ็กต์ (object array transfer) และการส่งข้อมูลตัวเลขขนาดใหญ่ (data transfer of large numerical data) ผลปรากฏว่า ไม่มีโปรแกรมการใช้งานเชิงวัตถุชนิดใดสามารถทำงานได้อย่างมีประสิทธภาพครบทุกกรณี และ HORB มีประสิทธิภาพดีโดยเฉพาะการเรียกใช้งานออบเจ็กต์จากระยะไกล แต่เทคโนโลยีเชิงวัตถุทั้งหมดที่พิจารณาควรได้รับการปรับปรุงด้านการส่วนสำรองข้อมูล (buffer management) และการทำสำเนาข้อมูล

1.5.2 การประเมินความสามารถของตัวกลางการติดต่อสื่อสารบนเครือข่ายความเร็วสูง (Measuring the Performance of Communication Middleware on High-Speed Networks)

Aniruddha S. Gokhale และ Douglas C. Schmidt ในปี ค.ศ. 1996 ได้ทำการทดลองหาแหล่งที่มาของส่วนเกินในการส่งข้อมูลภายในการพัฒนาคอร์บาโดยทั่วไปบนเครือข่าย ATM ปรากฏว่า โปรแกรมที่พัฒนาด้วย CORBA ให้อัตราความเร็วในการส่งข้อมูลต่อหนึ่งหน่วยเวลาน้อยกว่าการส่งด้วย C/C++ wrapper และ hand-optimized RPC version of TTCP วิธีการแบบ CORBA ทำงานได้ผลน้อยสุดในการส่งข้อมูลที่มีความซับซ้อน เช่น ข้อมูลชนิด struct เนื่องจากส่วนเกินของข้อมูลที่มากเกินไปซึ่งมาจากการทำสำเนาข้อมูลและการแปลงพารามิเตอร์ (marshalling/demarshalling) และการเก็บสำรองข้อมูลเป็นส่วนย่อยๆ มากเกินไป

จึงสรุปว่าการปรับปรุงการทำงานให้มีประสิทธิภาพยิ่งขึ้นต้องประยุกต์การทำงานของ CORBA ในส่วน stub ของไคลเอนท์และ skeleton ของเซิร์ฟเวอร์ เพื่อลดส่วนเกินในการแปลงพารามิเตอร์ การทำสำเนาข้อมูล และการส่งสัญญาณการเรียกใช้งาน

1.5.3 ประสิทธิภาพของตัวกลางการเรียกใช้วัตถุแบบจาวา (Performance Characteristic of a Java Object Request Broker) ศึกษาโดย David Miron และ Samuel Taylor ในปี ค.ศ. 2002 กล่าวถึงประสิทธิภาพการทำงานของ คอร์บาในการส่งแฟ้มข้อมูลขนาดใหญ่ผ่านระบบเครือข่ายในโครงการจัดการและเผยแพร่รูปภาพ (Imagery Management and Dissemination Group: IMAD) ซึ่งประเมินความสามารถการทำงานของคอร์บาด้วยการวัดจำนวนข้อมูลที่ส่งได้และความหน่วงที่เกิดขึ้น โดยเปรียบเทียบผลที่ได้จากการทำงานของคอร์บากับการติดต่อแบบ socket-to-socket ของจาวา พบว่าปริมาณงานจากการส่งข้อมูลขนาดใหญ่ผ่าน ORB ได้จำนวนน้อยกว่าการใช้วิธีส่งผ่าน socket และความหน่วงที่เกิดขึ้นจาก ORB เพิ่มขึ้นกว่าการใช้ socket ด้วยเช่นกัน นอกจากนี้ยังได้ศึกษาขนาดของข้อมูลย่อยที่เหมาะสมในการส่งข้อมูล

1.5.4 เริ่มต้นการจัดการประสิทธิภาพการทำงานด้วยการออกแบบ IDL (Performance Management Starts with IDL Design) นำเสนอโดย Khanh Chau ในปี 2002 ได้ประยุกต์แบบแผนตัวทำซ้ำ (Iterator Pattern) ในระดับภาษาที่ใช้อธิบายการติดต่อสื่อสารกับวัตถุ (Interface Definition Language: IDL) เพื่อดูแลปริมาณข้อมูลที่ส่งผ่านบนเครือข่าย เนื่องจากสภาพแวดล้อมของคอร์บานั้น การออกแบบอินเทอร์เฟซด้วย IDL เป็นสิ่งสำคัญอย่างหนึ่งที่มีผลต่อประสิทธิภาพความเร็วในการทำงานของคอร์บา เมื่อเครื่องไคลเอนท์ร้องขอข้อมูลขนาดใหญ่ไปยังเครื่องเซิร์ฟเวอร์ เพื่อจัดเก็บและประมวลผล แบบแผนตัวทำซ้ำจะช่วยให้เครื่องเซิร์ฟเวอร์กระจายข้อมูลออกเป็นส่วนย่อย เพื่อลดการจองหน่วยความจำ ซึ่งมีผลต่อประสิทธิภาพการทำงานของระบบผ่าน ORB

1.6 ข้อจำกัดของงานวิจัย

เนื่องจากการวิจัยนี้มุ่งเน้นการออกแบบและกำหนดอินเทอร์เฟซด้วย Interface Definition Language ที่เป็นพื้นฐานการติดต่อสื่อสารระหว่างวัตถุภายใต้สถาปัตยกรรมคอร์บา ดังนั้นจึงมิได้ปรับปรุงเชิงลึกภายใต้สถาปัตยกรรมแบบ CORBA เนื่องจากบางวิธีการอาจทำให้ลดความสามารถบางอย่างของ CORBA นั้นเสียไป

บทที่ 2

สถาปัตยกรรมคอร์บา

(Common Object Request Broker Architecture)

2.1 บทนำ

Common Object Request Broker Architecture (CORBA) เป็นแนวทางการพัฒนาแอพพลิเคชั่นตามหลักการเทคโนโลยีการกระจายวัตถุที่กำหนดขึ้นโดย Object Management Group (OMG) ซึ่งได้รับความนิยมนำไปใช้งานและศึกษาค้นคว้าอย่างแพร่หลาย CORBA ได้กำหนด Object Request Broker (ORB) เป็นกลไกการทำงานมาตรฐานระหว่างออบเจ็กต์ของโปรแกรมประยุกต์ที่ต้องมีกระจายการทำงาน (distributed software object) และเครื่องคอมพิวเตอร์ที่เป็นไคลเอนท์ (Client) ให้ติดต่อสื่อสารกัน หลายบริษัทได้นำสถาปัตยกรรม CORBA ไปพัฒนาโปรแกรมทางด้าน ORB software โดยเริ่มพัฒนาตั้งแต่รายละเอียดการส่งคำร้องขอโอเปอร์เรชั่นที่ต้องการใช้งาน (operation request) และการตอบสนองต่อคำร้องขอดังกล่าว รวมทั้งรายละเอียดด้านการให้บริการต่างๆ ของ ORB

OMG ได้นำเสนอ CORBA 1.0 ในเดือนธันวาคม 1990 และตามด้วย CORBA 1.1 ในช่วงต้นปี 1991 โดยกำหนดภาษาที่ใช้ในการติดต่อสื่อสารที่เรียกกันว่า Interface Definition Language (IDL) ซึ่งคล้ายกับ Application Programming Interface (API) สำหรับให้แอพพลิเคชั่นติดต่อกับ Object Request Broker (ORB) ต่อมากำหนดให้ CORBA 1.x สร้างขั้นตอนที่สำคัญที่ทำให้ออบเจ็กต์ทำงานร่วมกันได้ อนุญาตให้ออบเจ็กต์ทำงานบนสภาพที่แตกต่างกันทั้งเครื่องที่ใช้งานสถาปัตยกรรม และการเขียนโปรแกรมด้วยภาษาที่แตกต่างกันให้ติดต่อสื่อสารระหว่างกันได้ CORBA เริ่มมีบทบาทในด้านการพัฒนาฟังก์ชั่นการทำงานของ ORB ในเดือนธันวาคม ปี 1994 CORBA 2.0 ได้นำเสนอออกมาอีกครั้ง เพื่อกำหนดโปรโตคอล (protocol) มาตรฐานที่ใช้ในการติดต่อระหว่าง ORB ที่มาจากหลากหลายผู้ผลิต ที่รู้จักกันในชื่อว่า Internet Inter-ORB Protocol (IIOP) โดยนำมาใช้บนระบบเครือข่ายที่เป็น TCP/IP เท่านั้น ซึ่งเป็นโปรโตคอลที่ใช้แพร่หลายในอินเทอร์เน็ตและอินทราเน็ต (Intranet) ด้วยเหตุนี้ผลิตภัณฑ์ ORB ที่มีคุณสมบัติตาม CORBA 2.0 เรียกว่าเป็น CORBA 2.0 compliant อย่างไรก็ตามมาตรฐาน CORBA ยังคงมีการพัฒนาอย่างต่อเนื่อง เพื่อให้ได้สถาปัตยกรรมที่สมบูรณ์และมีประสิทธิภาพยิ่งขึ้น

2.2 โครงสร้างพื้นฐานของสถาปัตยกรรมคอร์บา (Overview of Common Object Request Broker Architecture)

CORBA ประกอบด้วยองค์ประกอบที่สำคัญ 5 ส่วนดังต่อไปนี้

- *Object Request Broker (ORB)* ถือเป็นออบเจ็กต์บัส (object bus) หรือเส้นทางการสื่อสารของออบเจ็กต์ โดย ORB จัดได้ว่าเป็นมิดเดิ้ลแวร์ (middleware) ทำหน้าที่เป็นตัวกลางในการติดต่อระหว่างแอพพลิเคชั่นทางฝั่งไคลเอนท์ที่ต้องการบริการและแอพพลิเคชั่นทางฝั่งเซิร์ฟเวอร์

- *Interface Definition Language (IDL)* เป็นภาษาที่ใช้ประกาศและกำหนดโครงสร้างของอินเทอร์เฟซของออบเจ็กต์ (object interface) เป็นภาษาที่ทำงานโดยไม่คำนึงถึงวิธีการพัฒนา

- *Interface Repository* เป็นฐานข้อมูลแบบออนไลน์ (online) สำหรับเก็บคำนิยามและข้อกำหนดต่างๆ เกี่ยวกับออบเจ็กต์ ใช้ในระหว่างที่แอพพลิเคชั่นกำลังทำงาน สำหรับการเรียกใช้ด้วยวิธีไดนามิก (dynamic method calls) การระบุตำแหน่งคอมโพเนนต์ของซอฟท์แวร์ที่นำกลับมาใช้ได้อีก และการตรวจสอบชนิดของ method signature

- *Object Adaptor* สร้างขึ้นสำหรับจัดเตรียมสภาพแวดล้อมระหว่างการทำงานให้เหมาะสมกับแอพพลิเคชั่นทางฝั่งเซิร์ฟเวอร์ และควบคุมการเรียกใช้งานจากทางฝั่งไคลเอนท์

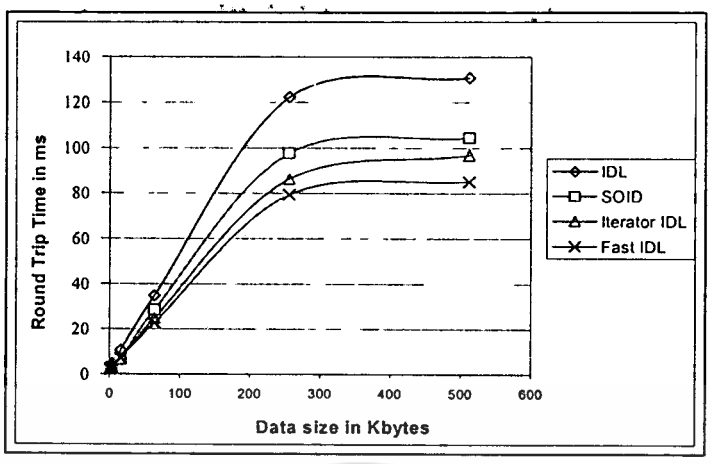
- *CORBA Services* เป็นส่วนขยายฟังก์ชั่นการทำงานของ ORB โดยทำหน้าที่ให้บริการสำหรับการจัดการสถานภาพของออบเจ็กต์ (persistence) ทรานแซคชั่น (transactions) การสอบถามฐานข้อมูล (database queries) เป็นต้น

2.2.1 ส่วนการทำงานของ CORBA

โครงสร้างของสถาปัตยกรรมคอร์บาประกอบด้วยส่วนการทำงานที่สำคัญดังภาพที่ 2.1 มีรายละเอียดดังนี้

- **Servant** มีหน้าที่พัฒนาขั้นตอนการปฏิบัติงานที่กำหนดด้วยอินเทอร์เฟซของ IDL โดยใช้ภาษาที่สนับสนุนการเขียนโปรแกรมเชิงวัตถุ (object-oriented programming) เช่น C++ และ Java servant ที่พัฒนาขึ้นอาจใช้การทำงานจากออบเจ็กต์มากกว่า 1 ออบเจ็กต์ servant จะจำแนกด้วย Object reference ซึ่งใช้บ่งชี้และจำแนก servant ที่อยู่ในกระบวนการทำงานทางฝั่งเซิร์ฟเวอร์

- **Client** เป็นส่วนหลักของโปรแกรมที่เรียกใช้โอเปอร์เรชั่น (operation) บน servant นั้น Client อาจเรียกใช้จากภายในหรือภายนอกก็ได้ ซึ่งมีวิธีการเรียกใช้คล้ายกับการเรียกใช้โอเปอร์เรชั่นที่อยู่ภายใน เช่น object - > operation(args) ตามรูปที่ 1 ได้แสดงให้เห็นส่วนประกอบที่ ORB ใช้ในการส่งคำร้องขอ (request) จาก client ไปยัง servant สำหรับการเรียกใช้โอเปอร์เรชั่นจากระยะไกล (remote operation invocations)



ภาพที่ 2.1 โครงสร้างของสถาปัตยกรรมคอร์บา

- **ORB Core** เมื่อไคลเอนท์ (client) ร้องขอโอเปอร์เรชั่นบน servant ORB Core จะรับผิดชอบเกี่ยวกับการส่งคำร้องขอไปให้แก่ servant และการส่งผลการทำงาน (response) คืนให้แก่ไคลเอนท์ สำหรับ servant ที่ถูกเรียกใช้งานจากระยะไกล CORBA-compliant ORB Core จะติดต่อผ่าน General Inter-ORB Protocol (GIOP) และ Internet Inter-ORB Protocol (IIOP) ซึ่งทำงานบนโปรโตคอลที่ทำหน้าที่ส่งข้อมูลที่เรียกว่า TCP โดย ORB Core จะถูกพัฒนาขึ้นในลักษณะที่เรียกว่า run-time library แฝงอยู่ในแอพพลิเคชั่นทางฝั่งไคลเอนท์หรือเซิร์ฟเวอร์

- **ORB Interface** ORB อาจจะถูกพัฒนาได้หลายแบบ อาทิเช่น เป็นกลุ่มของโปรเซสการทำงาน (processes) หรือกลุ่มของไลบรารี (libraries) ORB interface จึงต้องจัดเตรียมโอเปอร์เรชั่นมาตรฐานที่ใช้แปลง object reference เป็น string และแปลงค่ากลับคืนรูปเดิมอีกครั้ง นอกจากนี้ยังสร้างรายละเอียดของอาร์กิวเมนต์สำหรับคำร้องขอผ่านส่วนที่เรียกว่า Dynamic Invocation Interface (DII)

- **OMG IDL Stubs and Skeletons** ทำหน้าที่คล้ายกับตัวเชื่อมการติดต่อระหว่างไคลเอนท์และเซิร์ฟเวอร์ และระหว่าง ORB การเปลี่ยนแปลงจากรูปแบบของ IDL เป็นภาษาที่ต้องการใช้ในการเขียนโปรแกรมจะเป็นไปโดยอัตโนมัติผ่าน IDL complier ของภาษานั้นๆ IDL complier จะตัดข้อมูลทั่วไปที่ผิดพลาดในการเขียนโปรแกรมระดับเครือข่าย และจัดรูปแบบให้เหมาะกับคอมไพเลอร์ (complier)

- **Dynamic Invocation Interface (DII)** เป็นอินเทอร์เฟซที่ปรับเปลี่ยนได้เพื่อให้ไคลเอนท์เข้าถึงกลไลพื้นฐานสำหรับการส่งคำร้องขอภายใต้ ORB หากว่าออบเจ็กต์ของแอพพลิเคชั่นไม่มีข้อมูลของอินเทอร์เฟซที่ต้องการ และยังอนุญาตให้ไคลเอนท์เรียกใช้งานแบบ deferred synchronous ซึ่งจะแยกส่วนการร้องขอและส่งคืนผลการทำงานที่ได้ เพื่อหลีกเลี่ยงการระงับส่วนการร้องขอของไคลเอนท์จนกว่า servant จะมีการตอบกลับ

- **Dynamic Skeketon Interface (DSI)** ทำงานคล้าย DII ของไคลเอนท์ จะอนุญาตให้ ORB ส่งคำร้องขอไปยัง servant ที่ไม่มีข้อมูลออบเจ็กต์ที่ต้องการ ในช่วงเวลาที่มีทำงานของ IDL interface (compile time) โดยไม่ต้องทราบว่า ORB บนฝั่งเซิร์ฟเวอร์นั้นใช้ skeleton แบบใด

- **Object Adaptor** จัดเป็นส่วนการทำงานสำคัญของ ORB ในการส่งคำร้องขอไปยังออบเจ็กต์และกระตุ้นออบเจ็กต์ให้ทำงาน อีกทั้งยังช่วยการสร้างออบเจ็กต์เพื่อทำงานร่วมกับ ORB

จากโครงสร้างข้างต้นสังเกตได้ว่าทั้งทางฝั่งไคลเอนท์และเซิร์ฟเวอร์ต้องทำการติดตั้ง ORB และติดต่อสื่อสารกันผ่าน ORB CORBA ได้กำหนดโปรโตคอลมาตรฐานที่ใช้ในส่วนการติดต่อสื่อสารระหว่างเครื่องเซิร์ฟเวอร์และเครื่องไคลเอนท์ เรียกว่า General Inter-ORB Protocol (GIOP)

2.2.2 General Inter-ORB Protocol (GIOP)

GIOP เป็นโปรโตคอลมาตรฐานที่ออบเจ็กต์ในสถาปัตยกรรม CORBA ใช้สำหรับติดต่อสื่อสาร และแลกเปลี่ยนข้อมูลระหว่างเครื่องไคลเอนท์และเซิร์ฟเวอร์ ประกอบขึ้นด้วย

- **การกำหนดรูปแบบทั่วไปของการนำเสนอข้อมูล (Common Data Representation : CDR)**

CDR เป็นโครงสร้างประโยคที่ใช้แปลงข้อมูลชนิดต่างๆ ของ IDL ไปเป็นรูปแบบที่จะนำส่งระหว่าง ORB โดยมีจัดแบบแผน CDR สำหรับ IDL ทุกชนิด GIOP ยังมีการกำหนดรูปแบบข้อความ 8 แบบ คือ Request, Reply, CancelRequest, LocateRequest, LocateReply, CloseConnection, MessageError และ Fragment

- **การจัดรูปแบบข้อความของ GIOP (GIOP Message Format)**

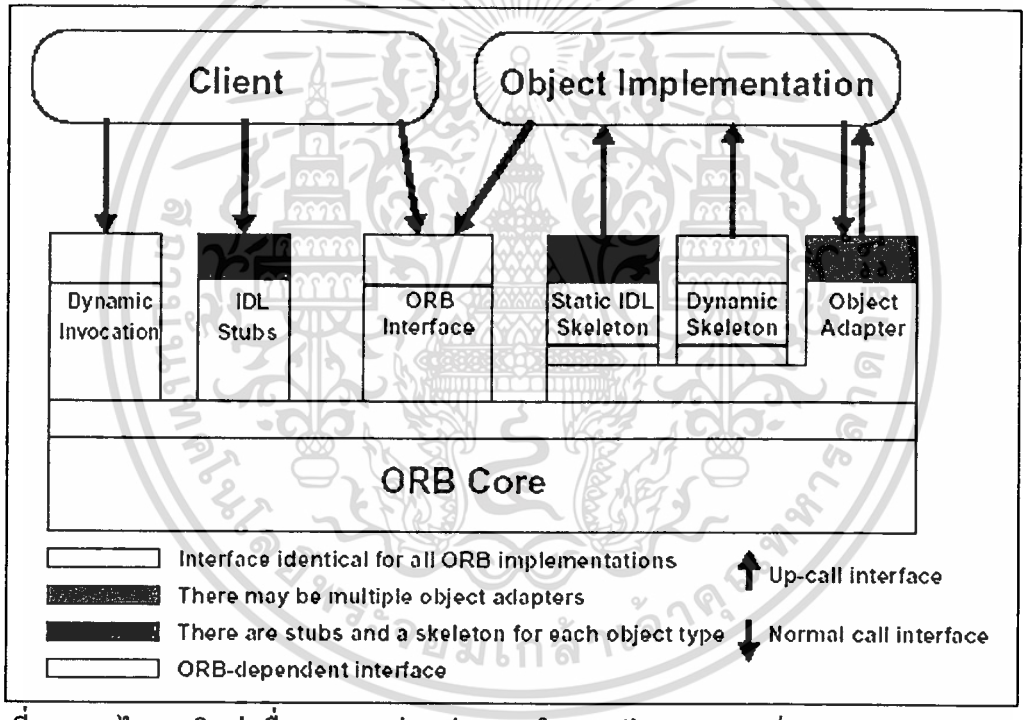
รูปแบบข้อความของ GIOP นั้นจะมีการแลกเปลี่ยนระหว่าง ORB เพื่อสนับสนุนการเรียกใช้งานออบเจ็กต์ การระบุตำแหน่งออบเจ็กต์ที่มีการพัฒนาแล้ว และการจัดการช่องทางการสื่อสาร

- **การเข้านำส่งข้อมูลผ่าน GIOP (GIOP Transport Assumption)**

นอกจากนี้ CORBA ยังกำหนดส่วนการติดต่อสื่อสารผ่านเครือข่ายอินเตอร์เน็ต เรียกว่า Internet Inter-ORB Protocol (IIOP) ซึ่งเป็นรูปแบบเฉพาะของ GIOP สำหรับติดต่อผ่านโปรโตคอล TCP/IP และมีการระบุวิธีการเชื่อมต่อยัง TCP/IP และนำมาใช้สำหรับการถ่ายโอนข้อมูลของ GIOP

2.3 กลไกการติดต่อสื่อสารของคอร์บา (CORBA Communication Mechanism)

OMG กำหนดกลไกพื้นฐานในการติดต่อระหว่างองค์ประกอบที่กล่าวถึงในข้อ 2.2 [16] ORB ทำหน้าที่รับผิดชอบทุกกลไกที่ต้องถูกเรียกใช้ได้แก่ การค้นหาออบเจ็กต์ที่ถูกร้องขอ เตรียมออบเจ็กต์ให้พร้อมรับคำร้องขอที่จะเกิดขึ้น และการสื่อสารข้อมูลระหว่างการจัดการตามคำร้องขอนั้นๆ ในภาพที่ 2.2 แสดงถึงกลไกการติดต่อสื่อสารของคอร์บา โดยโครงสร้างหลักที่อธิบายไว้ในข้อ 2.2 จะแสดงแทนด้วยรูปสี่เหลี่ยมที่มีแถบสี และลูกศรแสดงถึงการทำงานของ ORB เมื่อมีการเรียกใช้งานออบเจ็กต์เกิดขึ้น ไคลเอนท์สามารถเรียกใช้งานออบเจ็กต์นั้นผ่าน Dynamic Invocation Interface หากว่าออบเจ็กต์ของแอพพลิเคชั่นไม่มีข้อมูลของอินเทอร์เฟซที่ต้องการ หรือเรียกใช้งานออบเจ็กต์ผ่าน IDL Stub ในกรณีที่มีข้อมูลอินเทอร์เฟซนั้นเปิดให้บริการอยู่ นอกจากนี้ไคลเอนท์ยังสามารถติดต่อกับ CORBA โดยตรงเพื่อเรียกใช้งานบริการพื้นฐานอีกด้วย

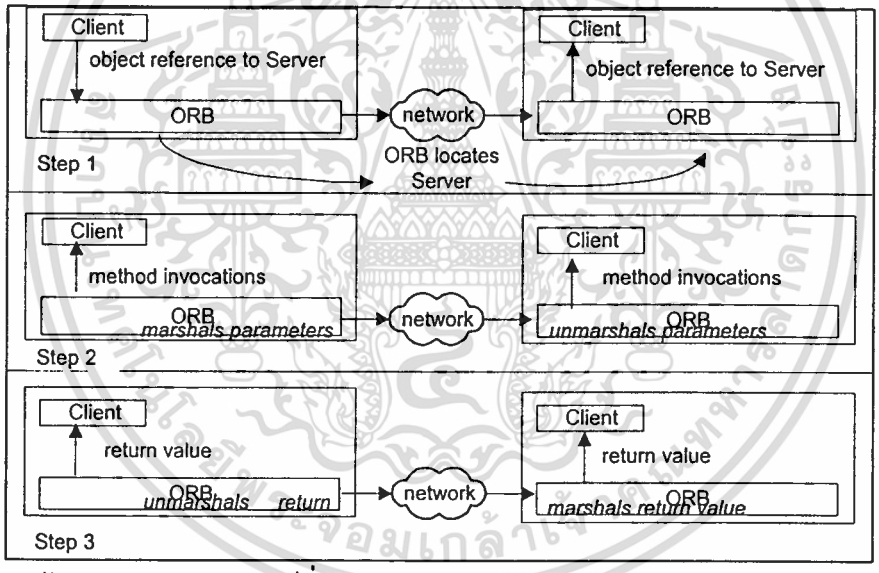


ภาพที่ 2.2 กลไกการติดต่อสื่อสารของส่วนประกอบในสถาปัตยกรรมคอร์บา

เมื่อส่วนการทำงานของออบเจ็กต์ได้รับคำร้องขอหรือการเรียกใช้งานอินเทอร์เฟซดังกล่าวจะส่งคำร้องขอเหล่านั้นขึ้นผ่านไปยัง OMG IDL เพื่อสร้าง IDL skeleton หรือ dynamic skelaton ขณะเดียวกันส่วนการทำงานของออบเจ็กต์นั้นอาจออกคำสั่งไปยัง Object Adapter และ ORB ระหว่างที่มีการประมวลผลคำร้องขอก็ได้ การกำหนดอินเทอร์เฟซกับออบเจ็กต์สามารถกำหนดได้สองแบบ อินเทอร์เฟซสามารถกำหนดตายตัวไว้ใน IDL ซึ่งเรียกว่า OMG Interface Definition Language (OMG IDL) หรือไอดีแอล ภาษา IDL ทำการนิยามชนิดและโครงสร้างของออบเจ็กต์

และโอเปอร์เรชั่นเพื่อประมวลผลพารามิเตอร์ที่ส่งเข้ามา ขณะเดียวกันอินเทอร์เฟซเหล่านั้นสามารถเพิ่มได้ในการให้บริการที่เก็บรวบรวมอินเทอร์เฟซ (Interface Repository service) บริการดังกล่าวนั้นนำเสนอส่วนประกอบของอินเทอร์เฟซดังเช่นออกเจ็กต์ อนุญาตให้มีการเรียกใช้ขณะกำลังทำงาน ไคลเอนท์จะปฏิบัติตามคำร้องขอได้นั้นต้องมีคำอนุญาตให้เข้าไปใช้งานออบเจ็กต์เหล่านั้น และรู้จักชนิดและโอเปอร์เรชั่นที่ต้องเรียกใช้งาน ซึ่งอินเทอร์เฟซทั้งแบบ dynamic และ stub จะยอมรับเฉพาะคำร้องขอที่มีลักษณะเหมือนกันเท่านั้น หลังจากนั้น ORB จะทำการหาตำแหน่งการทำงานที่เหมาะสมของออบเจ็กต์ ส่งผ่านพารามิเตอร์และโอนการควบคุมไปยังส่วนการทำงานของออบเจ็กต์ผ่าน IDL static skeleton หรือ dynamic skeleton โดย skeleton เหล่านั้นจะเฉพาะเจาะจงไปสู่อินเทอร์เฟซและ object adapter เพื่อประมวลผลคำร้องขอที่ได้รับ และเมื่อคำร้องขอถูกประมวลผลเสร็จเรียบร้อย ส่วนควบคุมและผลลัพธ์ที่ได้จะถูกส่งกลับไปยังไคลเอนท์

เมื่อนำกลไกดังกล่าวข้างต้นไปพัฒนาแอพพลิเคชั่นสามารถอธิบายเป็นขั้นๆ ตามภาพที่ 2.3 ได้ดังนี้



ภาพที่ 2.3 กลไกการอ้างอิงออบเจ็กต์ที่ต้องการใช้งานของ ORB

1) แอพพลิเคชั่นที่พัฒนาตามสภาพแวดล้อม CORBA เริ่มการทำงาน โดยเรียก ORB และ Basic Object Adapter (BOA) ให้เริ่มทำงานผ่านฟังก์ชั่นในการเรียกใช้ ได้แก่ ORB_init() และ BOA_init() เพื่อรองรับการทำงานและให้บริการแก่เกี่ยวกับกับออบเจ็กต์

2) ORB จะทำการลงทะเบียนออบเจ็กต์และโอเปอร์เรชั่นต่างๆ ของออบเจ็กต์ผ่าน BOA ซึ่งเป็นออบเจ็กต์ประเภทหนึ่งของ ORB ทำหน้าที่ให้บริการฟังก์ชั่นพื้นฐานในการติดต่อระหว่างออบเจ็กต์ แต่ละออบเจ็กต์จะได้ object reference เป็นตัวชี้อินเทอร์เฟซที่ใช้ติดต่อสำหรับออบเจ็กต์นั้นๆ

3) เมื่อไคลเอนท์ร้องขอบริการโอเปอร์เรชั่นใดบนออบเจ็กต์ จะส่ง object reference ของออบเจ็กต์ที่ต้องการ พร้อมทั้งพารามิเตอร์ (parameter) ไปให้แก่เครื่องเซิร์ฟเวอร์หรือโปรเซสที่มีออบเจ็กต์นั้นอยู่ โดยผ่านเส้นทางการทำงานของออบเจ็กต์ ซึ่งก็คือ ORB นั่นเอง

4) ORB รับคำร้องขอของไคลเอนท์และทำการหาตำแหน่งที่อยู่ของออบเจ็กต์ แล้วสร้างเส้นทางเชื่อมการติดต่อระหว่างไคลเอนท์กับเซิร์ฟเวอร์หรือโปรเซสที่มีออบเจ็กต์นั้น หลังจากนั้นจะส่งการร้องขอการทำงาน (method invocation) ให้แก่ทั้งสองฝ่าย ตามภาพที่ 3 (Step 2)

5) หลังจาก ORB รับพารามิเตอร์ที่ไคลเอนท์ส่งมาพร้อมกับโอเปอร์เรชั่นที่ร้องขอ จะทำการแปลงพารามิเตอร์เหล่านั้นให้อยู่ในรูปแบบที่สามารถส่งผ่านโปรโตคอลที่ใช้ในระบบเครือข่ายนั้นได้ เรียกกระบวนการนี้ว่า "Marhsaling" แล้วส่งไปให้แก่เซิร์ฟเวอร์ ORB ที่อยู่ทางฝั่งเซิร์ฟเวอร์ทำการแปลงพารามิเตอร์ที่ได้รับจากรูปแบบที่ส่งผ่านเครือข่ายมาให้อยู่ในรูปของออบเจ็กต์นั้นเข้ามาทำงานต่อไปเรียกว่า "Unmarshalling" เพื่อเรียกใช้โอเปอร์เรชั่นที่ต้องการตามภาพที่ 3 (Step 2)

6) ORB ทางฝั่งเซิร์ฟเวอร์จะส่งพารามิเตอร์กลับคืนให้แก่ไคลเอนท์ เมื่อทำการประมวลผลการร้องขอดังกล่าวแล้ว จะทำการแปลงผลลัพธ์โดยการทำ Marshalling และ Unmarshalling เช่นเดียวกับที่ไคลเอนท์ส่งคำร้องขอมาดังที่เห็นในภาพที่ 3 (Step 3) เพื่อส่งผลลัพธ์กลับไปยังเครื่องไคลเอนท์

ก่อนการส่งข้อมูลสู่ระดับเครือข่าย (Network layer) ข้อมูลจะถูกจัดเรียงตามรูปแบบของ General Inter-ORB Protocol (GIOP) ซึ่งระบุรายละเอียดรูปแบบและโครงสร้างมาตรฐานการส่งข้อมูลในระดับล่าง (low-level data representation) ที่เรียกว่า Common Data Representation (CDR) เป็นการเปลี่ยนรูปแบบข้อมูลจากชนิดข้อมูล IDL เป็นรูปแบบที่ใช้ส่งในระดับเครือข่ายต่อไป และรูปแบบข้อความสำหรับการติดต่อสื่อสารระหว่าง ORBs และทำงานผ่านช่องทางการติดต่อสื่อสารที่ต้องมีการเชื่อมต่อไว้ก่อน (connection-oriented transport protocol) ส่วน Internet Inter-ORB Protocol (IIOP) จะกำหนดรายละเอียดจะส่งหรือแลกเปลี่ยน GIOP message ผ่านการติดต่อแบบ TCP/IP พร้อมทั้งระบุโปรโตคอลมาตรฐานที่ใช้สำหรับอินเทอร์เน็ต

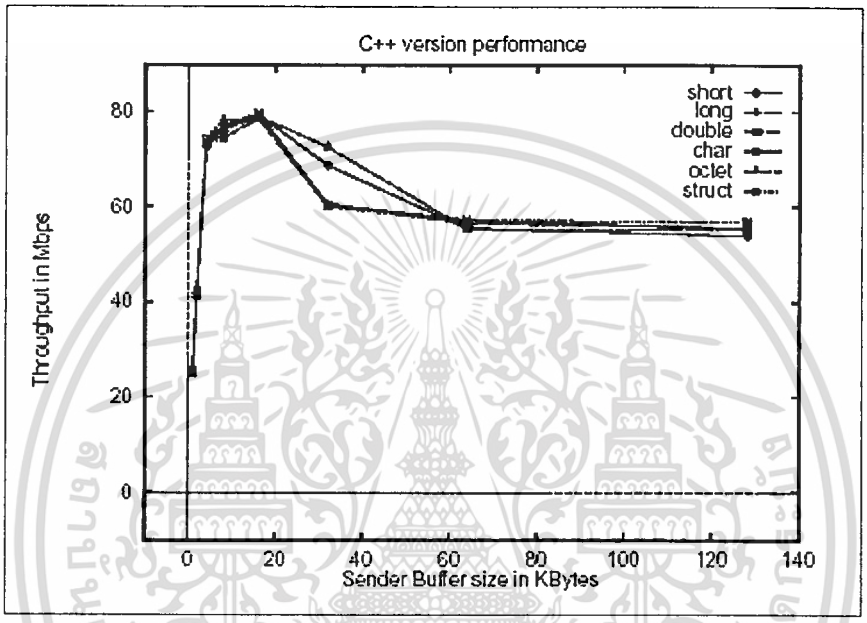
บทที่ 3
ความสามารถของกลไกการสื่อสารของคอร์บา
CORBA Object Communication Performance

เมื่อมีการนำเสนอเทคนิคใหม่มาใช้ในการพัฒนาระบบงานหรือแอพพลิเคชั่นมักมีคำถามเกิดขึ้นตามมาเสมอเช่น เทคนิคหรือเทคโนโลยีใหม่นั้นทำงานได้เร็วอย่างไร ทำงานแตกต่างจากเทคนิคที่ใช้ในปัจจุบันอย่างไร และแอพพลิเคชั่นหรือระบบงานที่พัฒนาตามเทคโนโลยีใหม่นี้ทำงานได้เร็วหรือช้ากว่าเทคโนโลยีที่มีอยู่ คำถามเหล่านี้เป็นคำถามที่เกิดขึ้นกับเทคโนโลยีเชิงวัตถุด้วยเช่นกัน โดยเฉพาะคอร์บาซึ่งเป็นมาตรฐานเทคโนโลยีเชิงวัตถุที่เปิดกว้างให้มีการศึกษาและนำไปพัฒนา ในบทนี้จะกล่าวถึงการประเมินประสิทธิภาพของแอพพลิเคชั่นที่พัฒนาด้วยกลไกการสื่อสารแบบคอร์บา เทียบกับกลไกการสื่อสารระดับล่างแบบอื่นๆ เช่น Java socket หรือ C socket และวิเคราะห์หาจุดบกพร่องของกลไกการทำงานของคอร์บา

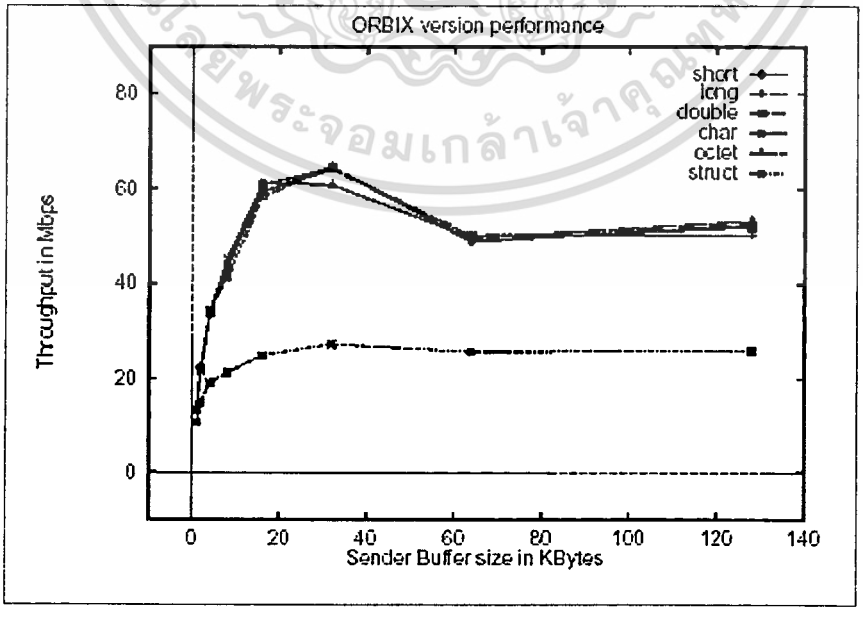
3.1 การประเมินประสิทธิภาพของกลไกการทำงานระหว่าง Java Socket C Socket และคอร์บา

กลไกสื่อกลางการติดต่อสื่อสาร (Communication middleware mechanism) ครอบคลุมกลไกการติดต่อผ่านเครือข่ายระดับล่าง (Lower-level network mechanism) ได้แก่ การติดต่อกับอินเทอร์เฟซของ C socket (socket-based C interface) และแรพเพรอะ (wrapper) ของ C++ สำหรับการติดต่อผ่านซ็อกเก็ต (C++ wrappers for socket) จนถึงกลไกการติดต่อระดับที่สูงขึ้น (Higher-level mechanism) อันได้แก่ Remote Procedure Calls (RPC) และคอร์บา การศึกษาเกี่ยวกับลักษณะประสิทธิภาพการทำงานของเทคโนโลยีวัตถุแบบกระจายที่ผ่านมาของ Gokale, A. และ Schmidt, D.C. ในปี ค.ศ. 1996 และ 1998 ได้เปรียบเทียบการทำงานระหว่างกลไกการติดต่อระดับล่างและกลไกการติดต่อสื่อสารด้วยตัวกลางบนเครือข่ายความเร็วสูงตามลักษณะข้อมูลพื้นฐานทั่วไป เช่น short, char, long, double เป็นต้น โดยประเมินปริมาณข้อมูลที่ส่งได้ต่อหนึ่งหน่วยเวลา (throughput) เปรียบเทียบกับขนาดการสำรองข้อมูล (buffering) ของ C++ socket wrapper และ CORBA ด้วยการกำหนดขนาดหน่วยสำรองข้อมูล (buffer size) ตั้งแต่ 1 กิโลไบต์, 2 กิโลไบต์, 4 กิโลไบต์, 8 กิโลไบต์, 16 กิโลไบต์, 32 กิโลไบต์, 64 กิโลไบต์ และ 128 กิโลไบต์ สำหรับส่งข้อมูลขนาด 64 เมกกะไบต์ ตามภาพที่ 3.1 และ 3.2 แสดงปริมาณข้อมูลที่ได้เทียบกับการกำหนดขนาดหน่วยสำรองข้อมูลของ C++ และ Orbix ซึ่งเป็น CORBA ORB ที่ผลิตโดยบริษัท IONA ตามลำดับ

พบว่าเมื่อขนาดหน่วยสำรองข้อมูลมีขนาด 1 ถึง 8 กิโลไบต์ ปริมาณข้อมูลที่ส่งได้ก็จะเพิ่มขึ้น จนขนาดสำรองข้อมูลอยู่ที่ 8 และ 16 กิโลไบต์จะสามารถส่งข้อมูลได้อัตราความเร็วสูงสุดที่เกือบ 80 เมกกะไบต์ต่อวินาที และจะตกลงหลังจากนั้น ส่วนประสิทธิภาพการทำงานของ CORBA ORB คือ ORBIX เป็นตัวแทนนั้น พบว่า อัตราปริมาณข้อมูลที่ส่งได้ต่อหนึ่งหน่วยเวลานั้นเพิ่มขึ้นเรื่อยๆ สำหรับขนาดสำรองข้อมูลตั้งแต่ 1 กิโลไบต์จนถึง 32 กิโลไบต์ ซึ่งเป็นจุดที่ได้อัตราการส่งข้อมูลสูงสุดประมาณ 65 เมกกะไบต์ต่อวินาที



ภาพที่ 3.1 แสดงความสามารถการทำงานของ C++



ภาพที่ 3.2 แสดงความสามารถการทำงานของ ORBIX CORBA ORB

ดังนั้นในบทนี้จึงได้ทำการศึกษาทดลองเพิ่มเติมในส่วนของข้อมูลที่เป็นลักษณะ byte array ซึ่งเป็นลักษณะข้อมูลที่คล้ายกับการส่งไฟล์ข้อมูลผ่านระบบเครือข่าย โดยอาศัยการจองหน่วยความจำสำรองในการส่งข้อมูลที่ 32 กิโลไบต์ ตามการศึกษาข้างต้น และรูปแบบการเรียกใช้งานข้อมูลของ Satoshi Hirano Yoshiji Yasu และ Hirotaka Igarashi ในปี ค.ศ. 1996 เพื่อเปรียบเทียบการทำงานระหว่าง C socket Java socket และ CORBA ORB ซึ่งในการทดลองนี้ใช้ Visibroker ของบริษัท Visigenic จำกัด

3.2 อุปกรณ์และสภาพแวดล้อมที่ใช้ในการทดสอบ

การประเมินประสิทธิภาพการทำงานของเทคโนโลยีวัตถุแบบกระจายจะทำการทดสอบกับระบบเครือข่ายที่ต้องการความเร็วในการประมวลผล เพื่อให้สอดคล้องกับการทำงานของระบบทั่วไปในปัจจุบัน ดังนั้นจึงเชื่อมต่อคอมพิวเตอร์ที่มีความเร็วสูงผ่านเครือข่าย 100Mbps Ethernet โดยคุณสมบัติของเครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบมีดังนี้

1) เครื่องเซิร์ฟเวอร์
   - เครื่องคอมพิวเตอร์ PentiumII 350MHz หน่วยความจำหลัก 128 MB
   - ระบบปฏิบัติการ Windows NT Server 4.0
   - Visibroker for Java 3.3 เป็นเครื่องมือที่มีโปรแกรม ORB ของบริษัท Visigenic
   - เครื่องมือที่เป็นตัวคอมไพล์ (complier) สำหรับการเขียนโปรแกรมด้วยภาษาจาวา คือ Just In Time complier (JIT) ใน Visual Café 3.1 Enterprise Edition ของบริษัท Symentec
2) เครื่องไคลเอนท์
   - เครื่องคอมพิวเตอร์ PentiumII 350MHz หน่วยความจำหลัก 64 MB
   - ระบบปฏิบัติการ Windows NT Workstation
   - Visibroker for Java 3.3 เป็นเครื่องมือที่มีโปรแกรม ORB ของบริษัท Visigenic
   - เครื่องมือที่เป็นตัวคอมไพล์ (complier) สำหรับการเขียนโปรแกรมด้วยภาษาจาวา คือ Just In Time complier (JIT) ใน Visual Café 3.1 Enterprise Edition ของบริษัท Symentec
3) ระบบเครือข่าย 100Mbps Ethernet เชื่อมต่อด้วยโปรโตคอล TCP/IP

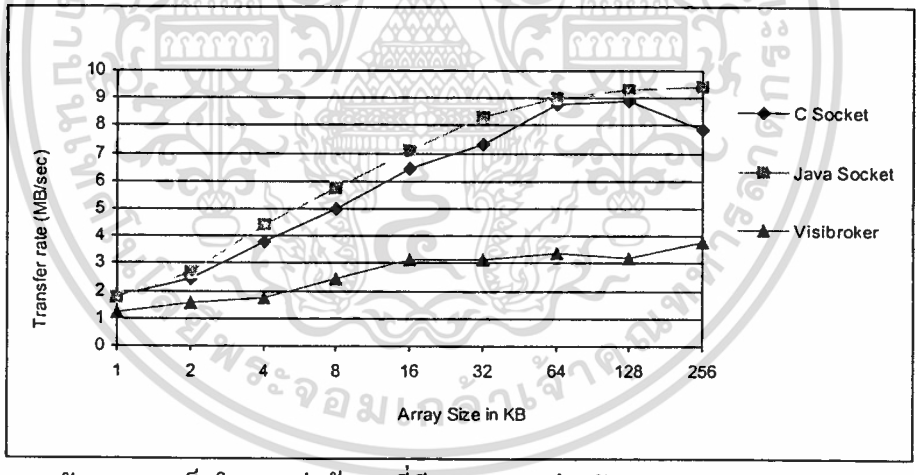
ในการทดสอบได้แยกการติดตั้งโปรแกรมการเรียกใช้งานไว้บนเครื่องคอมพิวเตอร์ที่ใช้เป็นไคลเอนท์และโปรแกรมที่ให้บริการไว้บนเครื่องเซิร์ฟเวอร์ โดยใช้ Just In Time 1.1.x

complier ของ Symentec สำหรับคอมไพล์โปรแกรมที่เขียนด้วยภาษาจาวา แล้วทำการเปรียบเทียบเวลาที่ใช้ในการส่งข้อมูลแบบทีละตัวอักษรระหว่างโปรแกรมที่เป็น ORB กับกลไกการสื่อสารระดับล่างอื่นๆ เช่น Java socket หรือ C socket เนื่องจากเป็นลักษณะข้อมูลที่คล้ายคลึงกับรูปแบบการส่งข้อมูลทั่วไป และวัดความเร็วที่ใช้ในการส่งข้อมูลไฟล์ข้อมูลในรูปของชุดลำดับของชนิดข้อมูลแบบ octet sequence ตามการกำหนดของ IDL ซึ่งเปรียบได้กับการส่ง byte array ของภาษา Java และ C

3.3 ผลการทดสอบประสิทธิภาพ

3.3.1 ผลทดสอบการส่งข้อมูลในรูปของ byte array

เมื่อมีการเรียกใช้งาน ขนาดของอาร์เรย์ที่ส่งไม่ได้กำหนดขณะที่โปรแกรมทำงาน Java socket ใช้ฟังก์ชั่นการทำงานของ BufferedOutputStream class ใน DataOutputStream class ซึ่งมีลักษณะคล้ายกับการทำงานของ CORBA และกำหนดขนาดที่ใช้รองรับข้อมูล (buffer size) 32 KB C socket นั้นถูกกำหนดการทำงานเช่นเดียวกับ Java แต่เขียนด้วยด้วยภาษาซี โดยมีการเรียงลำดับข้อมูลโดยอัตโนมัติ



ภาพที่ 3.3 แสดงอัตราความเร็วในการส่งข้อมูลที่มีขนาดแตกต่างกัน

ในรูปที่ 3.3 แสดงถึงผลการทดสอบการทำงานตามวิธีการที่กล่าวมาแล้ว พบว่า C socket ใช้ความเร็วในการส่งข้อมูลค่อนข้างสูงเช่นเดียวกันกับ Java socket โดยเฉลี่ยประมาณ 9.4 MB/sec. ในขณะที่อัตราความเร็วในการส่งข้อมูลของ Visibroker นั้นต่ำ ซึ่งชี้ให้เห็นว่าด้วยความเร็วของช่องทางการสื่อสารที่เพิ่มขึ้นในอัตราเท่าๆ กัน ประสิทธิภาพของ CORBA จะทำงานช้ากว่า low-level communication middleware เพราะต้องใช้เวลาเพิ่มขึ้นในการแปลงรูปแบบข้อมูลจาก presentation layer และการคัดลอกข้อมูล

3.3.2 ผลทดสอบการส่งข้อมูลแบบ octet stream ภายใต้สภาพแวดล้อมของ CORBA

การวัดความเร็วที่ใช้ในการส่งข้อมูลแบบ octet stream ของ Visibroker ที่เป็นโปรแกรม ORB-compliant เป็นการทดสอบการทำงานโดยภาพรวมของการส่งข้อมูลที่มีลักษณะใกล้เคียงกับการส่งไฟล์ข้อมูลผ่านระบบเครือข่าย เพื่อเปรียบเทียบขนาดของข้อมูลกับเวลาที่ใช้ ซึ่งปรากฏผลดังตารางที่ 1 Times เป็นระยะเวลาที่ใช้ในการรับส่งข้อมูล และ Rate เป็นอัตราการส่งข้อมูลที่วัดได้ต่อหนึ่งหน่วยเวลา โดยนำขนาดของ Octet Stream หารด้วย Times (*Rate = Octetstream ÷ Times*) พบว่าอัตราการส่งข้อมูลมีการเพิ่มขึ้นอย่างมีนัยสำคัญ โดยจะเพิ่มขึ้นเป็นสองเท่าเมื่อขนาดข้อมูลเพิ่มขึ้น ยิ่งจำนวนข้อมูลที่ต้องส่งเพิ่มขึ้น เวลาที่ใช้ก็เพิ่มมากขึ้น โดยมีส่วนที่เพิ่มขึ้นในการเรียกออบเจ็กต์แต่ละครั้ง ซึ่งมีผลโดยตรงจากจำนวนข้อมูลที่เพิ่มขึ้นด้วยเช่นกัน

ตารางที่ 3.1 อัตราการส่งข้อมูลแบบ octet sequence

Octetstream (KB)	Times (msec)	Rate (KB/msec)
1	0.711	1.406
256	0.661	387.272
512	0.601	851.913
1024	0.681	1572.964
2048	0.811	2525.277
4096	1.132	3618.374
8192	1.442	5680.998
16384	2.333	7022.717
32768	5.01	6540.519
65536	7.91	8285.208
131072	15.83	8279.975
262144	31.34	8364.518
524288	68.30	7676.252
1048576	136.2	7698.796

3.4 ปัจจัยที่มีผลต่อประสิทธิภาพของ CORBA (CORBA Performance Issues)

จากผลการทดสอบที่กล่าวมาข้างต้นและการศึกษาในออกแบบแอพพลิเคชั่นด้วย CORBA ของ Slama, D., Garbis, J., Russel, P (1999) และ Brose, G., Vogel, A., and Duddy, K (2001) ทำให้ทราบว่าการออกแบบออบเจ็กต์ด้วยภาษา IDL ของคอร์บานั้นมีผลต่อความสามารถในการทำงานของแอพพลิเคชั่น CORBA IDL ที่เขียนอย่างไม่มีรูปแบบสามารถทำให้ลดความสามารถในการทำงานของระบบและทำให้การขยายขนาดของระบบเป็นไปได้ยากด้วย การออกแบบ CORBA IDL Interface แตกต่างจากการออกแบบ interface สำหรับรูปแบบเชิงวัตถุแบบเดิมที่ทุกวัตถุอยู่กับ

ส่วนการทำงานเพียงส่วนเดียว การเรียกใช้งานคำสั่งอย่างง่ายที่อยู่หน่วยความจำ (in-memory method invocation) แสดงผลการทำงานได้ดีกว่าการเรียกใช้งานที่อยู่ระยะไกล (remote method invocation) แบบ CORBA โดยเปรียบเทียบได้จากขนาดและลำดับของระบบ ในมาตรฐานระบบเครือข่ายเฉพาะ (Local Area Network : LAN) จำนวน remote method invocation ที่ถูกเรียกใช้ย่อมมีผลกระทบต่อประสิทธิภาพการทำงานของระบบโดยรวม ส่วนระบบเครือข่ายที่มีความเร็วสูง (high-speed network) สิ่งที่ต้องเสียไประหว่างการทำมาร์แชลลิ่ง (marshalling) จะมีผลต่อระบบอย่างเห็นได้ชัด โดยปัจจัยที่มีเกี่ยวข้องกับผลการทำงานแบ่งออกเป็น 3 ส่วนใหญ่ ดังนี้

3.4.1 จำนวน remote method invocation ที่เกิดขึ้นภายในระบบ

การร้องขอใช้งานแต่ละครั้งที่ส่งผ่านบนการเชื่อมต่อเครือข่ายก่อให้เกิดปริมาณงานที่เพิ่มขึ้นซึ่งแฝงอยู่ภายในระบบเครือข่าย (net latency) ทำให้เกิดความหน่วง (delay) ภายในระบบ ความหน่วงที่เกิดขึ้นนี้สามารถวัดได้เวลาที่ใช้ในแต่ละครั้งเมื่อมีการเรียกใช้งาน CORBA แบบระยะไกล โดยเฉพาะระบบเครือข่าย TCP/IP ปัจจัยที่มีผลคงต่อเนื่องนี้เป็นผลมาจากจำนวนที่มีการเรียกใช้งานแบบระยะไกลมากกว่าปริมาณข้อมูลที่ถูกส่งพร้อมการร้องขอแต่ละครั้ง การค้นหาออบเจ็กต์ที่อยู่ฝั่งเซิร์ฟเวอร์จัดเป็นอีกปัจจัยหนึ่งที่ต้องเสียไปสำหรับการเรียกใช้งานแบบระยะไกลแต่ละครั้ง และส่วนเกินที่เกิดจากขาดการสนับสนุนการเรียกใช้งานพร้อมกันของเครื่องไคลเอนท์เมื่อมีการเรียกใช้งานผ่านระบบเครือข่ายจะยิ่งทำให้มีความสูญเสียเพิ่มขึ้น ซึ่งสังเกตได้จากการใช้เทคนิคมัลติเธรดดิ่ง (multithreading) อันแสดงถึงส่วนเกินจากการทำงานที่เพิ่มสูงขึ้น

3.4.2 ปริมาณข้อมูลที่ถูกส่งไปพร้อมกับการเรียกใช้งานแบบระยะไกลในแต่ละครั้ง

ถ้าสิ่งที่ถูกส่งออกไป (message) มีขนาดใหญ่เกิดไป อัตราปริมาณงานที่ได้ก็จะลดลงเนื่องมาจากทรัพยากรส่วนหนึ่งของระบบมารองรับข้อมูลของ TCP (TCP buffering) และเพิ่มการทำงานให้แก่ระบบด้วย ระบบเครือข่ายแต่ละแบบจะกำหนดขนาดการสำรองทรัพยากรของระบบแตกต่างกัน ส่วนสำคัญที่มีผลต่อความสูญเสียที่เกิดขึ้นทางฝั่งเซิร์ฟเวอร์คือวิธีการจัดส่งข้อมูลที่ยังไม่มีประสิทธิภาพเพียงพอ ส่วนหนึ่งมาจากการยึดติดกับรูปแบบที่กำหนดขึ้นเฉพาะของโปรโตคอล GIOP 1.0 และอัลกอริธึม (algorithm) ที่ใช้ทำการจัดเรียงและการจองพื้นที่ (alignAndReserve) นั้นกำหนดขนาดเริ่มต้นไว้น้อย อีกทั้งวิธีการที่ใช้ขยายขนาดพื้นที่ที่สำรองไว้ยังไม่ดีพอ นอกจากนี้ไม่มี ORB Compliant ใดที่มีการสำรองภายในเพื่อใช้สำหรับการเขียนและอ่านเมื่อมีการส่งข้อมูลบนเครือข่าย

3.4.3 ลักษณะข้อมูลที่ต่างชนิดกันมีผลต่อกระบวนการจัดรูปแบบข้อมูล

ข้อมูลต่างชนิดกันมีผลต่อส่วนเกินที่เพิ่มขึ้นอย่างต่อเนื่องสำหรับการจัดแปลงข้อมูล (marshalling/demarshalling) เพราะการจัดแปลงข้อมูลเป็นกระบวนการของการแปลข้อมูลจากรูปแบบที่ใช้นำเสนอบนโปรแกรมให้อยู่ในรูปแบบที่ส่งผ่านเครือข่าย และในทางกลับกันจะแปลงจากรูปแบบที่ส่งผ่านเครือข่ายไปเป็นรูปแบบที่ใช้นำเสนอต่อผู้ใช้ ส่วนเกินเหล่านี้ยังแตกต่างกันไปตาม ORB ที่ใช้งาน ภาษาที่ใช้ในการเขียนโปรแกรม ระบบปฏิบัติการ อุปกรณ์คอมพิวเตอร์ และประเภทเครือข่าย ซึ่งสามารถอธิบายแยกตามชนิดข้อมูลตามการศึกษาของ กราฟภาพที่ 3.1 แสดงการเปรียบเทียบส่วนเกินที่เพิ่มขึ้นระหว่างการจัดแปลงข้อมูลกับชนิดข้อมูลที่แตกต่างกัน ได้แก่

- Octet ส่วนเกินที่เพิ่มขึ้นมีจำนวนน้อยมาก ORB คำนึงถึงการเรียกลำดับไบต์ของข้อมูลแบบ Octet น้อยมาก

- Struct ส่วนเกินที่เพิ่มขึ้นสำหรับข้อมูลแบบ Struct มีปริมาณน้อย ทั้งนี้ขึ้นอยู่กับชนิดข้อมูลของสมาชิกที่ประกาศไว้ใน Struct

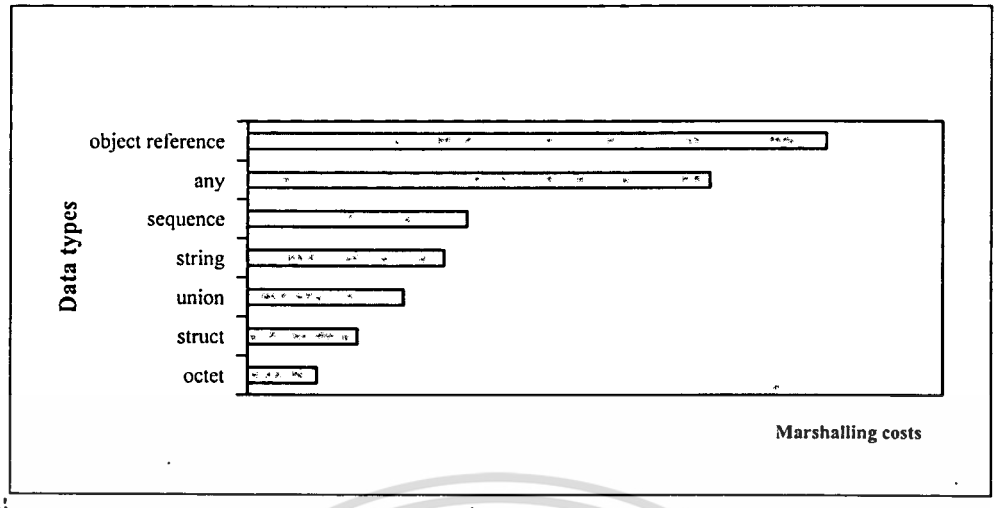
- Union ส่วนเกินที่เพิ่มขึ้นมากกว่าข้อมูลแบบ Struct ไม่มากนัก ส่วนเกินที่เกิดขึ้นในการจัดแปลงข้อมูลแบบ Union จะขึ้นอยู่กับชนิดข้อมูลที่อยู่ประกาศไว้ใน Union คล้ายกับการประกาศ Struct

- String ข้อมูลชนิด String เป็นตัวแปรข้อมูลที่มีขนาดไม่จำกัดซึ่งโดยทั่วไปจะทำให้เกิดการทำงานด้านการจัดการหน่วยความจำเพิ่มขึ้น ยกตัวอย่างเช่น เมื่อมีการคัดลอกข้อมูลชนิด sequence ของ string ORB จะไม่จัดสรรหน่วยความจำสำหรับข้อมูลชนิด string ไว้ทันที อย่างไรก็ตามต้องมีการจัดสารหน่วยความจำสำหรับแต่ละ string

- Sequence ข้อมูลชนิดนี้ไม่เพียงเป็นตัวแปรข้อมูลที่ไม่จำกัดขนาด ยังสามารถเก็บชนิดข้อมูลที่ซับซ้อนได้ ส่วนเกินที่เกิดขึ้นระหว่างการจัดแปลงข้อมูลสำหรับ sequence ขึ้นอยู่กับจำนวนของชนิดข้อมูลที่ประกาศไว้ใน sequence

- Any เป็นชนิดข้อมูลพื้นฐานที่ซับซ้อนที่สุดและคล่องตัว ต้องมีการจัดการปรับเปลี่ยนโค้ดที่ยืดหยุ่นระหว่างการจัดแปลงข้อมูล ส่วนเกินที่เกิดขึ้นนั้นขึ้นอยู่กับภาษาที่ใช้ในการเขียนโปรแกรมและการพัฒนา ORB

- Object reference เป็นชนิดข้อมูลแบบนามธรรม มีกลไกการจัดกลุ่มข้อมูล เช่น หมายเลขเครื่องแม่ข่าย (host address) และตัวจำแนกออบเจ็กต์ (object identifier) และมีวิธีการบ่งชี้การติดต่อเครือข่ายที่สามารถเข้าใช้งานได้ ส่วนเกิดที่เกิดขึ้นสำหรับการจัดแปลง object reference สูงทั้งฝั่งไคลเอนท์และเซิร์ฟเวอร์ เพราะมีการสร้างตัวแทนออบเจ็กต์ (proxy object) ทุกครั้งที่ถูกเรียกใช้



ภาพที่ 3.4 แสดงการเปรียบเทียบระหว่างชนิดข้อมูลกับ marshalling costs

ปัจจัยเหล่านี้จะลดลงหากมีการจัดการที่เหมาะสมตั้งแต่ช่วงการออกแบบ โดยจะทำให้ความหน่วงในการประมวลผลเกิดขึ้นเมื่อมีการส่งข้อมูล ด้วยเหตุนี้หากการออกแบบระบบโดยเฉพาะการกำหนดรูปแบบ IDL ที่ใช้งานภายในระบบจะช่วยลดจำนวนการส่งข้อมูลบนเครือข่ายซึ่งเกิดขึ้นจากการแลกเปลี่ยนข้อมูลกันระหว่างส่วนประกอบเชิงกระจาย จะทำให้ความสามารถและประสิทธิภาพของระบบดีขึ้น

บทที่ 4
เทคนิคการออกแบบไอดีแอลอินเทอร์เฟซ
(CORBA IDL Design Techniques)

จากการศึกษาผลงานวิจัยที่เกี่ยวข้องและการทดสอบประสิทธิภาพการทำงานของ CORBA ในบทที่ 3 แสดงข้อด้อยของระบบไคลเอนท์/เซิร์ฟเวอร์ที่พัฒนาด้วยสถาปัตยกรรมแบบ CORBA ด้วยเหตุนี้จึงมีการปรับปรุงประสิทธิภาพของ CORBA ด้วยหลากหลายเทคนิคและวิธีการ ขึ้นอยู่กับมุมมองของผู้ศึกษาวิจัย การรับส่งข้อมูลตามสถาปัตยกรรมการประมวลผลหลายระดับ (N-tier architecture) อย่าง CORBA ต้องพิจารณาถึงการจัดการความเสี่ยงในหลายด้าน อาทิเช่น ปัจจัยที่แฝงในเครือข่าย การตอบสนองของระบบ การใช้ประโยชน์จากบริการที่มีอยู่ การจัดการปริมาณงาน การเก็บซ่อนแบบกระจาย (distributed caching) การจัดเก็บวัตถุแบบกระจายที่ไม่ได้ใช้งาน (distributed garbage collection) และการจัดการระบบ เทคนิคการออกแบบอินเทอร์เฟซของออบเจ็กต์นับเป็นส่วนหนึ่งต้องพิจารณาในการสร้างส่วนประกอบเชิงวัตถุแบบกระจายสำหรับการพัฒนาระบบงาน รายละเอียดในบทนี้จะเทคนิคที่ใช้ในการออกแบบและเขียน CORBA IDL Interface เนื่องจาก IDL ถือว่ามีบทบาทสำคัญในการกำหนดขนาดของข้อมูลที่จะถูกส่งผ่านระบบเครือข่ายต่อไป

4.1 แบบแผนตัวทำซ้ำ (Iterator Pattern)

แบบแผนตัวทำซ้ำเป็นแบบแผนการออกแบบการจัดการข้อมูลขนาดใหญ่ทั่วไป โดยอนุญาตให้มีการเข้าถึงข้อมูลขนาดใหญ่ในลักษณะชุดข้อมูลขนาดเล็กหลายๆ ชุดมากกว่าเป็นชุดข้อมูลขนาดใหญ่ตามที่มีการร้องขอ วิธีการนี้ทำให้หลีกเลี่ยงการจองหน่วยความจำขนาดใหญ่ทั้งทางฝั่งไคลเอนท์และเซิร์ฟเวอร์ เครื่องไคลเอนท์สามารถประมวลผลข้อมูลแต่ละส่วนและลบข้อมูลดังกล่าวทิ้งก่อนมีการร้องขอต่อไป อีกทั้งฝั่งไคลเอนท์อาจยกเลิกการร้องขอข้อมูลส่วนที่เหลือและทำลายตัวทำซ้ำนั้น เพื่อลดการจองหน่วยความจำ และยังสามารถหลีกเลี่ยงการประมวลผลข้อมูลที่อาจจะไม่ถูกเรียกใช้ทางฝั่งเซิร์ฟเวอร์ด้วย แบบแผนตัวทำซ้ำถูกนำมาใช้ในเกณฑ์การพัฒนา CORBA ในแง่ของการบริการระบุชื่อ (Naming service) และการบริการแลกเปลี่ยนออบเจ็กต์ (Trading service)

Khanh Chau ได้นำเสนอตัวอย่างการนำแบบแผนตัวทำซ้ำไปใช้ในการสอบถามข้อมูลสินค้าที่ผลลัพธ์ที่ได้นั้นมีขนาดใหญ่ ซึ่งผลที่ได้เป็นชุดข้อมูลของตามจำนวนสินค้าที่เรียกดู ถ้าผลลัพธ์ประกอบด้วยสินค้าจำนวน n จำนวน ดังนั้นข้อมูลสินค้า n จำนวนเหล่านั้นจะถูกแปลง

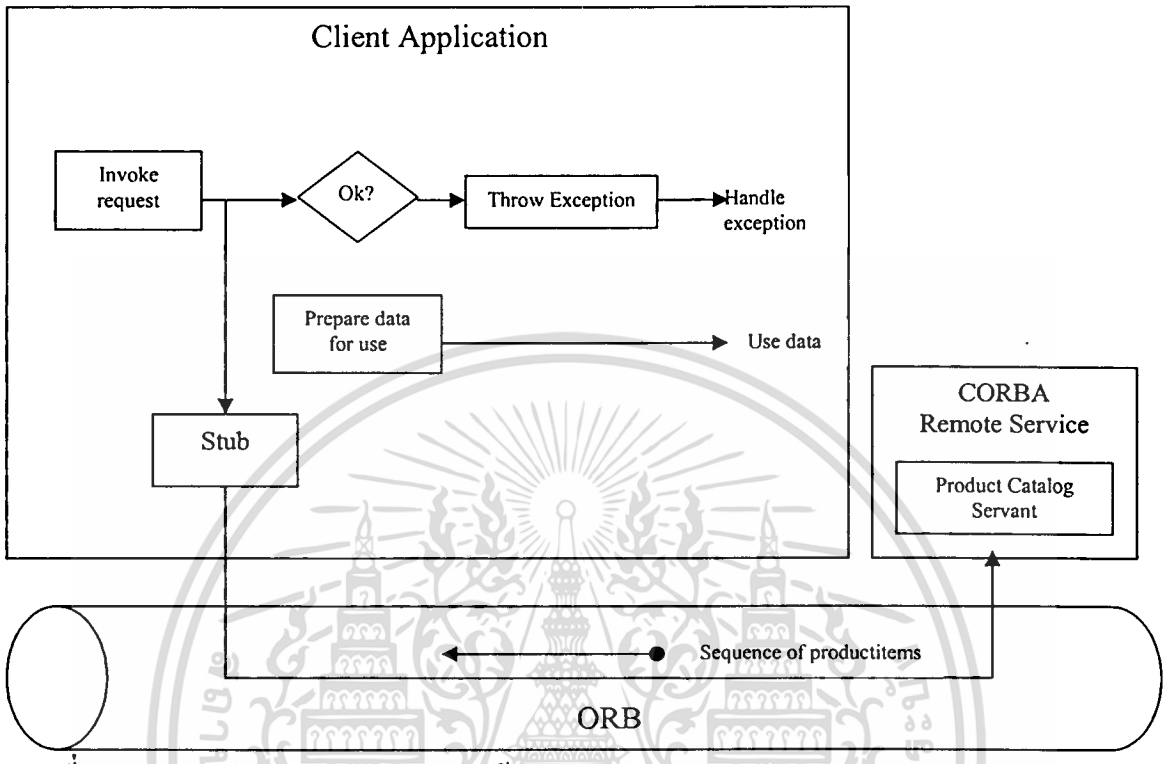
ให้อยู่ในรูปพร้อมที่จะส่งผ่านเครือข่าย อินเทอร์เฟซของสินค้าตามการเขียน IDL จะประกอบด้วยโอเปอร์เรชั่นเพียงโอเปอร์เรชั่นเดียวคือ getProductItems ซึ่งกำหนดการส่งคืนข้อมูลในรูปชุดลำดับของสินค้า ตามที่แสดงในภาพที่ 4.1 เมื่อไคลเอนท์ต้องการเรียกดูข้อมูลสินค้า จะมีกระบวนการทำงานตามภาพที่ 4.2 ดังนี้ ตัวจัดการคำร้องทางฝั่งไคลเอนท์ (client proxy) จะจัดเตรียมคำร้องขอนั้นให้อยู่ในรูปแบบมาตรฐานแล้วส่งให้ stub เพื่อส่งคำร้องขอดังกล่าวไปให้ Product Catalog Servant ที่อยู่ทางฝั่งเซิร์ฟเวอร์ประมวลผลตามคำร้องขอ โดยข้อมูลที่ส่งกลับจะอยู่ในรูปของ sequence of ProductItems ดังนั้นหากมีการเรียกดูข้อมูลสินค้า n จำนวน จะมีการติดต่อระหว่างไคลเอนท์และเซิร์ฟเวอร์ n+1 ครั้ง

```
module productcatalog
{
        struct ProductItem {
                int     productID;
                string  productName;
                float   productPrice;
                ...
        };
        typedef sequence<ProductItem> ProductItemList;

        interface ProductCatalog {
                ProductItemList getProductItems(in string group,
                        in string category, in string status)
                  raises (SomeRemoteException);
        };
};
```

ภาพที่ 4.1 ตัวอย่าง IDL สำหรับกำหนดออบเจ็กต์สินค้า

เมื่อนำหลักการ Iterator Pattern มาใช้ปรับปรุงการเขียน IDL จะต้องมีการกำหนดรูปแบบ IDL ให้สามารถแบ่งข้อมูลขนาดใหญ่ออกเป็นส่วนย่อยๆ ให้เหมาะสมกับตัวทำซ้ำที่ใช้ดึงข้อมูล โดยกำหนดอินเทอร์เฟซของตัวทำซ้ำพื้นฐาน (base iterator interface) ซึ่งระบุพฤติกรรมของตัวทำซ้ำระยะไกล (remote iterator) ด้วย IDL โมดูล *iterator* ในภาพที่ 4.3 ประกอบด้วยอินเทอร์เฟซ BaseIterator และ BaseListIterator เพื่อใช้อ้างอิงสำหรับตัวทำซ้ำระยะไกลอื่นๆ เช่น ProductIterator ในภาพที่ 4.4 เป็นตัวอย่าง IDL ที่มีการอ้างอิงถึงตัวทำซ้ำพื้นฐานในโมดูล ProductCatalog โดยผลลัพธ์ที่ได้จากประมวลผลจะถูกส่งกลับเป็น ProductIterator แทนที่จะเป็นชุดของ ProductItemList เมื่อนำไปพัฒนาภายในโปรแกรมทางฝั่งเซิร์ฟเวอร์ ProductIterator มีบทบาทจัดการจำนวนข้อมูลที่จะโอนย้าย ProductIterator จะปรากฏขึ้นเมื่อมีการเรียกใช้ฟังก์ชั่น getProductItems(...) ข้อมูลที่อยู่ภายในจะขึ้นอยู่กับผลลัพธ์ที่ได้จากค้นหา นอกจากนี้ยังต้องดัดแปลงฟังก์ชั่น getProductItems(...) ในโมดูล

ProductCatalog ให้ส่งคืนผลลัพธ์ด้วยการอ้างอิงออบเจ็กต์ ProductIterator แทนการส่งชุดของ ProductsItem



ภาพที่ 4.2 กระบวนการมาตรฐานจัดการคำร้องขอ (Standard Procedure to Handle Requests)

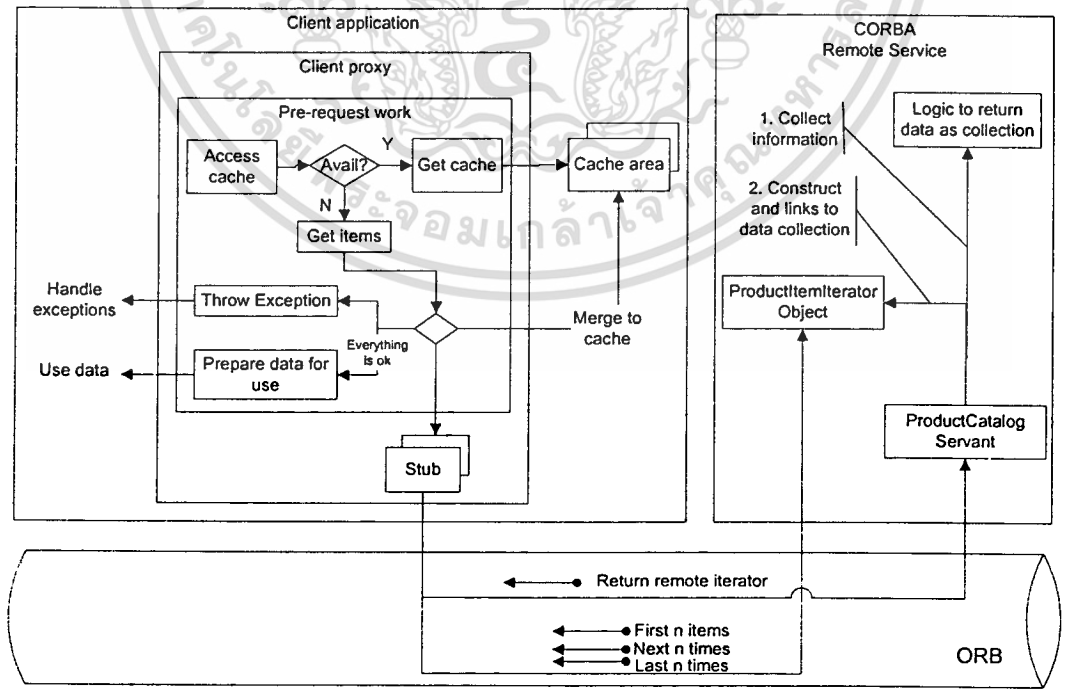
```
module  iterator
{
    interface BaseIterator {
        boolean  hasNext( )  raises (SomeRemoteException);
        short count( ) raises (SomeRemoteException);
    };

    interface  BaseListIterator {
        boolean hasPrevious( ) raises (SomeRemoteException);
        short previousIndex( ) raises (SomeRemoteException);
        boolean hasNext( )  raises (SomeRemoteException);
        short nextIndex( ) raises (SomeRemoteException);
        short count( ) raises (SomeRemoteException);
    };
};
```

ภาพที่ 4.3 แสดงตัวอย่างการเขียน Base Iterator ใน IDL

```
module productcatalog
{
        struct ProductItem {
                int       productID;
                string    productName;
                float     productPrice;
                ...
        };
        typedef sequence<ProductItem> ProductItemList;

        interface ProductIterator : iterator :: BaseListIterator {
                ProductItemnext( ) raises (SomeRemoteException);
                ProductItemList nextBlock(in short size ) raises (SomeRemoteException);
                ProductItemprevious( ) raises (SomeRemoteException);
                ProductItemList previousBlock (in short size ) raises (SomeRemoteException);
        };

        interface ProductCatalog {
                ProductIterator getProductItems(in string group,
                        in string category, in string status)
                  raises (SomeRemoteException);
        };
};
```

ภาพที่ 4.4 ตัวอย่าง IDL ใหม่สำหรับกำหนดออบเจ็กต์สินค้า

ภาพที่ 4.5 แสดงการติดต่อระหว่างส่วนประกอบต่างๆ เมื่อโปรแกรมทางฝั่งไคลเอนท์ระบุคำร้องขอค้นหาข้อมูลครั้งแรก พร็อกซี่ออบเจ็กต์ (proxy object) ที่อยู่ฝั่งไคลเอนท์จะทำงานแยกแต่ละคำร้องขอโดยตรวจเทียบกับข้อมูลที่สำรองไว้ ถ้าไม่พบข้อมูลที่ต้องการจะส่งคำร้องขอไปยังออบเจ็กต์ทางฝั่งเซิร์ฟเวอร์เพื่อเรียกดูข้อมูลต่อไป



ภาพที่ 4.5 Remote Iterator in IDL

4.2 ตัวระบุออบเจ็กต์ลำดับรอง (Secondary Object Identifier)

ในสภาพแวดล้อมแบบกระจาย แบบแผนการเข้าถึงออบเจ็กต์ที่ให้บริการถือเป็นอีกปัจจัยหนึ่งที่สำคัญ รูปแบบการเข้าถึงออกเจ็กต์มีได้หลากหลายวิธี วิธีหนึ่งที่จะกล่าวถึงต่อไปนี้คือ ตัวระบุออบเจ็กต์ลำดับรอง (Secondary Object Identifier) [13,15,21] กรณีที่มีการส่งคืนค่าอ้างอิงออบเจ็กต์ (Object reference) จำนวนมากกลับไปยังเครื่องไคลเอนท์ตามที่มีการร้องขอมาด้วยการส่งเพียงครั้งเดียวนั้น ส่วนใหญ่เป็นไปได้ที่จะทำให้สมรรถภาพของระบบไม่ดีนัก สาเหตุที่ทำให้สมรรถภาพของระบบลดลงเพราะส่วนเกินที่เสียไประหว่างการทำมาร์แชลลิ่งของ object reference เมื่อการส่งผ่านค่า object reference จำนวนมากมักจะตามมาด้วย remote method invocation จำนวนมากเช่นกัน เนื่องด้วยตัวรับหรือรีซีฟเวอร์ (receiver) จะแยกรับ object reverence แต่ละตัว ยกตัวอย่างเช่น IDL สำหรับการจัดการหุ้นแบ่งออกเป็น 2 ส่วน คือ StockWatch.idl ซึ่งอธิบายโครงสร้างข้อมูลของหุ้นที่ต้องการ และ Portfolio.idl ซึ่งเป็นออบเจ็กต์จัดการข้อมูลหุ้น ดังแสดงในภาพที่ 4.6 และ 4.7 ตามลำดับ ในอินเทอร์เฟซ Portfolio นั้นมีการอ้างอิงอินเทอร์เฟซของ Stock ที่อยู่ใน StockWatch.idl เมื่อมีการเรียกดูข้อมูลหุ้นในแต่ละครั้ง จะมีการเรียกใช้งานเมธอด 2 เมธอด คือ getStock() ที่อยู่ใน Portfolio และ getSymbol ที่อยู่ใน StockWatch ดังแสดงการเรียกใช้งานออบเจ็กต์ในภาพที่ 4.8 พบว่าหากมีการเรียกดูข้อมูลหุ้นจำนวน N หุ้น จะทำให้เกิดคำร้องขอทั้งสิ้น 1+2*N ครั้ง เพราะฉะนั้นหากเพิ่มจำนวนคำร้องขอมากขึ้นผ่านการทำงานดังภาพที่ 4.8 จะทำให้เวลาที่ใช้ในการส่งคืนข้อมูลช้าลง

```
// IDL : StockWatch Example
module StockWatch {
    // Basic type definitions
    typedef string Date;
    typedef float Money;
    typedef string Symbol;
    typedef sequence<Symbol> SymbolSeq;
    // Exceptions
    exception rejected { string m_reason; };
    // Structs
    struct PriceInfo {
      Money m_price;
      Date m_when;
    };
    typedef sequence<PriceInfo> PriceInfoSeq;
    // Interfaces
    interface Stock {
      Symbol getSymbol();
      string getdescription() raises(rejected);
      Money getCurrentPrice() raises(rejected);
      PriceInfoSeq getRecentPrices ()  raises (rejected);
    };
    interface StockWatch {
      SymbolSeq getsymbols () raises (rejected);
      Stock  getStockBySymbol (in Symbol asymbol)  raises (rejected);
    };
};
```

ภาพที่ 4.6 IDL Interface ของ StockWatch

```
// CORBA IDL
module Portfolio {
  #include <StockWatch.idl>
  interface Holding;
  typedef sequence<Holding> HoldingSeq;
  interface Portfolio {
    Money getCurrentValue ( );
    HoldingSeq getHoldings ( );
  };
  interface Holding {
    unsigned long getNumberOfShares ( );
    Stock getStock ( );
  };
};
```

ภาพที่ 4.7 IDL Interface ของ Portfolio

ด้วยเหตุนี้จึงมีการนำตัวระบุออบเจ็กต์ลำดับรองมาใช้แทนการส่งคืนค่า object reference จำนวนมากพร้อมกันคราวเดียว และจัดเตรียมเมธทอด (method) ที่จะให้ไคลเอนท์เข้ามาแลกเปลี่ยนCORBA object reference กับ Secondary OID ซึ่ง Secondary object reference คือตัวแปรตามชนิดข้อมูลของ IDL ที่อ้างอิงออบเจ็กต์ตัวอื่น เนื่องจากอินเทอร์เฟซ Portfolio ถือเป็นออบเจ็กต์ที่ทำหน้าที่จัดการส่งคืนข้อมูล จึงมีดัดแปลงให้ Portfolio อ้างอิงและเรียกใช้ออบเจ็กต์ที่กำหนดไว้ใน StockWatch โดย IDL ใหม่ของ Portfolio ตามภาพที่ 4.8 ได้มีการตัดชุดข้อมูลของ Holding ชื่อ HoldingSeq (typedef sequence<Holding> HoldingSeq;) และฟังกชั่น getHoldings() ออกไป แล้วแทนที่ฟังก์ชั่น getsymbols() ซึ่งอ้างถึง SymbolSeq ที่อยู่ใน StockWatch และเพิ่มฟังก์ชั่น getHoldingBySymbol(...) การเปลี่ยนแปลง IDL ดังกล่าวจะทำให้สามารถเรียกดูชุดข้อมูลหุ้นด้วยคำร้องขอเพียงครั้งเดียว ทำให้ลดจำนวนคำร้องขอ ปริมาณข้อมูลที่ต้องส่ง และชนิดของข้อมูลที่ต้องส่งด้วย

```
module Portfolio {
  #include <StockWatch.idl>
  interface Holding;
  interface Portfolio {
    Money getCurrentValue ( );
    SymbolSeq getsymbols ( );
     Holding getHoldingBySymbol (in Symbol asymbol);
  };
  interface Holding {
    unsigned long getNumberOfShares ( );
    Stock getStock ( );
  };
};
```

ภาพที่ 4.8 IDL ที่มี secondary object identifier ของ Portfolio

4.3 การจำกัดขนาดข้อมูล (Data Size Blocking)

Kashima (1999) นำเสนอแนวทางสำหรับการสร้างระบบงานองค์กรบนเว็บด้วยเทคโนโลยีเชิงวัตถุ CORBA โดยเสนอวิธีการออกแบบโครงสร้างแอพพลิเคชั่นและการออกแบบการจัดการระบบ ได้เปรียบเทียบอัตราการส่งข้อมูลที่ต้องการสำหรับ object reference และเวลาที่ใช้ในการส่งข้อมูลแบบ octet string ที่มีเนื้อหาข้อมูลแบบเดียวกันนั้น จะถูกส่งได้ 3 แบบด้วยกัน ได้แก่

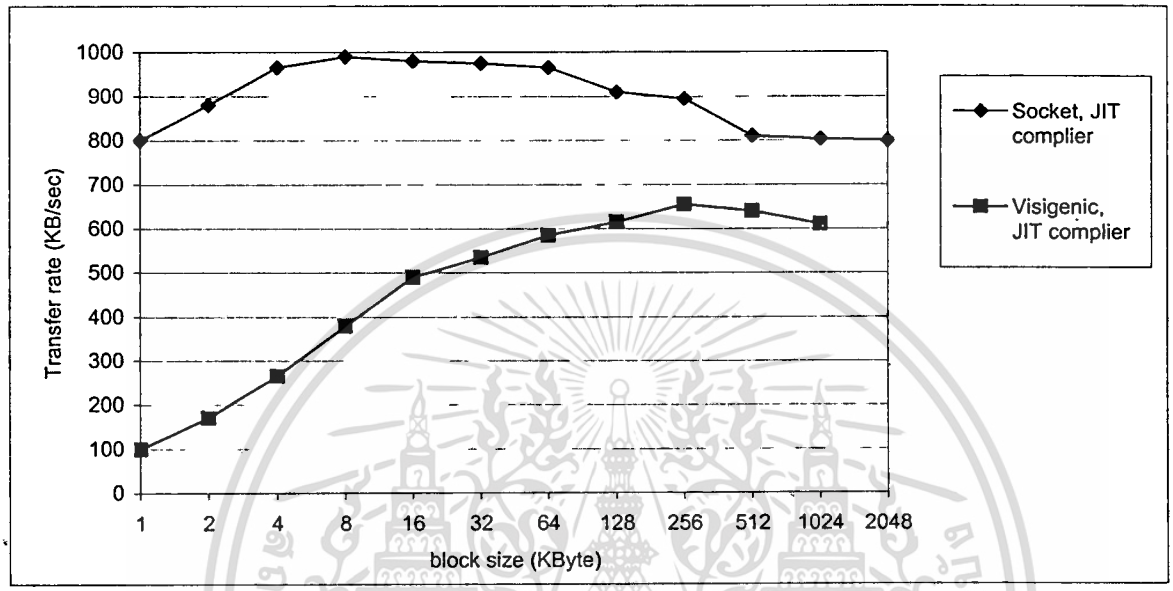
- การส่งค่าแบบอ้างอิงออบเจ็กต์ ซึ่ง 1 Kilobyte (KB) เท่ากับข้อมูล string ขนาด 4 ไบต์ (byte) หรือเรียกว่า คลาส (class)
- การส่งข้อมูลแบบแรกด้วยข้อมูลชนิด sequence (class string)
- กานส่งข้อมูลแบบแรกในรูปของ octet sequence (byte string)

ผลเปรียบเทียบมิได้มีนัยว่าวิธีการส่งแบบใดเป็นวิธีที่ดีที่สุด แต่แสดงผลที่ควรพิจารณา ดังนี้

- object reference ใช้เวลาในการส่งข้อมูลมากกว่าแบบการส่งข้อมูลแบบ byte string ถึง 10 เท่า เพราะเป็นการส่งปริมาณข้อมูลที่มีทั้งหมด
- การส่งข้อมูลด้วย class string ไม่ได้ปรับปรุงเวลาที่ใช้ในการส่งข้อมูล เพราะเป็นการส่งผ่านข้อมูลแบบ sequence ของ object reference เพื่อไม่ให้มีการเปลี่ยนแปลงจำนวนการเรียกใช้งาน
- การวนรอบ (turnaround) ของการเรียกใช้งานออบเจ็กต์แต่ละครั้งเป็นปัจจัยที่มีผลกระทบต่อประสิทธิภาพ หมายถึงประสิทธิภาพจะลดลงตามจำนวนการเรียกใช้งานที่เพิ่มขึ้น
- เป็นไปได้ที่จะคาดการณ์จำนวนการเรียกใช้ที่จะเกิดขึ้นได้จากรูปแบบการเขียน IDL และตรรกะที่ใช้ในการเขียน ดังนั้นประสิทธิภาพสามารถที่จะประเมินล่วงหน้าได้ถูกต้องอย่างมีนัยสำคัญ

Miron and Taylor (1996) ได้ทำการศึกษาลักษณะเฉพาะของประสิทธิภาพของ Java Object Request Broker ด้วยการวัดอัตรการส่งข้อมูลแบบ raw byte ด้วยวิธี SII ซึ่งข้อมูลดังกล่าวจะไม่มีการกำหนดชนิดข้อมูลพื้นฐาน (primitive data type) บนเครือข่ายที่แตกต่างกันระหว่างการส่งข้อมูลผ่าน socket กับการใช้ ORB เพื่อแสดงการติดขัดที่เกิดขึ้นภายใต้ ORB เมื่อมีการส่งข้อมูลขนาดใหญ่ พบว่า อัตราการส่งที่สูงเป็นผลจากการส่งข้อมูลที่มีขนาดใหญ่ทั้งหมดเพียงครั้งเดียว ดังนั้นการแบ่งข้อมูลขนาดใหญ่ออกเป็นชุดของข้อมูลย่อยๆ แสดงผลอย่างมีนัยสำคัญต่ออัตราการส่ง ประสิทธิภาพของ ORB จะไม่ดีนักเมื่อขนาดบล็อคข้อมูลย่อย (block size) มีขนาดเล็กจนเกินไป แต่จะดีขึ้นเมื่อขนาดบล็อคเพิ่มขึ้น ผลการทดสอบอัตราการส่งข้อมูลต่อหนึ่งหน่วยเวลาในภาพที่ 4.9 อัตราการส่งข้อมูลสูงสุดที่วัดได้เมื่อชุดข้อมูลย่อยมี

ขนาดประมาณ 256 กิโลไบต์ อย่างไรก็ตามขนาดข้อมูลย่อยยังมีการผันผวนอยู่ในแต่ละอัตราการส่งข้อมูลที่ได้ ทั้งนี้เนื่องจากอัลกอริธึมนาเกิล (Nagle's algorithm) ที่ใช้ในการเรียงข้อมูลซ้อนกันใน TCP เพื่อทำการสำรองและรวมกลุ่มข้อมูล (packet) ขนาดเล็กเข้าด้วยกันเพื่อทำการส่งข้อมูลไปยังระดับต่อไป



ภาพที่ 4.9 แสดงขนาดข้อมูลที่มีผลต่ออัตราความเร็วในการส่งข้อมูล

4.4 เทคนิคฟาสต์ไอดีแอล

เทคนิคฟาสต์ไอดีแอลเป็นเทคนิคการเขียนอินเทอร์เฟซของ CORBA IDL ที่ผู้วิจัยนำเสนอขึ้น โดยผสมผสานการออกแบบออบเจ็กต์ด้วยการระบุออบเจ็กต์ลำดับสอง (Secondary Object Identifier) แบบแผนการทำซ้ำ (Iterator Pattern) และการกำหนดขนาดข้อมูลย่อย (Data Blocking) ตามที่กล่าวมาแล้วข้างต้นมาใช้ร่วมกัน ซึ่งมีกระบวนการออกแบบดังนี้

4.4.1 การออกแบบตัวระบุออบเจ็กต์ลำดับสอง

ส่วนใหญ่ตัวระบุออบเจ็กต์ลำดับสองจะถูกกำหนดให้เป็นออบเจ็กต์ที่ทำหน้าที่จัดเตรียมข้อมูลให้อยู่ในรูปแบบที่ต้องการส่งหรือแสดงผลให้กับทางฝั่งไคลเอนท์ เริ่มด้วยการกำหนดอินเทอร์เฟซของออบเจ็กต์ที่ต้องการเรียกใช้แล้ว ต้องมีการกำหนดอินเทอร์เฟซที่จะใช้เป็นตัวจัดการการส่งออบเจ็กต์ที่กำหนดไว้ก่อนหน้านี้ โดย ออบเจ็กต์กลุ่มแรกที่ออกแบบนั้นเป็นออบเจ็กต์สำหรับเรียกเก็บข้อมูล ส่วนออบเจ็กต์ที่ออกแบบครั้งหลังนี้เป็นตัวกำหนดลักษณะการส่งข้อมูลที่ถูกเรียกเก็บไว้ในออบเจ็กต์กลุ่มแรก ในภาพที่ 4.10 (ก) แสดงการ

ออกแบบ IDL โดยทั่วไป มีการกำหนดโครงสร้างข้อมูลแบบ struct เพื่อจัดเก็บชนิดข้อมูลอื่นไว้ด้วยกัน (struct <struct_name>) และมีโอเปอร์เรชั่นสำหรับส่งค่า struct ดังกล่าว (returnType();) และภาพที่ 4.10 (ข) แสดงการเขียน Fast IDL เมื่อมีการกำหนดตัวระบุลำดับสอง จะเพิ่มการกำหนด (data_type)Seq เป็นตัวระบุออบเจ็กต์ลำดับสอง เพื่อจัดกลุ่มข้อมูล struct แต่ละตัวเป็นชุดเดียวกัน (sequence<struct_name>) ซึ่งทำให้ลดจำนวนคำร้องขอที่จะถูกส่งมาให้ระบบเครือข่ายได้ เพื่อโอเปอร์เรชั่น returnType() จะส่งคืนค่า (data_type)Seq แทน struct_name ในภาพที่ 4.10 (ก)

```
module <module_name> {
    ....
    struct <struct_name> {
        (date_type) DataTypeName1;
        (date_type) DataTypeName2;
        ...
    };

    interface <interface_identifier> {
        (struct_name) returnType();
        ...
    };
    ...
};
```

(ก)

```
module <module_name> {
    ....
    struct <struct_name> {
        (date_type) DataTypeName1;
        (date_type) DataTypeName2;
        ...
    };
    typedef sequence<(struct_name)>
                                (data_type)Seq;

    interface <interface_identifier> {
        (data_type)Seq returnType();
    };
    ...
};
```

(ข)

ภาพที่ 4.10 (ก) การเขียน IDL โดยทั่วไป

(ข) Fast IDL หลังการประยุกต์ Secondary Object Identifier

4.4.2 การประยุกต์ใช้แบบแผนตัวทำซ้ำ

ถึงแม้ว่าการเขียนออบเจ็กต์แบบ Secondary Object Identifier จะช่วยลดจำนวนคำร้องขอที่จะเกิดขึ้นในระบบเครือข่าย แต่หากจำนวนข้อมูลที่มีการร้องขอมีขนาดใหญ่ก็จะทำให้เวลาที่ใช้ในการรับส่งข้อมูลมากขึ้น ไม่ว่าข้อมูลเหล่านั้นจะถูกนำไปประมวลผลทางฝั่งไคลเอนท์หรือไม่ก็ตาม แบบแผนตัวทำซ้ำถูกนำมาใช้ประกอบกับการเขียนออบเจ็กต์แบบ Secondary Object Identifier ในข้อ 4.4.1 เพื่อจัดแบ่งปริมาณข้อมูลที่มีขนาดใหญ่ให้เป็นขนาดย่อย รูปแบบทั่วไปของตัวทำซ้ำใน IDL ได้แก่ BaseIterator ในภาพที่ 4.11 เป็นการออกแบบตัวทำซ้ำอย่างง่าย เนื่องจากไม่ต้องมีการพัฒนาส่วนสำรองข้อมูลเพิ่มเติมเพื่อเก็บข้อมูลที่ถูกเรียกมาแล้ว นั่นคือถ้ามีการเรียกข้อมูลชุดถัดไป ข้อมูลชุดใหม่นั้นจะถูกจัดเก็บแทนที่ข้อมูล

เดิม เมื่อนำไปใช้อ้างอิงใน IDL ที่ได้จากข้อ 4.4.1 IDL ใหม่จะมีลักษณะดังภาพที่ 4.12 โดยกำหนดอินเทอร์เฟซสำหรับ Iterator เข้าไปเพิ่มเติม

```
module iterator {
    interface BaseIterator {
                    boolean  next_N();
                    short count( );
                    void destroy( );
    };
};
```

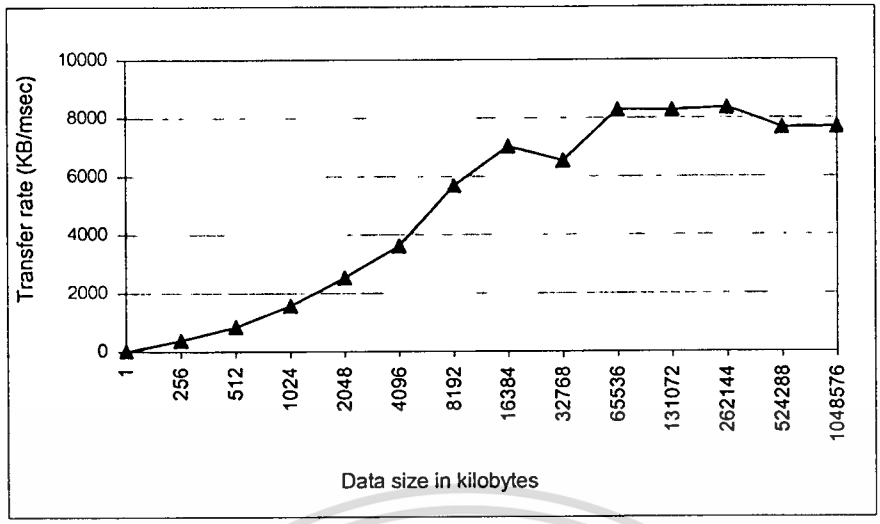
ภาพที่ 4.11 IDL ของแบบแผนตัวทำซ้ำ (Base Iterator)

```
#include "iterator.idl"
module <module_name> {
    ....
    struct  [struct_name] {
        (date_type) DataTypeName1;
        (date_type) DataTypeName2;
        ...
    };
    typedef  sequence<(data_type)>  (data_type)Seq;

    interface <interface_identifier>Iterator : iterator {
        boolean  next_N(in unsigned long how_many,
                        out (data_type)Seq the_list);
        ...
    };

    interface <interface_identifier> {
        <interface_identifier>Iterator  returnType();
    };
```

ภาพที่ 4.12 การประยุกต์ใช้ Iterator Pattern และ Secondary Object Identifier ใน Fast IDL

4.4.3 การกำหนดขนาดชุดข้อมูล

ในการใช้แบบแผนตัวทำซ้ำนั้นต้องมีการกำหนดจำนวนชุดข้อมูลย่อยที่จะส่ง ตามภาพที่ 4.12 โอเปอร์เรชั่น next_N(...) มีตัวแปร how_many กำหนดจำนวนข้อมูลที่จะใช้ส่ง เพื่อให้การทำงานของตัวทำซ้ำได้ผลดียิ่งขึ้น จึงต้องมีการกำหนดขนาดจำนวนข้อมูลที่จะส่งให้อยู่ในปริมาณที่เหมาะสม จากการทดลองในบทที่ 3 พบว่าขนาดข้อมูลตั้งแต่ 1 กิโลไบต์ ถึง 16384 กิโลไบต์ อัตราการส่งข้อมูลเพิ่มขึ้นเป็นลำดับ และจะตกลงเมื่อขนาดข้อมูลเพิ่มขึ้นถึงแม้ว่าอัตราการส่งข้อมูลจะเพิ่มขึ้นอีกครั้ง แต่ก็มิได้นำมาพิจารณาเพราะหลังจากที่อัตราการส่งข้อมูลเพิ่มขึ้นอีกครั้งก็จะตกลงไปอีก ดังกราฟที่แสดงในภาพที่ 4.13 การกำหนดขนาดชุดข้อมูลย่อยที่ต้องการอาจกำหนดในส่วนจำนวน sequence ของ Secondary object identifier หรือในส่วนฟังก์ชั่น next_N(...) ของ iterator



ภาพที่ 4.13 อัตราความเร็วในการส่งข้อมูลขนาดต่างๆ ของ ORB

IDL ที่ได้จากเทคนิค Fast IDL นี้มีแนวโน้มที่จะใช้เวลาในการทำงานน้อยลง เพราะการใช้ Secondary Object Identifier มาใช้จะช่วยลดจำนวนคำร้องขอที่จะเกิดขึ้นในระบบเครือข่าย กรณีที่ผลลัพธ์ที่ได้จากคำร้องขอนั้นมีขนาดใหญ่ ก็จะทำให้เวลาในการทำงานเพิ่มขึ้น จึงนำแนวความคิด Iterator Pattern มาใช้ในการแบ่งปริมาณข้อมูลขนาดใหญ่ออกเป็นส่วนย่อยที่เหมาะสม เพื่อให้ความเร็วในการส่งข้อมูลดีขึ้น

บทที่ 5

การวัดประสิทธิภาพของเทคนิคฟาสต์ไอดีแอล

(Performance Evaluation for Fast IDL Technique)

รูปแบบการเขียน IDL ตามที่ได้กล่าวมาในบทที่ 4 นั้นสามารถนำไปปรับปรุงประสิทธิภาพการทำงานของกลไกการสื่อสารของ CORBA ได้ในระดับหนึ่ง เมื่อนำมาใช้กับการส่งข้อมูลที่มีขนาดใหญ่มากนั้นยังต้องมีการปรับปรุงเพิ่มเติม โดยเทคนิคที่ผู้วิจัยนำเสนอเรียกว่า ฟาสต์ไอดีแอล (Fast IDL) นั้นได้ประเมินความเร็วการสื่อสารของ CORBA ที่เขียนด้วยฟาสต์ไอดีแอลในการรับส่งข้อมูล ดังจะกล่าวในรายละเอียดต่อไป

5.1 การวัดประสิทธิภาพกลไกการทำงานของ ORB

การวัดและประเมินประสิทธิภาพของ ORB เพื่อนำไปใช้พัฒนาระบบงานนั้นจะพิจารณาถึง ระยะเวลาทั้งหมดที่ใช้ในการรับส่งข้อมูล (RTT – Round Trip Time) ปริมาณงานที่ได้ต่อหนึ่งหน่วยเวลา (throughput) และผลการถดถอยของประสิทธิภาพ (performance degradation หรือ scalability) ภายใต้สภาพแวดล้อมระบบทั้งที่มีเครื่องไคลเอนท์เพียงเครื่องเดียวและหลายเครื่องเพื่อนำไปประเมินสภาพการทำงานดังนี้

- ชนิดข้อมูลที่แตกต่างกันจะมีผลต่อระยะเวลาที่ใช้ในการรับส่งข้อมูลอย่างไร โดยพิจารณาชนิดข้อมูลพื้นฐานต่างๆ ชนิดข้อมูลที่เป็นตัวอักษร ชนิดข้อมูลที่ผู้ใช้กำหนดเอง และชนิดข้อมูลแบบอ้างอิงตามออบเจ็กต์
- ขนาดข้อมูลมีผลอย่างไรต่อประสิทธิภาพของระบบ ในรูปแบบชุดข้อมูลตั้งแต่ขนาด 1 ไบต์ไปจนถึง 16384 ไบต์

1) <u>เวลาที่ใช้ในการรับส่งข้อมูล (Round Trip Time)</u>

เวลาที่ใช้ในการรับส่งข้อมูลโดยเริ่มนับเวลาตั้งแต่ไคลเอนท์ส่งคำร้องขอไปยังเครื่องเซิร์ฟเวอร์จนกระทั่งไคลเอนท์ได้รับผลที่ได้จากคำร้องขอคืนจากทางฝั่งเซิร์ฟเวอร์ ซึ่งในการทดสอบจะใช้เวลาของเครื่องไคลเอนท์มาใช้ในการคำนวณ โดยการคำนวณเวลาที่ใช้นั้นจะทำทางฝั่งไคลเอนท์ใช้สูตรดังนี้

$$Elapse\_time = Start\_time - End\_time \quad \text{.....(1)}$$

โดย Start_time คือเวลาเริ่มต้นที่ไคลเอนท์เริ่มส่งคำร้องขอไปยังเซิร์ฟเวอร์

End_time คือเวลาสุดท้ายที่ไคลเอนท์ได้รับข้อมูลตามที่ร้องขอครบ

Elapse_time คือผลต่างของเวลาเริ่มต้นและเวลาสุดท้าย เป็นเวลาทั้งหมดที่ใช้ในการรับส่งข้อมูล

2) <u>อัตราปริมาณข้อมูลที่ส่งได้ต่อหนึ่งหน่วยเวลา (Data Throughput)</u>

อัตราปริมาณข้อมูลที่ส่งได้ต่อหนึ่งหน่วยเวลาหาได้จากการนำปริมาณข้อมูลทั้งหมดที่มีการร้องขอหารด้วยเวลาที่ใช้ในการรับส่งข้อมูล โดยการคำนวณหาปริมาณข้อมูลที่ส่งได้ต่อหนึ่งหน่วยเวลาจะทำทางฝั่งไคลเอนท์ หลังจากไคลเอนท์ได้รับข้อมูลครบตามที่ร้องขอ มีสูตรการคำนวณดังนี้

$$Throughput\_rate = TotalDataSize/Elapse\_time \quad \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

โดย TotalDataSize คือขนาดข้อมูลทั้งหมดที่ทำการส่ง

Elapse_time คือเวลาที่ใช้ในการรับส่งข้อมูลตามขนาดที่กำหนด (TotalDataSize)

Throughput_rate คืออัตราส่วนของปริมาณข้อมูลที่ส่งได้ต่อหนึ่งหน่วยเวลา จากการคำนวณโดยนำขนาดข้อมูลทั้งหมดหารด้วยเวลาที่ใช้

5.2 รายละเอียดการทดสอบประสิทธิภาพ (Details of the Experiments)

ในการประเมินประสิทธิภาพการทำงานของ CORBA จะทำการทดสอบกับระบบเครือข่ายที่ต้องการความเร็วในการประมวลผล เพื่อให้สอดคล้องกับการทำงานของระบบทั่วไปในปัจจุบัน จึงจัดสภาพแวดล้อมในการทดสอบดังนี้

5.2.1 สภาพแวดล้อมของระบบที่ใช้ในการทดสอบ

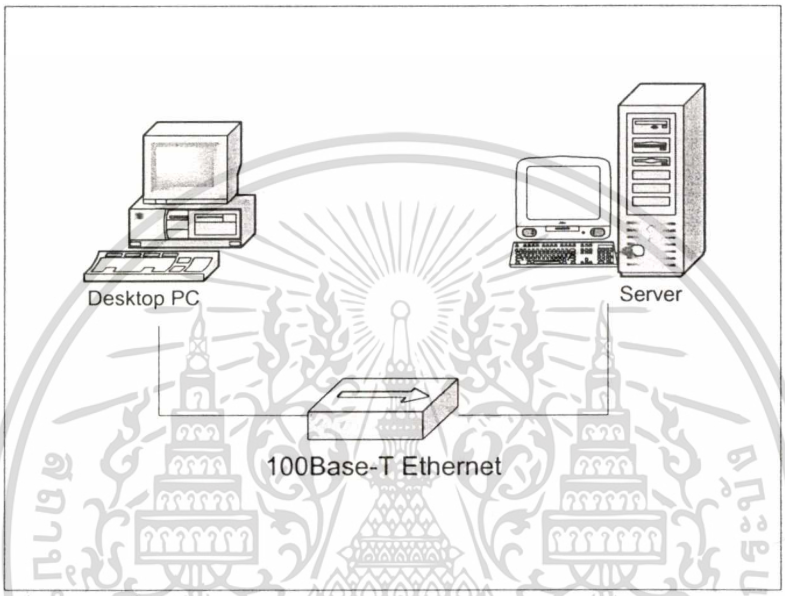
ฮาร์ดแวร์และซอฟท์แวร์ที่ใช้ในการทดสอบมีดังนี้

1) เครื่องคอมพิวเตอร์ที่ใช้ในการทดสอบ

- เครื่องคอมพิวเตอร์ Pentium III 800MHz หน่วยความจำสำรอง 256 MB สำหรับใช้เป็นเครื่องเซิร์ฟเวอร์ 1 เครื่อง
- เครื่องไคลเอนท์ใช้ Pentium III 800MHz หน่วยความจำสำรอง 128 MB จำนวน 1 เครื่อง
- ระบบปฏิบัติการ Windows NT Server 4.0 service pack 6 สำหรับเครื่องเซิร์ฟเวอร์ และระบบปฏิบัติการ Windows 98 สำหรับเครื่องไคลเอนท์

- Visibroker for Java 3.4 เป็นเครื่องมือที่มีโปรแกรม ORB Compliant ของบริษัท Borland
- เครื่องมือที่เป็นตัวแปลภาษา หรือคอมไพเลอร์ (complier) สำหรับการเขียนโปรแกรมด้วยภาษาจาวา คือ Java 2 SDK Standard Edition v1.2.2

2) ระบบเครือข่าย 100Mbps Ethernet เชื่อมต่อด้วยโปรโตคอล TCP/IP และ Ethernet Hub-switching



ภาพที่ 5.1 การเชื่อมต่อเครื่องคอมพิวเตอร์และเครือข่ายในการทดสอบ

5.2.2 วิธีการทดสอบ

เครื่องเซิร์ฟเวอร์และเครื่องไคลเอนท์ต้องมีการลงโปรแกรม Smart Agent ของ Visibroker ทั้งสองฝั่ง ซึ่งเป็นส่วนที่จัดการเกี่ยวกับการติดต่อระหว่างเครื่องคอมพิวเตอร์และการเรียกใช้งานออบเจ็กต์ นอกจากนี้ยังต้องมี Java Virtual Machine (JVM) เวอร์ชั่น 1.2 ขึ้นไป โดยโปรแกรมทางฝั่งเซิร์ฟเวอร์นั้นต้องเปิดให้บริการออบเจ็กต์ที่กำหนดไว้ รวมทั้งออบเจ็กต์ที่เป็น abstract ของ Iterator ด้วย ส่วน โปรแกรมที่อยู่ทางฝั่งไคลเอนท์จะทำการติดต่อและสร้างขนาดข้อมูลที่ต้องการเรียกใช้งานโดยอัตโนมัติ โดยมีขนาดข้อมูลตั้งแต่ 1 – 16,384 กิโลไบต์สำหรับข้อมูลแต่ละชนิดที่นำมาทดสอบ

5.2.3 ลักษณะข้อมูลที่นำมาทดสอบ

เพื่อให้การทดลองมีลักษณะการจำลองการทำงานคล้ายกับสถานการณ์จริง ลักษณะข้อมูลที่นำมาใช้ทดสอบได้แก่ข้อมูลชนิดพื้นฐาน ได้แก่ short, long, float, double, Boolean, char, octet และข้อมูลชนิดผสม โดยใช้ชนิดข้อมูลแบบ struct เนื่องจากการประกาศชนิดข้อมูลแบบ struct นั้นสามารถประกาศชนิดข้อมูลพื้นฐานและชนิดข้อมูลที่ซับซ้อน เช่น ชนิดข้อมูลแบบ sequence

ภายในตัวได้ และประกาศชนิดข้อมูลแบบ sequence ของข้อมูลชนิดพื้นฐาน (sequence<data_type>) ที่ต้องการส่งข้อมูลเป็นชุด นอกจากนี้ยังมีการกำหนดข้อมูลในอยู่ในรูปของ object reference เพื่อเปรียบเทียบการส่งข้อมูลแบบ sequence<object_reference> ดังนั้นการออกแบบ IDL interface สำหรับระบบงานได้ดังตารางที่ 5.1

ตารางที่ 5.1 การแปลงลักษณะข้อมูลที่ใช้งานจริงเป็นรูปแบบโครงสร้างของ IDL

ชนิดข้อมูลที่ใช้งานจริง	ชนิดรูปแบบข้อมูลที่กำหนดด้วย IDL

เรคอร์ด (record) ที่เก็บข้อมูลในตาราง	Struct *Record_Name*{
	boolean b;
	octet o;
	short s;
	long l;
	long long ll;
	float f;
	double d;
	};

ตาราง (Table) ที่อยู่ในฐานข้อมูล	typedef Sequence <*RecordName*> *Table_Name*;

Object reference	interface myObject {
	attribute octetSeq a;
	};

5.3.3 ลักษณะโปรแกรมที่ใช้ในการทดสอบ

โปรแกรมที่ใช้ในการทดสอบเป็นโปรแกรมที่ใช้สำหรับถ่ายโอนข้อมูลตามชนิดข้อมูลต่างๆ ที่ได้กำหนดไว้ โปรแกรมแบ่งออกเป็นส่วนที่ทำงานบนเครื่องเซิร์ฟเวอร์และส่วนที่ทำงานบนเครื่องไคลเอนท์ โปรแกรมที่ทำงานบนฝั่งเซิร์ฟเวอร์มีขั้นตอนการทำงานดังนี้

- การลงทะเบียน Smart Agent เป็น Basic Object Adapter สำหรับออบเจ็กต์ และทำหน้าที่เป็นตัวกลางในการติดต่อระหว่างไคลเอนท์และเซิร์ฟเวอร์

- ออกแบบออบเจ็กต์ตามรูปแบบของ Fast IDL ที่อธิบายไว้ในข้อ 2.1 โดยมีการกำหนดขนาดข้อมูลที่จะส่งในแต่ละครั้งตั้งแต่ 256 กิโลไบต์จนถึง 16384 กิโลไบต์

- พัฒนาออกเจ็กต์ตามรูปแบบที่กำหนดไว้และลงทะเบียนออบเจ็กต์เหล่านั้นไว้ยัง Smart Agent เพื่อเปิดให้ใช้บริการ

ส่วนโปรแกรมที่ทำงานทางฝั่งไคลเอนท์มีขั้นตอนการทำงานดังนี้

- เปิดใช้ Smart Agent ของ Visibroker เพื่อเป็นตัวกลางในการติดต่อเช่นกัน

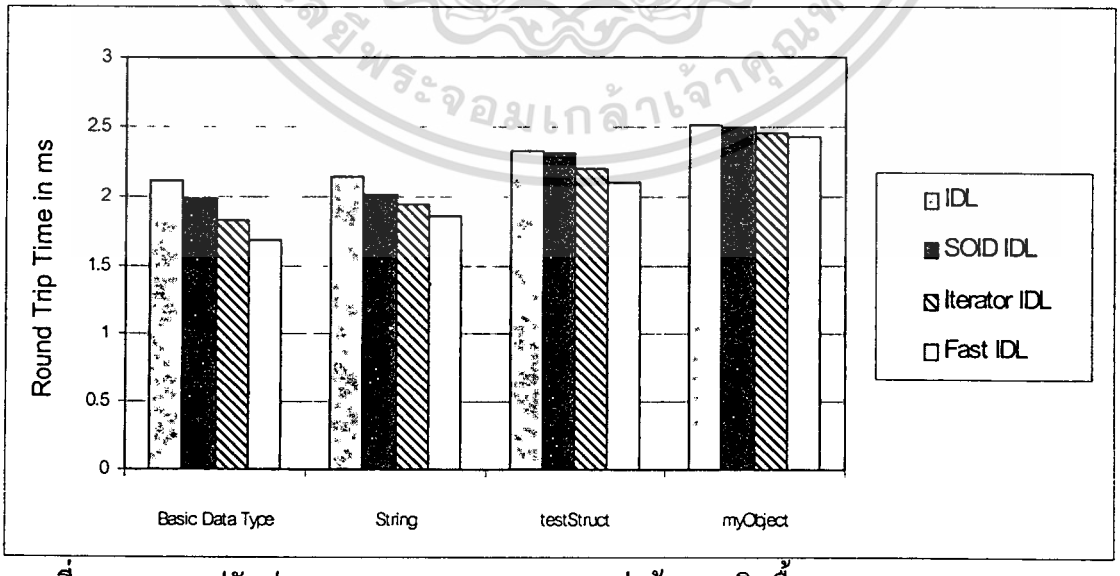
- เมื่อมีการเรียกใช้โอเปอร์เรชั่น จะทำการติดต่อกับ ORB บนฝั่งเซิร์ฟเวอร์ เพื่อตรวจสอบว่าออบเจ็กต์ที่เรียกใช้งานเปิดให้ใช้บริการได้

- เมื่อแน่ใจว่าออบเจ็กต์นั้นเปิดให้ใช้บริการอยู่ จะเรียกใช้งานโอเปอร์เรชั่นที่ต้องการ และทำการวัดและคำนวนระยะเวลาที่ใช้ในการส่งข้อมูล และอัตราการส่งข้อมูลต่อหนึ่งหน่วยเวลาของข้อมูลแต่ละชนิด

5.3 ผลการทดลอง

5.3.1 Round Trip Time สำหรับการรับส่งข้อมูลชนิดพื้นฐานและแบบผสม

จากการวัด Round Trip Time ของข้อมูลพื้นฐานแต่ละชนิดได้แก่ boolean, char, byte/octet, short, int, long, float, and double ซึ่งเป็นการส่งคืนค่าปกติโดยยังไม่มีการกำหนดขนาดข้อมูลนั้น ระยะเวลาที่วัดได้มีค่าใกล้เคียงกัน จึงสรุปเป็นค่าเฉลี่ยของข้อมูลชนิดพื้นฐาน เพื่อเปรียบเทียบกับลักษณะข้อมูลแบบ string testStruct และ myObject โดยเปรียบเทียบการเขียน IDL แบบปกติ IDL ที่อ้างอิง Secondary Object Identifier (SOID IDL) IDL ที่มีการเรียกใช้ Iterator และการเขียน IDL ด้วยเทคนิค Fast IDL ดังแสดงในภาพที่ 5.2 แสดงให้เห็นว่า Fast IDL ใช้เวลาในการส่งข้อมูลชนิดพื้นฐานน้อยกว่า IDL แบบปกติประมาณ 0.4 ms คิดเป็นเปอร์เซนต์ได้ 20% และน้อยกว่า IDL ที่ใช้เทคนิค Secondary Object Identifier หรือ Iterator เพียงอย่างเดียว ซึ่งใช้เวลาในการรับส่งข้อมูลน้อยกว่า IDL แบบปกติเพียง 5.8% และ 13.7% ตามลำดับ



ภาพที่ 5.2 ผลการปรับปรุง Round Trip Time ของการส่งข้อมูลชนิดพื้นฐาน

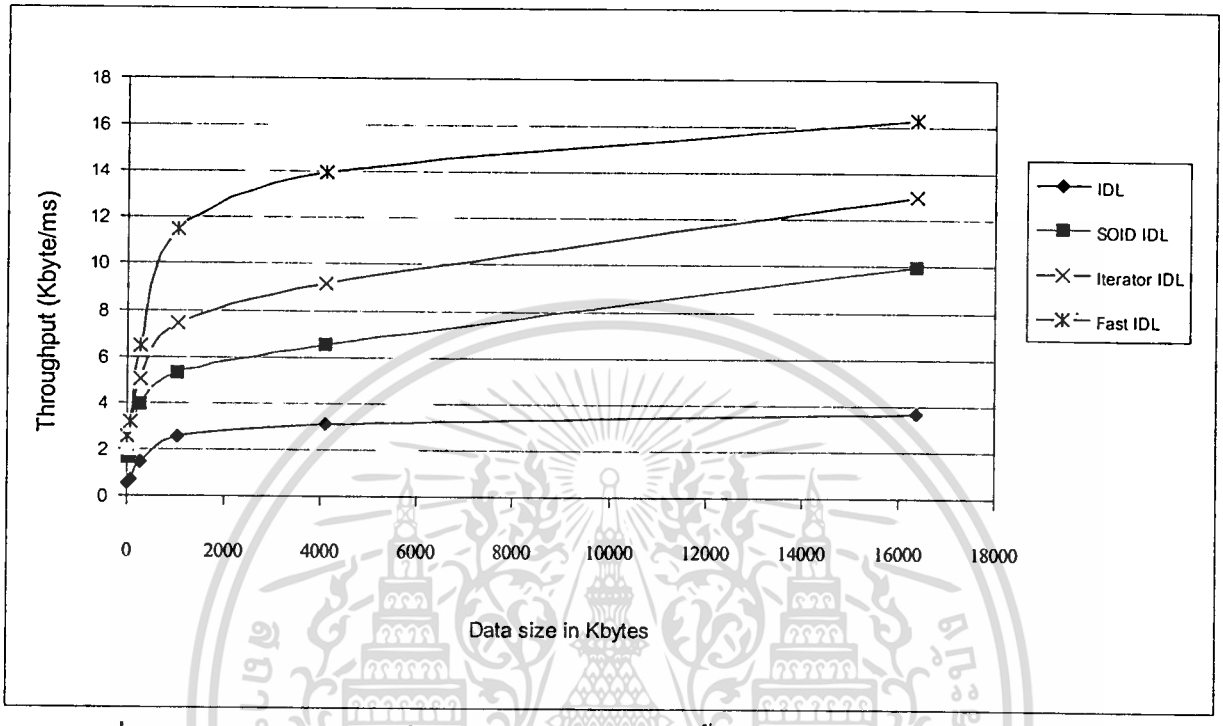
สำหรับเวลาที่ใช้ในการส่งข้อมูลประเภท string นั้น การใช้เทคนิค Secondary Object Identifier พัฒนาจะทำให้เวลาที่ใช้ในการส่งข้อมูลลดลงประมาณ 6% และการใช้ Iterator จะทำให้เวลาที่ใช้ในการส่งข้อมูลลดลงประมาณ 9.5% ในขณะที่เทคนิค Fast IDL ใช้เวลาในการส่งข้อมูลน้อยกว่า IDL แบบเดิมเกือบ 0.3 ms คิดเป็นเปอร์เซนต์ได้ 13% เมื่อเทียบกับการใช้ IDL แบบเดิม ส่วนข้อมูลแบบ testStruct นั้น การใช้เทคนิค Secondary Object Identifier ทำให้เวลาที่ใช้ส่งข้อมูลลดลงไม่มากนัก เพียง 0.6 เปอร์เซนต์เมื่อเทียบกับการใช้ IDL แบบเดิม แต่น้อยกว่าการใช้เทคนิค Iterator และ Fast IDL ที่ใช้เวลาในการส่งข้อมูลลดลง 5% และ 11% ตามลำดับ และสำหรับการส่งข้อมูลแบบ object reference จะใช้เวลาน้อยลงจากการใช้ IDL แบบเดิมประมาณ 3% เมื่อใช้ Fast IDL ส่วน IDL ที่ออกแบบด้วย Secondary Object Identfier หรือ Iterator อย่างใดอย่างหนึ่งจะใช้เวลาในการส่งข้อมูลไม่แตกต่างจาก IDL แบบเดิม นอกจากจากการเปรียบเทียบลักษณะข้อมูลที่มีผลต่อระยะเวลาที่ใช้ในการรับส่งข้อมูล ลักษณะข้อมูลแบบผสม (compound data type) ซึ่งได้แก่ testStruct และ myObject ใช้เวลาในการรับส่งข้อมูล (RTT) มากกว่าลักษณะข้อมูลแบบพื้นฐานอีกด้วย

5.4.2 Throughput ที่วัดได้จากการส่งข้อมูลที่มีขนาดแตกต่างกัน แยกตามลักษณะของข้อมูลแต่ละชนิด

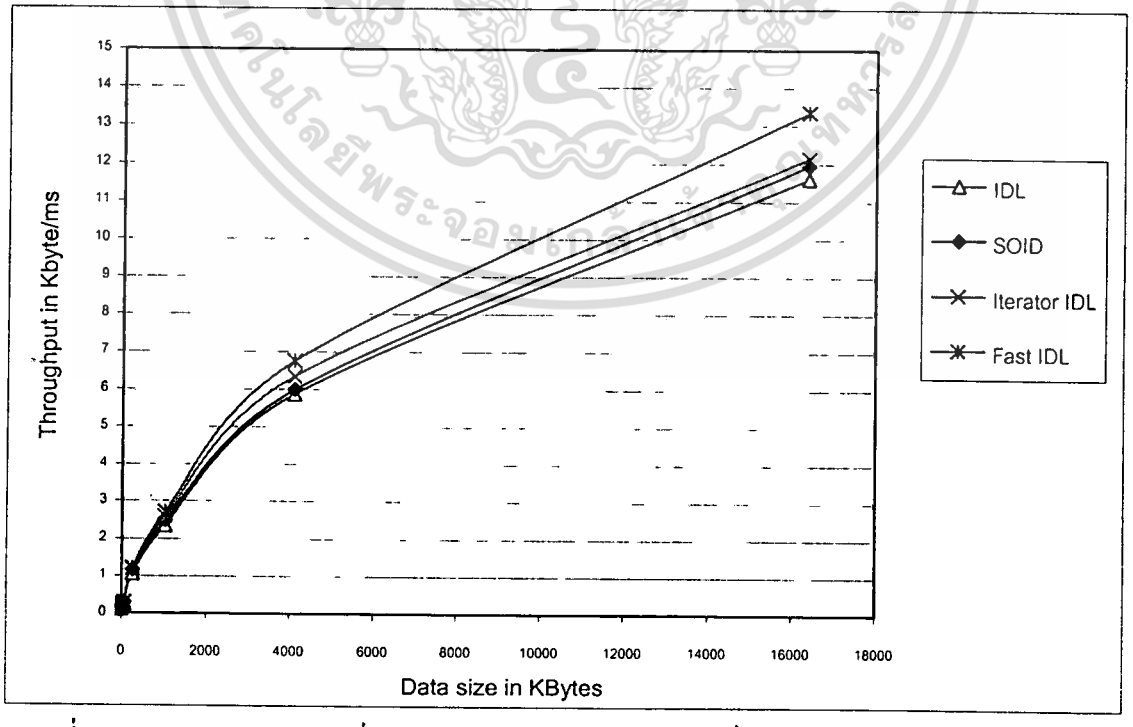
ลักษณะขนาดข้อมูลถือเป็นอีกปัจจัยหนึ่งที่มีผลต่อ Round Trip Time และ Throughput ของ CORBA จากการปรับปรุงกระบวนการ marshalling และ demarshalling ค่าเฉลี่ยของ Throughput ที่วัดได้เปรียบเทียบกับขนาดข้อมูลของชนิดข้อมูลพื้นฐานแบบต่างๆ โดยทำการส่งข้อมูลตั้งแต่ขนาด 1 ถึง 16,384 กิโลไบต์ ดังแสดงในภาพที่ 5.3 Fast IDL สามารถได้ผลสูงสุดเกือบถึง 16 Kilobytes/ms จากเดิมส่งได้เกือบ 4 Kilobytes/ms เท่านั้น เมื่อเทียบอัตราส่วนแล้ว Fast IDL ส่งข้อมูลได้มากกว่าเดิม 4.5 เท่า การใช้เทคนิค Secondary Object Identifier จะได้อัตราการส่งข้อมูลมากกว่าการใช้ IDL แบบเดิม 2.8 เท่า และ IDL ที่ใช้ Iterator จะให้ความเร็วมากกว่าเดิม 3.7 เท่า อย่างไรก็ตามถึงแม้ว่าเทคนิค Fast IDL จะให้อัตราความเร็วในการส่งข้อมูลมากกว่า Secondary Object Identifier แต่ในช่วงขนาดข้อมูลตั้งแต่ 1 – 64 กิโลไบต์ การใช้ Iterator กับ IDL นั้น จะได้ผลใกล้เคียงกับการใช้ Fast IDL แต่เมื่อข้อมูลมีขนาดมากขึ้น Fast IDL จะให้อัตราการส่งข้อมูลมากกว่า

สำหรับชนิดข้อมูลแบบ string นั้น ตามข้อกำหนดของ OMG นั้น IDL complier จะทำการแปลงข้อมูลแบบ string ไปเป็น Java.lang.String ซึ่งเป็นคลาสหนึ่งของภาษาจาวา Throughput ที่วัดได้ก่อนปรับปรุง IDL ถือว่าอัตราการส่งข้อมูลไม่ค่อยดีนัก เมื่อมีการประยุกต์ใช้ Fast IDL ในส่วนการจำกัดขนาดข้อมูลที่จะส่งใน Iterator พบว่า ปริมาณ Throughput ที่วัดได้สูงขึ้นแต่ไม่มากนัก โดยเฉพาะหากสังเกตช่วงขนาดข้อมูลตั้งแต่ 1 – 64 กิโลไบต์ ตามกราฟในรูปที่ 5.4 จะเห็นว่า

Throughput สำหรับข้อมูลแบบ string ที่วัดได้ของการเขียน IDL ด้วย Secondary Object Identifier การใช้ Iterator และเทคนิค Fast IDL จะมีอัตราการส่งข้อมูลใกล้เคียงกับการเขียน IDL แบบเดิม แต่จะเพิ่มขึ้นเมื่อขนาดข้อมูลมากกว่า 256 กิโลไบต์



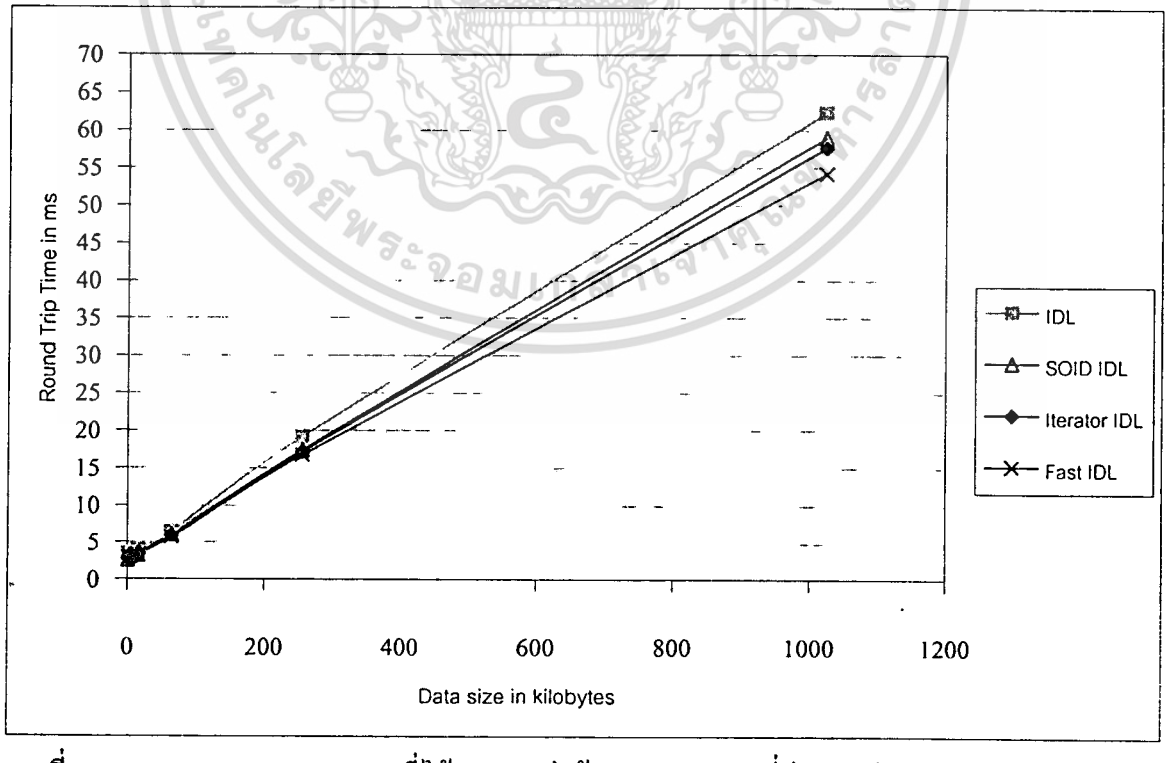
ภาพที่ 5.3 แสดง Throughput ที่ได้จากการส่งข้อมูลชนิดพื้นฐานที่มีขนาดต่างกัน



ภาพที่ 5.4 แสดง Throughput ที่ได้จากการส่งข้อมูลชนิด string ที่มีขนาดต่างกัน

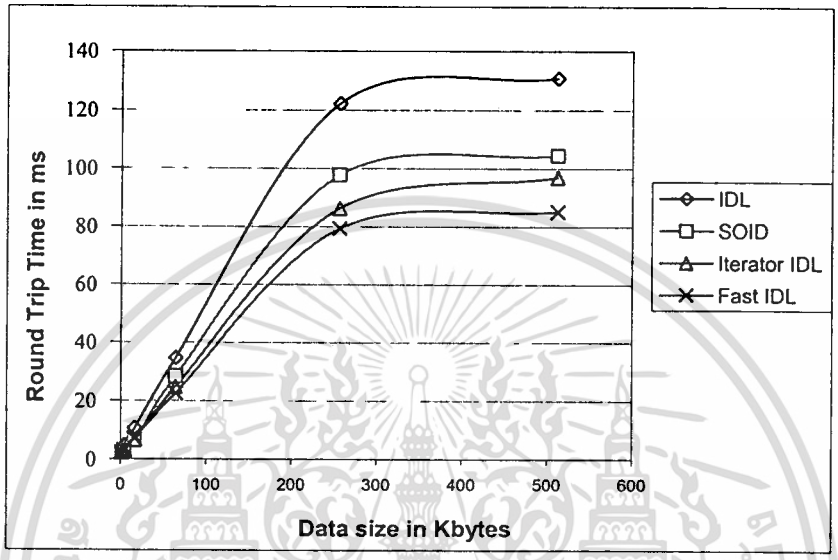
5.4.3 Round Trip Time ที่วัดจากการส่งข้อมูลแบบผสม เปรียบเทียบกับขนาดข้อมูล

ชนิดข้อมูลแบบผสมที่ใช้ในการทดสอบนี้ เป็นข้อมูลที่ผู้ใช้กำหนดขึ้นที่มักพบได้บ่อยในการพัฒนาโปรแกรม ณ ที่นี้แบ่งออกเป็น 2 แบบ คือ testStruct และ myObject สำหรับโครงสร้างแบบ testStruct นั้น IDL จะมองเสมือนเป็นโครงสร้างข้อมูลปกติ โดยสามารถส่งผ่านพารามิเตอร์ด้วยวิธี object-by value ในรูปของ array of testStruct สำหรับชนิดข้อมูลที่กำหนดใน myObject จะส่งค่าผ่าน object reference ในรูปของ sequence ของชนิดข้อมูลที่กำหนดไว้ ทั้ง IDL array และ IDL sequence จะถูกแปลงให้อยู่ในรูป Java array เพราะไม่มีการกำหนดขนาดตายตัวของ Java array ประสิทธิภาพของ Round Trip Time ของลักษณะ array และ sequence ของ testStruct จากการเพิ่มส่วนที่เป็น Iterator และทดสอบกับการจำกัดขนาดข้อมูลตามเทคนิคของ Fast IDL พบว่าระยะเวลาในการส่งข้อมูลที่วัดได้สำหรับขนาดข้อมูลตั้งแต่ 1-64 กิโลไบต์ วัดได้ใกล้เคียงกับระยะเวลาที่ใช้ในการรับส่งข้อมูลด้วยการเขียน IDL แบบเดิม และเมื่อขนาดข้อมูลมากกว่า 64 กิโลไบต์ ระยะเวลาที่ใช้ในการรับส่งข้อมูลจะน้อยลงกว่า 60 ms ซึ่งคิดเป็นเปอร์เซนต์ลดลงถึง 15% ตามกราฟที่แสดงในภาพที่ 5.5 ในขณะที่การใช้เทคนิค Secondary Object Identifier และ Iterator Pattern อย่างใดอย่างหนึ่งในการออกแบบ IDL ทำให้เวลาในการส่งข้อมูลน้อยลงเพียงเล็กน้อย ประมาณ 5 และ 7 เปอร์เซ็นต์ ตามลำดับ แต่ในการทดลองไม่สามารถวัดเวลาที่ใช้ในการรับส่งข้อมูลที่มีขนาดเกิด 1,024 กิโลไบต์ เนื่องจากระหว่างการทำงานทดลองนั้น หน่วยความจำที่ใช้ในการสำรองข้อมูลไม่เพียงพอ



ภาพที่ 5.5 แสดง Round Trip Time ที่ได้จากการส่งข้อมูล testStruct ที่มีขนาดต่างกัน

สำหรับผลการปรับปรุงประสิทธิภาพของชุดข้อมูลแบบ object reference ตามที่กำหนดไว้ใน myObject นั้น ซึ่งพัฒนาการจำกัดข้อมูลในตัวทำซ้ำนั้นกำหนดการจำกัดข้อมูลแบบ object reference เป็นจำนวนออบเจ็กต์ที่จะถูกส่งกลับในแต่ละครั้ง เวลาที่ใช้ในการรับส่งข้อมูลเปรียบเทียบกับขนาดข้อมูลที่รับส่งน้อยกว่าก่อนทำการปรับปรุงรูปแบบของ IDL ดังแสดงในภาพที่ 5.6



ภาพที่ 5.6 แสดง Round Trip Time ที่ได้จากการส่งข้อมูลแบบ myObject ที่มีขนาดต่างกัน

บทที่ 6
บทสรุปและข้อเสนอแนะ

6.1 บทสรุป

การออกแบบ CORBA IDL สำหรับโครงสร้างออบเจ็กต์ถือเป็นส่วนหนึ่งที่สำคัญในการพัฒนาระบบงาน ผู้พัฒนาควรศึกษาเทคนิคการออกแบบทั้งในด้านความยืดหยุ่นของระบบ ประสิทธิภาพของระบบ และการเชื่อมโยงการทำงานร่วมกับแบบแผนโครงสร้างอื่น เนื่องจาก IDL เป็นปัจจัยหนึ่งในการกำหนดข้อมูลที่จะถูกส่งระหว่างการทำงาน จึงมีผลต่อประสิทธิภาพการทำงานของแอพพลิเคชั่นที่พัฒนาด้วยสถาปัตยกรรม CORBA เพราะหากการออกแบบ IDL ไม่ได้ดึงความสามารถที่มีอยู่มาใช้ได้อย่างเต็มที่และเหมาะสม จะทำให้ประสิทธิภาพในการทำงานของระบบลดลง การศึกษาวิจัยนี้จึงนำเสนอการออกแบบ IDL ที่มีชื่อว่า ฟาสต์ไอดีแอล (Fast IDL) การออกแบบออกเจ็กต์ลำดับสอง (Secondary Object Identifier) เพื่อกำหนดออบเจ็กต์สำหรับควบคุมจัดการชุดข้อมูลที่จะถูกส่ง และนำแบบแผนตัวทำซ้ำที่ใช้สำหรับการให้บริการตามเหตุการณ์ (Event Service) มาประยุกต์ใช้ในการออกแบบ IDL สำหรับส่งชุดข้อมูลขนาดใหญ่ผ่านระบบเครือข่าย เพราะการใช้ตัวทำซ้ำจะมีผลในการแบ่งข้อมูลขนาดใหญ่เหล่านั้นออกเป็นข้อมูลขนาดย่อย ซึ่งสามารถยกเลิกการรับส่งข้อมูลจำนวนย่อยที่เหลือหากไม่เป็นที่ต้องการ และการใช้เทคนิคตัวทำซ้ำยังมีผลต่อการจองหน่วยความจำทั้งฝั่งไคลเอนท์และเซิร์ฟเวอร์ ดังนั้นวิธีจำกัดขนาดข้อมูลจึงถูกนำมาใช้เพื่อหาขนาดที่เหมาะสมสำหรับการแบ่งข้อมูลที่มีขนาดใหญ่ออกเป็นขนาดเล็ก จากผลการวัดประสิทธิภาพกลไกการทำงานของ CORBA ที่พัฒนาด้วยเทคนิค Fast IDL เปรียบเทียบกับการใช้เทคนิค Secondary Object Identifier และ Iterator Pattern เพียงอย่างเดียว ขนาดข้อมูลที่เหมาะสมที่ใช้ในการกำหนดส่วนสำรองข้อมูลที่ใช้ในการทดสอบคือ 256 กิโลไบต์ โดยกำหนดผ่านขนาดชุดข้อมูลที่จะส่งในแต่ละครั้งผ่าน IDL และการออกแบบ Fast IDL จะช่วยให้กลไกการทำงานของ CORBA รับส่งข้อมูลได้เร็วขึ้นกว่าการใช้เทคนิค Secondary object identifier หรือ Iterator pattern เพียงอย่างใดอย่างหนึ่ง สรุปได้ดังนี้ เวลาที่ใช้ในการรับส่งข้อมูลพื้นฐานของเทคนิค Fast IDL เร็วกว่าการส่งข้อมูลที่ออกแบบด้วย IDL แบบปกติประมาณ 20% และเร็วกว่าการออกแบบด้วย Secondary object identifier และ Iterator pattern ประมาณ 14% และ 6% ตามลำดับ เทคนิค Fast IDL ยังช่วยให้อัตราความเร็วในการส่งข้อมูลขนาด 256 กิโลไบต์ขึ้นไปสูงขึ้นกว่าการออกแบบการส่งข้อมูลด้วยเทคนิคการออกแบบ IDL แบบเดิมถึง 4.5 เท่า และมากกว่าอัตรา

ความเร็วที่ส่งข้อมูลด้วยเทคนิคการออกแบบ Secondary object identifier และ Itertor pattern ประมาณ 1.7 เท่า และ 2 เท่า ตามลำดับที่ขนาดข้อมูล 4096 กิโลไบต์

การออกแบบโครงสร้างออบเจ็กต์สำหรับการส่งข้อมูลแบบ string ด้วยเทคนิค Fast IDL ทำให้เวลาที่ใช้ในการส่งข้อมูลที่ออกแบบด้วยเทคนิค IDL ปกติถึง 13% ในขณะที่การออกแบบออบเจ็กต์ด้วยเทคนิค Secondary object identifier และ Iterator pattern ทำให้เวลาที่ใช้ในการส่งข้อมูลมากกว่า IDL ปกติประมาณ 6% และ 9.5% ตามลำดับ แสดงให้เห็นว่าเทคนิค Fast IDL ทำให้เวลาที่ใช้ในการส่งข้อมูลแบบ string น้อยกว่าการออกแบบด้วยเทคนิค Secondary object identifier ถึง 2 เท่า และน้อยกว่าการใช้เทคนิค Iterator pattern ประมาณ 1.4 เท่า ส่วนอัตราความเร็วในการส่งข้อมูลของการออกแบบด้วย Fast IDL ส่งข้อมูลได้มากกว่า IDL เดิมถึง 15% ตั้งแต่ข้อมูลที่มีขนาด 256 กิโลไบต์ขึ้นไป ถึงแม้ว่าเทคนิค Secondary object identifier ช่วยให้อัตราการส่งข้อมูลเพิ่มขึ้น 10% แต่ก็ยังน้อยกว่าความเร็วที่ได้จากเทคนิค Fast IDL ส่วนเวลาการส่งข้อมูลชนิดผสมอันได้แก่ struct และ object reference ที่วัดได้นั้นมีแนวโน้มที่จะสูงขึ้น โดย Fast IDL ใช้เวลาในการส่งข้อมูลแบบ struct และ object reference น้อยกว่า IDL แบบเดิมถึง 15% และ 30% ตามลำดับ ซึ่งเทคนิค Fast IDL จะใช้ได้ดีเมื่อขนาดข้อมูลมากกว่า 256 กิโลไบต์สำหรับข้อมูลแบบ struct และ object reference ส่วนข้อมูลที่มีขนาดเล็กกว่า 256 กิโลไบต์นั้น เทคนิค Secondary object identifier จะใช้เวลาในการส่งข้อมูลน้อยกว่า Iterator patter และ Fast IDL

สรุปได้ว่าการออกแบบโครงสร้างออบเจ็กต์ด้วยเทคนิค Fast IDL ช่วยให้ความเร็วในการส่งข้อมูลเพิ่มขึ้น และเหมาะกับการส่งข้อมูลที่มีขนาดตั้งแต่ 256 – 16,384 กิโลไบต์ หากข้อมูลน้อยกว่า 256 กิโลไบต์จะส่งข้อมูลได้ช้า จึงเหมาะสมกับการใช้งานระบบงานที่มีการเรียกข้อมูลขนาดใหญ่จากฐานข้อมูล เช่น สินค้าคงคลัง ข้อมูลหุ้น เป็นต้น และการโอนย้ายไฟล์ทั่วไป ถึงแม้ว่าเทคนิค Fast IDL สามารถนำไปใช้กับการโอนย้ายไฟล์ข้อมูลทั่วไปได้ แต่ไม่เหมาะนำไปใช้กับการโอนย้ายไฟล์ข้อมูลภาพและเสียง (Audio/Video Streaming) เพราะการจำกัดขนาดข้อมูลไม่ตรงกับลักษณะข้อมูลภาพและเสียง ทำให้ความเร็วที่ส่งข้อมูลไม่ดี นอกจากนี้ในการประเมินประสิทธิภาพในงานวิจัยนี้ไม่ได้ทำการทดสอบขนาดข้อมูลย่อยที่มีผลต่อการรับส่งข้อมูล

6.2 ข้อเสนอแนะ

การปรับปรุงความเร็วในการสื่อสารของคอร์บานั้น นอกจากการปรับปรุงการออกแบบโครงสร้างออบเจ็กต์ IDL ด้วยเทคนิค Fast IDL แล้วนั้นยังสามารถปรับปรุงในส่วนการทำงานภายในของ ORB Core อาทิเช่น การทำงานของคอมไพล์เลอร์ที่ใช้ในการแปลงภาษา IDL ไปอยู่ในรูปเฉพาะของภาษาที่ใช้ในการพัฒนาโปรแกรม หรือการปรับปรุงในส่วนการ

ออกแบบวิธีการเข้าถึงออบเจ็กต์ ในส่วนการพัฒนาแอพพลิเคชั่นด้วยสถาปัตยกรรม CORBA นักพัฒนาอาจเพิ่มประสิทธิภาพในการทำงานทางฝั่งไคลเอนท์ โดยใช้เทคนิคแคชชิ่ง (Caching) ทางฝั่งไคลเอนท์ เพื่อเก็บข้อมูลที่ถูกส่งไปในแต่ละครั้งของตัวทำซ้ำ โดยเมื่อมีการค้นหาข้อมูลครั้งต่อไปให้ไปค้นหาข้อมูลในส่วนเก็บข้อมูล (Cache) เสียก่อน หากไม่พบข้อมูลที่ต้องการจึงทำการเรียกตัวทำซ้ำ เพื่อดึงข้อมูลชุดถัดไป

บรรณานุกรม

ภักดี มูซอ และคณะ. 2540. "การเขียนโปรแกรมใช้งานร่วมภายใต้ระบบเครือข่าย." ปริญญานิพนธ์วิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.

Brose, G., Vogel, A., and Duddy, K. 2001. **Java Programming with CORBA**. 3rd Ed. USA : John Wiley & Sons, Inc.

Chau, K. 2002. **Performance Management Start with IDL Design**. [Online]. Available: http://www.trcinc.com/knowledge/articles/ Performance_Mgmt_Starts_With_IDL_Desig-n_TRCInc.pdf

Emerald C. P. et al. 1999. **DCOM and CORBA Side by Side, Step by Step, and Layer by Layer**. [Online]. Avalilable: http://www.cs.wustl.edu/~schmidt/submit/Paper.html.

Farley, J., Crawford, W., and Flanagan, David. Java Enterprise in a Nutshell. 2nd Ed. O'Reilly & Associates, Inc. 2002.

Gokhale, A. and Schmidt, D.C. "Measuring the Performance of Communication Middleware on High-Speed Networks." Proceeding of SIGCOMM '96, Stanford, Calif., Aug. 1996.

Gokhale, A. and Schmidt, D.C. 1998. "Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks." **IEEE Transactions on Computers.** pp.391-413.

Hirano, S., Yasu, Y. and Igarashi, H. 1999. Performance Evaluation of Popular Distributed Object Technologies for Java. [Offline].

Inprise Corporation. 1996. **Visibroker for Java, Programmer's Guide**.

Juric, M.B., Welzer, T. and Rozman, I. 1999. **Performance Assessment Framework for Distributed Object Architectures**. [Online]. Available : http://lisa.uni-mb.si/~juric/ PerfAssessmentFrw.pdf.

Lurie, J. 2002. **Develop a Generic Caching Service to Improve Performance**. [Online]. Available : http://developer.java.sun.com/developer/technicalArticles/ALT/ cachingservices.

Miron, D. and Taylor, S. 2002. **Performance Characteristics of a Java Object Request Broker**. [Online]. Available : http://www.dstc.edu.au/Publications/index.html.

Mishra, S. and Shi, N. 2002. "Improving the Performance of Distributed CORBA Applications." **Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)**. pp. 1-6.

MLC Systeme GmbH and Distributed Systems Research Group. 1998. **CORBA Comparison Project - Final Project Report**. [Offline].

Noriel, N. 2002. "CORBA Design Patterns In Distributed Systems." **Proceedings of MASPLAS'02 The Mid-Atlantic Student Workshop on Programming Languages and Systems**. Pace University. 2002. pp. 17.1-17.11.

Object Management Group. 2002. **The Common Object Request Broker: Architecture and Specification**. Revision 2.3. [Online]. Available : http://www.omg.org/cgi-bin/doc?ptc/99-10-07.pdf.

Object Management Group. 2002. **OMG IDL to Java Mapping Specification**. Revision 2.3. [Online]. Available : http://www.omg.org/cgi-bin/doc?ptc/99-10-10.pdf.

Orbix Research at United Kingdom. 1999. **Performance Improvements in CORBA Applications**. [Online]. Available: http://www.cse.buffalo.edu/pub/WWW/dobs/projects/poster/index.html.

Orfali, R. and Harkey, D. 1997. **Client/Server Programming with JAVA and CORBA**. USA : John Wiley & Sons, Inc.

Rosenberger, J. L. **Teach Yourself CORBA in 14 days**. Indiana : Sam Publishing. 1998.

Slama, D., Garbis, J., Russel, P. 1999. **Enterprise CORBA**. Prentice Hall.

Wolfgang, E. 2000. **Engineering Distributed Objects**. London : John Wiley & Sons, Ltd.

ภาคผนวก

ผลงานวิจัยที่ได้รับการตีพิมพ์

Arveesri Butpo and Bunjong Piyatanrong. "CORBA Object Communication Management." The 2000 IEEE International Conference on Management of Innovation and Technology (ICMIT 2000), November 2000. pp. 748-751.

CORBA OBJECT COMMUNICATION MANAGEMENT

Arveesri Butpo* and Bunjong Piyatamrong**

*Faculty of Information Technology, King Mongkut's Institute of Technology, Ladkrabang, Bangkok, 10520, Thailand
Email: s1067044@kmitl.ac.th
**Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology, Ladkrabang,
Email: bunjong@ce.kmitl.ac.th

ABSTRACT

Distributed Object Technologies (DOTs) was becoming common and widely in the software development. Application and system development using Common Object Request Broker Architecture and Java provides robustness, flexibility, and reusability. So the performance of CORBA IIOP implementation is crucial in high-performance network computing systems. This paper aims at studying and evaluating the object communication mechanism in CORBA environment in order to analyze the object operations compared with the conventional network communication programming, find the key sources for reducing the overhead of communications and improving the communications performance.

Keywords: Distributed Object Technology, Common Object Request Broker (CORBA), Java

1. INTRODUCTION

Distributed client/server computing is becoming common in software development and application with the increasing popularity of the Internet, the Web in the near future. There are various ways in which applications can interact in a distributed, networked computing environment. Examples are low-level sockets, remote procedure calls, CORBA, Java RMI, and some others. Any such mechanism must support interaction between applications running on various operating systems and hardware architectures, and developed using different languages. This is becoming even more important with the increasing popularity of the Internet and the Web. Common Object Request Broker Architecture (CORBA) defines the object request broker (ORB), a standard mechanism, which provides and handles the communication between objects on clients and servers. It includes making information describing the objects in a system and their interfaces available to any objects in the system without regard to operating platforms and host locations. CORBA automates common network programming tasks; for examples object location, object activation, parameter marshalling/demarshalling, framing, and error handling. Also CORBA provides the basis of defining higher layer distributed services, such as naming, events, replication, transactions, and security.

However, the primary drawback of using CORBA ORB as the higher-level communication middleware is the performance of data transferring reduction compared with conventional network programming techniques as C\C++ socket, including the lower throughput because of conventional CORBA ORBs incur significant overhead when delivered large data rates to application. The earlier works pinpointed that some key sources of overhead are from data presentation layer data conversion and data copying. So this paper aims at studying the object communication mechanism in CORBA environment in order to conceive their operations and evaluate its performance in data transfer and the throughput of CORBA ORBs.

2. OVERVIEW OF CORBA

The Common Object Request Broker Architecture, or CORBA is a multi-vendor standard for the distributed object framework, developed by a consortium 700+ companies called the Object Management Group (OMG). It is a set of specifications for providing interoperability and portability to distributed object-oriented applications. CORBA ORBs allow clients to invoke operations on distributed objects without concern for object location, programming language, Operation platform, communication protocols, and hardware equipment. The CORBA core provides basic functions such as remote object connection and remote method call, while the CORBA services and

remote method call, while the CORBA services and facilities arrange extended functions. CORBA services provide the low-level functionality needed by objects such as persistence, naming and directory services, transaction services, and so on, while CORBA facilities provide a high-level framework that can be used by applications to invoke user-level facilities such as compound document management, help facilities, and system administration.

The core of CORBA architecture is the Object Request Broker (ORB) that acts as the object bus over which objects transparently interact with other objects located locally or remotely. The CORBA 2.0 specification particularizes a protocol for the communication between a client and an object server running on ORBs provided by the same or different vendors. The General Inter-ORB Protocol (GIOP) specifies transfer syntax and a standard set of message formats for ORB interoperation. It is simple and scalable, and is designed for implementation on any reliable, connection-oriented protocol such as TCP/IP. The Internet Inter-ORB Protocol (IIOP) specifies how GIOP is implemented over TCP/IP.

The CORBA architecture composes the primary components shown in Figure 1. The responsibility of each component in CORBA is described below [2]:

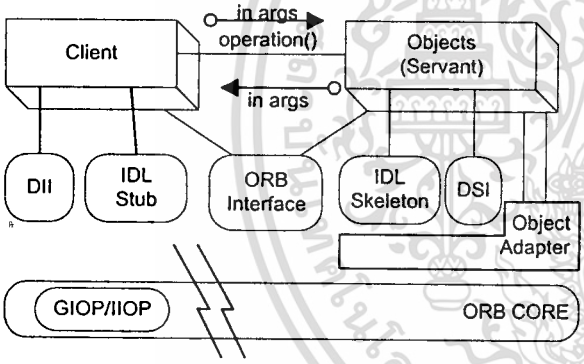


Figure 1 Components in the CORBA Reference Model

• Servant: This component implements the operations defined by an OMG Interface Definition Language (IDL) interface and uses object-oriented programming languages like C++ and Java. A servant is identified by its object reference, which identifies the servant in a server process.

• Client: The program entity invokes an operation on the servants. Accessing the services of a remote object should be transparent to the caller.

• ORB Core: When a client invokes an operation on the servant, the ORB Core is responsible for delivering the request to the servant and returning a response, if any, to the client. For remote servants executing, a CORBA-compliant ORB Core communicates via the GIOP and IIOP, which runs over the TCP transport protocol.

• ORB Interface: An ORB is a logical component that may be implemented in various ways. To decouple applications from implementation details, the CORBA specification defines an abstract interface for ORB. This ORB interface provides standard operations that convert object references t strings and vice versa.

• CORBA IDL stubs and skeletons: CORBA IDL stubs and skeletons serve as the "glue" between the client and server applications and ORB. Stubs provide the static invocation interface (SII) that marshals application data into a common packet-level representation and demarshal the representation back into typed data to an application.

• Dynamic Invocation Interface (DII): The DII allows a client access the request transport mechanisms provided by the ORB at run time. It serves as a generic interface that does not require stubs.

• Dynamic Skeleton Interface (DSI): DSI binds incoming method calls for such object that do not have IDL-based compiled skeletons at run time. The DSI is the server equivalent of a DII. It can receive either static or dynamic client invocations.

• Object Adaptor (OA): Object Adapter provides the core run-time functionality required by servers. It is on top of the ORB's communication services and accepts requests for service on behalf of the server's objects. It demultiplexes incoming requests to the servant and dispatches the appropriate operation apical on that servant.

In the following sections, Figure 2 illustrates the overall architecture of CORBA, which divided into the primary three layers [6]. The top layer is the basic programming architecture, which is visible to the developers or the clients and object server programs. The middle layer is the remoting architecture, which transparently makes the interface pointers or object references meaningful across different processes. The bottom layer is the wire protocol architecture, which further extends the remoting architecture to work across different machines or platforms.

At the top layer, the basic programming architecture, the client and the server programs communicate as if they reside in the same location. A server object can be activated and obtain its object reference by using bind() operation. ORB starts a server that contains that object and then the server initializes all supported objects. Server will registers all objects with Basic

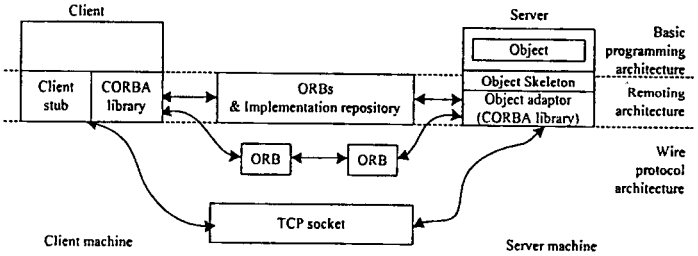


Figure 2 The overall architecture layers of CORBA

0-7803-6652-2/2000/$10.00

Object Adapter (BOA) to tell ORB that it is ready to accept client requests.

The middle layer consists of the infrastructure necessary for providing the client and the server with the illusion that they are in the same address space. To send data across different address spaces requires a process called marshaling and unmarshaling. Marshaling packs a method call's parameters (at a client's space) or return values (at a server's space) in the Common Data Representation (CDR) format before putting it into a standard format for transmission. Unmarshaling, the reverse operation unpacks the standard format to an appropriate data presentation in the address space of a receiving process.

The bottom layer specifies the wire protocol for supporting the client and the server running on different machines. When the client-side ORB gets a request, it consults a locator file to choose a machine supported and sends the request to the server-side ORB via TCP/IP. The server is started by the server-side ORB, the object is instantiated and BOA creates a socket endpoint. For clients talking the IIOP protocol, the server generates an Interoperable Object Reference (IOR) that contains a machine name, a TCP/IP port number, and the object reference. The BOA registers the object reference with the ORB and the object reference is returned to the client side to establish a socket connection to the server. The request is sent to the target server through the established socket connection. The target skeleton is identified by the object reference ID. After invoking the actual method on the server object, the skeleton marshals the return values and sends them via IIOP throughout TCP/IP port number.

3. PERFORMANCE EVALUATION

This experiment has focused on measuring the throughput and round-trip times of data transferring of widely CORBA ORB product, Visigenic's Visibroker 3.3. Although there are many CORBA products, Visibroker is one of the most popular. To evaluate CORBA in likely be the common environment for high-performance network computing, the client program was run on a separate server computer. All PCs were connected via 100 Mbps Ethernet. The environment was as follows:

- CPU: 2 Pentium II 350 MHz computers with 128 MB memory for server and 64 MB for client
- OS: Windows NT Workstation 4.0 SP 4
- Network: 100 Mbps and TCP/IP
- Java VM: Symantec's Visual Café 3.1
- Java Compiler: Symantec's JIT 1.1.x
- Visigenic's VisiBroker for Java 3.3 (Java-implemented CORBA IIOP)

This testbed mention some observations about the ease of writing programs using CORBA but it is difficult to express ease-of-use in value.

3.1 The Testbed Environment Setting

This section describes the CORBA testbed and out outlines the experimental methods. To compare the times used to pass data, byte stream data between hosts is compared between several implementations of CORBA and other lower-level mechanisms like sockets. Byte stream data is representative of applications like bulk file transfer. However, byte stream array does not test the overhead of presentation layer conversions because untyped data is not necessary to be marshaled or demarshalled. So to test how fast does an ORB transfer data; data acquisition systems require high bandwidth of data transfer. The uses of typed data like octet in sequence is representative of file transferring. In addition, measuring typed data transfer reveals the overhead of presentation layer conversions and data copying for the communication middleware mechanisms used.

3.2 Experimental Results

For the evaluation of data transfer, there is used a method that take a byte array as arguments. The size of array is undefined at the compile time. The Java Socket version used the DataOutputStream class with the BufferedOutputStream class to act like CORBA. The buffer size was 32 KB. The C socket version is equivalent to the Java socket version, but written in

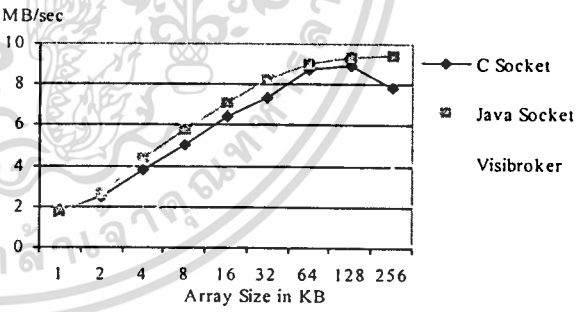


Figure 3 Byte in array

C. It does byte reordering for the network byte order. Figure 3 shows the results of byte array transfers respectively. The horizontal axis indicates the size of an array to be transferred to a remote method in KB. The vertical axis is the bandwidth of transfer in MB/sec. For byte array transfer, C Socket showed very high data transfer rate and the performance of Java Socket is similar too. Their average was at near 9.4 MB/sec. In contrast, the data transfer rates of Visibroker was low. These findings illustrate that as channel speeds increase, the performance of the CORBA implementations become worse relative to that achieved by low-level communication middleware since the overhead of presentation layer

0-7803-6652-2/2000/$10.00

conversions and data copying become increasingly dominant.

When evaluated the performance of only Visibroker, the data transfer rate of octet stream like file transferring is shown in Table 1.

Table 1: Data transfer rate of Octet stream.

Octetstream (KB)	Times (msec)	Rate (KB/msec)
1	0.711	1.406
256	0.661	387.272
512	0.601	851.913
1024	0.681	1572.964
2048	0.811	2525.277
4096	1.132	3618.374
8192	1.442	5680.998
16384	2.333	7022.717
32768	5.01	6540.519
65536	7.91	8285.208
131072	15.83	8279.975
622144	31.34	8364.518
524288	68.30	7676.252
1048576	136.2	7698.796

The data transfer rate is increasingly significant. When increasing the data size, it is added in double of round-trip times. It shows that unless a large number of complex data is passed, the constant overhead of an invocation overshadows the impact of data sizes. Hence, it is imperative to eliminate this overhead so that CORBA can be used effectively to build flexible and reliable middleware capable of delivering very high data rates to applications.

4. CONCLUSIONS

Although CORBA is good for distributed object computing, especially to allow many request invocations in the same communication path. From the evaluation above, CORBA is much slower than conventional network programming techniques because of data handling specific; for examples, parameter marshalling prepares data for transfer between the client and server. Moreover, parameters must be copied between multiple layers of transfer. The CORBA implementation performed worse when sending large and complex data because of excessive copying and marshalling/demarshalling overhead and write resulting from small size write-buffers. The results in this paper indicate that the efficient optimization need to be applied to the CORBA client-side stubs and server-side skeletons to reduce the marshalling, data copying, and overhead. Including the buffer management that is very important for high-performance network computing. In byte transfer test, the cost for data copying degraded the bandwidth largely for CORBA. The further study is to improve the integrating techniques and tools that simplify application development, optimize application performance, and measure application processes in order to pinpoint and alleviate performance bottlenecks.

REFERENCES

[1] A. Gokhale and D.C. Schmidt, "Measuring the Performance of Communication Middleware on High-Speed Networks," presented at the IEEE Conference on **Proc. SIGCOMM '96**, Stanford, Calif., Aug. 1996.

[2] A. Gokhale and D.C. Schmidt, 'Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks', **IEEE Transactions on Computers**, April 1998, p.391-413.

[3] J. L. Rosenberger, **Teach Yourself CORBA in 14 days**. Indiana: Sam Publishing, 1998.

[4] MLC Systeme GmbH and Distributed Systems Research Group, "CORBA Comparison Project - Final Project Report," Online. http://www.kav.cas.cz/~buble/corba/comp/. June 17, 1998.

[5] Orbix Research at United Kingdom. "Performance Improvements in CORBA Applications," Online. http://www.cse.buffalo.edu/pub/WWW/dobs/projects/poster/index.html. March 3,1999.

[6] P. Emerald Chung ET all, "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer," Online. http://www.cs.wustl.edu/~schmidt/submit/Paper.html, September 30, 1999.

[7] R. Orfali and D. Harkey, **Client/Server Programming with JAVA and CORBA**, John Wiley & Sons, Inc., 1997.

[8] Satoshi Hirano, Yoshiji Yasu and Hirotaka Igarashi, "Performance Evaluation of Popular Distributed Object Technologies for Java," Online, December 15, 1999.

 0-7803-6652-2/2000/$10.00

ประวัติผู้เขียน

ชื่อ – สกุล	นางสาวอาวีศรี บุตรโพธิ์
วันที่เกิด	18 พฤศจิกายน 2519
สถานที่เกิด	จังหวัดกรุงเทพมหานคร
วุฒิการศึกษา	วิทยาศาสตรบัณฑิต (วท.บ.)
	สาขาวิทยาการคอมพิวเตอร์
	มหาวิทยาลัยมหิดล
	สำเร็จปีการศึกษา 2540